TSO Attach: A multipurpose communication channel to IBM Database 2

by K. R. Hammond
M. R. Zimowski

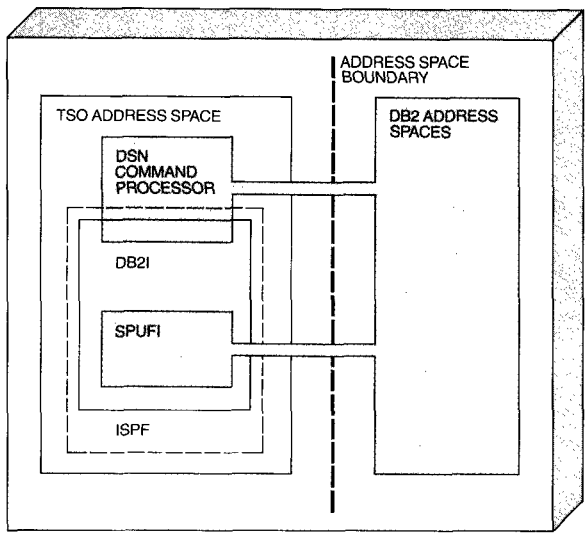
TSO Attach provides IBM Database 2 capabilities in a productive work environment that appears as a natural extension of the Time Sharing Option (TSO) and the Interactive System Productivity Facility (ISPF). It was designed and built with careful consideration for the varied and complex user group for which it was intended. Ease of use and ease of development and maintenance were among the significant factors in the design. These factors and others are addressed in this paper, which discusses the basic design decisions made in building the TSO Attachment Facility.

As general-purpose data base management systems evolve, data base capabilities are being made more available to users within their primary working environments. At the same time, a conscious effort is being made to ensure that user interfaces do not appear as highly stylized complex protocols. Rather, they are being designed as natural extensions of existing user environments. Also, data base capabilities are being presented through these interfaces in a manner consistent with the skills and working habits of the intended audience. Reducing the amount of learning required prior to their use is an important concern. In addition, an increased emphasis is placed on satisfying more and more of the needs of larger segments of the user community. A primary objective is to promote increased user productivity.

System R[1] and INGRES[2] are the earliest relational data base management systems that used this approach. Both of these prototypes permitted access to data through interactive interfaces that supported unified data manipulation and data definition languages. System R uses Structured Query Language (SQL) and INGRES uses QUEry Language (QUEL). The earliest approaches supported a linear syntax whereby a single statement could be entered in a straight-line fashion. Later approaches introduced interfaces that were more graphic in nature; Query-by-Example (QBE)[3] and CUPID[4] are well-known examples. In addition, each of the prototypes permitted its language to be embedded within application programs written in the programming languages most commonly used by the anticipated user communities. The syntax used in embedding the data manipulation and data definition statements was designed to appear as a natural extension of the application programming language.[5]

IBM Database 2 (DB2) furthers the goal of making data base capabilities available to users within the environments that they utilize daily. This goal is accomplished through attachment facilities, which connect the user's environment to DB2. DB2 has attachment facilities for the most commonly used Multiple Virtual Storage (MVS) environments: Customer Information Control System (CICS), Informa-

Figure 1 Basic structure of TSO attachment facility



tion Management System (IMS), and Time Sharing Option (TSO). TSO users access DB2 by using the TSO Attachment Facility. It makes the set of external controls and services required for DB2 application development, operation, and maintenance available to TSO users. In addition, it transforms the TSO environment into a DB2 transaction environment. This paper describes TSO Attach by first presenting the functional requirements and usability objectives of the TSO attachment package. It then discusses the manner in which these requirements and objectives were addressed, emphasizing the significant design decisions that were made.

Figure 1 presents an overview of the structure of TSO Attach. Each TSO user has a separate TSO address space, including a copy of the TSO Attach code. TSO Attach consists of three main blocks of code: the DSN[6] command processor, Database 2 Interactive (DB2I), and SQL Processor Using File Input (SPUFI). These three are described later in the paper. DB2I, which includes SPUFI, runs only under the Interactive System Productivity Facility. DSN runs as part of DB2I or by itself. Note that there are two communications paths to DB2.

