The flexibility and usefulness of an integrated data base can be limited by constraints inherent in the data base management system. One such system, IBM's Information Management System (IMS), has been extended in terms of data independence, access control, data integrity, and user communication by a group of techniques integrated into an IMS application called the Product Development Communication and Control (PDCC) system. This paper describes the IMS extensions provided by PDCC and discusses the system's implementation.

PDCC is the prototype of the IMS Application Development Facility, an IBM Installed User Program.

Design techniques for a user controlled DB/DC system

by G. F. Heyne and C. J. Daniel

Most business organizations have a continuing and often critical need for timely, accurate information. Management and operational personnel need information that is concise and appropriate for the purpose at hand. To satisfy these requirements, a wide range of information can be stored in a single, nonredundant data base that is accessible to many users for a variety of applications.[1] Access to such a data base is controlled by a data base management system, such as IBM's Information Management System (IMS),[2] which provides a structure for organizing data to satisfy diverse requirements and has facilities for controlling on-line access to and modification of the data.

With IMS, as with most data base management systems, one or more programs must be written to enable each application to gain access to the data base or to update it. This requirement can limit the flexibility and usefulness of the data base because of the volatility and rapid change that are characteristic of today's business environment. There is a need for application systems that can be implemented in anticipation of continuously changing business needs.

The desired flexibility and responsiveness can be attained in a system that provides comprehensive data base access control and a high degree of data independence. IMS provides data independence at the segment level in the data base structure, giving programs access to data on a segment by segment (that is, record by record) basis. However, data independence is required at a finer level—at the level that recognizes the location, attributes, and content of each field within a segment.

Another requirement is for more positive control of access to the data base to prevent erroneous or unauthorized updating. Also needed is an automated facility for notifying operating departments in an organization when crucial modifications have been made to the data base. Such notification should be the responsibility of the data base system, not of each application program.

This paper describes a group of techniques developed at IBM's General Systems Division laboratory in Rochester, Minnesota, to provide data base flexibility, communication, and control in a product development environment. The techniques are integrated into an IMS data base/data communications (DB/DC) application called the Product Development Communication and Control (PDCC) system.

PDCC simplifies the interface to integrated data bases for both the application programmer and the application user. Programming and maintenance activity are reduced by removing from the application code those portions, such as input verification routines and segment layouts, that are common to several applications or that are frequently changed.

In a conventional system, the user gains access to a data base by means of transactions whose format is rigidly prescribed. The PDCC user merely specifies the keys for segments at various levels within a hierarchy. At the heart of the system is an architecture, transparent to the user, that allows applications to be implemented and maintained quickly and easily by means of rules that define application variables, and also by accommodating different types of processing.

The PDCC system was implemented for the development of computer hardware at the Rochester facility in January 1974, and the routines used in the system have been implemented since then in other DB/DC applications at Rochester and other IBM locations. In addition, PDCC routines have been incorporated into an IBM Installed User Program called the IMS Application Development Facility,[3] the objective of which is to simplify the task of developing IMS/VS application programs and to extend the capabilities of IMS/VS in terms of data independence, access control, data integrity, and communication among application system users.

The techniques used in the PDCC routines give the user customized data processing capabilities, authenticity and consistency of data, and ease of use in a display terminal environment. The system is adjustable to meet the user's changing needs, and it can be extended to new segments, data bases, and applications without program modification. Thus, it can improve programmer productivity and eliminate much maintenance activity.

Before PDCC was implemented, application programs using IMS frequently had to include substantial code to describe the layout of the data to be accessed by the programs as well as the specifications of input and output data. These requirements caused redundancies in data specifications and layout when many programs had access to common data. The programs were difficult to code, maintain, and modify because every audit change, field respecification, and format change required recompilation.

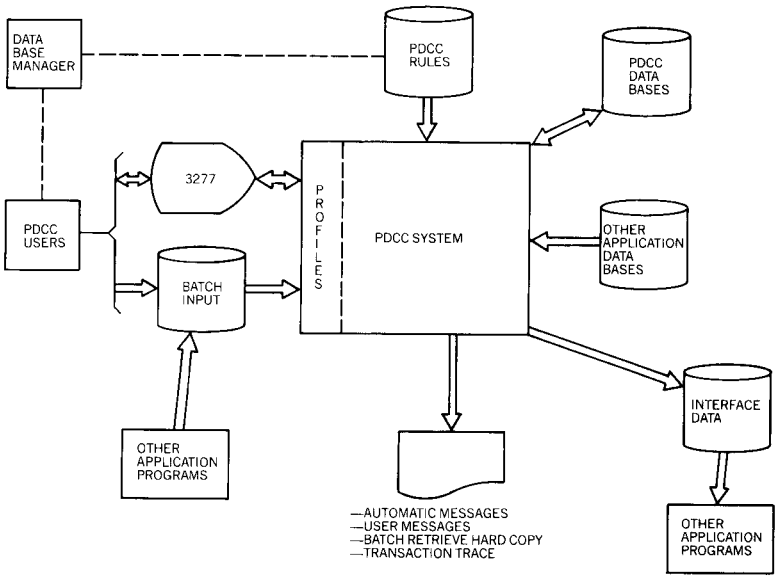
These deficiencies demanded an environment in which data and control variables would remain external to transaction processing programs. The variables had to be bound during program execution so the user would be able to modify his environment with minimal programmer interface or support. This separation of application and control variables could greatly reduce the application system maintenance effort and the expense associated with adapting complex systems to the dynamics of changing business environments. The functions provided by data dictionary/directories,[4] query processors, and report generators[5] could greatly reduce the development and maintenance effort required of such application systems.

