The Distributed Processing Programming Executive (DPPX) is a new, full-function operating system designed to support distributed processing with the IBM 8100 Information System. The functional requirements of distributed processing and their solutions in DPPX are discussed. The structure of the operating system is outlined, and its advantages are analyzed. Highlighted are particular characteristics of the DPPX structure that uniquely support distributed processing.

An operating system for distributed processing—DPPX

by S. C. Kiely

Small system architectures embodied in mini- and microprocessors offer significant price/performance gains for certain usages. These systems have made technically and economically feasible the distribution of processing among several smaller processors in place of one larger processor, and offer reliability and availability advantages because of the reduced scope of the effect of the failure of any one component. These technological advances have made possible the satisfaction of requirements that have been evolving in the data processing user community over the past several years. The motives for this evolution are diverse and wide-ranging. Large centralized computers often tend to become overloaded, to the extent that some displacement of their workload is required. Placement of processing and data closer to the end user can more closely reflect the organization of business responsibilities in the organization of the data processing system. Evolving needs and technologies have now come together in the low-cost, full-function distributed processing systems embodied in the IBM 8100 Information System and the Distributed Processing Programming Executive (DPPX) operating system. This article discusses the requirements imposed on an operating system intended to support distributed processing and describes the approaches taken with DPPX to deal with them.

Distributed processing requirements

DPPX was designed using a definition of distributed processing that reflects much of the work done on distributed processing requirements by the GUIDE Futures division[1] and in the SHARE DDP Project.

The essential properties of distributed processing were assumed to be the following:

- Both function and data are dispersed in a network (not necessarily geographically dispersed);
- Function and data are dispersed on a coordinated basis, with the coordination performed by one logical manager in the network;
- Nodes may be autonomous to some extent, but ultimately are interdependent with other nodes for coordination and control, and in some cases for other function or data.

Analysis of this definition has resulted in the identification of the specific functional requirements, discussed in the following paragraphs, for a distributed processing system.

Centralized management and control functions. The functions of the system associated with managing or controlling the system—such as data base administration or program library control—have to be provided in a way that permits a single central site to perform these functions for a network of distributed processors. This capability permits an enterprise to implement meaningful control while at the same time dispersing the data processing closer to the end user.

Centralized maintenance and problem determination facilities. The distributed system cannot require specialized data processing skills at dispersed sites for reasons of geography and economics. However, data processing systems must be maintained and errors corrected or circumvented. With the present state of the art, these operations require the participation of trained personnel. Consequently, facilities are needed that provide the DP professional with necessary access to the dispersed systems and their data.

Interactive transaction support. The applications for a distributed computer are essentially the same as those for the larger, centralized computer. In general, the difference between the systems is that distributed applications are simpler and require fewer operating system options. Both batch and interactive applications will be implemented for distributed configurations. As is the case for larger systems, most new applications will be interactive transaction applications. Some applications require that stored data records be maintained with integrity, so that the end user can rely on

the result of an update operation. Many applications also require complete recoverability of data, so that damage to data can be repaired without loss of any completed updates. These requirements have been satisfied traditionally by supporting applications as *transactions*, with a defined *scope of recovery* associated with each transaction.

Distributed communications. Extensive interconnection capability is required at a distributed node, including communications with peer systems, with hierarchically defined hosts, and with terminals. For migration and marketability, a wide variety of terminal types and line disciplines must be supported. The distributed system must also be very flexible, and network control must function dynamically, without requiring the system to be stopped to introduce logical or physical configuration changes.

Support for distributed data. Applications must be able to access or update data stored at host or peer systems. Data access may be required interactively in real time, or the application may transfer data in bulk in a batched mode.

Ease of installation, operation, and maintenance. The end users of distributed systems should not be required to be skilled data processing professionals. Therefore, the design must support the operation and use of the system—as well as its installation and support—in a simple and straightforward way. The system cannot require a complicated SYSGEN process to be installed. The end user cannot be required to undertake extensive training to operate the system. The application of maintenance—both system corrections and application program revisions—must be accomplished without knowledge of programming or complex tools.

Ease of application development. Distributed processing is partly the outgrowth of continuing improvements in computing price/performance as technology has advanced. Another implication of this trend is that the cost of providing end-use data processing applications is shifting from hardware costs to application programming costs at an increasing rate. To the extent that application programmer productivity is the principal bottleneck to producing new applications, high productivity of the programmer is a principal objective of the system.

Growth and migration. The collection of distributed nodes represents a significant user investment that must be protected from changes in the application and the system configurations that will certainly occur over time. Change, as a necessary adjunct to application growth, must be accommodated by the system in a forgiving, flexible, easy-to-implement fashion. Migration from an existing installed base of applications to new, unforeseen applications must also be supported.

