AN EXTENDED OPERATING ENVIRONMENT FOR THE SUPPORT OF APPLICATION PROGRAMS

RICHARD A. BERGQUIST AND STEVEN M. COOPER
AMERICAN MANAGEMENT SYSTEMS, INC.

The Brownboard Order and Rollstock Distribution System (BOARDS) supports order processing, invoicing, inventory control, and planning for the Shipping Container and Containerboard Marketing Division (SCD) of the Weyerhaeuser Company. The function of BOARDS is described more fully in the paper, "Decision Support System for the Management of Containerboard Logistics" by P. DiGiammarino and R. Schwartz.[1]

American Management Systems, a management consulting and system development firm, began work on BOARDS with Weyerhaeuser in September, 1976. Several subsystems are in production use; the system will be fully operational in early 1979.

BOARDS is written in COBOL, SPL, and FORTRAN, in order to utilize the advantages of each language. COBOL was chosen for the majority of the system because of its widespread use and report generating abilities. SPL was chosen for its efficiency and ability to interface with all aspects of the Operating System. FORTRAN was chosen for number processing routines such as those that employ linear programming techniques.

In order to provide an enhanced environment for the programmer without the need of becoming familiar with all aspects of the system, sets of common routines were developed. These routines also insure consistency and compatiability across the system and allow for easy maintenance of these technical functions.

Sets of common routines have been provided that extend the services provided by MPE, KSAM, IMAGE, and DEL. These sets of routines are summarized in Figure I and are described in more detail in the remainder of this paper.

BATCH JOB SUBMISSION COMMON ROUTINES

MPE provides a convenient method of introducing batch jobs through the use of the STREAM command. However, one problem associated with STREAMing jobs is security. To be STREAMed, the User ID must be provided along with all appropriate passwords. The problem here is that either the user must

FIGURE I

COMMON ROUTINES	FUNCTION
BATCH JOB SUBMISSION	Handles User ID's and passwords. Parameter substitution. Provides interactive front-end for batch job submission.
FORMS	Replacement for DEL, enhancements include: - Protected variable data. - Dynamic screens with multiple forms and repetitions. - Mixed line and page mode transfer.
VERSION NUMBERS	Solves concurrent update problem without locking data base for entire transaction. Identifies transactions that failed during data base modification either because of program failure or system failture. Extends data base lock across process boundries.
DATA BASE ACCESS	Performs IMAGE calls. Performs 'before' logging to protect against program aborts. Performs 'after' logging to protect against file system errors. Prevents 'Deadly Embraces'.

be prompted for the password, the password must be hardcoded into a program, or passwords must be kept within the jobstreams on disc. Prompting the user is unacceptable for human engineering reasons, hardcoding does not allow for changing of User Id's or passwords, and leaving passwords on disc leaves them accessible to anyone who can STREAM the file. Finding none of these alternatives acceptable, a common routine to stream batch programs was designed. The common routine is passed the name of a template file, job parameters, and a parameter string.

The common routine creates a jobstream from a JOB statement which it creates and the statements contained in the template file. The JOB statement is based on the user running the program and includes all needed passwords. The User ID and passwords are kept in a table in a separate, hidden SL procedure. This allows passwords to be changed by simply modifying an SL segment, while allowing only legitimate procedures to access the passwords. As a further security measure, the User ID which is used does not have interactive capability -- i.e., it is restricted to batch access.

The remainder of the jobstream comes from the template file. The streaming procedure substitutes the run parameters which where passed to it into the jobstream wherever an ampersand occurs as the first character. This allows an interactive program to prompt the user for parameters and to then substitute the parameters into the batch system.

HP2645A COMMON ROUTINES

The BOARDS terminal network consists of HP2645A terminals with 12K memory connected at 2400 baud via multiplexers (one time-division mux and one statistical mux), as well as various other ASCII devices that dial into standard BELL-103 type modems. Early in our design phase, we determined the needs of programs that would use the HP2645A's in Forms Mode, and evaluated the Hewlett Packard product, DEL/3000[2], in light of these requirements. This analysis resulted in two observations: 1) We needed certain features not provided by DEL/3000; and 2) DEL/3000 had features we did not need or want to pay for in terms of processing time. We therefore developed a set of COBOL-callable, SPL common routines, that are used as a total replacement for DEL/3000. Some of the differences between DEL/3000 and the BOARDS routines are described below.

