Application development productivity is a broad-based concern. A system answering this concern is the IBM Health Care Support/DL/I-Patient Care System announced by IBM in late 1977. The system is of general importance because its application development system architecture is not application specific and thus can be used for the rapid development of many types of on-line systems. It has an elegant simplicity, and it uses the standard facilities of such operating system components as CICS/VS and DL/I. The application productivity has been clearly and successfully demonstrated in the real working environment of the Dallas County Hospital District (Parkland Memorial Hospital) and other sites. The paper provides an architectural overview followed by a description with an example of CRT (cathode ray tube) screen and print format design and coding and an examination of a data collection list to demonstrate the power of that facility.

Application development system: The software architecture of the IBM Health Care Support/DL/I-Patient Care System

by D. J. Mishelevich and D. Van Slyke

As evidenced by the recent Conference on Application Development Systems,[1] significant interest is beginning to focus on application development systems and, thus, application development productivity. True productivity enhancement must be two-sided: for the designer/implementor on the one hand, and for the user on the other. A major contribution in this area is the IBM Health Care Support/DL/I-Patient Care System (PCS) which was developed by Duke University Medical Center with guidance from IBM as the Duke Hospital Information System (DHIS). It is both a new software architecture with significant productivity features *and* a successfully implemented large application. PCS has been available as an Installed User Program (IUP) from IBM since November 1977. The Dallas County Hospital District (DCHD) in Texas, which owns and operates the Parkland Memorial Hospital (PMH), became intimately involved with the system in mid-1977 when the hospital became the validation site to show that DHIS was indeed transportable. The DCHD implementation[2-7] is called POIS (Parkland On-Line Information System).

A Hospital Information System (HIS) of this type is a comprehensive on-line system with CRT (cathode ray tube) terminals (display devices) and printers located in nursing stations and other patient care areas and in the ancillary departments (e.g., Radiology, Laboratories, Pharmacy, Central Supply, etc.). Entry of orders, inquiry into order status, message switching, patient profiles, medication profiles, and a number of other features, including the reporting of results, where appropriate, are supported.

hardware and systems software environment

The current DCHD system is on a System/370 Model 168-3 (eight megabytes) with IBM 3350 disk drives. By the summer of 1980, over 300 CRTs (primarily IBM 3278 display devices, many with light pens because of the primary menu-selection approach of the HIS application) and 100 printers (primarily IBM 3287 printers) were installed. System control programming is MVS (Multiple Virtual Storage), and the additional software consists of CICS/VS (Customer Information Control System/Virtual Storage) and DL/I (Data Language I), TSO (Time Sharing Option), and SPF (Structured Programming Facility), as well as COBOL, PL/I, assembly language, and appropriate utilities.

The Good Samaritan Hospital in Cincinnati, where the DOS (Disk Operating System) version of PCS was done on a System/370 Model 138, indicates that smaller CPUs are practical for use of the system, too.[8,9]

Application development system

PCS is intended to do more than just provide an application development system for data processing. PCS extends the application development tool directly to professionals outside of those in data processing and makes productive the entire application development process from the initial conception to production to maintenance/enhancement.

Some of the essential elements for productivity enhancement in an application development system of this type are

1. Data independence
2. Logic independence
3. Ease of logic implementation
4. Ease of CRT screen design and coding
5. Ease of printer format design and coding
6. Application and architecture extensibility
7. Ability of user personnel (non-data-processing) to perform functions of Items 3, 4, and 5
8. Human-engineered, user-friendly production system

PCS meets these criteria, as will be demonstrated in some detail in

this paper for all items except 6, which is being effectively accomplished but is beyond the scope of this presentation. Item 8 is evidenced by literally hundreds of users at Parkland from ward secretaries to physicians who use the system effectively after minimal training. Item 3 is accomplished via the Data Collection List (DCL) facility.

Patient Care System architecture

A detailed architectural presentation is beyond the scope of this paper. For additional material, the reader is referred to the available IBM documentation[9-13] as well as a paper by the authors.[14]

It is important to note that a major facet of PCS is that much of the analysis, design for, and incorporation of both the CRT screens and printer formats is done by health-care professionals (usually nursing or ancillary department personnel) whose skill is a detailed knowledge of the application rather than data processing expertise. We call our group of user-coordinators POIS Associates.

Having user personnel in the designer/implementor role is multifold. For example, it provides for a built-in understanding of the application problem, job enrichment, commitment to the specific application field, and frequently even a commitment to the specific organization. This is particularly important when data processing personnel are often very scarce and very mobile.

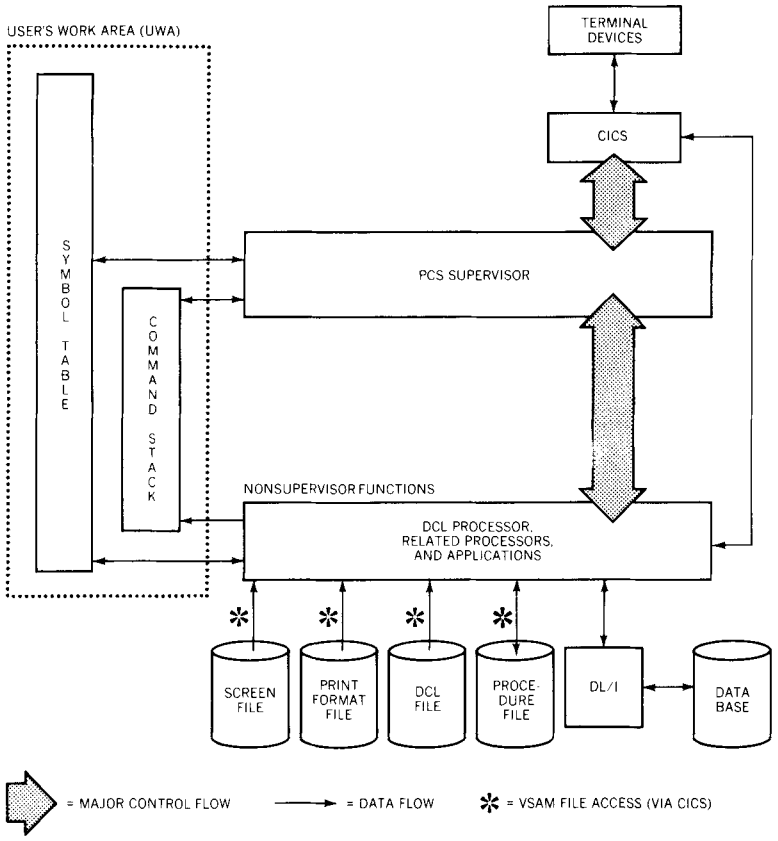
architecture overview

A graphic presentation of the on-line architecture is shown in Figure 1. As previously noted, terminal users interact with PCS via CICS. The PCS Supervisor consists of the Supervisor Nucleus, Symbol Table management, Nonconversational Screen Management, Execution Debug, and Error Handler. Program control flows through the PCS Supervisor to and from the nonsupervisory functions which consist of the Data Collection List Processor, Edit Facility, Data Manager, Print Manager, Conversational Screen Manager, Security Facility, and application programs.

Each user has a User's Work Area (UWA), which contains the CICS Transaction Work Area (TWA), the user's Symbol Table and Command Stack. The TWA contains status information which may be used by the various supervisory and nonsupervisory programs. All functions can add to or access elements of the Symbol Table. Programs and screens can add commands to the Command Stack in the UWA, but only the PCS Supervisor removes/executes them.

The screen, print format, and DCL files shown in Figure 1 are relatively static (e.g., static in the production environment) files

Figure 1 PCS architectural overview (DCL = Data Collection List; screen file is also available to the nonconversational screen manager within the PCS supervisor box)



required in any PCS implementation. The DCL itself is read in from the DCL file by the DCL processor. However, since many similar procedures can sometimes share a DCL, the procedure file (see below) may be accessed first to include default information specific to one procedure in the Symbol Table.

The procedure file is a relatively static application-specific file. In the Hospital Information System context, it is a charge description master file. The procedure record also contains the name of a master DCL that is used to control the application logic flow for the procedures sharing a DCL.