Operating system structure. The structure must support all the other user requirements and generally must offer a high degree of flexibility and an absence of constraints. This leaves the user with the ability to choose the desired level of function, to configure the system's function to meet specific needs of the application program, and to modify and extend the system. The flexible operating system offers stable and well-defined programming interfaces.

DPPX as a distributed system

The Distributed Processing Programming Executive (DPPX) has been designed to meet these requirements. It provides a comprehensive set of control program services and input-output services, with high-level capability for interactive command processing, transaction processing, simple batch processing, and several facilities for interconnection with other processors, particularly System/370.

Facilities for central management, control, maintenance, and problem determination are provided in conjunction with two System/370 Program Products, the Distributed Systems Executive (DSX) and the Host Command Facility (HCF). DSX provides a general capability for bidirectional data set transfer between the 8100 Information System and System/370, and HCF provides connectivity for a System/370-attached IBM 3270 keyboard/display with the interactive command facility of DPPX. In DPPX, programs, display maps, program corrections (PTFs), configuration definitions, data set definitions, etc.—all system objects—have been implemented as data sets. Therefore, the DSX capability in fact provides system definition and content control from the System/370. Because DPPX implements a single command language for user access to all system functions, the HCF actually provides access to the entire range of local DPPX capability from the System/370 terminal, including DPPX operator function. The effect of DSX and HCF, in concert with DPPX, is that of a unified, single multinode system with a single resource management system.[2]

Requirements for transaction processing are addressed by the DPPX Data Base and Transaction Management System (DTMS), a licensed program that, in close cooperation with the base DPPX, provides full transaction scheduling and transaction data recovery and backout capabilities.

Distributed communications and data support are offered by DPPX through the combination of rich, device-independent communications and data management support local to an 8100, with application-to-application communications interfaces for a host System/370 and other 8100 systems.

Extensive interactive application development tools are provided under an Interactive Command Facility (ICF), including full-screen editing, interactive display map definition, language translators, interactive debugging, and the DPPX Development Management System (DMS), a licensed program that provides a question/answer mode of interactive program development.

The 8100 with DPPX has a number of advanced design features required for ease of installation, maintenance, and operation in a distributed environment. For example, DPPX is designed to be installed directly, without a systems generation (SYSGEN) process or off-line assembly of tables or system code. Tuning, selection of options, or setting up of the configuration can be done on line during production operation. In a few cases (primarily those related to tuning), the system must be reloaded (IPL) to make the changes effective.

Dynamic change of system configuration, transaction processing, and data base definitions illustrates the power of the DPPX design features. A device type, such as tape drive support not previously used, can be defined to the system on line during production operation. DPPX dynamically builds the required control program support for the device type. Adding new data bases or transactions can be done on line during production operation as well. A new transaction processing program and transaction can be added for testing during production operations and tested with production data bases. After completion of the test, any data base changes are reset.

Single definition of resources has been incorporated. Previous systems with transaction processing have frequently required that resources be defined several times. For example, a data base may have been defined to both the transaction processing system and (as a data set) to the operating system. Similarly, terminals, users, and programs may have been defined to the transaction processing system, to the operating system, and to other components. In DPPX, however, resources are defined once, and all components share this common definition. The transaction processing system in DPPX, that is, DTMS, has no knowledge of the network or of terminal devices. It uses the authorization and user characteristics provided to the operating system, and, in turn, provides the operating system with information about its data bases.

The result is simpler system software and greater ease of setting up or modifying the system. Also, a *programmed operator*, which is a system application, can be written to handle exception conditions or to guide the operation of the system. The structure of DPPX allows any terminal, as well as a local or remote 8100 application, to control the operation of the system. This application

can handle as many conditions as it chooses, sending others to a terminal or printer for human operator action.

Restart after a failure is fully automatic. After each IPL, DPPX and its optional transaction processing component, DTMS, check whether the system closed properly the last time. If not, action is automatically taken to back out incomplete transactions.

Problem determination and hardware diagnostics can be performed on line, during production operation.

All these functions, including full operation of the system, can be performed by a remote host System/370 terminal operator. Management of change (changing the configuration, replacing programs, applying fixes, adding new programs) can be done centrally without user action at the distributed sites. Changes can be applied to one or to many systems, with the central system keeping track of the status of each system and ensuring that the changes are made even if a failure occurs.

These features provide the ability to install and operate an 8100 with minimal impact upon users and personnel at distributed sites, and, consequently, with greatly lessened personnel cost associated with the remote systems. By designing DPPX so that change can be dynamic, largely unattended operation is possible and most service and administrative functions can be performed dynamically on line by a small central group at a host system. Users embarking on distributed data processing can gain the benefits of economy of scale in operations personnel associated with any data processing installation configuration.

