*Discussed is the evolution of a computerized airline reservation system from its early form up to the present version. Data base allocation, accessing techniques, and data communications of the system are described. The system consists of the Programmed Airline Reservation System (PARS) and its control program called ACP.*

# A high-performance DB/DC system

**by J. E. Siwiec**

To satisfy applications with very high performance requirements, a specialized operating system is needed. In this paper, one such system that evolved from early automated airline reservation systems is described, the result being the present DB/DC (data base/data communications) system. Initially developed for the growing air passenger travel industry, it was adopted for use in other, completely diverse businesses which required similar system attributes, e.g., the automation of the clerical, record-keeping function with no impact on the casual and natural dialogue between the reservations agent and customer during the transaction. Terminal response, availability, reliability, and recoverability were the overriding systems design considerations.

Today, such a system typically has 3000 terminals and performs a real-time inventory control application 24 hours per day at a rate of more than 50 messages per second against a five billion-byte data base.

This paper deals with the development of that system, explaining in what ways it differs from other systems. The history describing its evolution through all the releases of the software and supported component devices is documented. The elements that give the system its unique performance characteristics are described, including the dispatching of work units and storage management. Data base allocation and accessing techniques are discussed. The data communication section describes the evolutionary sequence from the specialized Airlines Line Control to SNA/SDLC (Systems Network Architecture/Synchronous Data Link Control). The importance of test tools and data collection-reduction facilities in the system are emphasized.

## History of the system

The earliest passenger name reservation system was developed with American Airlines and was called SABRE, a name suggesting speed and accuracy. The original acronym, SABER, stood for Semi-Automatic Business Environment Research.[1] The primary motivation for the system was to automate the airline reservation function which was mostly a manual process. The objective was to improve on the accuracy and capacity of the reservation function without impacting the personal and natural dialogue between the reservations agent and passenger. The initial on-line system contained 100000 instructions, 40000 of which made up the control program and support functions. The system was implemented on the IBM 7090 processor and used IBM 1301 disk files. Specialized terminals (IBM 1003) and concentrators (IBM 1006) were also developed for SABRE.

Some important technical results of the work on SABRE were:

- Line concentration techniques.
- Medium- and low-speed data set development.
- Fast random access.
- Front-end processing.
- Large volume disk and drum storage devices.
- Relocatable, reentrant software.

Two similar systems were developed later that benefited from the experience gained on the SABRE project—Deltamatic for Delta Airlines using IBM 7074 processors and PANAMAC for Pan American Airlines using IBM 7080 processors. These systems were functionally equivalent, the primary difference being system capacity (throughput).

The American Airlines SABRE system went "on-line" in 1963. Delta and Pan American started up their systems in 1964.

**PARS**

In 1965, IBM began to develop a generalized airline reservation package called PARS (Programmed Airline Reservation System). The plan was to apply the knowledge gained from the three prior implementations, together with the System/360 technology, to produce a modular system usable by airlines of any size. Today, PARS is widely used in the airline industry.

PARS was implemented on System/360 processors Models 40 through 75 but was used primarily on the System/360 Model 65. Large Core Storage (LCS) was used as a data file for frequently accessed records, and less active data was stored on an IBM 2314 Direct Access Storage Facility. The off-line support programs ran under DOS. The system supported IBM 2703 Transmission Control Units and terminal concentrators (2948) to

which special CRT terminals (2915) with reservation-functional keyboards were attached. The system had several special features to provide the performance and application requirements. For example, a special buffer for the storage control unit freed the channel for a major part of the access time, providing more available channel time and, hence, more accesses.[2] The terminal concentrators and transmission control units were modified for similar reasons.

In 1968, the control program portion, called ACP, and related utilities were separated from the airline reservation application in PARS and have since been separately enhanced to support new devices and provide functional improvements. These enhancements, coupled with the increased use of ACP for other applications, have had the effect of creating software with the characteristics of an operating system particularly suitable for the support of high transaction rates.

**ACP characteristics**

The ACP system, which has evolved over the years, can be defined as a high-performance, real-time, message-driven operating system. It consists of approximately 500000 lines of code. (The reservation application in PARS totaled 150000 lines.) It is characterized by a high volume of unpredictable inputs, requiring limited function and flexibility but extremely high availability and rapid, consistent responses. These characteristics imply that demands on the system are dynamic in nature and dictated by some external stimulus not under control of the system; that the data base must be on-line and current; that fallback, restart, and recovery functions must be fast and accomplished with little or no awareness by the user and almost no impact to the performance of the system.

These requirements are addressed in ACP through the evolution of a control philosophy and new facilities. Dynamic load control was implemented to allow maximum utilization of the various system components. The user (the terminal operator) was made an integral part of the system, and the impact of any errors was isolated to affect only the specific user while the system remained available to other users. This philosophy was extended to include the operator as part of the system, and checkpointing, restart, recovery, and reconfiguration facilities were provided to allow restart in one to two minutes or reconfiguration in three to five minutes while attempting to limit the impact of an outage to only those users directly affected.

Several key factors were involved in achieving the high availability and recoverability characteristics:

**human element**

- Certain system software decisions and functions could be off-loaded by considering the reservations agent-operator as

an *intelligent* system component. Human judgments regarding the transaction were made by that component. However, the agent could take no action, make decisions, or tie up resources that could affect other transactions or the system as a whole. This philosophy not only streamlined the process but enabled a natural dialogue between the agent and passenger.[3]

**restart and switchover**

- Because the system is for the most part configured at system generation (not initialization) time, the restart process is rapid and uncomplicated, thus making most restarts successful and achievable in seconds. Moreover, the switchover and restart procedures are identical, enabling either process to be effected with the same high reliability expectations.

**trivial computation**

- The structured nature of the system enables an extremely high degree of multiprogramming. Thus, many units of work will be active at any given time (50 to 75 are not uncommon). Because each unit will use only a tiny resource, waiting time (by each unit) is reduced to a minimum. This design ensures system accessibility (availability) to the highest possible number of users.

**backup CPU**

- A backup processor designated as the standby machine is made available in the event that an unscheduled switchover must be performed. A switchover is usually due to some uncorrectable hardware failure in the on-line CPU. This occurrence, while relatively rare, is nonetheless of crucial significance when it does occur during peak activity. In practice, most switchovers are of the "scheduled" type for preventive maintenance.

**duplicate data base**

- The data base is selectively duplicated depending on the data type and relative importance as designated by the user. While duplicated records increase system overhead (each record must be written twice), an advantage is gained when referencing the data because more than one path exists to it.

Typically, an ACP system network consists of from 2000 to 5000 terminals, although some ACP systems also exist with a few hundred terminals, and 10000 terminal systems have been envisaged for the near future. However, many terminals do not necessarily imply many messages. The entry frequency for messages will vary from one application to another. A message input of one per minute was assumed as the design point for highly active terminals. Less active systems were designed for a lower frequency of input per terminal. The required average response (according to the design point) to the terminal is under two seconds, where response time is the period between the moment when the enter key is depressed and the first response character appears on the screen.