Functional requirements and usability objectives

A typical TSO installation serves a wide range of users and provides various ways of invoking TSO services and user applications. Likewise, a primary requirement of TSO Attach was to satisfy the DB2 processing needs of a varied audience, while naturally extending and taking advantage of the invocation flexibility of the TSO environment. Ease of use was identified as a DB2 product objective from the very beginning. Accordingly, TSO Attach was engineered with an ardent consideration for ease of use. In addition, the implementation of TSO Attach was guided by the more traditional concerns for ease of development and maintenance.

Versatility. TSO Attach was designed primarily to satisfy the DB2 processing needs of data processing professionals, with a special concern for the needs of application programmers, data base administrators, system administrators, and DB2 operators. It was to be a comprehensive DB2 application programming environment for TSO users. Further, DB2 application development facilities were to be provided for the IMS and CICS transaction environments. These facilities were to include the capabilities for generating table declarations, precompiling application programs, creating and controlling application plans, compiling or assembling, and link-editing application programs. However, only programs written for the TSO environment were to be executable from TSO Attach. CICS and IMS programs were to run in their respective environments.

Another requirement was to permit on-line processing of SQL statements. One objective of this facility was to enhance the application development environment by permitting early debugging of SQL statements that were to be embedded in application programs. Also, the on-line SQL processing facility was expected to include functions to create and save command streams for later execution, as well as to include a basic report formatting capability.

A further objective was to provide an interface for use by data base and system administrators. Data base administration requires the use of Data Definition Language (DDL) SQL statements for the declaration of data bases and objects within data bases. Additional DDL statements permit an installation to enhance or modify previous definitions. Furthermore, authorization to access data base objects must be explicitly granted and controlled. System administration requires similar DDL usage to control system objects such as storage groups and buffer pools. The basic requirement for these users was a facility to execute the appropriate SQL statements for the creation of DB2 objects and for the granting and revoking of authorizations from the TSO environment. It seemed most natural to provide this capability on

line. In addition, we anticipated that the ability to execute individual statements or groups of statements, with the capability of either backing out or committing them at appropriate user-selected points in time, would simplify the data base and system administration tasks. Another requirement was the ability to invoke the DB2 utilities from TSO. This requirement was also primarily for data base and system administrators. A final requirement was the ability to execute DB2 commands from the TSO environment.

The TSO environment supports several user interfaces and permits a great deal of flexibility in their invocation. We intended that TSO Attach appear as an integrated part of the TSO environment, taking full advantage of the user interfaces and invocation flexibility. TSO Attach commands were to be implemented using the standard TSO command processor interface. A linear syntax interface was to be introduced for TSO users who like to enter commands in a linear fashion, a line at a time. An Interactive System Productivity Facility (ISPF) panel interface was to be included for TSO users who prefer ISPF. Duplication of functional capabilities was to be provided as seemed appropriate. Execution through command lists (CLISTS), and in both foreground and background, was to be supported whenever possible.

The design of TSO Attach was guided by several usability objectives.

Ease of use. The design of TSO Attach was guided by several usability objectives. It was to appear as a natural extension of the MVS/TSO environment. It was to require little training before use, and to provide on-line "help" during use. TSO Attach was to permit users to tune their session characteristics to meet their particular needs. Furthermore, TSO Attach was to provide clear and meaningful feedback describing the system response to any user request.

Ease of development and maintenance. From the beginning, we designed TSO Attach with a concern for ease of development and ease of maintenance. We used available TSO and ISPF capabilities whenever possible. We tried to avoid needless duplication of effort. TSO debugging aids were used and extended as deemed necessary. In addition, specialized tools, designed to ease the serviceability task, were implemented where our development needs and experience indicated that they would be worthwhile.

Versatility

A key requirement of TSO Attach was to perform many different functions for many different users. This need is reflected in many of our fundamental design decisions. It was apparent that the functional requirements could be handled by a group of commands. Different jobs naturally require different commands or groups of commands. It seemed that the set of all TSO DB2 commands could be handled by a single standard TSO command processor. One processor allows different users to work in the same command environment but to select only the subcommands that they need for their current task. We chose the name "DSN" for this general-purpose command processor. DSN became our basic mechanism for serving the needs of a large and varied user community.[6]

Different commands for different users. Specific DSN subcommands were introduced for the various anticipated users. For application programmers, the DSN command processor has a RUN subcommand. When a program runs, DSN makes the necessary connection to DB2, then passes control to the application. Applications must be bound before they can run. The BIND subcommand does this. The REBIND and FREE subcommands work with the BIND subcommand to perform the basic functions of plan creation, modification, and deletion.