The remaining requirements for growth and migratability and the structure of the operating system are the subjects of the rest of this article.

DPPX structure

DPPX is constructed as a hierarchically defined set of *layers*. Layers are logically self-contained elements of the system, with defined roles that can be thought of as levels of abstraction[3,4] within the system providing increasing independence and decreasing awareness of details to the using programs as one advances vertically upward through the hierarchy.

Within practical limits, functions are not duplicated among layers. Thus if one layer provides a capability, other layers use that layer when the same capability is needed.

The elements of the system, then, relate hierarchically to one another. In addition, the design of each system element has been

carried out with independence from other elements. Specifically, this means that data relationships among elements are controlled and formal (as contrasted with implicit and informal, as with many System/360 and System/370 operating system designs). Furthermore, an element must have what Myers calls *strength*[5] in its functional definition, so that it can be both conceptually and physically separable from other elements.

structural advantages of layering

Because the function requirements for distributed data processing are so extensive and broad, economy of design is crucial to making the overall objectives achievable in a practical manner. The layered structure of DPPX avoids redundancy, thereby ensuring consistency while reducing total effort required to provide function. Moreover, the separateness of system elements and the layered structure lend an intellectual manageability to the product that reduces the complexity of developing and later supporting or extending the product. By avoiding redundant function (and redundant user interfaces), the structure also contributes to an inherent simplicity that supports easy user-program design and development. For example, DPPX data management has only one data-recording format supporting all modes of data access by applications—sequential, relative, indexed, and recoverable data base support. As a result, all the DPPX utilities, both on-line and stand-alone, that support the manipulation of data can be common across the different logical organizations of the data. The result is fewer commands to learn and use in developing and supporting one's applications.

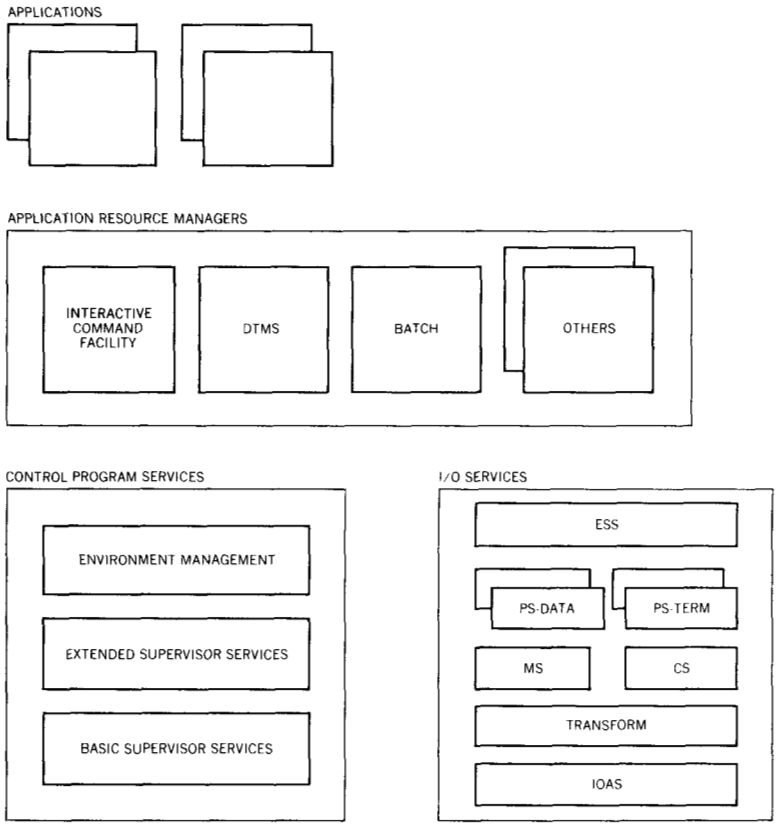
control program design

Figure 1 depicts three major structural areas of DPPX: Control Program Services, I/O Services, and Application Resource Management.

The Control Program is structured into three layers of function. The *base layer* creates and supports the tasking structure of DPPX and supplies the synchronization services that are implemented in DPPX through queues and locks. A higher layer of *extended supervisor functions* is based on the task structure and supports the management of processor storage, logical storage, program contents, timer services, and error management. A top layer of *resource management functions* (called "environment management" in DPPX) supports the allocation and deallocation of resources and the creation and termination of the *environment*, a collection of resources that is the basis for resource allocation in DPPX.