A DEL form is made up of protected fields, whose contents are fully defined when the form is created, and unprotected fields, whose contents are alterable by the user and are read back to the computer whenever the form is read. In addition to these field types, we have defined variable-data protected fields. These fields may be filled by the program at run-time, but are protected on the screen so that they are unalterable and are not transmitted when the form is read. Three examples where these fields proved useful are:

- The user specifies that (s)he would like to enter an order for customer code 'ABC'. The name and address for this customer are then retrieved from the data base and displayed on the screen in variable-data, protected fields, for visual-verification by and information for the terminal user.

- Whenever an unprotected field is found to be in error, it is set blinking and a two-character error code is written to a variable-data, protected field at the beginning of the line that contains the field in error.

- The second line of every screen in the system is called the Message Line. It is an 80-byte variable-data, protected field in which the application program can inform the user as to the status of the processing and inform the user as to what is next expected of him/her.

DEL/3000 allows only one form to be on the screen at one time. Forms may be chained together, but this means that after one form has been displayed and processed, the screen will be cleared and the next form will be displayed. The BOARDS form routines allow multiple forms and multiple repetitions of forms to be displayed on the screen (and in terminal memory) at one time. This feature has given us the capability of having dynamically sized screens whose length is determined at run-time. It has also given us the ability to produce composite screens where each piece is selected at run-time from the form file. Similarly, the same form may be used as part of more than one screen.

The BOARDS common routines allow for the programatic control of the memory-lock feature of the HP2645A. All of the form routines function correctly whether or not memory-lock is set.

All screens in BOARDS have a similar first line containing, among other things, a one-byte unprotected field, called the Control Field. Various values may be entered by the user into this field in order to alter the normal flow of a program. Several of these values are processed by the common routines and the application program is not aware that this processing occurred. For instance, 'E' means exit, 'R' means redisplay the data in the unprotected fields. In several of these cases, once the Control Field is read, there is no need to read the rest of the screen. So when the ENTER key is depressed, the computer addresses the cursor to the Control Field and triggers a field read. If a value has been entered, it is processed and the rest of the screen is not read. If a value is not found, the terminal is programatically strapped-for-page, and a page read is triggered. This allows us to minimize terminal I/O's which can become important since some of our screens contain several thousand bytes of unprotected data.

Another Control Field value that is automatically handled by the common routines is 'P'. Whenever this value is entered, a hardcopy printer listing of the screen is produced. If the terminal is equipped with an HP2631/240 character printer, the terminal is instructed to copy the contents of terminal memory to the printer. For terminals without attached printers, the contents of the screen (unprotected, protected, and variable-data, protected fields) are written line-by-line into a printer spool file.

Many program functions must operate in inquiry mode on both HP2645A and other, non-screen mode, terminals. We therefore developed common routines to allow the same application program to operate on a variety of different terminal types for display only functions. When a program is run from a non-HP2645A terminal, the escape-sequences are stripped out and the unprotected, protected, and variable-data fields are combined on a line-by-line basis and printed, in character-mode, to the terminal.

Finally, I/O error recovery is attempted in the common routines. This is especially useful in dealing with the multiplexers, since saturation conditions can arise that would result in lost data. When an input error is detected, the read is re-initiated repetitively until it is either successful or the maximum retry value is reached. If none of the retries succeed, the screen is blanked and the form is re-written to the terminal under the assumption that either an output error occurred while writing the screen or the terminal user inadvertently damaged or erased the screen.

VERSION NUMBER ROUTINES

The version number routines serve three purposes. First, they protect against concurrent update of a data base without locking the entire data base throughout the transaction. Second, they allow detection of incomplete transactions caused by system or program failures. Third, they extend a data base lock across process boundries.

Within a data base, a logical data path is formed by data which are logically grouped together but which may physically cross data set boundries. An example of this is a purchase order contained in a header record and line items which reside in a detail data set. The logical path in this case would consist of the header record and all of the line items.

If two users were to update the same path concurrently, some changes might be lost. Figure II shows how two users concurrently changing a path might 'lose' a change.

FIGURE II

Time	User A	User B
Ø	User A gets purchase order XYZ in order to update it	
1		User B also gets purchase order XYZ in order to update it.
2	User A adds two bolts to the purchase order and updates the data base.	
3		User B adds three nails to the purchase order and updates the data base. However, User B is unaware that the purchase order has changed since it was originally obtained.
4	At this point the purchase order does not reflect the changes that User A applied. That change was lost when User B applied his change using the data base record retrieved before User A's change was made.	

To protect against concurrently updating the same path, one must lock the data base when one begins the transaction and unlock it when the transaction is completed. If there are many users trying to update records in the same data base, this method is unacceptable; one user might not finish a transaction in a timely manner and the data base will be tied up for an extended period of time.