Extensions to IMS

The implementation techniques described in this paper are extensions to the IMS DB/DC facilities and are completely external to IMS. The major areas of extended support are:

- Security and data base integrity. A user profile controls who uses the system and what transactions the user can perform without transaction passwords. The profile controls the type of access or mode of operation (retrieve, update, add, remove) that the user can perform on the transactions.
- Field level control. All actions within the system are controlled at the field level with execution-time binding of data and attributes. All data base input and output are handled through IMS by Data Language/I (DL/I)[2] even though the PDCC system has field level control. No application module contains the layout of a segment in its source code.
- Process control. Transaction process control allows execution-time binding of modules, screens, keys, input, and audits for each transaction for each user in either the batch or display-work-station environment. In addition, PDCC modules control the chain of events by passing to IMS the transactions required to complete the user's requested selection. This conversational processing provides for continuous transaction processing in a work-station environment.

Figure 1 PDCC overview



- Ease-of-use features. The user can generate transactions through menu selection techniques and look-ahead determination of segment keys. To operate the system, therefore, he does not have to know the IMS transaction codes or entire segment keys. These features help experienced and inexperienced users alike. Attribute byte setting of fields through the IMS message format service[6] provides for dynamic field-level control by the user.
- Authenticity and consistency of data. Point-of-entry auditing is provided for all input and related data by field. The entire transaction is checked for completeness and validity before any data are stored in the data bases. Duplicate data elements and constants in the system are controlled and automatically updated by the auditing facility. Data base integrity is ensured, since only logically complete and consistent data are stored.
- Message sending. This extension automatically detects critical changes in data, generates the required messages, and routes the messages to the functions and the users affected by the changes.

System overview

Figure 1 presents an overview of PDCC. The system receives its intelligence from rules (Figure 2) which govern the interactions among user terminals, batch input, PDCC data bases, other application data bases, and interface data. All access to the system is governed by user profiles which describe the functions or

Figure 2 PDCC rule types

Rule types	*Storage form*
User Profile	Data Base
Field	Load Module*
Transaction Process	Load Module*
Option Menu	Load Module*
Key Selection	Load Module*
Text Processing	Load Module*
Audit (Data Verification)	Data Base
Message Sending	Data Base
Transfer	Load Module*

*Generated by user-controlled macros

capabilities each user is allowed. Users have access to the system by means of IBM 3277 Display Stations or batch input, and all their interactions are controlled externally by rules. A rules-driven system has the following advantages:

- It provides data, data description, and processing information at execution time. Thus, data and processing information are independent of the application programs.
- It allows the user to modify system processing as his workday environment changes.
- It supports many different users, yet meets the needs of each.
- It allows the same processing modules to be used for both batch and on-line versions of the system.

The rules are controlled by a data base manager (DBM), who is responsible for all changes in rules that control transaction authorization, field definitions, auditing, messages, and transaction processing. With this authority, the DBM can explore the full processing potential of the system. Each user has some subset of the DBM's authority, tailored to the user's specific needs and authority.

The rules also govern input to the batch system. All non-PDCC application programs interface with PDCC through batch input. PDCC generates one or more interface data files for processing by these other application programs. This kind of interface helps to guarantee the reliability, accuracy, and integrity of the system within itself.

The major routines in the PDCC system are:

- Sign-on, the user's entry point, which validates his authority to use the system.
- Option menus, modules that provide individual users with customized option lists for transaction selection.