The structure of DPPX separates the allocation of resources from the dispatching unit or task (DPPX uses the term *thread* to describe the dispatching unit.) In earlier systems, the task served both purposes, thereby imposing unnecessary constraints on the program's flexibility in accessing resources. The DPPX thread may

Figure 1 DPPX structure



in the course of its execution change the basis for its resource accounting from one resource pool to another. For example, a system function, such as the Data Base Manager, can change from its own environment, i.e., resource pool, to the user's environment without having to switch execution to a different task.

I/O services design

The I/O Services are consistent with the functional layering of the SNA architecture. Several illustrations of the advantages of careful layering of function can be seen by examining the properties of the DPPX I/O Services design,[6] as shown in Figure 2. The layered I/O structure supports three facilities: (1) access to stored data on disk, diskette, and tape; (2) access to presentation media (terminals and printers); and (3) data exchange among application programs. The problems to be solved in each case are significantly different. Although their solutions in DPPX are structured similarly, the roles of the layers in the three cases are different.

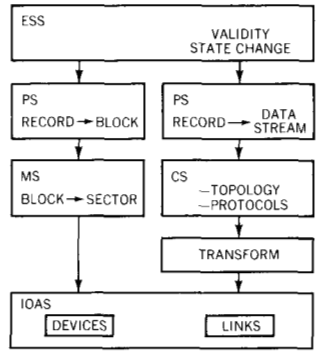
The layers of I/O Services should be viewed as independent and configurable. Each layer is designed autonomously, presuming

only the logical function and interface of the layers below it. This design allows layers to be added, so as to vary the function perceived by the application. Layers can be eliminated, so that unneeded function is omitted (not just bypassed). Also, layers can be replaced without interacting with subordinate or superior layers. For stored data, the layers have the following roles.[7]

External Support Services (ESS) adapt the application to the layers below it, handling changes in mode from application mode to privilege mode, validity checking, and synchronization with the application. This processing is organized into a separate layer so that the other layers can use each other without the overhead required for this processing. Thus mode changes, checking, etc. are performed only once per application request.

Presentation Services (PS) provide the application program with a logical view of data. For example, records of a sequential (entry-sequenced) file on a disk may be stored at random locations on the storage media, wherein the sequentialness is logical. The same stored collection of records with the same physical organization can be viewed by an application as being randomly addressable on the basis of relative record numbers, that is, a different logical view of data. Thus, a different logical view of the data is provided by a different Presentation Service. Note that within the DPPX structure, the DTMS Data Base Manager is also a PS layer. As a result, Data Base support can be provided for batch programs as well as for interactive DTMS programs.

Figure 2 I/O layers



Media Services (MS) provide the common management functions associated with the storing and retrieving of data and provide for the sharing of direct-access data volumes on a data set basis. This layer is the foundation for all stored data management in DPPX. The MS interface is expressed in terms of a byte-addressable sequential space with storage and retrieval performed in units of transfer called logical blocks. This interface is device-independent within either of two classes of devices, as represented by tape and disk. The result is that the access methods in DPPX (i.e., Presentation Services) are structurally independent of unique service characteristics and geometry, and therefore need not be modified when a new device is added to the system.

I/O Attachment Services (IOAS) support physical I/O. This layer supports a channel program-like device-dependent interface to the using layer or application, similar in function to the OS or DOS EXCP. It allows for full exploitation of all device features and characteristics, at the cost to the application of providing for all device and data management function. In the case of stored data devices, the user of the IOAS interface is the owner of the device and handles any sharing of the device.

For data presentation (keyboard/printers, keyboard/displays, printers, card reader/punches) and data exchange (with other programs), the layers are different from those for stored data. The overall structure, however, is symmetric with that of stored data. Layers for communications are discussed in the following paragraphs.

Presentation Services (PS) manage the application's logical view of the data and map the application data record (logical record) to the appropriate transmission record (e.g., the IBM 3270 display data stream) for presentation at the device or for delivery to another application. There are a variety of PS layers provided for DPPX, including the Distributed Presentation Services (DPS) licensed program that supports displays and printers in a manner analogous to IMS Message Formatting Services (MFS) or CICS Basic Mapping Service (BMS). These Presentation Services have user-defined maps to convert between the application logical record and the device data stream. Other Presentation Service layers handle a line-at-a-time format for printers, keyboard/printers, and keyboard/displays (called PS-line/page) and the exchange of logical records between application programs (either other DPPX applications, System/370 IMS or CICS applications, or applications in certain SNA cluster controllers, such as the IBM 3630).

Communication Services (CS), in a link-independent manner, manage the transmission of data in a network. They control the path over which data flow in the network, thereby implementing the path control and transmission control functions of SNA.