The system is message-driven, which means that the source of all input is the result of polling the remote concentrators on a

periodic basis. The units of work in the CPU (called entries) are initiated by some external stimulae (messages). Multiprogramming is accomplished by allowing as many entries to be activated as the system resources will accommodate. Application programs are relocatable and reentrant and, hence, can be shared between entries. Each entry will proceed until it is required to "wait" for an external (file) reference, at which time another entry will be initiated or allowed to proceed. All messages are handled on a first-in, first-out basis and, hence, no processing overhead is incurred for task switching as in a preemptive scheduling system.

**ACP Models 5 and 6**

ACP was first announced as a system separate from PARS in November 1970 and first delivered in August 1972. It was called ACP Model 5. The anticipated users were airlines, and this model, implemented on a System/360 Model 195, provided support of the following devices:

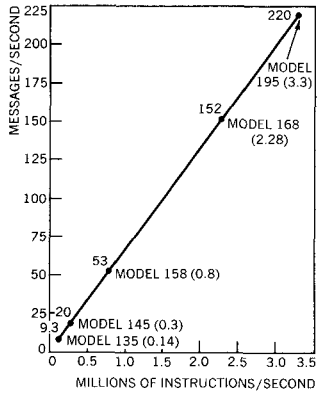
- 2880 Block Multiplexor Channel.
- 2305 Fixed Storage Model 2.
- 2969 Programmed Terminal Interchange (PTI), which is a special transmission control unit designed to extend the communications line attachment capability for Model 195 systems.

The off-line utilities were converted from DOS to run under OS in Release 5. The support of the System/360 Model 195 necessitated a major restructuring of the control program to accommodate the increased capacity of that system. Restrictions on numbers of terminals and device attachments were removed. Since the system still supported the other System/360 models, new device support had to be modular and had to be able to be system generated. Note that the modularization necessitated by the larger CPU ultimately assisted the capability to run on the smaller System/370 CPUs (e.g., System/370 Model 135).

A companion release, ACP Model 6, was announced in January 1972 and shipped in May 1973. It extended support to include the 3330 Direct Access Storage Facility and was implemented on the System/370 Model 165.

Comprehensive tests of Model 5 using System/360, Model 195s were performed with special emphasis on the performance capabilities. Projected peak message activity required a throughput capability for processing 180 typical reservation messages[4] per second at 85 percent of CPU utilization. The average response time per message during peak loads was to be within two seconds with a requirement for no more than 10 percent of the messages to exceed four seconds. The average processing time per message was to be within 4.7 milliseconds.[5]

Figure 1 CPU throughput for PARS messages




The tests validated all the stated objectives (with the system driven by a second CPU simulating the network at various rates up to 161 messages per second). One important observation made was that the processing time per message (4.2 milliseconds were actually measured) remained reasonably constant at all message rates. This knowledge permitted linear extrapolation with high confidence levels up to the maximum capacity of the CPU. Also, it means that given the rated capacity of any CPU, its PARS capability could be quickly assessed.

Thus, for ACP, the message-processing capacity of any CPU is a linear function of its MIP (millions of instructions per second) rate:

$$\text{Messages per second} = \frac{\text{MIP rate}}{k}$$

Figure 1 depicts the relationships and extrapolations for several CPU types assuming the typical PARS message at 15000 instructions per message and up to 100 percent CPU utilization. However, in practice, systems are designed for up to 85 percent CPU utilization beyond which some degradation in response may occur.

**cohabitation**

Since the ACP systems were designed for the future peak message rate and were required to operate 24 hours per day, and since the load demands of the application followed a cyclical pattern, there were long periods of low system utilization. A software hypervisor was developed to make use of the unused CPU capacity. This facility was announced in ACP Model 7 in August 1972 and made available in December 1973. The hypervisor allowed an ACP user on a System/370 to capitalize on the excess computing power by operating OS/VS jobs during the low periods of on-line activity.

Some unique objectives had to be met in the hypervisor development process. Firstly, the processing overhead on ACP had to be minimal (under five percent) to retain the performance capabilities of the system. Secondly, essentially no impact on ACP system reliability (uptime) could be tolerated. Thirdly, the feature had to be transparent to the applications under ACP or OS/VS and to OS/VS itself.

The first objective was met by designing the hypervisor as a logical extension of the ACP interrupt handler routines. Since interrupt handlers are crucial to ACP performance, and since they already contained logic to recognize which interrupts do and do not belong to ACP, only a slight amount of additional overhead was necessary to process the OS/VS interrupts.

The second objective was met by designing the ACP hypervisor as a special kind of "virtual machine" system. Only two operating systems are supported (ACP and OS/VS); OS/VS runs only in problem state and is always physically enabled. Interrupts occurring while OS/VS is in control of the CPU allow ACP to seize control of the CPU if it solicits the interrupt, but OS/VS solicited interrupts, occurring while OS/VS is logically disabled, are preserved (stacked) until OS/VS is logically enabled. Privileged instructions executed by the OS/VS supervisor are simulated by the hypervisor, keeping total control of the status of the OS/VS "virtual machine." This control prevents impairment of the ACP system reliability due to unforeseen damaging conditions emerging during the execution of instructions in the OS/VS domain.

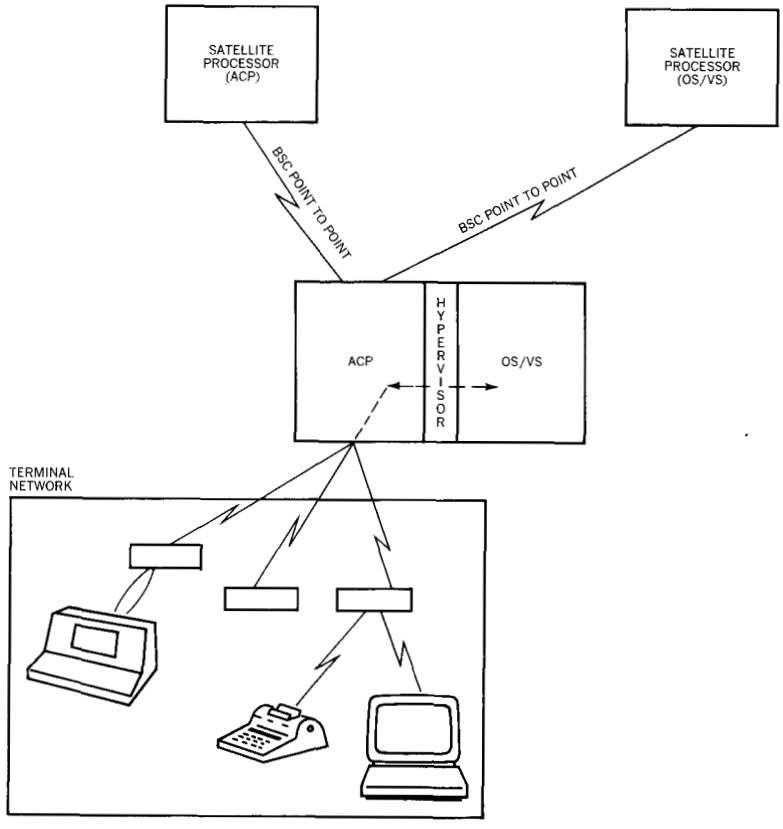
The third objective was also met. Adjustments had to be made dynamically by the hypervisor to the OS/VS address translation tables because the I/O supervisor modules were relocated to make way for ACP. Beyond those adjustments, transparency was maintained. As might be expected, the simulation of privileged instructions executed by OS/VS cause a degradation of OS/VS jobs, depending on the percentage of privileged instructions executed in OS/VS, which varies from job to job. A later version (ACP Model 9) invokes the facility of the Virtual Machine Assist feature to reduce the overhead significantly.