The application-specific (HIS), nonstatic data is contained in DL/I data bases and accessed via the DL/I Data Base Management System.

Major factors: Data and logic independence

The productivity-enabling aspects depend on two PCS basic concepts: data independence and logic independence. Even though

PCS uses CICS and DL/I as do other IBM transaction-processing environments, a major difference is the independence in these two areas achieved with PCS.

data independence

In a traditional (nonapplication development) on-line programming environment (such as that represented by CICS and DL/I), data are generally accessed from the terminal devices or data base via calls within the internal application program logic. In application development environments (such as the Development Management System (DMS)[15] or the Application Development Facility (ADF)[16,17]), the application data are usually accessed through higher-level functions such as screen and data base managers. Although data may be accessed on the screens by name, the application programs are generally limited to access by predefined data structures as the data flow between the application programs, the data base, and the terminal devices. Also, in most cases, no facility exists for the application designer to easily maintain active data; therefore, repeated access to the data base or terminal may be required.

In PCS, however, more data independence is achieved. The data are referenced by different programs via the use of a Symbol Table, and the data base management functions of PCS take care of the DL/I calls. The application code need not know about the intricacies of CICS because the PCS screen and print managers take care of the CICS interactions. The situation is shown schematically in Figure 2. The PCS application program gets its input data from and puts its output data in the Symbol Table without concern for either the source (CRT screen, data base, etc.) or destination (CRT screen, printer, data base, etc.) of data.

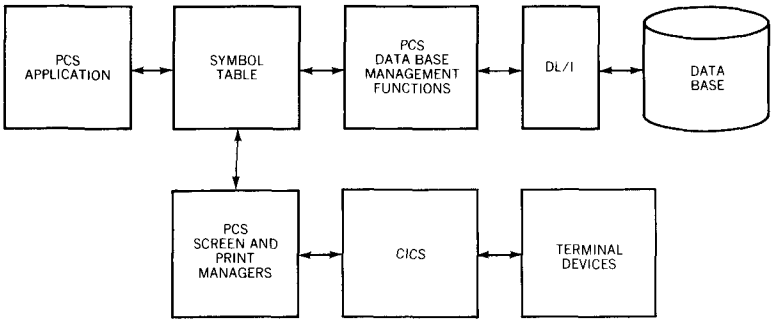
The functionality of the Symbol Table is somewhat demonstrated by conceptually similar data structures in other environments, for example, the data area within an APL work space[18] or the variable pool within SPF (Structured Programming Facility[19,20]). The Symbol Table provides sufficient data independence so that all of the logic required for application development can be maintained in a modular fashion, thus allowing new application functions to be generated out of existing modular components, programs, screens, print formats, etc. In addition, the designer may retain active data in the Symbol Table, thereby improving performance and usability.

logic independence

One might now reasonably ask how the relationships between application programs, data base functions, and terminal I/O functions are controlled.

In a traditional on-line programming environment, application logic flow and application logic for data collection or data update are determined internally by program logic. In application development environments, application logic flow can usually be deter-

Figure 2 Diagram of data independence in PCS



mined externally to the program logic. However, externally specified application logic is generally limited to logic following a given sequence of events (hereafter called sequence-driven logic).

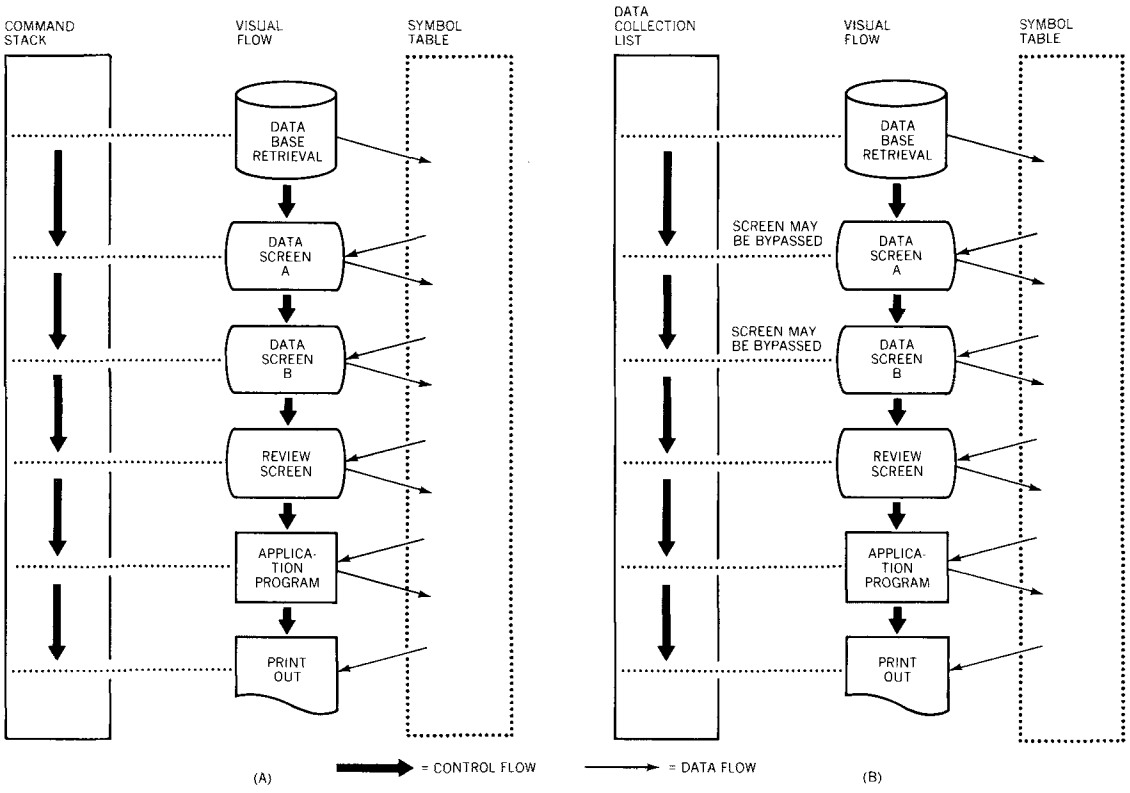
In PCS, however, more logic independence is achieved. Application logic flow may be specified externally to the program logic in either screen definitions (sequence-driven logic) or in DCLs, which will be referred to as data-driven logic. PCS is the only known system with this facility. In either case, a modular approach is used, putting the various components together as building blocks.[21] A change in the one program or screen element used in a DCL will suffice to incorporate that change in any DCL that invokes that element.

Sequence-driven logic flow specified in multiple screen definitions may be combined at execution time (due to operator selections on multiple screens) into a logic flow holding area called the Command Stack. Parenthetically, the use of the stack approach contributes to the inherent structured nature of the system.

Data-driven logic flow also allows a sequence of events to be specified. However, at execution time, some data collection events may be skipped if the data were defaulted (by the DCL, previous screens, or programs) or were obtained in some previous interaction with the terminal user. Thus, the potential exists for greater terminal user productivity. Data that have been previously gathered need not be requested again during subsequent sequences.

The situation is shown schematically in Figures 3A (a sequence-driven approach) and 3B (a data-driven approach). In both figures, a visual representation of the application logic flow is depicted in the center column entitled Visual Flow. The right side of the figure indicates that the data interaction of each of the components is with the Symbol Table. The left side of each figure

Figure 3 Diagram of logic independence in PCS, (A) Sequence-driven, (B) Data-driven



depicts graphically the logic flow holding area corresponding to the visual flow. Although this example is illustrated using either type of logic, the data-driven approach is more flexible for the terminal user.

Many application flows can be done with either approach, but sequence-driven logic is most appropriate for flow of control, and data-driven logic is best used where collection (or update) of data is required.

Screen design and coding

Screen design and coding have been provided to permit users (in this case usually health-care professionals) to easily code their own screens. The screen images in our environment are for practical purposes all designed by the health-care professionals of the POIS Associates group. Such design is possible because screen coding has been greatly simplified compared with most other

techniques. Every effort has been made to extend the screen and print format coding process to the user who is not familiar with data processing.