Transform Layers support non-SNA terminals by mapping the communications protocols, function management protocols, and if necessary the data stream of a non-SNA device to the appearance of an SNA terminal or cluster controller. For example, the IBM 2741 Start/Stop Terminal is mapped to the same SNA appearance as the similar IBM 3767 Terminal. Locally attached IBM 3277 Terminals and IBM 3284 Printers are mapped to the appearance of an SNA IBM 3276 cluster controller and its attached displays and printers. Because of its placement in the structure, the implementation of a Transform Layer can result in universal support of the non-SNA terminal, i.e., for DTMS, ICF, and for applications, depending only on the completeness of the mapping implemented.

I/O Attachment Services (IOAS) provide physical link transmission function. IOAS supports several line disciplines [e.g., Synchronous Data Link Control (SDLC), Binary Synchronous Control (BSC), or Start/Stop (S/S)] and provides physical link/station management. IOAS is independent of the types of terminals or stations attached to the link and is unaware of network topology. It is the analog to the Data Link Control (DLC) layer of SNA. This delineation of function permits the using layer(s) to be independent of the

link methodology used to attach a terminal (e.g., local or remote, link or loop). For SDLC multipoint links, IOAS provides the appearance of a point-to-point link with each of the attached stations.

The various versions of layers are generally intermixable. Therefore, programming support for additional SNA equipment can theoretically be no more than a new table definition specifying a different combination of layers. In fact, during the design of DPPX, additional terminals were included in the set of supported terminals on that basis alone.

application resource management design

The Application Resource Management layer, based on the Control Program Services layer and I/O Services layer, creates a logical context within which applications can operate. An application context is a specific usage of system primitives for managing resources such as storage or processor resources. For instance, the Application Resource Manager for transaction processing (DTMS) sets up environments and schedules their use on a transaction-by-transaction basis. As a result, main storage for the DTMS user is allocated to optimize for short-duration processing.[8] DTMS does not manage storage itself; rather, it creates the storage context for the application. The actual management of the resource is performed by the Storage Management element of the Control Program.

In contrast, another Application Resource Manager, the Interactive Command Facility (ICF), is designed for interactive command processing and background job processing. It sets up environments with a one-user-to-one-environment relationship, recognizing a requirement for a user's address space (one of the resources of an environment) to persist across many commands in sequence. The result is a different appearance of storage, and yet the same control-program Storage Manager has been used.

The Application Resource Managers also provide application services that supplement the services provided by the Control Program and I/O Services. For instance, ICF supports command definition and parsing. DTMS supports program-initiated queuing of transaction requests and special interprogram communication facilities that permit performance optimizations.

One of the roles of an Application Resource Manager, then, is to differentiate the appearance of the system to a using program in a way that is meaningful and appropriate for that class of application. Yet differentiation for its own sake is not introduced. A command processor, such as DEBUG PROGRAM, is supported not only by ICF but also by DTMS and the Remote Job Entry (RJE) Workstation Facility (WSF) as well. DTMS provides its differentiated view of the system with the ICF programming and user

interfaces included as a proper subset. The RJE Workstation Facility is built as a set of ICF Command Processors using an ICF environment. Thus an RJE/WSF operator has full access to ICF function and commands. The various Application Resource Managers are structured in relation to one another so as to provide for global uniformity and consistency across the system in both the application programming and end-user interfaces.

DPPX binding services

Binding services are those functions that define, qualify, or name resources managed by a service of the system, and thus concretely relate those resources to the programs using them. Binding functions are those that cause a fixed relationship to be established between the various logical objects of the system. For example, binding functions specify that a program be located in a defined address space at particular addressing values with access to a specifically addressed terminal and a particular data set, etc., on behalf of a named user.

Structurally, the relationships of the binding functions in a system to the other service functions are very important. The areas of relationship that are important are as follows:

- Dependence or layering relationship of the binding function with the other services of the system, i.e. whether run-time services make use of or depend on the presence of the binding services.
- Packaging of the binding function with respect to the packaging of other service function (e.g., OPEN vs. GET/PUT).
- Time at which binding is permitted to occur.
- Scope of a binding, whether permanent or temporary or reversible or not.
- Granularity of the binding of various elements with respect to other elements, i.e., can one binding (e.g., address space) be changed without rebinding other entities such as devices or data sets?

binding tradeoffs

We now discuss these areas of relationship in terms of the choices made in the design of DPPX and the reasons for them.