**ACP Model 8**

In December 1972, ACP Model 8 was announced. This version included enhancements that began to give ACP the characteristics of a generalized DB/DC system. The following functions provided in Model 8 were made available in three separate releases:

- This version could be implemented on System/370 Models 135, 145, 158, 168, and 195.
- At a read/write level, the IBM 3211 Printer, 3505 Card Reader and 3525 Card Punch could be attached. It may be interesting to note that the support for such devices was developed to satisfy the need for application expansion. Being totally interactive in nature, ACP allows all system communications by terminal or system console, thereby rendering card readers and the Job Control Language type of input unnecessary.
- A load and dump facility through ACP for the emulation program in the IBM 3705 communications controller was available.
- SCP functions in support of consumer finance users (particularly terminals and concentrators) were provided.
- A 3340 direct access storage device could be attached.
- The 3270 Information Display system could be attached through a System/7 used as a remote multiplexor.

Figure 2 ACP Models 8 and 9 message routing




- The hypervisor was enhanced to allow channel sharing between OS/VS and ACP and to provide a "bridge" through which applications under OS/VS could access and update the ACP on-line data base.
- Improved system implementation facilities were also provided.

The network facilities, as part of the third release of ACP Model 8, extended the scope and function of existing data communications support facilities. The fundamental structure and method of terminal operation remained unchanged, but the terminals, lines, and other communications facilities could now be shared by other processing systems (satellite processors) operating under control of OS, OS/VS, or ACP.

Also, direct communications between applications under ACP and those in other CPUs were made possible, thereby allowing data-base records and other internally generated information to be sent from one system to another. The data is sent as the text

of a "message" between applications in each processor with appropriate indicators specifying the source and destination of the data. ACP remains the focal control point for the network of terminals and satellite processors. It establishes connections between terminals and application programs in the ACP system or in one of the satellite processors.

The satellite processors are normally attached to the ACP system by point-to-point communication lines utilizing a binary synchronous procedure.[6] Figure 2 shows the interprocessor connections utilizing the message-routing facility of ACP. Note that one of the satellite processors is an ACP system. In the case where the OS/VS systems share a CPU with ACP via the hypervisor, the binary synchronous communications connection becomes "virtual," and (as indicated by dotted lines) data is transferred over an internal interface.

**ACP Model 9**

Announced on July 15, 1975 and made available on November 26, 1976, ACP Model 9 is the most current version of the system. It is a true implementation of IBM's Systems Network Architecture for the 3600 Finance Communication System via the 3705 controller with NCP/VS (Network Control Program). The SNA/SDLC protocol can coexist in the same CPU with the previously supported ACP data communication protocols.

A binary synchronous data link for interprocessor communications extends the networking capabilities of the previous version to multipoint. (ACP Model 8 provided only point-to-point capabilities.) Also, the Virtual Machine Assist feature could be invoked in the ACP software hypervisor.

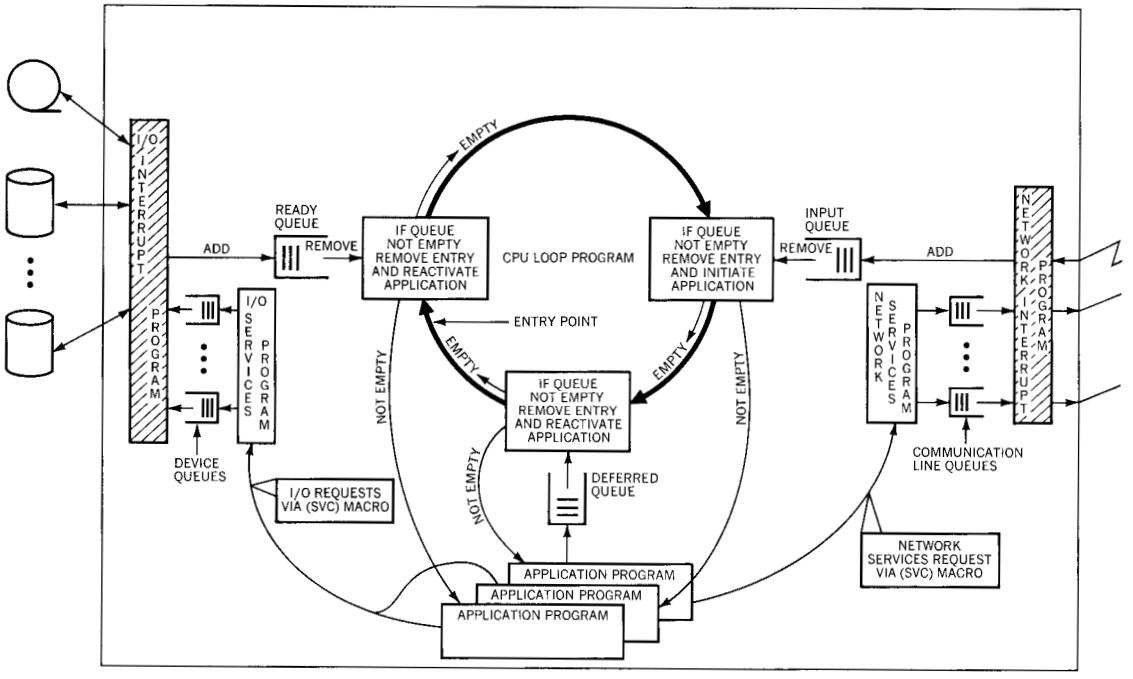
## ACP operational characteristics

**dispatcher**

The system is basically controlled through a mechanism called the CPU loop program (Figure 3). This program continually interrogates a series of queues for work to be done. Whenever an interrogated queue is not empty, the first item (entry) in the queue will be dispatched. Interrupts (as a result of I/O completions) cause entries to be placed in the queues. The interrupt routines do not perform any scheduling and always return to the interrupted code (which may be the CPU loop program itself).

The three primary queues are called the ready, input, and deferred queues. The order in which the queues are interrogated establishes the processing priority. The ready queue is always interrogated first and all the items dispatched before the input queue is tested. This is consistent with the philosophy that work in process (ready queue) takes priority over work to be processed (input queue). The ready queue contains units of work

Figure 3 The ACP dispatcher




that are ready for further processing. Entries are placed in the queue by the I/O completion interrupt routines which were the result of an application program request for system service (e.g., data retrieval).