Screen coding consists of providing two sets of information: (1) a "picture" of how the screen looks, and (2) a list of the data elements that are to occur on the screen (involved in either input or output). Although this information can be entered by any data entry medium, including cards, we use the CRT as our input device.

After the above items are entered, they are processed by the Screen Compiler. Output of the Screen Compiler is a printed report showing the "picture" and list of data elements in hard copy form and includes any applicable warning and error messages.

self-documenting

In our experience at Parkland Hospital, it is this output which is first reviewed with the end users. Thus, the screen coding is *self-documenting* since it is both coded and printed in "picture" format. In addition, the screen coding is executable even before any of the programs or data bases have been established because PCS data and logic independence allow separation of the functional components: screens, programs, DCLs, data bases, print formats, etc.

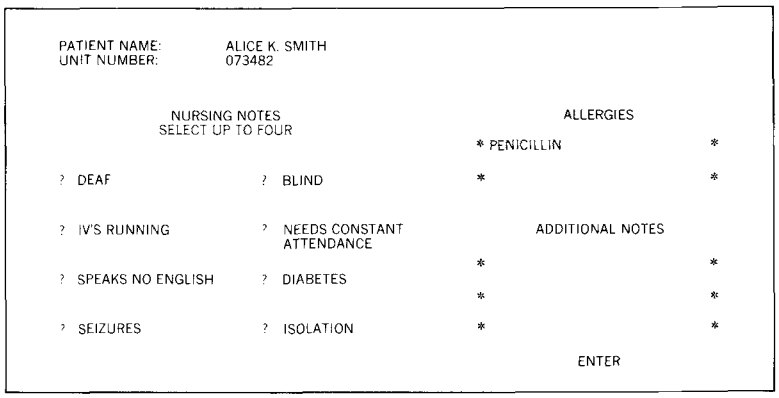
Once the screen flow is envisioned and coded by the designer, the application functions can be demonstrated on line to the end user. Our technique is to do this second review with the end users as soon as possible since experience has shown that changes are still likely at this stage. (The phenomenon of desire for change at this point is real even though in most environments the paper form of the screen design and flow would be the one implemented for production.)

early demonstration of application

The ability to *demonstrate* the application functions during the design phase is the most productive design capability of PCS. There are several factors involved: (1) At this stage, there are no data bases or program specifications to change, and the effect of the changes on the design process is helpful; (2) The user is pleased/encouraged to see a working model so soon and becomes more involved in an earnest review of the system; (3) At least one rework after development or production is avoided (due to the phenomenon noted above). A critical element is that the screens shown to the user to get user feedback are already coded for actual use.

Note that many of the idea, design, develop, test, demonstrate, rework, educate, etc. stages can now operate in parallel. Other time-compressing benefits accrue: (1) The designer is guided by the early feedback from the end user and moves quickly ahead;

Figure 4 Typical CRT screen as initially seen by the user (The question marks before each item in the two columns on the left indicate that those items are delayed-detectable. Asterisks are there simply to delineate areas in which the user can key in input.)



(2) The end user begins to make administrative plans for incorporation of the new feature or system (a calendar time-consuming process); (3) User manuals and other types of education and training material can be started. The major benefit is the ability to produce a better application design in a shorter time using fewer resources.

Screen coding considerations

The techniques used for the simplified screen coding are of particular interest because they seem natural, and one eventually wonders why screen coding has not always been done this way. In addition, the menu-selection method used in the HIS application is "user friendly" and allows complete operation of the application with a minimum of training. Screen coding considerations follow.

A sample screen as initially seen by the user is shown in Figure 4. In the normal course of events in running an application, "static data" like "unit number is" or "variable data" like "073482" (the unit number for the particular patient under consideration), are transmitted from PCS to the CRT terminal. These items show what choices are available for light-pen-mediated menu selection by the user and/or what items are to be keyed in by the user. The data keyed in by the user are also called "variable data."

menu selection types

Menu selection is of two types: *delayed-detectable*, in which case the selected item is "tagged" as having been selected, but the screen remains so that additional menu selections and/or key entries can still be made, and *immediate-detectable*, in which case the screen is transmitted back immediately to the host computer, and a new screen is sent to the CRT.

Figure 5 Screen in Figure 4 as typically filled out by the user just before transmission of the screen back to the host (Note that the menu-selected items in the two left-most columns are indicated by having their question marks changed to >.)

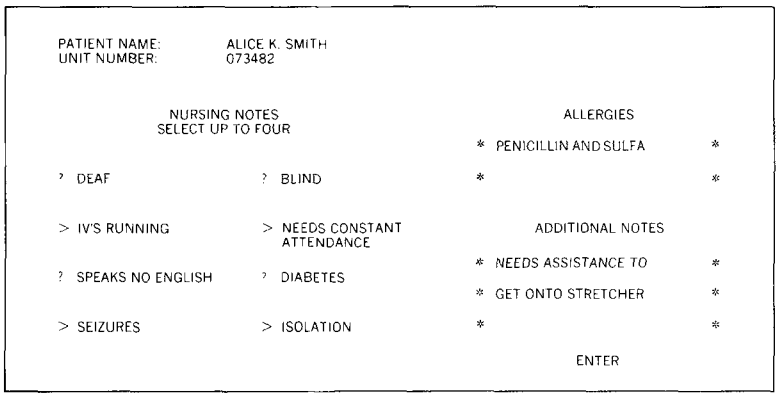
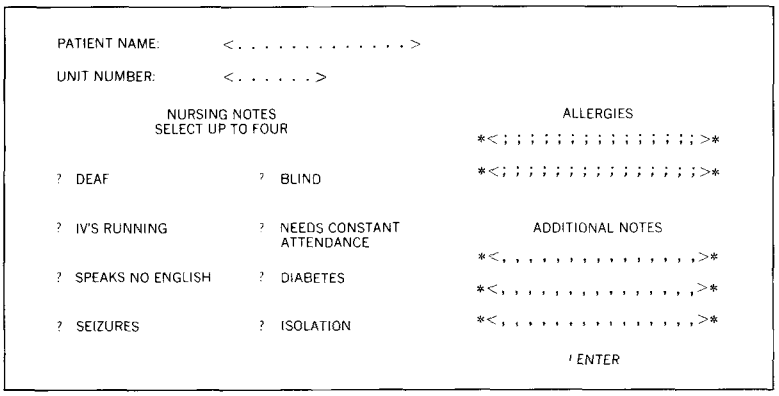


Figure 6 Design layout of the screen considered in Figures 4 and 5



After the user has interacted with the screen (just prior to transmission back to the host), the screen looks as shown in Figure 5. The design layout for the screen is shown in Figure 6. Here <.....> fields represent variable data with the periods indicating that those particular fields are for data output (from the computer to the CRT). The <,,,,,,,,,> fields represent variable data with the commas indicating that these particular fields are for data input (from the CRT to the computer). The <;;;;;;;;;> fields represent variable data with the semicolons indicating that these particular fields are for both data output and data input. This last representation is used since the allergy text shown in the figures may be retrieved from the patient's profile which is stored in the data base and may be updated at this time. As to the pen-detectable fields, the ones preceded by question marks are *delayed-de-*

tectable, whereas the exclamation point appearing just before ENTER in the lower right corner of the screen in Figure 6 indicates that field is *immediate-detectable*.

Although not shown in the figures, it is possible to include simple edit characters in the output or input/output variable data fields by including the edit characters among the periods or semicolons. Consider a date and time field internally carried as MMDDYYHHMM or 1122791005. This field could be displayed to the user as

```
CURRENT DATE AND TIME
 11/22/79              10:05
```

with the following screen design layout:

```
CURRENT DATE AND TIME
 <../../..             ..:..>
```

In any case, the number of characters between the less than (<) and greater than (>) signs indicates the maximum number of characters permitted as input and/or output in the field.

Some other considerations are:

1. The character string transmitted from the CRT terminal to the computer for light-pen-detectable fields is called a *generated value* and may or may not be the same as the phrase appearing on the screen. For example, the word "isolation" might be transmitted as 007 or some other alphanumeric value.