- Key selection, a module that guides the user to his target data by making it unnecessary for him to enter the entire IMS key.
- Transaction driver, a generalized module that handles standard or special processing of all transactions.
- Segment handlers, a group of modules that perform all data base input and output. A single module handles input and output for each segment of the data base. Segment handlers are the only modules in the system that contain segment search arguments.
- Screen formatter and deformatter, two modules that control the formatting and interpreting of data on display screens.
- Auditor, a module that verifies and validates data and initiates automatic message sending.
- Text processing, a module that allows the user to process bulk textual data in a block-edit mode.
- Data transfer, a module that accepts user definitions to control the transfer of data to or from the system, by field, to agree with internal and external data maps.
- Sign-off, a module that performs any housekeeping and accounting required at the termination of a session.
- Special processors, modules that perform special processing for segment manipulation or that process multiple segment types. These modules contain the application's unique logic, hence they contain the only application dependent code in the PDCC system.

The rules provide the logic to direct the processing of a transaction using this modular structure. Thus the modules, by interpreting the rules, provide the means of implementing a viable method of execution-time data binding in a user-controlled system.

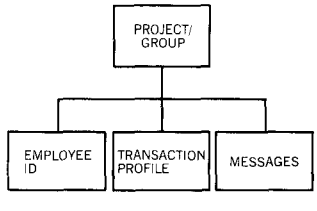
Implementation techniques

Some of the rules and techniques used in implementing a user-controlled DB/DC system are described below. For details of the techniques and on-line system flow, see References 7 and 8.

rules

The key to the entire system is a set of capabilities that are bound at execution time and tied together by rules. Variable data are external to the modules; thus the modules process the data with respect to rules. A summary of some of the system's rules is given in the following paragraphs.

Figure 3 Profile data base structure



User authorization rules, implemented by a user profile data base (see Figure 3), verify that a user is authorized to sign on to the system. If the user's sign-on information is not in the profile, he cannot sign on or perform any transactions. Once a user has passed the sign-on validation, the profile is used to dynamically

build his primary option menu, which shows only the major functions that the user is authorized to perform.

The profile also provides security at the transaction level by controlling the mode of operation a user can perform on a transaction. The modes of operation are so ordered that anyone who has the authority to update a segment can also retrieve it, but cannot add or remove it. For a survey of protection techniques that can be used to provide controlled sharing of information and user authentication, see Reference 9.

Message segments in the profile data base contain all messages sent by individual users, or sent automatically by the system, to particular projects or groups (that is, to any user-defined organizational structure that is independent of the system). If a modification to a specific field within a segment is critical to some other project or group, a message is sent automatically to that project or group. The user who makes the modification need not be aware that the message is being sent.

A field rule, or segment map, provides a complete description of each field in a segment. There is one field rule for each segment in the data base. Data independence, security, and processing are achieved by the field rule, as control is at the field level instead of the segment level.

For each field within a segment type, the field rule contains the field name, offsets, attributes, and a communication area. Each field in the system is identified by a unique name.

The communication area within each field entry may contain several types of data. For example, when a field is audited, the area may contain the number of an error message if the field is in error, or it may contain the message code for informative messages if message sending is required for the field and all audits are successfully passed.

A transaction processing rule provides processing information for a user transaction for both the terminal and batch drivers. It describes to the driver the input to be processed and contains control information indicating which resources are to be used in that process. It provides transaction, screen, segment, and field control, and it establishes the mode of operation by field. This rule is uniquely identified by project or group and transaction, allowing each project or group to have different processing capabilities for the same transaction. Appended to the transaction processing rule are audit field entries that contain unique audit control for the transaction, providing for individualized auditing of the data in a given field.

The rule contains a field entry for each field required by the transaction. The name used in the field and audit field entries is identical to the name in the field rule for the segment. The mode of operation specification at the field level in the transaction processing rule allows different users to have different capabilities by field. For a further discussion of privacy protection as related to the control of a user's access to data from the data base, see Reference 10.

A transfer rule defines the mapping of data into or out of data base segments and allows data to be transferred directly from buffer to buffer. Thus data can be transferred from or to any segment in the data base, and the system can generate application interfaces in the format defined in the transfer rule. This facility is especially helpful in reading records or segments from another system and then storing the data in a revised format.

A transfer control entry is generated for each field to be processed. These entries can combine data from several records or segments into one new record or segment. Information about the field is found through the field name, which directs the routine to the associated field rule. Changes in data values or characteristics are also possible when processing in this mode.

segment handlers

In IMS applications, communication between modules and data bases is at the segment level. Each type of segment has unique data keys, the segment search arguments, which control access to that segment type. In PDCC, however, the segment handler takes care of all input and output activity (insert, delete, update, retrieve) for a particular segment.