The TSO environment was chosen as the environment where users prepare their programs for execution. Included are CICS and IMS users, as well as TSO users. This decision was based on the presence of useful system services (ISPF, parsing, foreground, background, etc.) and the simple fact that all MVS users have TSO. Support for this function is provided by DSN and some auxiliary CLISTS. Program preparation is a multistep process. It begins with the PL/I macro phase (for PL/I programs only), then does the DB2 precompile, performs the BIND, compiles or

assembles the program, and finally link-edits it. For TSO applications, the preparation process continues by executing the program. We wanted to provide automated program preparation support because of the complexity (BIND and precompile) that DB2 adds to the already sufficiently complex preparation process.

In order to facilitate access to DB2 controls, the decision was made for DSN to support most of the DB2 operator commands. This support offers the flexibility of multiple operator consoles. So, with only one exception, TSO users can control and interrogate DB2 from TSO. The exception is the -START DB2 command. DSN does not support -START because DSN must be connected to DB2 to do any work. It cannot be connected ("identified") unless DB2 is active. The only way around this restriction would have been to include knowledge of the MVS subsystem interface in DSN. Inclusion of this knowledge would have added significantly to the overall complexity of DSN while returning a fairly minor payback. We decided against it.

SPUFI is an ISPF panel option that permits on-line processing of SQL statements.

Data base administrators (DBAS) need to perform certain utility functions. DB2 provides an extensive collection of utility programs for this purpose. TSO Attach helps users invoke these programs. The utilities run only in background mode and so require appropriate Job Control Language (JCL) to support their execution. TSO Attach has an on-line service for building and customizing the JCL for these runs. DSN also participates in this service. In particular, it lets users terminate jobs or display their status. We decided against on-line support for the commands of individual utilities because the commands tend to be long and complex (suggesting that they be stored) and because the utilities tend to be long-running jobs. We did not think that users would want to tie up their terminals waiting for utility jobs to complete.

DBAS and system administrators have other requirements. Some of them are not handled by the DSN subcommands. The need to execute SQL statements (without writing a program) is a prime example. To process these statements, we wrote the SQL Processor Using File Input (SPUFI).

On-line SQL. SPUFI is an ISPF panel option that permits on-line processing of SQL statements. Unlike the Query Management Facility (QMF),[7] which is a query facility for a broad spectrum of users, SPUFI is an interactive interface for application programmers, data base administrators, and system administrators. Thus, many of our design decisions were guided by the assumption that users would have considerable data processing experience.

As a selectable option of an ISPF panel, SPUFI is easily accessible within the TSO/ISPF programming development environment. This accessibility allows application programmers to readily invoke SPUFI. SPUFI was designed to aid their efforts in several different ways. To begin, SQL statements can be entered and executed, permitting early debugging of SQL Data Manipulation Language (DML) statements that are to be embedded in application programs. It can be done before the application program is even written, let alone compiled or debugged.

Deciding to use ISPF EDIT and BROWSE was a major design decision in the implementation of SPUFI. Using ISPF EDIT, one or more SQL statements can be entered and saved in a user-specified data set. When this data set containing SQL statements is executed, the results are stored in another user-specified data set. SPUFI invokes BROWSE against this data set, so that users can examine the results. Thus, command streams of SQL statements can be created and saved, permitting easily repeated cycles of modification, execution, and analysis of results, until the SQL statements produce exactly the results needed in the application. It also allows repeated execution of SPUFI input data sets. This capability is particularly useful for DBAS or system administrators, who repeatedly require the same information. A new output data set may be specified to receive the results of each input data set executed, or the same output data set may be reused again and again. Both input and output data sets are named and selected for use in the fashion typical of TSO ISPF panels. Thus, SPUFI

data set naming and selection conventions are identical to those that already exist in the TSO environment.

The results of command stream executions may be selectively committed or rolled back for the current user-defined interval. This decision allows SPUFI to provide a truly experimental environment. Tentative sequences of SQL statements can be attempted, and the results can be scrutinized for the desired effect. The results may then be committed for future use or discarded. Also, the degree to which one user's committed changes are made visible within another user's session can be controlled. Thus, the effects of varying levels of isolation on SQL statement execution can be understood prior to actual execution of the completed application programs.