The input queue contains input messages that have not yet been given the status of an entry. These messages are added to the queue by the network services interrupt routines that completed the message input processing.

The deferred queue contains entries designated as low priority by the application. It is a convenient way to defer processing until triggered by some required event taking place (e.g., a real-time indicator is set). Thus, the application can "wait" without usurping system resources by looping.

Figure 3 shows these queues and depicts the ACP dispatcher philosophy. Because the ready queue contains only entries for work in process, the application program is in all cases being reinvoked. In contrast, the input queue contains fresh messages that will initiate an application program for the first time. Note also that the entry point to the CPU loop program is always where the ready queue will be interrogated first, thereby maintaining the work-in-process priority.[7]

ACP storage management has the significant attribute of homogeneity. Storage blocks may alternately serve as: program, work space, data, and control blocks, using the same expedient storage management techniques. The storage block is part of an active entry and thus has a short life. It is returned to the storage area, or pool, for reuse by another entry.

**storage management**

Storage is presegmented into small, fixed block sizes and dynamically dispensed as required for use by the applications. The blocks are returned to a common pool when not in use. Four different storage areas called *pools* are maintained in different sizes:

128 bytes— used primarily as input buffers for network control and also as work space for the application or ACP.
381 bytes— used as an entry control block, message block, data block, program block, or work space.
1055 bytes— used the same as the 381-byte block except for entry control.
40 bytes— used exclusively by ACP to control I/O operations.

The main storage allocated to each pool is identified at ACP initialization time, and the size of each pool (number of blocks) is a function of the type and volume of system activity. It is important to be able to predict the activity to initially create the four different pools and subsequently measure it in order to "fine tune" the system for optimum performance. The data collection programs are used in this activity and will be discussed later.

A set of "lists" is associated with each storage pool. The list is a "pushdown stack" of addresses of all blocks in a given pool. A pointer is maintained and always points to the list item that contains the address of the next available storage block. Figure 4 depicts the storage pool and list concept. The pointer contains the address (A) of the location in the list that contains the address of the next available 1055-byte block (B). Managing the lists consists of the simple expedient of dispensing (and returning) main storage block addresses from and to the list and incrementing or decrementing the pointer to the list item.

The list concept is an improvement over an earlier design, in which the available main storage blocks were chained together, and the chain words, manipulated as blocks, were dispensed from and returned to the pool. Since the storage pools are unprotected from application programs, the chain words were often modified erroneously. Such modification resulted in broken chains and lost main storage blocks, which caused system errors. In the current design, the list itself is protected, whereas the storage pools remain unprotected. The error incidence has been reduced significantly as a result.

Figure 4 Storage management

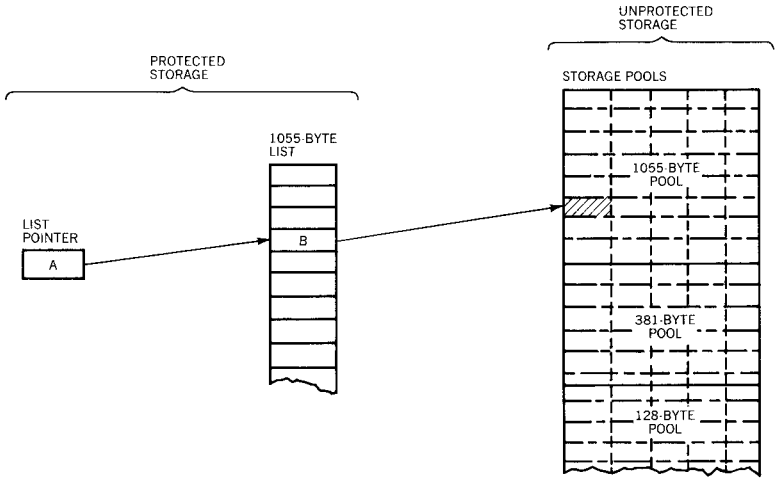
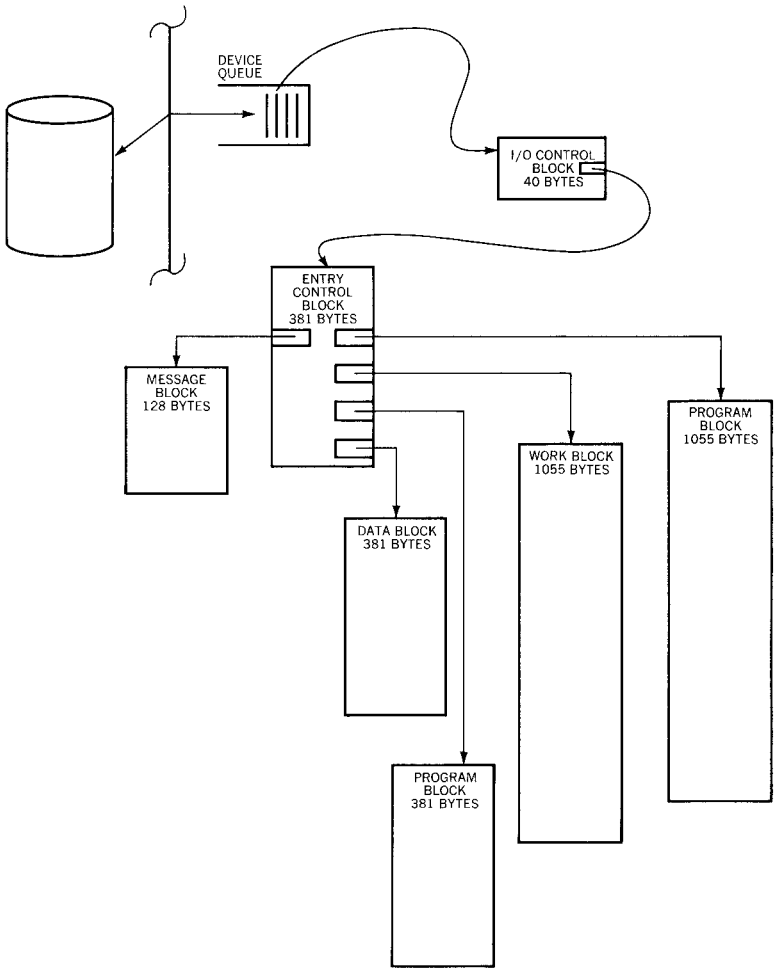



Figure 5 Application entry environment



The environment in which the application program performs its function is within a group of storage blocks interrelated by reference (chain) words. This environment is initially created when a message is removed from the input queue (see Figure 3) and chained to a 381-byte entry control block and an application program is dispatched. Figure 5 shows the application entry environment.

The application then processes the message, uses work space, retrieves and updates data, responds to the originating terminal, and exits. All related blocks are chained to the entry control block because it is the primary controlling block.

An entry control block is pointed to by an I/O control block that is used by the system to perform I/O operations on behalf of the entry. The I/O control block is, in turn, referenced as an item in the device queue.

One additional point should be made about the program block. Since all application programs must be relocatable and reentrant, one copy of the program could be shared by more than one entry. Thus, the program blocks shown in Figure 5 could actually be referenced and used by another entry in the system.