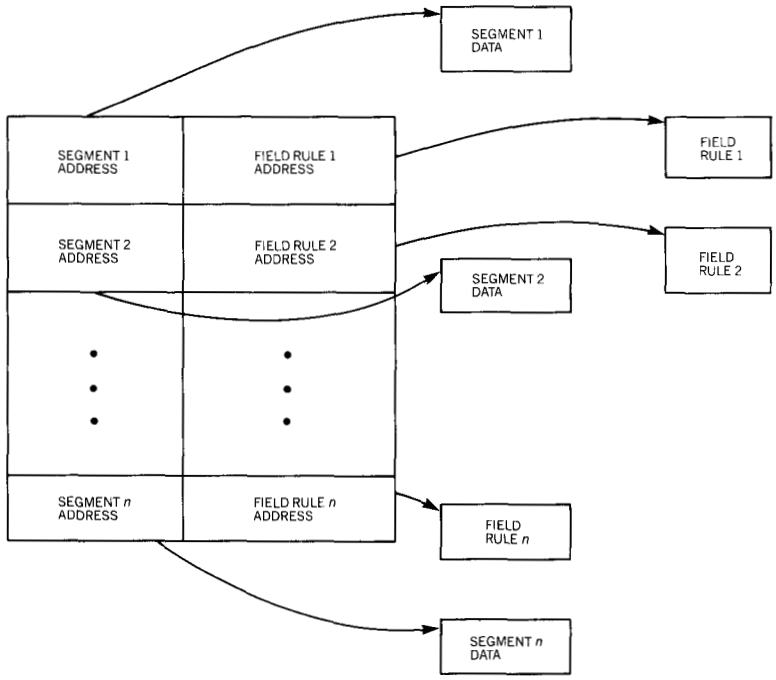
Thus the module requesting a particular segment does not need specific information about segment input and output activity (such as segment name, key field name, and segment search arguments) since the segment handler contains that information.

Usually, many modules in a data base system work with the same segment type, and each application module would build in the specific data requirements for the segment. Thus a change in the data requirements for a specific segment would require changes in every application module that used the changed segment. With the segment handler concept, modifications can be made once, and only in one module. Further, the concept reduces the education requirement for programmers working on the application modules, since all the data base code is in the segment handler. This concept significantly reduces the amount of maintenance required.

the auditor

All data stored by the system in the data bases is verified for authenticity and consistency by the auditor, if required by the

Figure 4 Segment table format

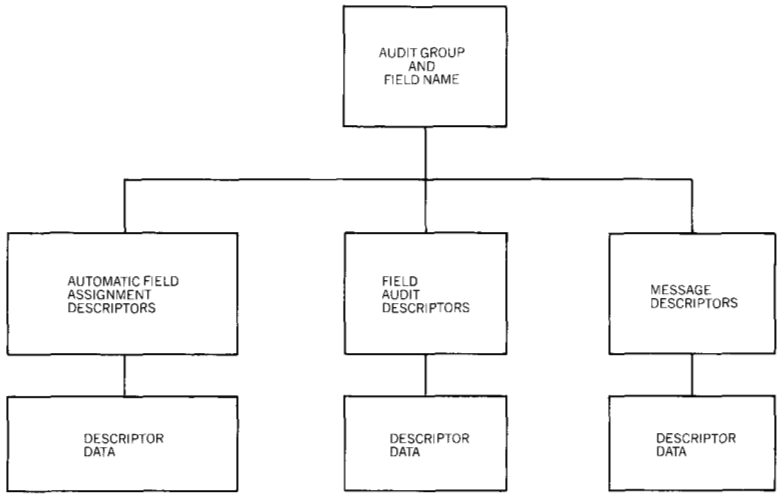


user. No segment is stored in a data base unless all auditing requirements have been met. In addition, the auditor indicates diagnostic message numbers by field for each error or message.

The auditor has a built-in logical capability which is driven by rules called descriptors, which are stored in a data base. This data base can be updated dynamically by the DBM through the text segment processor. Thus, an audit can be changed immediately by the DBM without reassembling any modules or rules. The auditor retrieves, from the audit data base, the descriptors needed to audit the fields. The auditor does not update either the audit data base or any other application data bases in the system.

All segment data to be audited are passed to the auditor through a segment table (Figure 4). This table contains the addresses of the segments to be audited or referenced and their corresponding field rule addresses. The table is used also to pass the same information to the transfer, message generating, and message sending modules. The segment table is the communication vehicle by which the auditor finds the data involved in the audit and by which the auditor informs the calling program of fields that are in error.

Figure 5 Audit data base structure



The logic of the auditor is controlled by descriptors, or logical instructions, which control the sequence of operations in the auditor. Examples of descriptor functions are *field value must be in a range of values, assign a value to a field from a related field,* and *assign a current date.*

Each unique descriptor has a corresponding subroutine in the auditor. Subroutines are easily added. The only change in the auditor when adding a new type of descriptor is the descriptor's program code and the extension of a table to include its identifier and the address of its subroutine.