Also, we designed SPUFI to aid application development in other ways. The DB2 catalog is a DB2 data base. Therefore, application programmers can use SPUFI to query the DB2 catalog for information. Requests for information about tables, such as column names or column attributes, or for verification that requested indexes are available, are easily fulfilled. In addition, the on-line help facility allows application programmers to look up the syntax of SQL statements at the terminal. Information is also available concerning SQL return and warning codes.

SPUFI provides these same capabilities for data base administrators and system administrators. Any of the SQL DDL statements can be executed from a SPUFI input data set. DB2 data bases, tablespaces, tables, indexes, storage groups, and buffer pools can be created, dropped, or altered. In addition, any of the GRANT and REVOKE SQL authorization statements may be executed from a SPUFI input data set. Thus, data base designs and system configurations that are intended for application programmers or end users can be defined and tested, or just experimented with. Further, each of the SQL statements required to establish a particular data base design or system configuration can be saved in a single data set. Related statements can be kept together, providing a single repository of definitions for easy execution and modification.

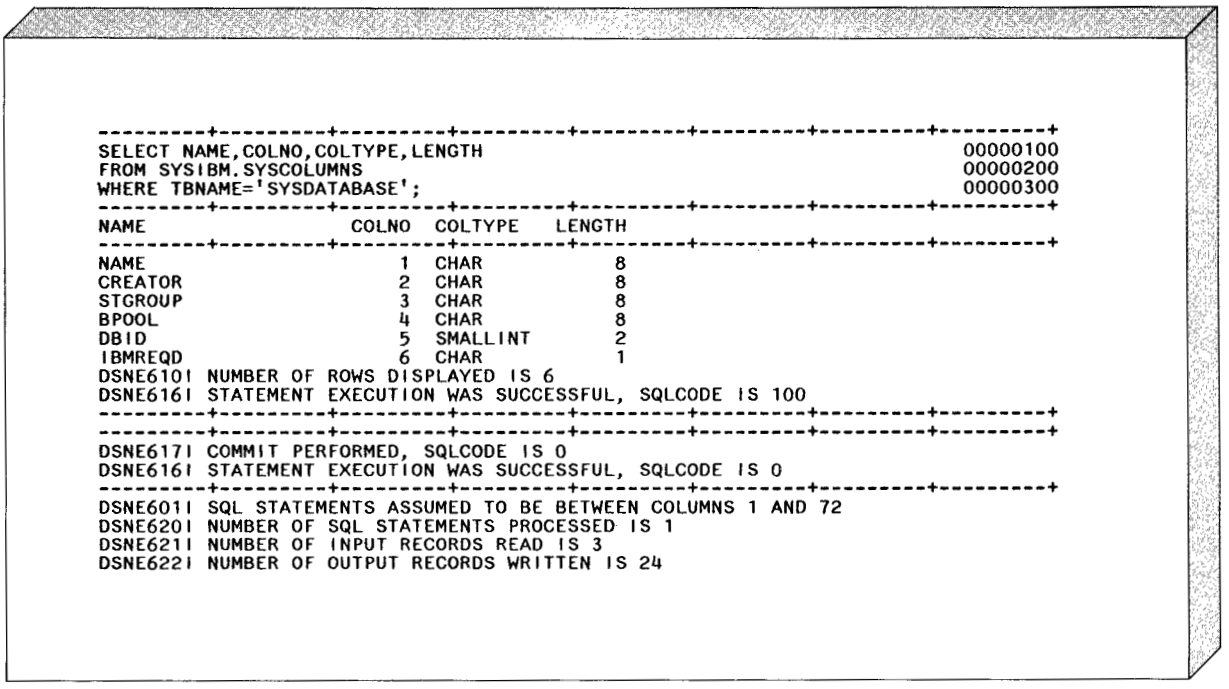
The creation of a data base design or system configuration can also be accomplished while other users continue to use the DB2 subsystem. Such creation is done by postponing the COMMIT until the entire definitional sequence is complete. Only then are the effects apparent. Thus, partially prepared environments are not visible to the intended users. If an error occurs in the definitional process, the portion of the definitional sequence that was completed can easily be rolled back. Each of these design decisions contributes to the flexibility required in an environment where experimentation is the norm. An AUTOCOMMIT option is also provided. This automatically commits all SQL statement results upon successful execution of the input data set. Any SQL error that occurs during the execution of an input data set causes automatic ROLLBACK processing.