   A generated value may also be a command or set of commands, which is, of course, an *extremely* powerful facility. For example, if the choices on a CRT screen were

   ```
   CANCEL EXAM
   RESCHEDULE EXAM
   ```

   and reschedule exam was selected, then the generated value might be

   ```
   $S=RCANREAS,$P=RRESCHD;
   ```

   which means display the screen RCANREAS to get the reason for the cancellation and then execute the DCL called RRESCHD. Note that $P= invokes the DCL. A $PROG= command can also be a component of such a command string.

   Clearly the data names and values must be coordinated among the screens, DCLs, and programs.

2. Displays can include three degrees of character intensity for the IBM 3278 display: bright, normal, and dark. Keyed-in characters, characters in variable output fields, and pen-selectable phrase strings are *always* of a bright intensity rather than normal. Static output fields may also be made bright by prefixing

them with a double-quote mark ("). The system also supports the IBM 3279 color display.
3. Some data names by convention serve special purposes. For example, $AUTO causes the same generated value (e.g., T-STATUS=31;) regardless of what the user enters, $COMMON is used for common functions like ENTER that are likely to appear on many screens, and $CMD01 is used to call command strings (such as $S=RCANREAS,$P=RRESCHD;).

Again, in addition to the layout of the screen, "supplemental" information is also provided by the screen designer to the screen compiler about which data names are to be associated with which input and/or output fields. An example for an output case follows.

Let us say the layout specification calls for

```
ORDER ENTERED
     DATE<.....>          TIME<....>
SPECIMEN COLLECTION
     DATE<.....>          TIME<....>
```

The fixing of the relationship between these variable fields and their corresponding data names can be done by specifying the following fields in a vertical or horizontal sequence. Thus

```
OUTPUT....V....ORDERDAT
               COLCDAT
               ORDERTIM
               COLCTIM
```

(where V indicates that a vertical precedence over left to right sequence is being followed) is appropriate and equivalent to

```
OUTPUT....H....ORDERDAT
               ORDERTIM
               COLCDAT
               COLCTIM
```

(where H indicates that a horizontal precedence over top to bottom sequence is being followed).

The input supplemental information requires a bit more specification. For the whole screen a vertical or horizontal field sequence must be designated. For each statement the following can or must be specified:

1. !, ?, or < indicating immediate-detectable, delayed-detectable, and keyed-in, respectively (mandatory)
2. Data name (mandatory)
3. Number of light-pen-selectable phrases (if applicable)
4. Maximum number of phrases to be selected (if applicable)
5. Generated values or "AS IS" (if applicable)

As an example, let us assume that we wish a screen to enter in a vertical sequence the name of the physician (keyed in) ordering the procedure plus the scheduling and transportation (both to be menu-selected). The screen layout specifications call for

```
                    ORDERING PHYSICIAN
                    <,,,,,,,,,,,,,,,,,,,,>
SCHEDULING                                  TRANSPORTATION
   ? STAT                                     !WHEELCHAIR
   ? TODAY                                    ! STRETCHER
   ? TOMORROW                                 ! BED
```

The supplemental information would be

```
V....?SCHED          003 001    STAT;TODY;TOMR;
   <MDNAME
   !TRANSPRT         003 001    WC;ST;BD;
```

with definitions according to the list above. The ?SCHED fields are listed before <MDNAME because vertical definition is requested and the question marks are further to the left than the less than sign. Also note that a horizontal sequence is not particularly applicable since the scheduling and transportation options are listed in columns on the same lines. Horizontal sequence would cause the following definitions:

```
H....<MDNAME
   ?SCHED            001 001    STAT;
   !TRANSPRT         001 001    WC;
   ?SCHED            001 001    TODY;
   !TRANSPRT         001 001    ST;
   ?SCHED            001 001    TOMR;
   !TRANSPRT         001 001    BD;
```

which are much more difficult to code. In addition, a problem is created since the operator can now select all three scheduling options.

The screen facility of PCS has been developed and implemented not only as a very powerful component of PCS, but also as a human-engineered system that can be extremely effective when used by a designer not familiar with data processing, such as one of our health-care professionals. Thus, PCS application systems (which need not be health-care related) are developed and implemented in the main by users for users.

Print format design and coding

Some application development systems have a high-level method of specifying screens, but the designer has to revert to program code for printer output. With PCS, a high-level and spooling-type print facility serves the designer's needs.

The procedures and conventions for print formats are generally the same as for screens, except, of course, that only output fields are permitted. The layout and sequential information statements are processed by the print compiler program. There are two format types:

- Report—if several items with the same format are to occur on the same printed page and the number of these items is not predictable so that a new page is to be started when the current page is filled.
- Message—if report conditions are not applicable, that is, if the length and content of the format are fixed.

Some additional considerations are:

1. The specification of repetitive lines can be abbreviated. Thus

```
CLINICAL HISTORY:
    <.....>
    <.....>
    <.....>
    <.....>
```

can be abbreviated as

```
CLINICAL HISTORY:
    <04 LINES.....>
```

2. One can include a stamp on the top or bottom of the printed data *message* (or *report*) containing the source, destination, date, and time.
3. One or more destinations and/or groups of destinations can receive the *message* or *report*. A group might be all the nursing stations. Home (the printer associated with the station where the message is being initiated) is a legitimate destination and may be used along with other destinations if desired.
4. Blank lines can be suppressed. For instance, in the clinical history above, if only one line of the variable information was keyed in, the printing of the other three that appear as blank lines can be suppressed.
5. Printout may be rerouted to the same or any alternate destination (including the system printers) for a period of 24 hours (based on disk queuing space) after the initial print request.

Again, the print facility is another powerful tool to quickly and economically specify the desired result and one that is easily handled by appropriate non-data-processing personnel.

Data Collection List

DCLs provide the primary mechanism for data gathering and distribution (including data base and Symbol Table update). A DCL

provides the specification of the various data elements needed for a given function/procedure including the source or destination of the data. To get the data, screens may be presented and/or programs executed. A DCL does not change the logical system flow, but once a given function/procedure is invoked, it makes sure that the required steps are taken to complete that function/procedure. The DCL maintains control until its function has been completed. After screens are displayed, messages printed, and/or designated programs executed, control returns to the DCL processor to ensure system flow integrity. As with the screens, design and programming are merged to a great extent. The DCL and Symbol Table, which are unique to PCS, are naturally linked to each other and have a symbiotic/synergistic relationship.

Quite naturally, the bulk of DCLs in a Hospital Information System will deal with placing and executing orders. Examples include order entry, order display, charging, canceling orders, result reporting, and displaying a result. Other functions, such as patient transfer, are also quite reasonable.

As an example of data collection, if one places a radiology order, a radiology DCL would cause data entry screens to be displayed to gather (more than one data item can be obtained via a single screen if appropriate) procedure type, clinical history, ordering physician, nursing history, allergies, transportation, and scheduling (not necessarily in that order).

Data can be obtained (1) because a default value is specified (and not overridden), (2) from a CRT screen, or (3) from a program. Data editing can occur regardless of data source.

life cycle support

A natural concept is that of orders or procedures having a life cycle. Many systems, (e.g., parts inventory, airline reservations, truck or rail shipping, etc.) have life cycles. However, the DCL facility that easily and powerfully supports life cycles is unique to PCS. The life cycle consists of (1) Initiation, (2) Execution phase(s), and (3) Termination (may have alternate forms).

Various stages in a typical order life cycle might involve (with numerical keys to the three items just identified)

1. Order entry
2A. Verify order
2B. Record complete procedure
3A. Report final result
3B. Correct final result

An alternate form of termination could be 3C, order cancellation. After execution of each stage in the order life cycle, the order data may take on a new status such as unverified, outstanding, result pending, final result, or corrected result.

With respect to the life cycle, the DCL supports the stages of the life cycle through the use of status codes. Each of the successive stages is assigned a two-digit number (called status code) with ascending sequence used to denote the relative position of that stage in the life cycle. The execution of one stage in the DCL is controlled by setting a T-Status (Target Status) field to the status code value of that stage prior to execution of the DCL.