The audit data base consists of three dependent segment paths (Figure 5). A descriptor is not restricted to one segment path or another; all paths have the full processing capabilities of the auditor. Each path has the following functions:

- Automatic field assignment. This function is performed first and is based on the transaction performed. Values are assigned to fields automatically if so specified in the transaction process rule. Field assignments also can be made or not made, depending on other data or on specific data entered into the segment.
- Auditing. This path makes it possible to check the validity of the data associated with the transaction. The data values checked are those fields that the user is trying to update or those values stored by automatic field assignment.
- Message sending. This path determines whether messages need be sent to the various users of the system. This part of the auditor is interpreted only when a field has been changed,

when it has been flagged as a candidate for message sending, and when all audits performed in the second path of the auditor were successful.

If an error is determined by field in either of the first two functions above, the code for that error message is stored in the communication area for the field in error in the field rule, and the message sending path of the auditor is not activated. If no errors are determined, the message sending path is activated and the message codes, if any, are stored by field in the communication area of the field rule.

message sending

PDCC provides for the sending of two forms of messages, automatic and user-to-user. Automatic messages are sent when the system detects critical changes in data. The messages are routed automatically to individuals and applications that might be affected by the changes. User-to-user messages are those sent by one user of the system to another.

Three general categories of messages can be sent automatically: deleted segment messages, unconditional update messages, and conditional update messages. Rules control the determination that a message must be sent, which message is to be sent, and to whom. Deleted segment messages and unconditional update messages are controlled by transaction only; that is, they are sent only when a transaction is completed successfully. The auditor is not used to determine whether messages are to be sent, since the auditor works on fields within a segment and not on individual transactions.

Conditional update messages are controlled both by transaction and by the auditor. The auditor is used first to verify all the changes and additions to a segment. If all audits are successful, a message-sending check is performed by interpreting the message sending path of the audit data base for those fields that were changed and that are marked as candidates for message sending.

Like the auditor, the message sending module has a built-in logical capability which is driven by descriptors stored in the message rule data base. The application identification code and the message code from the communication area in the field rule are the means by which the message sending module determines to whom and where the message is to be sent. Examples of descriptor functions are SEND to one user a series of messages, and ROUTE one message to a number of users.

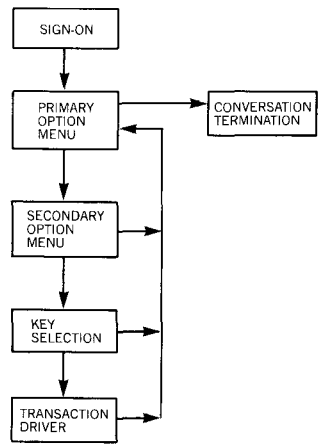
The DBM can dynamically update the message rule data base on line. Again, no modules need be modified for data or descriptor changes, so maintenance is simplified. As in the auditor, new descriptors are added as subroutines without affecting existing code.

Users can view the messages on line. In addition, all automatic and user messages accumulated throughout the day from the terminal system, as well as automatic messages generated during batch processing, are printed and sent to the user daily. Experience at Rochester indicates that the user relies heavily on the printed messages and does not use the on-line capability to a great extent.

conversational transaction control

A conversation, as defined by IMS, allows a user to interact with application programs in a message processing region and, between interactions, retain information in a scratch pad area. To start the transaction, the user must enter the transaction code, password, and data, and then he interacts with the transaction until his processing is complete. At that time, both the transaction and the conversation, which are of short duration, are terminated. To process another transaction, the user must enter that transaction code, password, and data. Therefore, to process in this mode, the user needs to know all the transaction codes, passwords, and data required to support the transactions.

Figure 6 Multiple IMS transactions within an IMS conversation



The method implemented in PDCC expands the concept of conversational processing as supported by IMS. It maintains transaction security and eliminates the requirement that the user know transaction codes and passwords. PDCC allows a continuous IMS conversation while the user is interacting with multiple IMS transactions.

The next IMS transaction to be executed is determined by the use of the profile data base, option menus, rules, and user selections. The programs take this information and build the IMS transaction code dynamically, and then pass the new code to IMS. This technique provides a work-station environment in which the user is not required to know IMS transaction codes and passwords. Figure 6 illustrates different IMS transactions involved in an IMS conversation.

sign-on security

Security is important in PDCC, as in most on-line systems. The user should be identified when he first signs on, and he should be limited to a specific set of transactions. In PDCC, these requirements are met with sign-on input consisting of employee serial number, project code, group code, accounting information, and a lockword. The profile data base, as described previously, is used to validate the employee sign-on information and identify the transactions the user may perform on the system. The non-display feature of the IBM 3277 Display Station is used for the lockword, which is stored in a scrambled data set. In all, one data base and two files are checked in the process of determining the user's validity.

dynamic option menu

Normally in a display terminal environment, an option menu has a fixed set of options from which a user can select. This fixed option menu is adequate in many terminal applications if all users are allowed to perform all options. In most DB/DC environments, however, the user is not allowed to perform all transactions. Yet from a fixed option menu he might select a transaction not in his profile, thereby generating an error message and causing frustration for the user. PDCC uses a more positive approach: individual option menus are generated through rules and show the user only the options he can perform. Thus he does not have to view options he does not understand or have a need to know.