The version number routines use a data item (version number) for each logical data path. When a transaction begins, the version number is saved. When the user has completed all of his or her modifications and is ready to update the data base, the data base is locked, the version number is re-read and compared with the original version number. If the version number has not changed, then the path has not been modified and the program may continue with the user's modifications. At the end of the transaction, the version number is incremented. If the version number that was re-read has changed, the user must begin the transaction again because the records within the path have changed since the transaction began. The use of version numbers allows the data base to be locked only when modifications are in progress while still protecting against concurrent updates.

To detect if a transaction was only partially completed, an entry is made in a table (called the Integrity Table) whenever a modification begins and removed when the transaction completes. The Version Number routines add the table entry when the version number is re-read and found to be unchanged. The entry is deleted at the end of the transaction when the version number is incremented. By examining the Integrity Table while the system is quiesced, transactions which were only partially completed can be identified.

Under IMAGE, a data base is locked by a process. If the process that has a data base locked is aborted, the data base is unlocked. To extend a path lock across process boundries (used for backing out of incomplete transactions as described in the next section) the Version Number routines examine the Integrity Table whenever a version number is read. If an entry is found in the table, then the path is currently locked or a transaction was only partially completed. In either case, the path is inaccessible and modifications are not allowed.

DATA BASE ACCESS COMMON ROUTINES

From a programmer's point of view, the data base routines are functional equivalents to their IMAGE counterparts. While performing as their IMAGE counterparts, the access routines are also performing 'before' and 'after' logging of all transactions as well as protecting against deadlocks when the data bases are locked.

Protection against deadlocks was accomplished by requiring that all data bases be locked at the same time (i.e., one call). The Data Base Access routine then locks the data bases in lexicographical order.

Two types of logging take place; 'before' and 'after'. 'Before' logging consists of saving a copy of all records associated with the transaction before any modifications are done. The 'before' log is essentially a snapshot of the data base before the transaction took place. 'After' logging consists of saving a copy of all records associated with the transaction after modifications are done. The 'after' log is essentially a snapshot of the data base after the transaction takes place.

'BEFORE LOGGING'

The 'before' log is used to restore the data base to its original state should the transaction process complete abnormally. 'Before' logging is done to Extra Data Segments to improve performance.

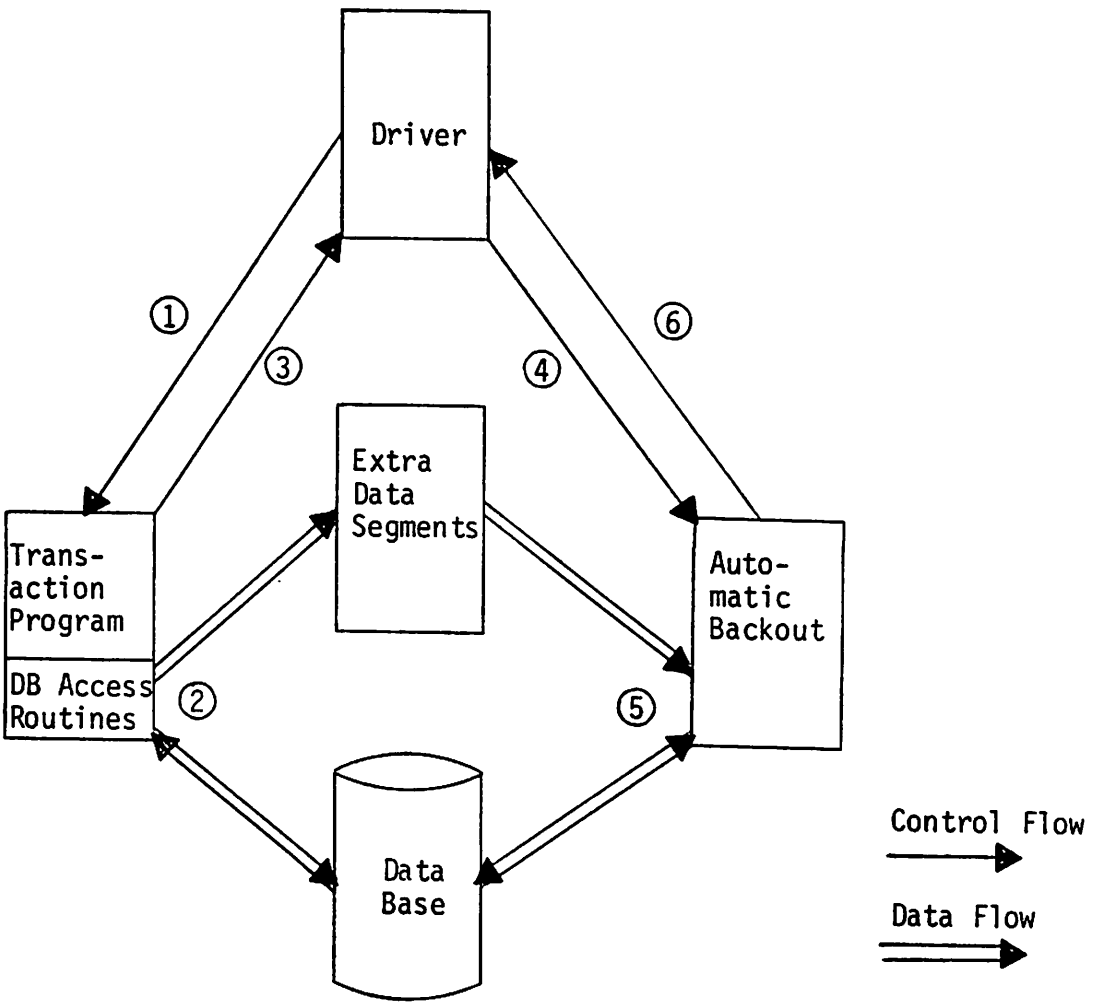
Since IMAGE requires a call to the GET procedure before a recond can be updated or deleted, all GETs are logged as they are performed. If the retrieved record is later updated or deleted, we record that information in the Extra Data segment. Likewise, all PUTs to the data base are also recorded in the EDS. Because the order of modifications is important, each individual data base modification is assigned a sequence number which is saved along with its buffer. If the transaction completes successfully, the 'before' log is purged. If the program does not complete successfully, the log remains which allows the data base to be restored to its state before the transaction began.