Items that will or can appear in a DCL include (1) a list of necessary data elements, and (2) a list of appropriate commands.

The general form for a Data Element DCL entry is

```
DATANAME    DEFAULT,SOURCE,EDIT,ERROR;
```

Suppose that we wish to obtain the type of transportation to be used for transfer of a patient from a nursing station to one of the ancillary (service) departments such as Radiology. In this case, a DCL entry for this item could be

```
ORDTRANS    WC,$S=TRANSPRT,$EP=EDITTRAN,$ERRS=TRANSPRT;
```

where

1. The name of the data element is ORDTRANS (with the characters "ORD" indicating this is part of an order).
2. WC is the default transportation method (wheelchair).
3. $S=TRANSPRT says that value is to be obtained from a screen whose name is TRANSPRT (transportation route).
4. $EP=EDITTRAN is the name of the program to edit the input value. Even though the particular example we are using involved a default value (WC), the editing process still makes sense since the terminal operator may be allowed to override the default (this is not shown here).
5. $ERRS=TRANSPRT indicates that if any error is detected, the TRANSPRT screen will be displayed again.
6. The semicolon indicates the end of this DCL entry.

Note that all of these operands are not required for each entry. With light-pen entry it is likely that an edit program and/or error screen is not applicable.

The general form for a Command DCL entry is

```
COMMAND    OPERANDS(S);
```

Command DCL entries are easily distinguished from data element entries because the first character of the command is always a dollar sign ($). This is important since command and data element entries are interspersed in the DCL as in the example below. Continuing our radiology example from above, suppose that we now wish to send a printout of data gathered for a chest x-ray to radiology in the form of a requisition. Typical DCL entries for this request would be

Table 1 Types of entries for a DCL

Statement (And purpose)	*Parameter(s) (And explanations)*
DATA ELEMENT Obtain data	DEFAULT,SOURCE,EDIT,ERROR: (see text for details)
$RS Perform review function	RSNAME; Name of appropriate review screen
$DM Request external data function	FUNCNAME=SEGNAME,FUNCNAM2=SEGNAM2; Name of function to be performed and name of applicable group of data
$PRINT Request printing	PRINTFORMAT,PRIORITY,DESTID; Name of print format to be printed, name of print priority symbol, name of symbol of destination identification
$ACCT Send accounting data	None. A specified set of data elements must already be present in Symbol Table

```
T-PRIOR     N;
T-DESTRR    RADIOL;
$PRINT      RADREQ,T-PRIOR,T-DESTRR;
```

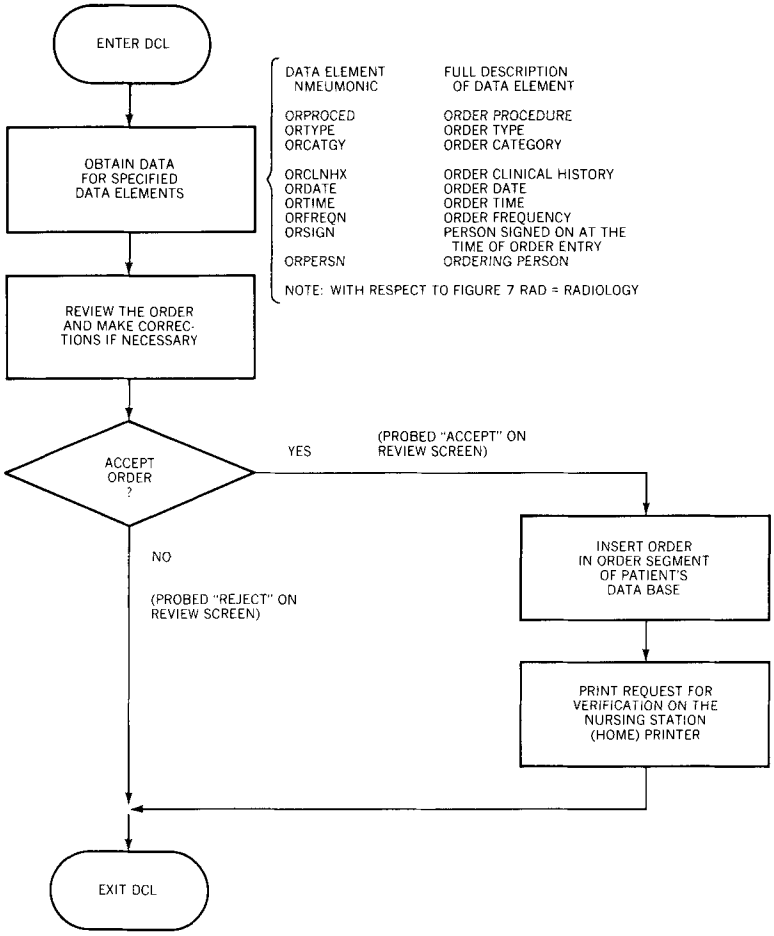
The first two lines show data element entries with only the default operand coded. The first entry defaults temporary data name T-PRIOR to a value of *N*. The second entry defaults temporary data element T-DESTRR to a value of RADIOL. Any data element beginning with T- is by convention a temporary data element that is used for a short time and is not stored in the data base. Note that in more complex illustrations, the data elements T-PRIOR and T-DESTRR could be assigned a value via a program or a screen.

The third line shows the actual print request which means execute a print command using a print format with the name RADREQ, an output priority of N (for normal) and a destination of RADIOL (logical destination indicating the printers in Radiology). Note that it is the values of the second and third operands that are used by the print request. In more complex cases, changing these values via screen input from the CRT operator allows the priority or destination to be changed.

Table 1 illustrates the types of entries that may be used in a DCL. Note that CICS and DL/I calls are made for you.

It would be possible for each stage in the life cycle of an order to be represented by a separate DCL. However, this would serve no useful purpose since many of the data elements and commands that must be specified in each stage of the order life cycle are common to all stages. Therefore, a mechanism exists to combine all stages of the life cycle into a powerful and highly compact structure.

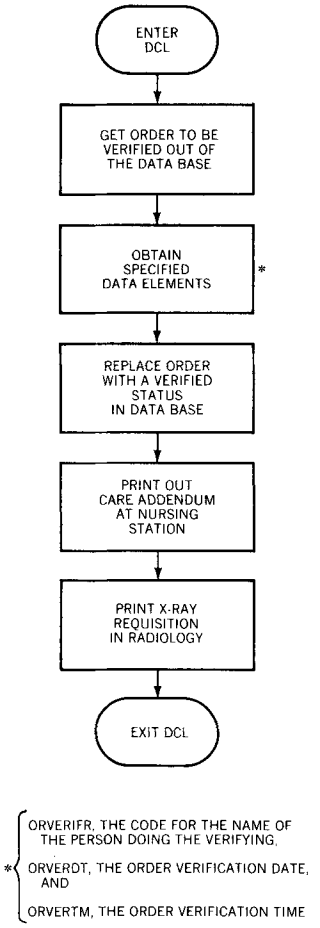
Figure 7 Flowchart of basic order process



The disposition of each entry in the DCL at any given stage is controlled by the comparison of the one or more status elements encoded for each line to the Target Status (T-Status). The value of T-Status at the time the DCL is entered will determine what stage is to be performed. Thus the *same* DCL can allow for such diverse stages in the order life cycle as:

1. Unverified order entry
2. Correct order before verification
3. Delete order before verification
4. Order verification
5. Enter an order verified
6. Order-associated materials received
7. Recording a complete procedure (RCP)
8. Recording an incomplete procedure (RICP)

Figure 8 Flowchart of order verification process



9. Report result
10. Correct result
11. Display order
12. Display result

Thus, the DCL is a powerful construct for permitting the coding of complex but related stages that share data elements and program calls. One can look at it as a series of statements to be executed (with data element lines handling one data element each) with (1) conditional invoking of each executable statement and (2) conditional treatment (acquiring, changing, using the existing value, disallowing the existing value, or using the default value in place of) for each data element.