In general, the selected method of binding reflects a tradeoff between flexibility and optimization for performance. The following example illustrates this point.

tight binding

Data sets are defined as collections of records that contain fields. A tight—or early—binding between a program and a data set is one in which the actual machine instructions for performing operations with the data set are physically contained within the pro-

gram. These instructions include absolute references to a disk volume address and other locator information (e.g., cylinder/track/record/field offset address). In this case, performance is optimized from the point of view that there are no extraneous linkages in the program to a Data Management routine. That is, there are no instructions included for translating from a symbolic representation of the location of data to the physical location. The only instructions in the path for data access are those absolutely required to transfer the data.

loose binding

In contrast to tight or early binding is loose—or late—binding between a program and a data set. Loose (or late) binding is one in which the program contains symbolic (or indirect) references to the data and to services that access the data. To take the opposite extreme from the preceding tight-binding example, consider a program that accesses logical fields of records within a data set through a dictionary. Such a program accesses logical records through an indexing technique, and accesses the physical location of the data set through a catalog. In this way, the program has achieved absolute isolation with respect to the data from changes in the surrounding environment in which it operates. The premium paid for this flexibility is that instructions must be executed to resolve each of the indirect references and to make each deferred decision.

It is apparent that a range of choices exists between the two extremes of tight and loose binding. A traditional approach groups some of the translation or mapping functions and most decisions about access to the data set into a service called OPEN that the program invokes only once during its execution. The program thus incurs the overhead for the binding action at initialization time, leaving a minimum of binding actions to be performed at execution time.

Another choice for implementing binding is called *prebinding*. Prebinding is any technique whereby the binding function, analogous to OPEN processing, is executed prior to any execution by the program. Real-time, sensor-based systems typically implement this form of binding via a program preparation step. The performance advantages of this approach must be weighed against the sacrificed flexibility. If the physical organization of the data sets on the disk is changed, for example, the prebound program must be reprocessed by the program preparation step in order to function properly. If several programs are prebound to several data sets, a change to the physical organization of the disk requires analysis to determine all programs using the affected disk areas and recreation of the affected object programs. This process can be complex and time consuming, and requires the system to be taken off line while both the disk organization changes and the new versions of the programs are simultaneously implemented.

In a distributed processing environment, given the requirements discussed earlier in this paper, system hardware and software configurations are subject to frequent change. Also, one of our givens has been that the system be maintainable without on-site data processing skill. (Unattended operation is preferable.) Where centralized system maintenance is implemented, the number of systems being centrally controlled tends to be large. The sum of these factors effectively precludes a prebinding strategy for general-purpose distributed systems. Flexibility and reconfigurability are primary requirements that constrain the binding technique implemented.

System stability requirements

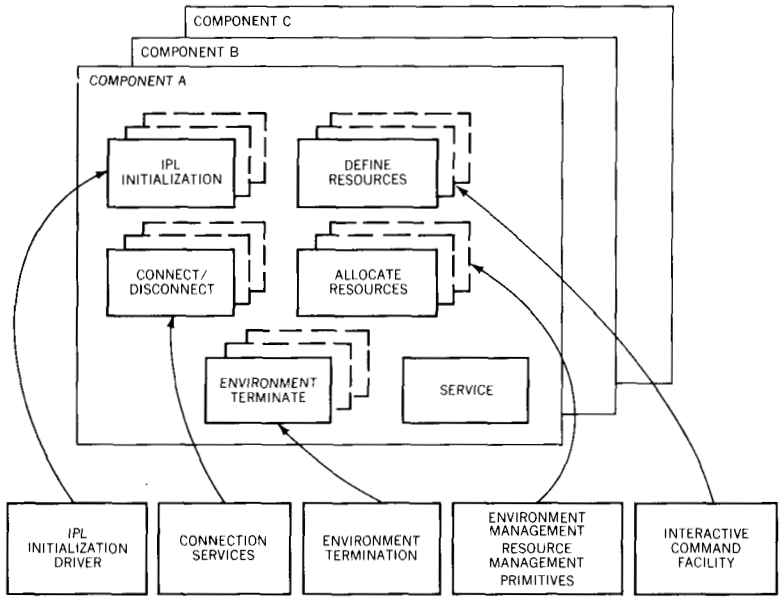
An additional factor in system configurability is the disruption potential of programming changes. If change is expected to occur frequently, and assuming the system must be very reliable, change obviously cannot be permitted to cause software instability. Past experience with operating systems and the disruption potential of change suggest areas that might profit from further exploration.

Examining motives for configuration growth and change, the most likely ones are either to add capacity or to add function. Added function may take the form of new or revised applications or a new device or terminal that offers additional new function. The problems of maintaining stability in the face of application changes are addressed in DPPX through such features of the system as storage isolation mechanisms, a test mode in DTMS, and security/integrity features. The problems of adding a new device should be examined in the context of overall system maintenance.

Generally, the risk of disruption is in proportion to the extent of change introduced. A change to one hundred modules is one hundred times more risky than a change to one module. It follows that a strategy for minimizing the disruption associated with a change should start by minimizing the number of elements involved in the change. In operating systems and programming, this approach is usually contradicted by the existence of corequisite and prerequisite requirements. Two functionally independent changes to a program become corequisite when they both modify a common segment of a program. The modified common segment implies that either both or neither of the changes may be installed, but there cannot be one change without the other.