As mentioned above, SPUFI results are stored in the data set the user specifies. A basic results-formatting capability is included to display the execution results. SPUFI output begins by echoing the SQL statement from the input data set. For Select statements, data values are placed beneath appropriate column headings. An execution summary, including the SQL return code and, if applicable, the number of rows manipulated, is provided for each SQL statement processed. Unusual situations such as data value or row truncation are reported. Comments present in the input data set are echoed in the output data set. SQL statement execution results can thus be delimited by user remarks. A final summary for the current SPUFI invocation is also provided. This summary includes information such as the number of SQL statements processed, input records read, and output records written. An example is given in Figure 2. Although very basic when compared to the elaborate QMF report formatter, the SPUFI results-formatting facility was designed to provide the essential execution results in a concise and readable fashion, suitable for the intended audience.

Another design decision of note was to make SPUFI part of the ISPF environment. This decision permits easy access to available TSO/ISPF capabilities. System administrators, for example, can look up volume and VSAM catalog information. Data base administrators can experiment with different EDIT and VALIDATION procedures. They can also verify tablespace and index primary and secondary space allocations. The ISPF panel print facility can be used to obtain a printed copy of both SPUFI input and output data sets. In general, the proximity to ISPF gives SPUFI users the full power of the ISPF Program Development Facility.

Multiple environments. Another fundamental way of being versatile is to run in more than one environment. Fortunately, TSO comes with a fine selection of software features and tools that made this

Figure 2 An example of SPUFI output



```
---------+---------+---------+---------+---------+---------+---------+---------+
SELECT NAME,COLNO,COLTYPE,LENGTH                                          00000100
FROM SYSIBM.SYSCOLUMNS                                                    00000200
WHERE TBNAME='SYSDATABASE';                                               00000300
---------+---------+---------+---------+---------+---------+---------+---------+
NAME                 COLNO  COLTYPE   LENGTH
---------+---------+---------+---------+---------+---------+---------+---------+
NAME                     1  CHAR           8
CREATOR                  2  CHAR           8
STGROUP                  3  CHAR           8
BPOOL                    4  CHAR           8
DBID                     5  SMALLINT       2
IBMREQD                  6  CHAR           1
DSNE610I NUMBER OF ROWS DISPLAYED IS 6
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
---------+---------+---------+---------+---------+---------+---------+---------+
---------+---------+---------+---------+---------+---------+---------+---------+
DSNE617I COMMIT PERFORMED, SQLCODE IS 0
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0
---------+---------+---------+---------+---------+---------+---------+---------+
DSNE601I SQL STATEMENTS ASSUMED TO BE BETWEEN COLUMNS 1 AND 72
DSNE620I NUMBER OF SQL STATEMENTS PROCESSED IS 1
DSNE621I NUMBER OF INPUT RECORDS READ IS 3
DSNE622I NUMBER OF OUTPUT RECORDS WRITTEN IS 24
```

goal relatively easy to achieve. For example, the DSN command processor was originally coded to run in TSO foreground. However, the TSO Terminal Monitor Program runs in either foreground or background. This capability means that DSN can run in foreground or background. The only DSN capabilities that are not supported in background are attention processing and prompting from the parse routines. These capabilities are not required in background processing as it is defined today.[8] The ability to run the DSN command processor in background permits users to be more productive without adding complexity. They do not need to learn a different language for each environment. The same command set does the job, only without tying up the terminal in the way foreground sessions do.

Another TSO facility that made it easy for us to write code that can run in different environments was the CLIST processor. It is easy to invoke DSN from a CLIST. The CLIST needs a DSN command, followed by a series of DSN subcommands, and finally, an END subcommand to terminate the DB2 session. Although many people are probably in the habit of running CLISTS in foreground, they can also run in background as long as the proper responses to CLIST prompts are known in advance. Figure 3 shows a JCL stream that invokes DSN, runs a program, and terminates the DSN session. This is followed by a CLIST invocation; the CLIST manages a second DSN session. In the figure, the JCL stream invokes the DSN command processor in background mode. Notice that the GO step invokes the TSO Terminal Monitor Program, which accepts standard TSO commands in its input stream (the lines following the SYSTSIN DD statement). After the DSN command processor runs the program named PROGA, it calls a CLIST, "H443722.TSO.CLIST(TEST)," which rebinds two plans and runs two programs, PROGB and PROGC.

Thus, we support linear syntax for DSN subcommands in three different environments: foreground, background, and CLISTS. For some users this will be sufficient, but many users prefer full-screen syntax to linear syntax, so we wanted to include a panel interface in the TSO Attach package. We were pleased to discover that ISPF provides excellent facilities for building panel dialogs.