To illustrate this concept, let us analyze an example. We will examine a DCL for order entry, order verification, recording a complete procedure (RCP), recording an incomplete procedure (RICP), and display order. The functions RCP and RICP indicate whether the outstanding order has been completed or definitely will not be performed, respectively. The flowcharts and data element names for these functions are shown in Figures 7 through 11.

The single DCL to accomplish all of the functions just described is shown in Figure 12. A major consideration is how control over the individual lines will be exercised, in terms of (1) which data element statements will be relevant, and (2) which commands ($DM, $RS, etc.) will be performed in a given execution of the DCL. One execution represents one stage such as enter orders. By using the T-Status at the time the DCL is entered, these questions are resolved by comparison of the T-Status to the status element or elements associated with each statement line of the DCL.

One or more status elements will appear on each DCL statement line whether the statement is a data element line or a command line. Thus, there is conditional control over both data and command. The three possibilities for the status elements are defined as

Status	*For data elements*	*For commands*		
S1	ALLOWED STATUS	LOW RANGE		...
S2	REQUIRED STATUS	...	*or*	EXACT VALUE
S3	NO CHANGE STATUS	HIGH RANGE		...

Each is a two-digit number, but how the interpretation is handled with respect to the comparison of the status elements to T-Status differs as to whether a data element entry or a command entry is involved. The status names only make sense in their own context and may be viewed simply as Status 1, Status 2, and Status 3. For data elements, one or more status values must be supplied. For commands, either Status 1 and Status 3 are coded or only Status 2 is coded.

With the above background, we can now apply the rules and see how overall functions are separated. In Figure 12, the right-most set of columns indicates the functions for which a given line (function or data element statement) will be active. The T-Status values upon entering the DCL are

Figure	*T-Status*	*Function*	*Purpose*
7	10	ORDER	Enter an order unverified
8	30	VERIFY	Verify a previously entered order
9	38	RCP	Record a complete procedure
10	82	RICP	Record an incomplete procedure
11	97	DISPLAY	Display order information retrieved from the data base

In each case the line selection may be compared to the logic in the corresponding flowchart as shown in Figures 7–11.

Please note in the lines of DCL code that 11 of the 31 lines serve more than one function because most of the common elements are supported. If we added just one more line after Line 18, S2 = 31, COMMAND=$DM, OPERAND=ISRT=ORDER;, we would get a sixth stage, "enter order verified."

Thus, one can see that all of the stages in an order life cycle can be encoded in a compact fashion (almost always a single page) using the powerful DCL construct.

Figure 9 Flowchart for record complete procedure

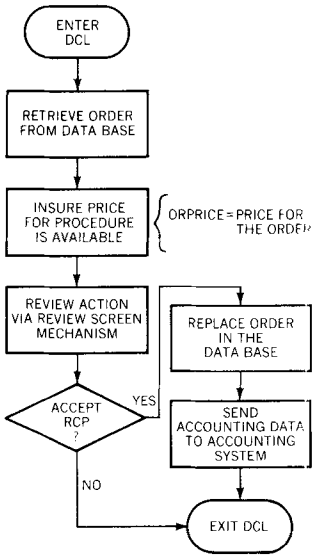
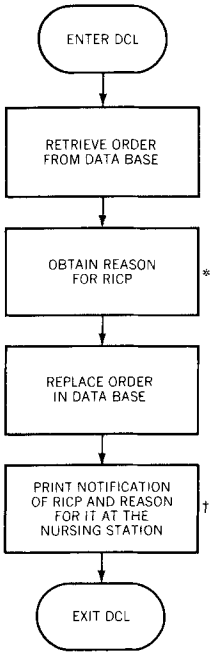


Figure 10 Flowchart for record incomplete procedure



* DATA ELEMENT NAME IS REASON

† HNURSTA = NURSING STATIONS WHERE RICP NOTIFICATION IS TO BE PRINTED (FROM LOCATION SEGMENT OF DATA BASE).

Other end-user-oriented features

While DCL, screen, and print format coding form the basis for the designer interface for those not working in data processing, other PCS facilities also play an important part in its end-user orientation. Some of these facilities are PCS Data Manager, Edit, Execution Debug, Security, and Error Handler. A brief discussion of each of these features follows.

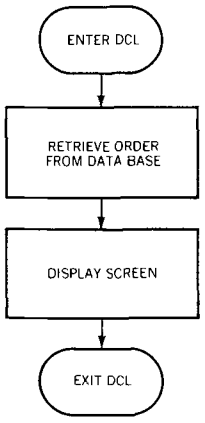
Data Manager

The Data Manager provides simple commands to access single or multiple segments/records without the need for programming. For instance, a line in a DCL shows

```
$DM        REPL=HPBASIC,GET=ORDERINF;
```

using the Data Manager command to: first, replace (REPL) Hospital Patient Basic (HPBASIC) information updated in the Symbol Table onto the data base, and second, retrieve (GET) order information (ORDERINF) about that patient. The general user is probably not aware that HPBASIC is a segment in the patient data base,

Figure 11 Flowchart for display order



whereas ORDERINF is a group of segments in the orders data base. The designer need only know that certain data names (generally key fields) need to be acquired before issuing a $DM request to retrieve or update a group of fields. The Data Manager automatically takes care of setting up the DL/I or VSAM (Virtual Storage Access Method) request, issuing the call(s), checking for errors, converting the data to display format, checking access security at the segment and field level, and mapping the data in or out of the Symbol Table. These functions are all handled automatically by the Data Manager using a compact, memory-resident data directory which is generally set up by the installation's Data Base Administrator. The function is called Data Manager rather than Data Base Manager because data may come from non-data-base sources such as VSAM files or even asynchronous input (e.g., via a Series/1 computer). The PCS-Data Manager was developed at the Cedars-Sinai Medical Center in Los Angeles.[22,23]

Edit

The Edit facility is designed to edit any field using a shorthand specification provided by the screen, DCL, or one of the installation's predefined edit tables. For instance, the following DCL lines

```
EDITQNTY    T(NN)L(003/RJ0)R(001-535);
QNTY        $S=KEYINQTY,$ET=EDITQNTY,$ERRS=ERSCREEN;
```

show the coding of the shorthand edit specifications (first line) and the Edit command (second line, command $ET=EDITQNTY). The Edit specification (EDITQNTY) can be used by one or more edit commands to check several fields. The specification ensures that: the type must be numeric, T(NN); the length is three and if not, it is right-justified using zero as the fill character, L(003/RJ0); and the range must be between one and 535 inclusively, R(001-535). This example shows three of the functions typically used by the designer. However, Edit provides 16 other functions such as DT — date, F — format, V — value is, VN — value not, and OV — override. Many of the functions such as Type shown above also have several subfunctions. Type has NN — numeric, NS — numeric signed, AA — alpha, AN — alphanumeric, and NM — name (last name comma first name). If the field is in error, Edit also provides standard flag, short, and/or normal error messages, and the designer may also code special error message text. PCS-Edit was developed by the Sisters of Charity Hospitals in Houston.[24,25]

Execution Debug

Just as assembler programmers have the CICS/VS On-Line Test/Debug II facility to single step transactions at the machine instruction level, and COBOL programmers have the CICS Execution Diagnostic Facility to single step their transactions at the CICS command level, so do non-data-processing designers also need