Further, when one change is implemented—assuming the implementation of prior changes—the prior changes become a prerequisite for the new change. In operating systems, if a large number

Figure 3 DPPX component structure



of changes are created they tend to form into a meshed network of corequisite and prerequisite changes. It becomes quite difficult if not impossible to separate one of a large set of interrelated changes from all the others. And yet it is this separation that supports the notion of reducing the quantity of change to a minimum in order to minimize disruption.

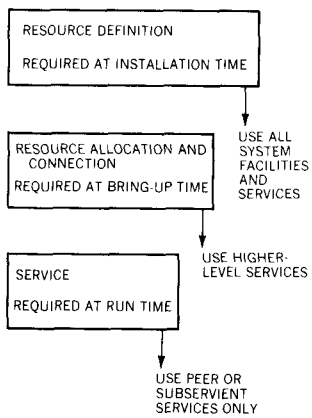
To attack this problem, DPPX has been provided with device support in a way that minimizes the number of elements of the system that must change to add a new device. Specifically, the device support in DPPX embodied in the I/O Services layered structure are designed to be self-contained, without corequisite dependencies on other system functions. The primary difference between DPPX and other systems in this regard is in the area of the binding functions related to device support. These are the functions of initializing the device and its control blocks in the system, allocating the device to a program, reclaiming the device at job termination, initializing an application program to use a device, and defining the device to the system. In OS/360, for example, these binding functions were all provided by separate components of the system, namely, device and control block initialization by NIP, program initialization by OPEN, device reclamation by CLOSE and ABEND, and device definition by SYSGEN.

In DPPX, an operating system component has complete responsibility for supporting its function in the system as well as providing the main-line function. The DPPX component structure is illustrated in Figure 3. Here, one of the layers of the I/O structure, for

example, is responsible for providing the logic to initialize, allocate or deallocate, connect or disconnect, and define resources. Driving or sequencing mechanisms are provided to control or initiate processing for these system services. These mechanisms are independent units and need not change. By structuring these supporting binding functions as an integral part of the main function, the number of system elements that must change for a new I/O device is reduced significantly. And since the component parts of the system have functional strength, there is little likelihood of a crossover corequisite between components.

The support for a new I/O device in DPPX thus generally consists of a single I/O layer to be either added or replaced. The isolation achieved through the layered structure further assists by masking the new device from the balance of the operating system. The affected I/O layer may be treated as a new layer, and only applications that employ the new support need use it. All other existing programs execute with no changed or new code in their execution paths through the system. Through the use of this structure, a technical base is firmly in place for nondisruptive growth in system functions and I/O device support.

Figure 4 DPPX component layering



Technology trends

Technology advances have continued to improve computing price/performance ratios at a dramatic rate, and changing cost relationships have resulted in changing system designs. The evolution of distributed processing is but one example of this effect.

By projecting the trends of technology into the future, distributed systems can be seen to be growing in two dimensions. The first dimension is function. If the price of computing is viewed as a constant, the effect of technology trends on future distributed systems is expected to result in significantly higher instruction processing rates. Improved price performance ratios, in turn, are expected to extend the capability of distributed systems to new applications with higher CPU loading characteristics. Foreseen applications are interactive graphics, noncoded information (NCI) handling, and image character recognition. Judging from past trends, gains of from five to ten times the current performance levels are conceivable in the next decade.[9]

The second dimension of growth, if the CPU performance level is held invariant, is expected to be reduced costs. Thus a processor capable of performing today's 8100 Information System applications might be expected during the next few years to perform at the same level at the approximate cost of a display terminal today. In other words, improving price/performance levels are projected to extend the distributed processing capability into the display terminal within ten years.

This view of future technology trends foresees distributed processing as spanning a broad range of processor configurations from a single terminal-like system to a multiple-terminal system designed for a complex application mix with a high-performance small processor.

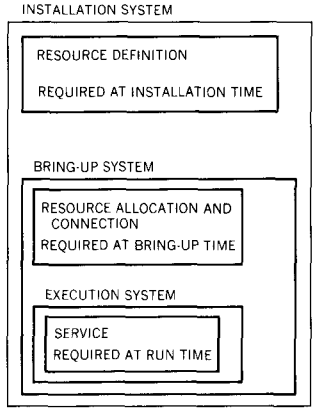
The hypothetical single-terminal system can be viewed as having all the requirements of the distributed system discussed so far. There is one intrinsic difference, however. The hardware configuration of the single-terminal system will probably be trivial compared to the current 8100 system. Such a system might even be fixed and unchangeable, in which case a different binding strategy would probably be desirable or necessary.

trends in DPPX binding structure

The design of DPPX has been influenced by these long-range considerations. Binding functions in operating systems have tended to be structurally inseparable from their related service functions, so that changes to one would dictate changes to the other.