We call the full-screen syntax portion of TSO Attach Database 2 Interactive (or DB2I). Refer to Figures 4A–4F to see samples of the DB2I panels. Figure 4A shows the ISPF primary option panel. It is normally the first panel seen on entering ISPF. Enter an "8" on

Figure 3 JCL stream invoking DSN

```
//SAMPLE   JOB ,
// NOTIFY=H443722,USER=H443722,REGION=300K,
// MSGCLASS=Z,TIME=(0,59)
//*********************************************************************
//* THIS JCL INVOKES A PROGRAM CALLED PLANA, THEN RUNS A CLIST
//* CALLED TEST.
//*********************************************************************
//GO       EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
DSN  SYSTEM(SSTR)
  RUN PROGRAM(PROGA) LIBRARY('H443722.TSO.LOAD') PLAN(PLANA)
END
EXEC 'H443722.TSO.CLIST(TEST)'
/*


PROC 0
/*********************************************************************/
/* THIS CLIST DEMONSTRATES THE WAY TO INVOKE THE DSN COMMAND          */
/* PROCESSOR FROM A CLIST. THIS CLIST CAN RUN EITHER IN FOREGROUND    */
/* OR BACKGROUND. IT RESIDES IN DATA SET 'H443722.TSO.CLIST(TEST)'.   */
/*********************************************************************/
DATA
  DSN SYSTEM(SSTR)
    REBIND PLAN(PLANB, PLANC)
    RUN PROGRAM(PROGB) LIBRARY('H443722.TSO.LOAD') PLAN(PLANB)
    RUN PROGRAM(PROGC) LIBRARY('H443722.TSO.LOAD') PLAN(PLANC)
  END
ENDDATA
EXIT
```

this panel to see the DB2I primary option panel, which is shown in Figure 4B. Use the DB2I panel to choose the particular DB2I function to be performed. If a "4" is entered, the first program preparation panel will appear, as seen in Figure 4C. This panel is used to begin the program preparation process. The first step is precompilation. The program preparation process continues with the panel in Figure 4D. It is used to bind, compile, link, and run a program. This panel is reached by typing YES on line 5 of the panel shown in Figure 4C and then pressing the enter key. Figure 4E is the main SPUFI panel. It is reached by entering a "1" from the main DB2I panel (Figure 4B). Figure 4F shows the SPUFI defaults panel. It controls the format of SPUFI output. This panel is reached by typing YES on line 10 of the main SPUFI panel and pressing the enter key.

DB2I adds a subtree of panels to the ISPF panel tree. These panels support the DSN functions, program preparation, and the DB2 utilities. SPUFI is also a DB2I panel option. Figure 5 shows the DB2I section of the ISPF panel tree. With it to round out our possible environments, users have the flexibility to use the product in the way that suits them best. From the ISPF primary option panel, DB2I can be entered by selecting option 8. From there, the function needed is chosen. For example, typing a "5" and pressing the enter key can run a program.

Ease of use

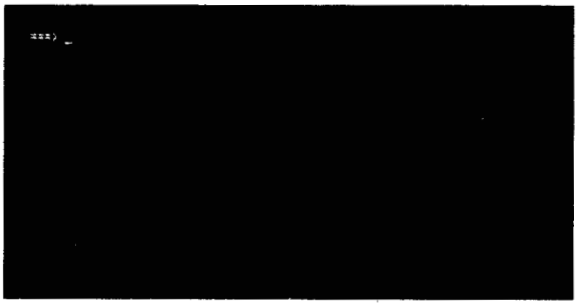
TSO Attach was designed to be easy to use. Although we assumed that most TSO Attach users would be experienced computer users, we still wanted to minimize the amount of knowledge they needed to have already, or to gain, in order to use it effectively. In doing that we supported another product objective: to minimize the length of time needed to start doing productive work.

A full-screen interface. For new TSO Attach users, our main strategy was DB2I. The full-screen syntax of DB2I can help new users in many ways. For any

Figure 4 (A) ISPF primary option panel; (B) DB2I primary option panel; (C) Program Preparation panel 1; (D) Program Preparation panel 2; (E) SPUFI panel 1; (F) SPUFI panel 2.