Figure 12 Sample Data Collection List

REF	PASS (See Note 2)	STATUS S1	STATUS S2	STATUS S3	COMMAND OR DATA NAME	OPERAND(S) TARGET STATUS →	EXPLANATIONS *FOR THIS PAPER ONLY* FUNCTIONS IN WHICH LINE ACTIVE (See Note 1 Below) O 10	V 30	RCP 38	RICP 82	D 97
1		33		97	$DM	GET=ORDER;			RCP	RICP	D
2		11		30	$DM	GET=ORDER;		V			
3				05	ORPROCED	(See Note 3)					
4				10	ORTYPE	RAD;	O				
5				05	ORCATGY	(See Note 3)					
6			10	32	ORCLNHX	$S=RADCLHX;	O	V			
7			10	32	ORDATE	$PROG=SYSDT;	O	V			
8			10	32	ORTIME	$PROG=SYSTM;	O	V			
9			10	32	ORFREQN	$S=RADFREQ,$EP=FREQEDIT	O	V			
10	P		10	11	ORSIGN	$PROG=SISIGN;	O				
11	P		10	32	ORPERSN	$S=ORDPERSN;	O	V			
12			05	64	ORPRICE	$EP=PRICE,$ERRS=PRICE;	O	V	RCP		
13		10		11	$RS	RREVIEW;	O				
14			38		$RS	RADRCP;			RCP		
15	P		30	32	ORVERIFR	$S=ORVERIFR;		V			
16			30	32	ORVERDT	$PROG=SYSDT;		V			
17			30	32	ORVERTM	$PROG=SYSTM;		V			
18			82		REASON	$S=RICPREAS;				RICP	
19			10		$DM	ISRT=ORDER;	O				
20		11		30	$DM	REPL=ORDER;		V			
21		35		89	$DM	REPL=ORDER;			RCP	RICP	
22		10		82	T-PRIOR	N;	O	V	RCP	RICP	
23		35		89	$ACCT				RCP	RICP	
24		10		32	T-DESTID	HOME;	O	V			
25			10		$PRINT	RORVER,T-PRIOR,T-DESTID;	O				
26	P	64		82	HNURSTA	$PROG=DOGETLOC;				RICP	
27			82		$PRINT	CANREAS,T-PRIOR,HNURSTA;				RICP	
28		30		32	$PRINT	RADADD,T-PRIOR,T-DESTID;		V			
29			97		$RS	RADDISP;					D
30			30	35	T-DESTRR	RADIOL;		V			
31		30		35	$PRINT	RADREQ,T-PRIOR,T-DESTRR;		V			

Note 1: O = Order (Refer to Figure 7), V = Verify Order (Refer to Figure 8), RCP = Record a Complete Procedure (Refer to Figure 9), RICP = Record an Incomplete Procedure (Refer to Figure 10), and D = Display Order (Refer to Figure 11). The respective T-STATUS values are 10 for 0, 30 for V, 38 for RCP, 82 for RICP and 97 for D. Control is expressed through the use of the T-STATUS value operative at the time the DCL is entered. The value of T-STATUS is generally set by the CRT operator selecting a function such as Enter Orders.

Note 2: The Pass column is just before the first Status column. When the DCL is finished, all the listed data elements are deleted except those that have been encoded with "PASS" (indicated by placing a P in this column) in which case the data element can be used by the succeeding routine or routines.

Note 3: ORPROCED (Order Procedure) and ORCATGY (Order Category) are determined before the DCL is entered when the CRT operator selects the procedure to be ordered such as Chest X-Ray. Procedure Information is accessed from the procedure file). Note that these fields cannot be changed since the T-STATUS in each case is greater than S3, the no-change status.

the PCS Execution Debug to single step their application at the functional component level.

The Execution Debug facility dynamically displays trace and status information allowing the designer to examine the Command Stack to view execution logic flow immediately prior to the

execution of each command. If necessary, the Command Stack can be modified by keying the appropriate application logic flow. Also, fields (data names and values) available in the Symbol Table may be displayed, and fields may be added, modified, or deleted. The application flow may be resumed at the designer's option, and the updated Command Stack and Symbol Table will be used in the logic flow.

This feature is very useful in testing the application logic. In addition, it allows each functional component (screen, DCL, program) to be individually tested. Thus, all components can be designed, developed, and tested in parallel. This is one of the major benefits of PCS applicable to designers and coders, including those who are familiar or unfamiliar with the operational details of data processing.

Security

As a system becomes more end-user oriented, the importance of security increases, since the user is guided as to how all the functions work and is no longer limited by a lack of know-how. PCS sign-on security requires that all functions be accessed after the user has identified himself or herself. Each individual user has access and demographic information stored in the User Profile data base. Users are limited to functions initiated from their own specific master screens (this technique is similar to one used by many installations with IBM's Structured Programming Facility, which is menu-selection driven). The user sign-on identifier, name, and security level are carried in the Symbol Table and may be included in data base segments, screen format, print formats, etc. as an audit trail of that user's access to the system. And as an additional security measure, the user's security level is compared by the Data Manager against the segment or field level security before the data are accessed.

Error Management

Error Management is invoked automatically when an abnormal PCS return code from any component is detected by the PCS supervisor. Each error is defined in the PCS Error File and contains the user error message (end-user oriented), the system error message (more specific information in data processing terms), the transaction dump option (no dump is generally needed to solve the problem), an optional user-written error program (if special handling of this error is required), and an optional error screen (this is used frequently) to be displayed to the user. If an error occurs, the user error message is displayed to the terminal user, the system error message is logged on the CICS system printer, requested error options are taken, and, unless prevented by the options taken (and this is unlikely), the terminal user is restored to normal operation. In general, the terminal user does not have to sign on again and may continue at the last successful function.

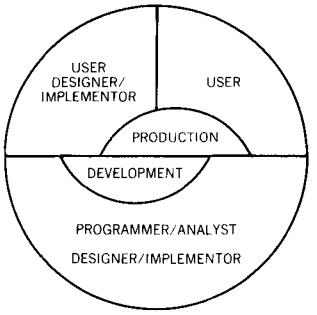
Review of the system

Now that the basic architecture has been covered along with the coding of data collection lists, screens, and print formats, it is appropriate to discuss several aspects of PCS.

user as designer/ implementor

Fundamental components of any on-line computing system include people on the one hand and the software/hardware on the other. Whereas in most systems the people quite typically are categorized as being either "users" or "programmer/analysts," PCS is different. As a productivity-oriented application development system, PCS materially benefits from incorporating the user as a designer/implementor in addition to the programmer/analyst. The pairwise bidirectional relationships of the three people components with the development system and the production system are shown in Figure 13. Note that the user as a designer/implementor overlaps both the production and the development systems. The potential complexities are evident, but in practice the mode is one of simplicity.

Figure 13 Relationships among people and system components



Parenthetically, depending on the individual situation, there may be a separate CICS component for a user training system facility, or the development system may be used for this purpose (as we generally do). In the latter case, there is a logical, conceptual difference in this interaction of the user with the development system as opposed to the user as a designer/implementor actually doing development, so Figure 13 remains valid.

PCS enables effective interaction to occur in the real world.

general applicability

Although PCS was developed and first applied in a health-care setting, there is nothing to preclude its application in nonhealth areas. At the Dallas County Hospital District (Parkland Memorial Hospital) we are using it for financial/administrative functions such as patient accounting and payroll/personnel. Because of the productivity aspect of the system, we would expect it to be widely applied outside the health industry.

architecture extensibility

PCS lends itself to extension. Modifications have been done or are going on in some customer sites in such areas as branching within a DCL (the so-called $IF facility), allowance for addition of new commands (command interface), and allowance for explicit indirect data names ("@" operand). The Edit and the Data Manager are architectural extensions.

productivity

Some 90 percent of the CRT screens and printer formats are coded by our group of health-care professionals (POIS Associates) as opposed to being done by programmer/analysts. In another environment, the personnel would be the users to be found in that setting. In our experience, for a system with a large number of CRT

screens and a large number of print formats to be coded, PCS significantly enhances productivity. Clearly this will be application dependent, but the power of PCS in facilitating screens and print format development will aid materially in almost any on-line system. The fact that the mainstream IBM operating system components (OS or DOS), such as CICS and DL/I, are used also facilitates the applicability, and since PCS handles and thus simplifies CICS and DL/I calls, it only improves the situation still further.

As to development productivity enhancement, we estimate that the effect has been improvement by a factor of four at a *minimum* when judged against other systems, even when they use CICS, DL/I, SPF, and TSO. For example, the Radiology System took only three months to develop using PCS and would have taken at least a year otherwise. This four-to-one factor does not include what we believe to be a better first production design implementation.