DPPX defines binding functions as distinct structural elements, logically separated from their related service functions. These binding functions are further mapped onto the layered functional structure, so that run-time services can operate as an independent package. This mapped structure, illustrated in Figure 4, permits a logical view of DPPX as the three distinct systems shown in Figure 5. Each successive system is based on its predecessor. A *run-time* system is the base system. The *bring-up* system adds the binding functions of connection and allocation. And the *installation system* adds the functions of resource definition (the symbolic level of binding), built on the bring-up system.

Figure 5 Logical view of DPPX



DPPX implements late binding with options for preconnecting (early OPEN processing) data sets and terminals in the interest of transaction processing efficiency. Given the structure described here, DPPX is structurally capable of supporting a future hypothetical alternative implementation of the binding function. By removing the bring-up system and replacing it with a prebinding processor that builds the control blocks needed by the run-time system, prebinding can be implemented if required to support a single-terminal distributed system. Such a system can be built using the same run-time system for both a prebound single-terminal system and a loosely bound multiapplication distributed system.

Summary

The 8100 Information System and the Distributed Processing Programming Executive, DPPX, have been designed to serve as a distributed processor, with objectives aligned to meet specific requirements considered to be essential for a centrally managed and

controlled distributed system. The functional content of the operating system reflects these objectives. Further, the structure of the system, including formal layering, functional isolation and formalization of interfaces, and a structured approach to implementing binding functions lends DPPX pervasive characteristics that are considered necessary for a distributed system.

ACKNOWLEDGMENTS

The author gratefully acknowledges the team of designers responsible for DPPX and their collective contributions to the concepts and principles discussed in this paper. Most particularly, I thank L. C. Thomason, H. R. Albrecht, W. P. Dunfee, and B. P. Lubart as the originators of the foundations of the DPPX design. I also acknowledge F. N. Stoppenbach for his substantive contributions to this paper and to DPPX in the definition of distributed systems requirements.

CITED REFERENCES

1. D. M. Bailey, D. Gade, J. Garneau, J. J. Lempe, A. L. Martin, and R. J. Miller, *Distributed Computing in the Early 1980's—the Environment and the Requirements*, Report available from Guide International Corporation, Futures Division, Chicago, IL (August 1977).
2. P. H. Enslow, Jr., *What is a 'Distributed' Data Processing System?*, IEEE Computer Society, Long Beach, CA (January 1978), pp. 13-21.
3. G. Goos, "Hierarchies," *Advanced Course of Software Engineering*, F. L Bauer, editor, Springer-Verlag, Heidelberg (1973), pp. 29-46.
4. E. W. Dijkstra, "The structure of the 'THE'—multiprogramming system," *Communications of the ACM* **11**, No. 5, 341-346 (May 1968).
5. G. J. Myers, *Reliable Software through Composite Design*, Petrocelli/Charter, New York (1975).
6. H. R. Albrecht and L. C. Thomason, "I/O facilities of the Distributed Processing Programming Executive (DPPX)," *IBM Systems Journal* **18**, No. 4, 526-546 (1979, this issue).
7. A. K. Fitzgerald and B. F. Goodrich, "Data Management for the Distributed Processing Programming Executive (DPPX)," *IBM Systems Journal* **18**, No. 4, 547-564 (1979, this issue).
8. F. C. H. Waters, "Design of the IBM 8100 Data Base and Transaction Management Systems—DTMS," *IBM Systems Journal* **18**, No. 4, 565-581 (1979, this issue).
9. T. A. Dolotta, M. I. Bernstein, R. S. Dickson, Jr., N. A. France, B. A. Rosenblatt, D. M. Smith, and T. B. Steel, Jr., *Data Processing in 1980-1985*, John Wiley & Sons, Inc., New York (1976), p. 68.

GENERAL REFERENCES

A. L. Scherr, "Distributed data processing," *IBM Systems Journal* **17**, No. 4, 324-343 (1978).

G. M. Booth, "Distributed information systems," *AFIPS Conference Proceedings* **45** (1976 National Computer Conference, June 7-10, 1976, New York City), 789-794 (1976).

An Introduction to the IBM 8100 Information System, IBM Publication order number GA27-2875-0, available through the local IBM branch office.

Distributed Processing Programming Executive (DPPX) General Information, IBM Publication order number GC27-0400-0, available through the local IBM branch office.

The author is located at the IBM System Communications Division laboratory, Kingston, NY 12401.

Reprint Order No. G321-5107.