There are several advantages in using this dynamic option-menu concept in an IMS environment. Option menus allow the user to select the next IMS transaction (through user terminology) without having to know the IMS transaction code. The option menu module transforms the selected option to an IMS transaction code through the use of rules, then passes the code to IMS. Adding or deleting IMS transactions does not affect the module generating the dynamic menus or the screen definition used in displaying the options. Only the rules or the user's profile need be changed, again easing the job of maintenance.

key selection

In most data base systems there are thousands of segments, each with a unique key. It is highly improbable that a terminal user could remember the keys of all the segments to which access is required without either writing them down or consulting a list. To avoid this problem, PDCC includes a key selection module that makes it easy for the user to supply the key information needed for access to the segments required by a transaction. The module:

- Allows the user to dynamically build the primary keys needed for access to any segment in any of an application's data bases.
- Requires the user to know only the root level of any segment in any data base.
- Provides a look-ahead feature so the user can select the dependent segments without prior knowledge of the keys.
- Derives its processing direction from a rule, without regard to segment type.
- Provides a method of key selection that is easy for both experienced and inexperienced users.

Each transaction that requires the key selection process has a unique rule, the key selection rule, which is processed by the key selection module to prompt the user for generation of the concatenated key required by the transaction. The rule specifies the screen definition to be used, and it specifies all the segment and field information required to reach the target segment.

The key selection module requires no segment search arguments or other segment data, as all segment input and output are accomplished through segment handlers. Thus the key selection module is independent of any segment in the data bases, and new segments can be added or deleted without modifying the module. Only a rule is generated for a transaction, and a screen display is defined describing the keys needed for access to the required segment.

The initial key selection display gives the user the opportunity to enter the concatenated key from the root segment to the target segment. For our application, it was mandatory for the user to enter the root segment key; however, this restriction is rule controlled. All information on this and subsequent displays is in the user's terminology. Therefore data base structure and terminology are removed from the user.

If the user does not know or want to enter the entire key on the initial display, the key selection module determines the key automatically if only one segment exists, or it gives the user a choice of segments. When a choice is given, each segment's primary key and pertinent segment data familiar to the user are displayed. Each key is preceded by a selection number, and the user need enter only the number, rather than the entire key of the selected segment.

Although the segment selection process was developed primarily for the inexperienced user, it has been found that experienced users, too, employ this technique rather than enter the entire concatenated key on the initial display. The technique allows the user to view all segment keys and pertinent data under some parent segment, which sometimes is all the user wants to know.

transaction driver

Transaction processing in PDCC generally is done in either of two modes:

- The standard segment mode, in which the transaction works with a single segment to delete, add, modify, or retrieve segments. It may have access to other segments, but it cannot modify them. There is no unique application code for this type of processing.
- The special processing transaction mode, in which the transaction may involve several segments to complete its processing, and the mode of operation upon the segment can be delete, add, modify, or retrieve. This type of processing provides computational capabilities, reports, and unique application-dependent logic that are outside the scope of standard segment processing.

In the processing of single segments, the order and type of processing are identical for each segment in the data base. For

example, to update a segment, the segment must be retrieved, the input data merged with the existing data, the input audited for validity, error messages generated if there are input errors, the data base updated if there are no input errors, and automatic messages sent to specified users if required by the transaction. This processing sequence is the same in both the on-line and batch environments. Processing in the special transaction mode is similar to processing in the standard mode except that special logic, unique to the transaction, is required.

Major difficulties are encountered in this type of processing. Unique modules for processing each segment require much program duplication. Segment fields, audits, and messages are coded within the module, thus requiring a change by the maintenance group every time one of the fixed data requirements is changed. In an ever-changing environment, this type of system is expensive to develop and maintain and soon becomes obsolete.