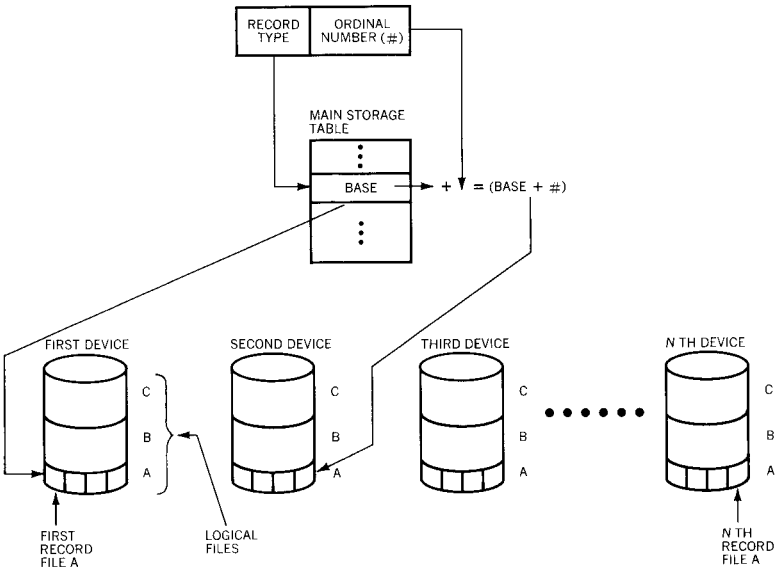
## Data-base support

The performance of a system largely depends on the number of file storage requests and the time required to transfer the data to main storage. The number of requests is determined by the application design. The request time includes the amount of time taken by the control program instructions, the time required to find the requested block on a physical device, the time in queue for the device, and the transfer time. Since each device can only handle one request at a time, multiple requests for a particular device must be queued.

The transfer time and seek time are based on the device characteristics. The queuing time, however, is dependent on the file organization. A first step in reducing the queue for any device can be to design a system that approaches equal queue sizes for each device. The average length of the queue on a device can then be reduced by increasing the number of devices.

In ACP, rapid access is provided to a large data base concurrently for many system entries by arranging PARS/ACP data in a unique pattern on the devices. The data is spread across all the physical devices rather than in a more traditional sequential-by-module

Figure 6 Data distribution and addressing




arrangement while maintaining the appearance of a sequential data set to the application. This scheme allows for optimum accessibility through the channels and control units to the devices because more paths are effectively maintained to the data. The file is not "opened" or "owned" by a particular application but is available to all applications at all times.

The organization might be envisioned as a group of disk packs, each identical in format and type of content. The logical files are layers on the packs, each pack with an equal percentage of the total logical file. Figure 6 represents N physical devices on which three logical files reside (A, B and C). The *N*th record on the *N*th module is the last record of logical file A. Examples of logical files might be inventory records or customer account records. A table resident in main storage contains the base (starting) address of each of the logical files (called record types in ACP), and an ordinal number designating the relative record position is added to the base to locate the specific record in the file. The ordinal number is assigned when the record is initially allocated and subsequently presented (by the application program) as a parameter together with the record type to effect the retrieval.

The purpose of this organization is to allow a larger number of concurrent accesses to any particular logical file than would otherwise be possible, thereby reducing the chance of excessive

queuing. It has an additional advantage in that it frees the application design from many of the physical device performance considerations.

Data is stored and retrieved at the physical record level, and the record sizes are fixed at the familiar 381- and 1055-byte size. Thus, the homogeneity of main storage is extended to the secondary levels of storage.

The data records fall into one of two general categories designated as "fixed" and "pool" records. "Fixed" records are used to contain permanent or static data. The data *content* may change dynamically, but the *number* of records generally will not. "Pool" records contain data that is more transient in nature and will occupy file space only while the data is active.

For example, fixed records might contain inventory counts of the number of seats available for a flight. Pool records might contain the names of the passengers on that flight. Since the flight is flown on a scheduled basis, it is necessary to retain the inventory records, whereas after the flight is flown, it is not necessary to retain the passenger name records on-line. They are then purged to a history file, and the space occupied made available for reuse; hence the name "pool."

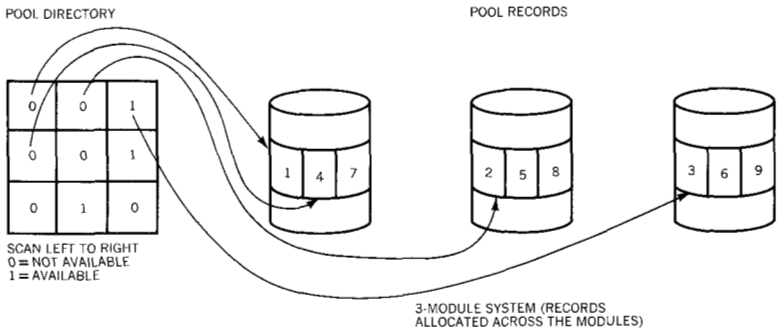
**pool storage**

Dynamic requirements for file storage are provided for in the pools. As in the fixed areas, pool records are similarly allocated across the physical devices during system generation. However, the pool record area represents a large repository of file storage common to all applications, and the records within the pool area are dispensed on an "as needed" basis. This means the availability of each record within the pool area is managed by service routines in ACP. (No such management is performed on records within a fixed record type). An application program may obtain a pool record address at a point in processing where the application needs the space.

A distinction is made between the lengths of time during which the data in the record is active. Record space from the short term pool is usually used only during the life of a transaction (i.e., seconds). Long term pool records are used for more permanent data (i.e., days, months). The pool space is freed for subsequent use by either the application or the control program. Large (1055) and small (381) records can be allocated to the pool.

Figure 7 shows the pool record allocation and addressing scheme. An entire pool appears as one record type in the system. Since record addresses must be dispensed dynamically, a bit matrix is used to locate the record and indicate its usage.

Figure 7 Pool record allocation




Matrix records are themselves chained and reside on file when not in use. Record space is freed or made unavailable by a simple bit setting in the matrix record.

Pool addresses are reinstated for use through on- and off-line utility programs. This reinstatement involves the manipulation of the bit matrix records (called directories), setting the bit "on" for the associated record address. Other utility programs "scan" the data base and "recoup" lost records (broken chains, unreturned addresses).

Data is retrieved and updated through a simple macro interface between the application programs and ACP. Some data-accessing macroinstructions are:

- FIND—retrieve a record.
- FILE —write a record.
- FINH—retrieve a record and hold (does not allow any other attempt to update but allows simultaneous reference).
- GETF—get file address (POOL).
- RELF—release file address (POOL).

**summary** ACP provides a simple direct-accessing capability on which a complex data management facility was built for PARS. Relocatable and reentrant subroutines are available to retrieve and update the data based on its input parameters, e.g., Flight/Date/Name. Other on-line subfunctions, such as recouping lost addresses, file maintenance, file reorganizers, and database capture, were also developed. Other applications (loan payment processing, car rental, credit inquiry) similarly tailored the data management function to their needs.