THE DRIVER PROGRAM AND TRANSACTION BACKOUT

The operating system, MPE, provides for multiple processes to communicate via the Job Control Word (JCW) facility. The JCW is set to an error value by MPE whenever a program terminates in an error state. A program may also set the JCW to a particular value indicating an unsuccessful transaction. By examining the JCW, a father process can tell if a son process completed successfully, was aborted by MPE, or terminated due to an error condition. This facility is used by a program known as the Driver to oversee the operation of all programs within BOARDS.

All programs within BOARDS are son processes of the Driver. It is the Driver's function to prompt the user for the function (s)he wishes to perform and then initiate the correct program to handle the user's function. The Driver then sleeps, waiting for the program to complete. If the program does not complete successfully, the Driver initiates a program known as Automatic Backout whose function is to restore the data base to its original state. Figure III shows the control and data flow in the case that a program aborts.

FIGURE III



1. The driver initiates a transaction program on Request from user.

2. The program through the Data Base Access routines retrieves and updates the data base. Logging takes place to extra data segments.

3. The program aborts. MPE or the program sets a JCW to an error value.

4. The Automatic Backout program is initiated to restore the data base to its initial state.

5. Automatic Backout restores the data base.

6. Control is returned to the driver and it is ready to process next users request.

Automatic Backout retrieves the Extra Data segments which contain the before record and applies the opposite operation as the aborted program applied. Figure IV summarizes the required operations. All transactions are done in the reverse order that they were done by the aborted program. This is necessary to insure that updates are done in the correct order and that IMAGE master/detail constraints are satisfied.

'AFTER LOGGING'

'After' logging is used to protect against data base transactions beging lost because of some error which makes the data base unusable. In such a case, an old version of the data base must be restored. If transactions were not logged, then all transactions that were entered since the time of the backup must be re-entered by the user. By logging the transactions as they occur to a tape, a program can be run to perform the data bases transactions which are recorded on the tape. In this manner, users need not re-enter data in order to recover from the loss of a data base.

The Data Base Access routines call upon the Malkin and Pinton transaction logging[3] system to perform the 'after' logging.

CONCLUSION

Through the use of common routines, the features of MPE and other HP-supplied system software have been expanded and utilized in COBOL application programs, without requiring that the programmers become familiar with MPE or SPL. These common routines serve as an Extended Operating Environment for the application programs within BOARDS.

FIGURE IV

DATA BASE OPERATION PERFORMED	FIXUP OPERATION	COMMENTS
PUT	DELETE	
UPDATE	UPDATE	with 'before' record
DELETE	PUT	

REFERENCES

1) DiGiammarino, P. and Schwartz, R., Decision Support System for the Management of Containerboard Logistics, 1978.

2) Data Entry Library Reference Manual, Hewlett Packard, Santa Clara California, 1977.

3) Malkin and Pinton Industrial Supplies, Transaction Logging System for the HP3000 Computer System, Vancouver, British Colombia, Canada, 1977.