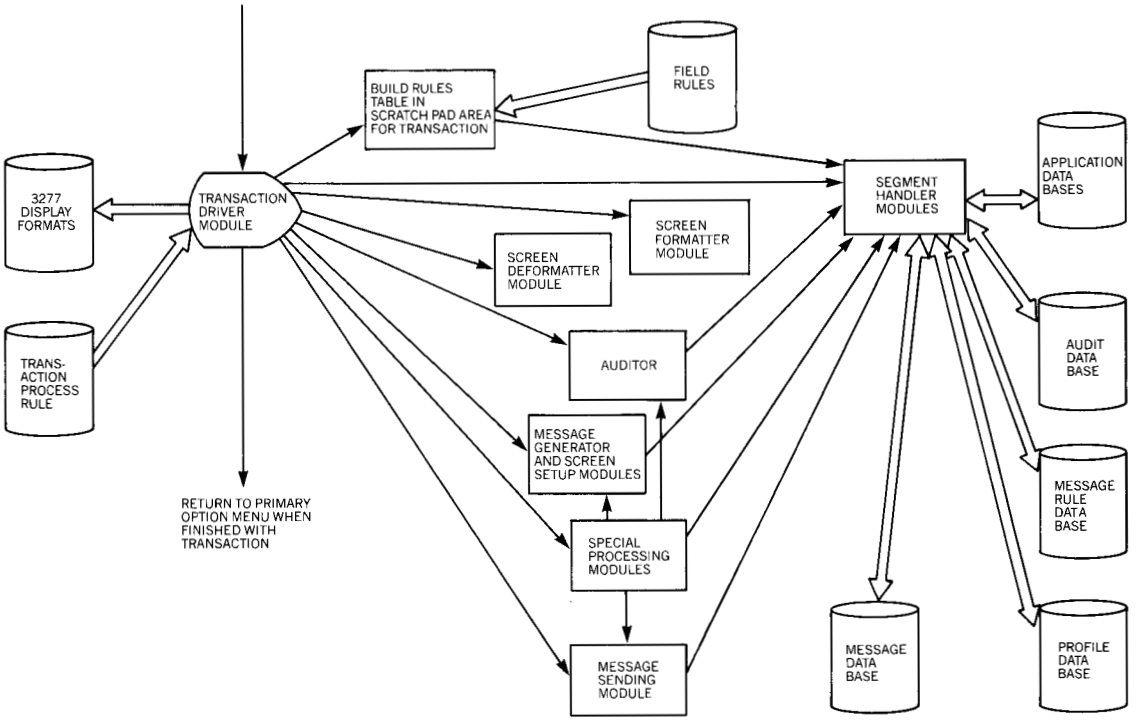
In developing PDCC, it became apparent that these limitations required an innovative approach. Therefore a transaction driver was implemented that:

- Performs the processing on all the segments in all the data bases used in the application.
- Loads the transaction process rule and field rules required for the transaction.
- Invokes the auditor for added or updated segments.
- Invokes the message generator and shows all error messages for the segments.
- Performs preprocessing for all special processing programs.
- Displays all error messages generated by the special processing programs.
- Invokes message sending for the standard segment mode.

The functions listed above are available to both batch and on-line systems. In addition, the transaction driver in an on-line system invokes a screen formatter and a screen deformatter, and it handles all interactions with IMS, including use of the scratch pad area and transmission of data from and to the terminal.

These capabilities were implemented by making the transaction driver rules-driven. The transaction process rule defines the input and data required for a transaction, and the transaction driver derives its processing from the rule. Segment handlers are used for all data base access and updating. The rules and segment data needed by the transaction are loaded by the transaction driver or by supporting modules. The screen formatter and deformatter use the rules to format the screen display and interpret any entries made by the user. During a segment update or

Figure 7 Transaction driver flow



addition, the transaction driver determines whether the audits were successful. If so, the data base is updated by invoking the proper segment handler. If errors were detected, the data base is not updated and the user is given an opportunity to make corrections. In all cases, the segment is not updated until all fields have been verified.

Specialized code in the transaction driver is eliminated for an individual transaction, reducing maintenance on the module and allowing new transactions to be added to the system without modification to modules. Only additional rules, audits, messages, and screen definitions need be generated. The simplification of these processes can be observed in the flow of the transaction driver as illustrated in Figure 7.

terminal screen formatting

User requirements determine to a great extent the types of routine that display segment data. First, the user must be able to retrieve, update, add, or delete data in any segment to which his profile allows him access. Second, field level security and control are needed, since some users can update a field while others cannot, and some fields might be confidential. Finally, the method implemented must be easy to use.

To meet these requirements, the screen formatter and screen deformatter modules were written for PDCC. The field rules and transaction process rule were used in conjunction with the field attribute feature of the IBM 3277 Display Station and the IMS message format service. The latter includes an option that allows an application program to dynamically modify or replace the attributes of a field described for the screen.

For each segment in the data base, there exists only one field rule; however, there may be different transaction process rules for different projects or groups. These rules describe the contents of the segment in detail. The screen formatter, which generates the attribute bytes and data for the screen, and the screen deformatter, which interprets only the modified data from the display screen, are driven by the needed field rules and the transaction process rule for the project or group.