## Data communications

The communications-handling portion of PARS/ACP is an integral part of the overall architecture. A message flows in and out of

the system, accomplishing its purpose of data-base inquiry and update with a minimum of processing. Incoming characters are assembled in a buffer storage block to form a message that is placed on an input queue. In its turn, the message becomes input to an entry created by the system. Applications, invoked as a result of input messages, retrieve and update data and are multiprogrammed with other entries as a result of I/O activity. Application processing terminates after a system request is made for the transmission of a response message.[8] Almost without exception, the above is the general processing sequence for all inputs.

The functions performed by the communications-handling portion of ACP (called Communications Control Program, or CCP) are:

- Polling and circuit assurance.
- Execution of I/O operations.
- Input/output character code translation.
- Main storage allocation for associated I/O operation.
- Preliminary input message assembly, including multiblock message assembly.
- Application program interface (invoking the application program).
- Hardware error detection and corrective actions.
- Communications hardware reconfiguration activations.
- Communications line queuing.
- Computer network control (Message Router) allows the sharing of terminals, lines, and other communication facilities by other (satellite) processors.
- Operator communications.

A detailed description of the data communications functions of ACP can be found in the document, *ACP System Concepts and Facilities*.[9]

**Airlines Line Control**

Early in the development of the original SABRE systems, the importance of efficient line control and terminal concentration was recognized. A special line discipline was developed for those early systems and is still in use today. It is called Airlines Line Control, or ALC (also known as SABRE Line Control or PARS Line Control). Implemented in reservation systems utilizing the special concentrators, ALC is a true, full duplex, synchronous transmission capability.

ALC uses synchronous full duplex transmission on dedicated communication lines that are either hub or roll call polled and may be either leased common carrier or private telephone lines. Each message contains two synchronization characters, the data text, and a character to aid in determining the validity of the message. Each character is six bits in length.

The line control takes advantage of the fact that the agent is an integral part of the system and the philosophy that every input message will result in an output message. The choice of message concentrators (asynchronous time division multiplexors), rather than individual terminal polling (synchronous time division multiplexing), further contributes to the communications efficiency.

For nearly 15 years, the PARS-ACP systems have used basically the same unique communication discipline. It remains an effective and stable discipline for applications that can be performed within the limits of the six-bit character set. For a more detailed analysis of the characteristics of ALC see Knight.[10]

As the use of ACP expanded, several external factors influenced the data communications architecture:

- Terminals and remote concentrators with more intelligence were being introduced, thus providing a more transparent communications interface to the application.
- ACP was increasingly being used for other applications beyond airlines reservations, requiring a larger character set.
- Hardware advances were reducing the necessity for special-purpose devices.
- A unified systems structure linking terminals and host computers called Systems Network Architecture (SNA)[11] was developed along with a line control discipline called synchronous data link control (SDLC).[12]

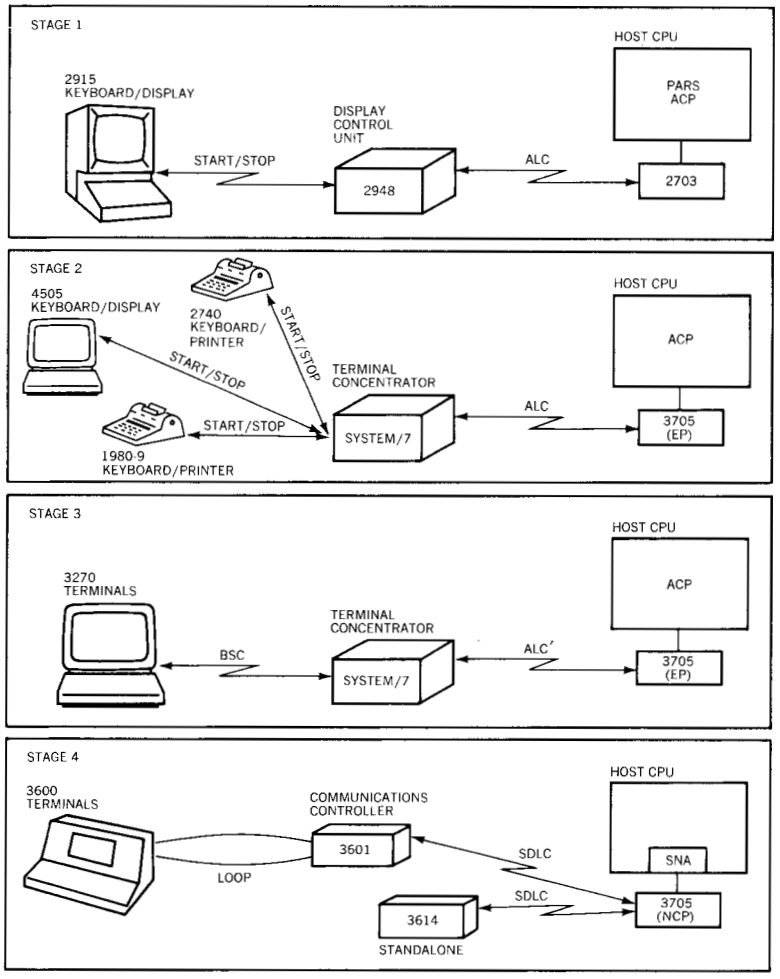
The evolution from ACP-ALC to SNA-SDLC was accomplished over a sequence of steps, each accommodating changes within the state of the art. Figure 8 shows four steps in the evolution.

*Stage 1* depicts the primary communication devices supported in earlier releases of PARS-ACP. All communication devices had custom hardware modifications to accommodate ALC. A start/stop mode of transmission was used between the terminal and display control unit.

The six-bit character (64-character set) representations provided adequate alphanumeric and special symbols for the airline reservations and certain other applications. A more powerful cyclic check (in contrast to a vertical character parity plus block check) of strings of characters provided for fewer undetected transmission errors as well as eliminating the parity bit per character. The advantage of a full duplex capability is to further reduce the error rate.

In *Stage 2*, System/7 was introduced as a remote concentrator. The six-bit character set was still adequate. Because the Sys-

Figure 8 Stages of communication




tem/7 is programmable, more than one terminal type could be attached to it. A regular keyboard/printer (IBM 2740) and two special-purpose terminals are shown (Airlines 4505/7411 and Consumer Finance 1980-9) in Figure 8.

*Stage 3* allows the attachment of the IBM 3270 Information Display System. This stage required the transmission to be accomplished via the binary synchronous communications (BSC) protocol using EBCDIC character representations. Nonetheless, the inherent efficiency of ALC was still required. The System/7 was programmed to convert the eight-bit code into one or two six-bit ALC characters for transmission over the ALC line (depicted as ALC′ in Figure 8). The characters were then reconverted in the CPU. A data link escape mechanism that has additional control characters which change the meaning of the characters that fol-

low it was used to achieve maximum efficiency over the line. Thus, the functional capability of the information display system was made available on the ACP high-performance communication facility.