In terms of application maintenance, which is almost always an ongoing critical activity, our experience leads us to estimate a productivity enhancement by a factor of ten at a *minimum* as compared with other systems. On several occasions POIS Associates have made changes in a few hours that would have taken a data processing professional several weeks in a situation where there was no application development system. Productivity, of course, is the key since projects also have life cycles. With the current rapidly accelerating time frames, a facility for quick development and easy change is mandatory for increasing productivity.

Conclusion

By presenting an architectural overview of PCS, this paper has demonstrated how application development system criteria are met. Such criteria included (1) data independence, (2) logic independence, (3) ease of logic implementation (in PCS via the screen coding and the data collection list mechanism), (4) ease of CRT screen design and coding, (5) ease of printer format design and coding, (6) application and architecture extensibility (the Edit and the Data Manager are examples of architectural extensions), (7) ability of user personnel (not data processing personnel) to perform criteria 3, 4, and 5, and (8) a human-engineered, user-friendly, production system (actually demonstrated by routine, daily use). An important sidelight is that there are commercially available packages that are PCS-based and allow transfer of technology to immediately enhance productivity of the receiving groups. An example is the Patient Care System-Radiology Installed User Program developed at the Dallas County Hospital District.[26-28] The application of these concepts has been successful, and again it must be stressed that they are not tied to a given application and thus are "industry-independent."

The implementation of a large on-line information system is a huge task. PCS has greatly aided in accomplishing the task. Having sample screen and printer formats from a productive application given to us as well as the PCS application development system was extremely helpful. Although we did not install the applications without modification (occasionally significant because of functional differences or because of adding desired features), in the majority of cases, we would not have had to do so. Even though other industries will likely also come up with large applications replete with actual screens and printer formats, this is not necessary for high productivity and effectiveness.

PCS has enabled us to literally weave the POIS (Parkland On-line Information System) Hospital Information System into the fabric of our institution rather quickly. The Patient Care System (perhaps preferably designated in a general context as PCS—Productivity Creating System) is powerful and easily applied.

CITED REFERENCES

1. M. L. Zolliker, *Proceedings of a Conference on Application Development Systems, ACM SIGBDP Database* **11** (120 pages) (1980).
2. D. J. Mishelevich, "Installation of the IBM Patient Care System at the Parkland Memorial Hospital," *Electronic Computing Health Oriented Boondocks* **7**, 36–56 (1978).
3. D. J. Mishelevich and L. D. Cranfill, "On-line hospital information systems," *THISS (Texas Hospital Information Systems Society) Installation Planning Guidelines*, 43–54 (1978).
4. B. G. Hudson, D. J. Mishelevich, E. I. Mize, and J. R. Roberts, "POIS —The Parkland On-Line Information System," *Electronic Computing Health Oriented Boondocks* **8**, 65–116 (1979).
5. D. J. Mishelevich, B. G. Hudson, D. Van Slyke, L. D. Cranfill, E. I. Mize, A. L. Robinson, H. C. Brieden, J. Atkinson, J. R. Willis, and J. Robertson, "The Parkland On-Line Information System (POIS): Installation of the IBM Health Care Support/Patient Care System at the Parkland Memorial Hospital," in press, *1979 National Computer Conference Proceedings in Computers in Health Care* (1980).
6. D. J. Mishelevich, B. G. Hudson, and E. I. Mize, "Parkland system: POIS update," in press, *Electronic Computing Health Oriented Boondocks* (1980).
7. D. J. Mishelevich, B. G. Hudson, D. Van Slyke, E. I. Mize, A. L. Robinson, H. C. Brieden, J. Atkinson, and J. G. Robertson, Jr., "The POIS (Parkland On-Line Information System) implementation of the IBM Health Care Support/Patient Care System," in press, *Proceedings of the Fourth Annual Symposium on Computer Applications in Medical Care* (1980).
8. *Health Care Support/DL/I Patient Care System Availability Notice*, G320-5756-2 (Unlicensed Material) — Program No. 5796-ANY, IBM Corporation (1977); available through IBM branch offices.
9. *Patient Care at Good Samaritan Hospital, Application Brief*, GK20-1203-0 (Unlicensed Material), IBM Corporation (1980); available through IBM branch offices.
10. *Health Care Support/DL/I Patient Care System Program Description/Operations Manual*, SH20-1955-0 (Unlicensed Material) — Program No. 5796-ANY, IBM Corporation (1977); available through IBM branch offices.
11. *Health Care Support/DL/I Patient Care System Terminal Operators Guide*, SH20-1956-0 (Unlicensed Material) — Program No. 5796-ANY, IBM Corporation (1977); available through IBM branch offices.

12. *Health Care Support/DL/I Patient Care System Systems Guide*, LY20-2306-0 (Licensed Material) — Program No. 5796-ANY, Feature 8922, IBM Corporation (1977); available through IBM branch offices.
13. *Health Care Support/DL/I Patient Care System Design and Coding Guide*, LY20-2307-0 (Licensed Material) — Program No. 5796-ANY, Feature 8923, IBM Corporation (1977); available through IBM branch offices.
14. D. J. Mishelevich and D. Van Slyke, "An overview of the software architecture of the IBM Health Care Support/Patient Care System," *Proceedings of the 1980 Conference on Application Development Systems* (Special Issue of the ACM Special Interest Group on Business Data Processing *Database*) **11**, 64–75 (1980).
15. *Development Management System/Customer Information Control System/Virtual Storage: General Information Manual*, SH20-2195-2, IBM Corporation (1979); available through IBM branch offices.
16. *IMS Application Development Facility: General Information Manual*, GB21-9869-1, IBM Corporation (1978); available through IBM branch offices.
17. *IMS Application Development Facility: Program Description/Operations Manual*, SH20-1031-3, IBM Corporation (1978); available through IBM branch offices.
18. A. Hassitt and L. E. Lyon, "An APL emulator on System/370," *IBM Systems Journal* **15**, No. 4, 358–378 (1976).
19. *TSO-3270 Display Support and Structured Program Facility (SPF/TSO) Version 2.2 Program Reference Manual*, SH20-1975-1, IBM Corporation (1978); available through IBM branch offices.
20. *TSO-3270 Display Support and Structured Program Facility (SPF/TSO) Version 2.2 Program Logic Manual*, LY20-2339-1, IBM Corporation (1978); available through IBM branch offices.
21. S. N. Zilles and P. G. Habalker, "Graphical representation and analysis of information systems design," *ACM SIGBDP Database* **11**, 93–98 (1980).
22. *Patient Care System—Data Manager Availability Notice*, G320-6343-0 (Unlicensed Material) — Program No. 5796-AYQ, IBM Corporation (1980); available through IBM branch offices.
23. *Patient Care System—Data Manager Program Description/Operations Manual*, SH20-6142-0 (Unlicensed Material) — Program No. 5796-AYQ, IBM Corporation (1980); available through IBM branch offices.
24. *Patient Care System—Edit Availability Notice*, G320-6344-0 (Unlicensed Material) — Program No. 5796-AYR, IBM Corporation (1980); available through IBM branch offices.
25. *Patient Care System—Edit Program Description/Operations Manual*, SH20-6143-0 (Unlicensed Material) — Program No. 5796-AYR, IBM Corporation (1980); available through IBM branch offices.
26. *Patient Care System—Radiology Availability Notice*, G320-6092-0 (Unlicensed Material) — Program No. 5796-AWJ, IBM Corporation (1979); available through IBM branch offices.
27. *Patient Care System—Radiology Program Description/Operations Manual*, SH20-2159-0 (Unlicensed Material) — Program No. 5796-AWJ, IBM Corporation (1979); available through IBM branch offices.
28. *Patient Care System—Radiology Terminal Operators Guide*, SH20-2160-0 (Unlicensed Material) — Program No. 5796-AWJ, IBM Corporation (1979); available through IBM branch offices.

D. J. Mishelevich is located at the Department of Medical Computer Science, University of Texas Health Science Center at Dallas, 5323 Harry Hines Boulevard, Dallas, TX 75235; D. Van Slyke is with the IBM Data Processing Division, Stemmons Empire Building, 8435 North Stemmons Freeway, Dallas, TX 75247.

Reprint Order No. G321-5134.