The use of these rules provides field-level data independence and security by user type. For example, all fields are protected when the mode of operation on a segment is RETRIEVE. For updating, the user follows the cursor on the display screen, which moves only to those fields that can be updated. Fields that cannot be updated are protected. This type of field level control by user would be extremely cumbersome to achieve if coded physically in the module.

new transaction entry

Upon completion of either the standard segment mode or the special processing transaction mode, the experienced user needs some method of initiating another transaction without going through the process of selecting the option, transaction identifier, and primary keys. By eliminating this process, the user can skip a number of screen interactions, and the total response time can be improved.

This goal was accomplished in PDCC by placing the option, mode, transaction identifier, and primary key in the control line of each display, allowing the user to enter all the information he knows about the transaction. For example, if a user enters the mode and transaction identifier on the option menu display, the next display—assuming that the user passes security—will be the key selection display. The secondary option menu display will be skipped if the IMS transaction determines that the user has already entered the required information. If the transaction identifier is incorrect, the secondary option menu module will show all available transaction identifiers that the user is allowed to select.

On the standard or special processing transaction input display, the user can change the mode, transaction identifier, and primary key as on the option menu. The user can then change one, two,

or all three control fields to initiate the next transaction. If the data entered are valid, the user can view the next transaction driver display without the three intervening displays for the primary and secondary menus and for key selection.

This technique allows experienced users to move along at their own pace and skip some screen interactions, while allowing the inexperienced user to be prompted step by step through a series of displays.

Summary

Implementation of PDCC at the Rochester laboratory has led to an increase in programmer productivity because of a reduction in application coding, easier program maintenance, and ease of change. User response has been good. The user's requirements have been met in terms of customization of displays and capabilities, authenticity and consistency of data, and ease of use in a display-work-station environment. The system is extendable to new segments, data bases, and applications. Instead of trying to anticipate future requirements, PDCC defines variable data in rules which are external to the processing modules. The system's application independent modules can interpret the various rules to provide dynamic and flexible transaction processing. Rules stored in data bases can be updated dynamically by an authorized user and be effective immediately.

Compared with an estimate based on traditional programming techniques, the first project to use the PDCC techniques in the Rochester laboratory required about 40 percent less time for designing and programming. Moreover, maintenance requirements were an estimated 30 to 50 percent less.

Aside from increased programmer productivity, the PDCC techniques have led to application independence, and they have enhanced the authenticity and consistency of data. They have proved viable in an environment characterized by a need for communication and control in terms of both data and the user organization.

ACKNOWLEDGMENTS

The authors wish to acknowledge Raymond G. Beard, Jerome E. Bonkoski, John A. Noid, and Dave Olson, IBM General Systems Division, Rochester, Minnesota, for their design and programming effort in developing the techniques incoporated in PDCC. Edward W. Hallbeck, John W. Justice, and Richard C. Peters, the user team, helped define the product functions and system features that led to the design of PDCC.

CITED REFERENCES

1. R. Ashany and M. Adamowicz, "Data base systems," *IBM Systems Journal* **15,** No. 3, 253–263 (1976).
2. W. C. McGee, "The Information Management System IMS/VS," *IBM Systems Journal* **16,** No. 2, 84–168 (1977).
3. *IMS Application Development Facility—General Information Manual,* order number GB21-9869, IBM Corporation, Productivity Application Development Department, 1501 California Avenue, Palo Alto, California 94304.
4. P. P. Uhrowczik, "Data Dictionary/Directories," *IBM Systems Journal* **12,** No. 4, 332–350 (1973).
5. *Generalized Information System GIS/360—Application Description Manual,* order number GH20-0892, IBM Corporation, Data Processing Division, White Plains, New York 10604.
6. *IMS/360 Version 2—Utilities Reference Manual,* order number SH20-0915, IBM Corporation, Data Processing Division, White Plains, New York 10604.
7. C. J. Daniel and G. F. Heyne, *Design Techniques for a User Controlled DB/DC System,* Technical Report TR07.575, (1957), IBM Corporation, Rochester, Minnesota 55901.
8. G. F. Heyne and J. E. Bonkoski, *Design Considerations for Display Terminals in a User Controlled System,* Technical Report TR07.603, (1976), IBM Corporation, Rochester, Minnesota 55901
9. J. H. Saltzer, "Protection and the control of information sharing in Multics," *Communications of the ACM* **17,** No. 7, 388–402 (1974).
10. N. Minsky, "Intentional resolution of privacy protection in data base systems," *Communications of the ACM* **19,** No. 3, 148–159 (1976).

Reprint Order No. G321-5057.