*Stage 4* shows the current step in the evolutionary process: SNA-SDLC support for the IBM 3600 Financial Communication System is indicated as an SNA module in ACP. The communications controller (3601) provides additional remote intelligence, enabling more terminal independence and added message integrity. SDLC provides a transparent[13] line protocol commensurate with the previous efficiency requirements. The Network Control Program in the communications controller (3705) replaces the emulator program, removing the last special-purpose hardware modifications in the communications network except that the communications controllers (2703 and 3705) in emulator mode required custom modifications to accommodate the ALC protocol.

As the use of ACP expanded to other applications, additional line controls and terminal support were provided. In addition to ALC, other protocols mentioned below are supported by the current system.

**synchronous data link control**

SDLC is the data transmission procedure used within IBM's Systems Network Architecture for information transfer over data communication channels. Transmission may be full duplex, half duplex, point-to-point, or multipoint. SDLC includes comprehensive detection and recovery procedures at the data link level. For example, ACP Model 9 supports the 3600 Financial Communications System within the framework of SNA/SDLC. Thus, all applicable SNA commands for the financial system are supported.

**low-speed controlled telegraph**

Low-speed controlled telegraph (LSCT) supports multidrop lines (multiple terminals on a line) in a start/stop mode of transmission. Each drop can be addressed and each drop must be roll call polled (i.e., the transmission of a short message to each drop to solicit input) for data. Half duplex lines operate at rates up to 75 baud (100 words per minute). The character size is five-bit Baudot,[14] and LSCT is required by users of 83-B type equipment for message-switching functions.

**low-speed free-running telegraph**

Low-speed free-running telegraph (LSFR) is point-to-point start/stop transmission. The line is free-running, eliminating poll messages and other control characters, but requiring more complex synchronization within the software. The lines operate at rates up to 75 baud in full duplex, half duplex, and simplex modes. Characters are five-bit Baudot.

**synchronous link control**

Synchronous link control (SLC) is a technique specified in the requirements of the National and International Air Transport Associations, which represent domestic and international airlines. Operation of SLC communications is synchronous on full duplex, private, leased, voice-grade lines. It requires point-to-point full duplex lines operating at rates up to 9600 baud. Each link, containing up to seven communication lines, represents a connection between CPUs. A single message may utilize all seven lines. No polling is associated with SLC. Each CPU continually listens to the receive lines. Dummy messages are transmitted whenever there are long periods without transmission to check on whether or not the line is still operating.

Data and control characters are eight bits long (seven bits of data plus parity). Each message contains a block check character for error detection, as well as a vertical redundancy check on all characters. All data messages are sequenced, allowing each CPU to detect missing or spurious messages, which permits the automation of corrective procedures. Corrective procedures usually entail the request for retransmission and/or system operator notification.

**asynchronous link control**

ARINC is an acronym for Aeronautical Radio Incorporated, a wholly owned subsidiary of the U.S. domestic airlines, which operates a message-switching network. ARINC support, which is Asynchronous Link Control, uses point-to-point, low line speed, full duplex, start/stop transmission methods. This includes message sequencing to detect lost messages. Application functions protect against the possibility of lost or garbled messages.

**binary synchronous communications**

The ACP system support of binary synchronous control is used for processor-to-processor communications. This use permits other system control programs to communicate with the ACP system (see Figure 2) in order to accomplish a point-to-point or multipoint data transfer.

## Test tools and support programs

The structure of PARS-ACP requires a rather rigid discipline be imposed on all the programs in the system. The structure is necessary to achieve high performance. Test tools are used to impose the structure on the programs. Structural violations are automatically detected, flagged, and then corrected by the programmer.

Some examples of structural violations are:

- Using nonreentrant code.
- Using more storage than optimally permitted.

- Using more CPU cycles than optimally permitted.
- Exceeding fixed segment (program and data) sizes.

**test tools**

The test complex that has evolved over the years provides a comprehensive environment in which an application is run in a controlled, but realistic, on-line environment to ensure that the on-line discipline is adhered to. This is of paramount importance since systems of this type can tolerate only very small outages for short periods of time. An example from the experience of Trans World Airlines, a user of PARS-ACP, serves to make the point: During 1976, overall uptime was 98.70 percent, and uptime performance against schedule was 99.85 percent. Looked at another way, that year the system was scheduled to be operational an average of 23 hours and 43 minutes a day. It was actually operational 23 hours and 40 minutes a day. And there were 270 days on which there was no unscheduled downtime. There were 131 total outages. The mean outage was 6.0 minutes despite the fact that a unique operational incident caused one single outage to last as long as 230 minutes.

The test environment must very closely approximate the real environment. In the initial systems, this approximation was accomplished by simulating every known condition that could occur in the real-time environment. In the later systems, the actual (ACP) system was used as the basis. Hence, the real-time environment was "real" to a much larger degree.

The resultant test facilities in PARS-ACP provide:

- A check on violations of programming conventions.
- An orderly progression from simple debugging through complex multiprogramming tests, including the entry of messages from terminals.
- A uniform data definition and data base for use in all levels of testing.
- The ability to batch various test runs.
- A flexible method to specify and/or modify data for each test, including the facility to restore the test data base between individual test cases.
- A method of simulating unavailable programs.
- Flexibility in specifying the types of output desired.
- On-line components to assist in the detection of faulty programs.
- Off-line components to print the results of a test.
- Stress testing, e.g., the capability to drive the system at high message rates for sustained periods of time.

Appropriate test methodology is required to give completeness and structure to the test sequence. This includes planning, preparation, control, executions, post analysis, and corrections.

Other functions required to support the on-line operation fall in the general category of support programs. Some (post-processors, data reduction) execute off-line in the backup machine. Others, which support the on-line operation directly, run in the on-line system. These programs were designed so as to minimize the impact on system availability. For example, the function of data base/capture must be performed periodically, so that in the event of a catastrophic data-base failure, it can be restored to at least a certain point in time. A stand-alone disk-to-tape utility could be used in an off-line environment to accomplish the task; however, this would require that the system go "off the air." In ACP, the function had to be designed as a major subsystem running on-line in real time with complex checkpointing subfunctions, while optimizing system resources (channels, drives) so as to complete the process in minimal elapsed time. The resulting facility can be invoked during regular operation and run in parallel with normal activity.

**support programs**

## Data collection-reduction

The PARS-ACP systems are designed for a six-to-eight-year life. Because they tend to be highly structured and, hence, not easily changed, a means is provided whereby:

- The system can be "tuned" to peak efficiency during installations and after the start-up phases.
- Daily monitoring of system performance can be readily achieved.
- Long-term trends can be observed, thus predicting growth in system load and justification for future expansion.

Data about system variables is collected during the operation of the on-line system and then correlated and interpreted so that decisions can be made to ensure efficient system operation and growth.

The data collection and reduction programs of PARS-ACP accomplish the required functions. As experience is gained with each successive installation, the importance of this aspect of the system becomes more apparent. Changes are continually being made to improve on the techniques.

The data collection programs operate on-line in real time. These programs are invoked on a periodic basis under operator control to: (a) read out counters (which are continuously updated), (b) intercept and record specific events, such as data and program reads, and (c) dynamically sample parameters that fluctuate with time (device queues, storage pool lists).

Table 1 Kinds of data to be collected

| |
|---|
| *Messages* |
| Arrival Time |
| Text |
| Length |
| Type |
| Output Queue Lengths |
| Total Number |
| Time Sent |
| *System* |
| Input Queue Length |
| Ready Queue Lengths |
| Count of High-Speed Messages |
| Count of Low-Speed Messages |
| Elapsed Time in Various CPU States |
| Storage Block Usage |
| Count of Active Entry Control Blocks |
| *Program* |
| Count of Program Execution |
| Program Residency |
| *Files* |
| Count of Record Accesses |
| Count of Program Accesses |
| Count of Tape Writes |
| Count of Device Queues |

The data is recorded, according to function, on tape. Each record is time-stamped to indicate chronological order and contains control information for reduction purposes. The kinds of data that can be collected are listed in Table 1.

The data reduction program runs off-line under OS/VS. The data is edited, sorted, and arranged chronologically ascending, uniquely by type, in single arrays. Arithmetic operations are performed to describe the system phenomena in user terminology, i.e., messages per second, percent of time CPU is busy, etc.

Some important observations, based on the PARS-ACP data collection-reduction experience, are:

- With real-time operations, a prime factor is the load and interference imposed by the collection programs. Minimum impact on normal processing operations is essential. Counters relevant to system variables are imbedded in and updated by ACP as part of the architecture: thus, no bias is introduced as the result of the accumulation of variables.
- It follows, then, that all data reduction can and should be done in the off-line system which would have no impact on the on-line system.
- Extreme care must be taken in the analysis of the reduced data. Judgments made from too little data can cause more problems than making no corrections.

- Software data collection and reduction facilities are not adequate to measure and predict all aspects of system performance. For example, the number of I/O accesses over a channel can be counted, but the time during which the channel is busy can only be approximated. For these reasons, special hardware monitors are used to supplement the performance measurement and analysis process.

## Concluding remarks

ACP is a special-purpose operating system that satisfies the needs of certain requirements (e.g., high message rates and high availability) for computationally trivial applications. As such, it imposes a rigid structure or discipline on the application in order to achieve the desired results.

Several of the methods and techniques discussed have transferable qualities. For instance, while other applications may not require 24-hour operation, they may have commensurately high uptime requirements for, say, a 12-hour day. All the high-availability attributes in ACP would then still apply.

The data-base concept is another example. The horizontal data distribution allows for optimum accessibility. Providing a fundamental access mechanism allows for either a customized high-performance data management system or a somewhat more general one to be built on that foundation.

The following are examples of the more common applications under ACP in use today:

*Airlines*:
Seat reservations
Fare quote
Ticketing
Boarding pass
Cargo

*Other*:
Police car dispatching
Car rental reservations and billing
Credit authorization
Hotel reservations
Loan/payment processing

In future systems, ACP is most often considered and recommended for:

*Bank teller systems*:
Demand deposit accounting
Time deposit accounting
Loan accounting
Financial switch, which is a message switching and forwarding application between various terminal devices (cash dispensers, teller machines, point-of-sale terminals) and participating member bank systems.

In addition to the interest in the financial industry, ACP is increasingly being considered in the domain of networking implementations. Most notably, it is conceptualized as a front-end processing system handling the "computationally trivial" transactions (with the most stringent response criteria) and passing the more complex ones to a more sophisticated data-base management system.

A final comment: Careful analysis and consideration must be given when designing this type of system. High message rates can be attained at the expense of some function or flexibility; that is, a "short cut" is always a trade-off for which compensation must ultimately be obtained.

## CITED REFERENCES AND FOOTNOTES


1. SABER bore a strong imprint in philosophy (indeed in name) from a predecessor system called SAGE (Semi-Automatic Ground Environment), an electronic air defense network developed during the 1950s.
2. This concept was originally developed for the IBM 1301 Disk Storage System used in SABRE and was called the "record ready" feature.
3. It is general knowledge that airline reservations systems are automated. Most people are surprised to learn, however, that the casual telephone conversation with the reservations agent usually results in five to ten messages generated to and from a remote computer, 70 data-base accesses, and 100,000 instruction executions during that conversation.
4. The "typical" PARS message is defined as 15,000 instruction executions and 10 data-base accesses—a relatively complex application profile for ACP.
5. Processing time as used here represents only that time taken for instruction execution and excludes all waiting and I/O transfer time.
6. *ACP System Message Routing Concepts*, No. GH20-1693, IBM Corporation, Data Processing Division, White Plains, New York.
7. The name CPU loop is somewhat of a misnomer but still descriptive of the concept. The original dispatcher was indeed a programmed loop. However, System/360 and System/370 architecture is better utilized by entering the WAIT state at a point where all the queues have been found empty—the conceptual "bottom" of the loop. Interrupt processing would eventually cause program execution to commence at the previously described entry point—the conceptual "top" of the loop.
8. A message can be "traced" through the system by envisioning it in the various processing stages as shown in Figure 3.
9. R. Heistand, *ACP System Concepts and Facilities*, "Data Communications," Chapter 6, No. GH20-1473, IBM Corporation, Data Processing Division, White Plains, New York.
10. J. R. Knight, "A case study: Airlines reservation systems," *Proceedings of the IEEE* **60**, No. 11, 1423–1431 (November 1972).

11. J. H. McFadyen, "Systems Network Architecture: An overview," *IBM Systems Journal* **15,** No. 1, 4–23 (1976).
12. R. A. Donnan and J. R. Kersey, "Synchronous data link control: A perspective," *IBM Systems Journal* **13,** No. 2, 140–162 (1974).
13. The information field is unrestricted in content; its content is transparent (invisible) to the components of the protocol.
14. A code named after its inventor. Five bits are used to represent each character. Since this allows for only $2^5$ or 32 character combinations, two shift characters are utilized and effectively increase the character representation to 57.

## GENERAL REFERENCES

1. G. Burck, "'On-line' in 'real time,'" *Fortune Magazine,* 141–145 (April 1964).

2. M. O. Duke, "Testing in a complex systems environment," *IBM Systems Journal* **14,** No. 4, 353–365 (1975).

3. H. Hellerman and T. F. Conroy, *Computer System Performance*, McGraw-Hill Book Co., Inc., New York (1975).

4. M. N. Perry and W. R. Plugge, "American Airlines SABRE electronic reservation system," *AFIPS Conference Proceedings, Western Joint Computer Conference* **19,** 593–601 (May 1961).

5. A. Shaw, *The Logical Design of Operating Systems*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey (1974).

6. *IBM Synchronous Data Link Control–General Information*, No. GA27-3093, IBM Corporation, Data Processing Division, White Plains, New York.

7. *IBM Systems Network Architecture–General Information*, No. GA27-3102, IBM Corporation, Data Processing Division, White Plains, New York.

8. *D.P. News*, National Edition, IBM Corporation, Armonk, New York (September 11, 1967).

Reprint Order No. G321-5051.