(A)

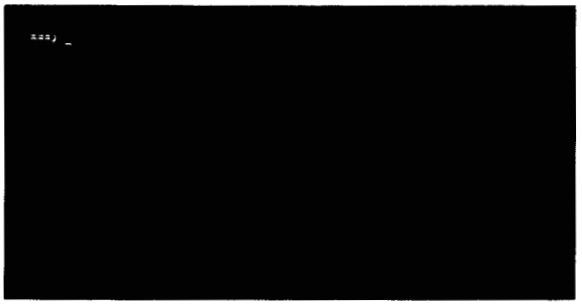
(B)



(C)

(D)

(E)



(F)

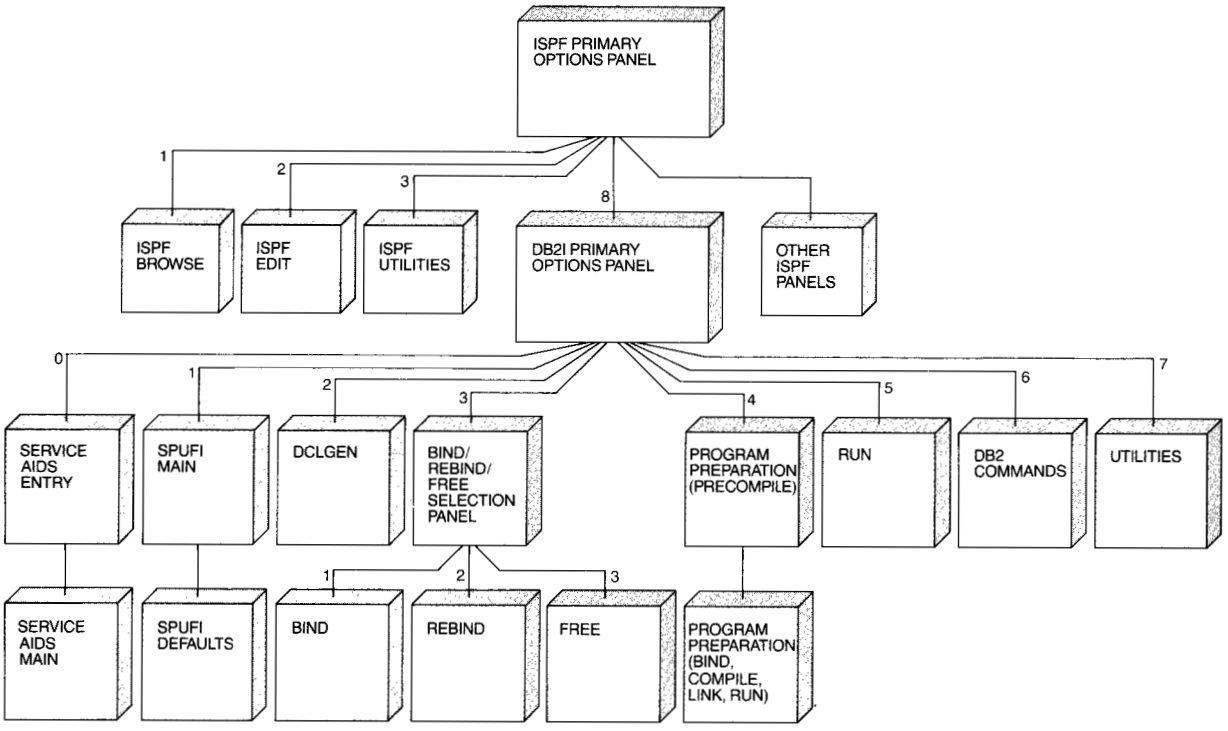
given task, it lists the possible parameters. It describes the permitted values for each parameter and provides defaults when possible. In addition, brief explanations of the different fields appear on the panels. More complete explanations can be seen by pressing the Help key. DB2I remembers field values from session to session, eliminating the need for users to do the remembering or to look up information.

Further, by using ISPF as the base for DB2I, we were extending an environment that is familiar to and well liked by many users. This avoided the need to introduce users to a new environment. For most users, we think that DB2I will be the preferred work environment.

A linear syntax interface. Some users prefer linear syntax. These users tend to be good typists and to know the product and particularly the command syntax well. Some users will want to run in background mode. And, of course, there will be users whose systems do not include ISPF. For all these people, we provided the basic linear syntax DSN command processor. They will be able to accomplish most of the TSO Attach functions (SPUFI being the notable exception) without entering ISPF.

Human factors. To ensure that the human factors of the project were reasonable, we carefully considered various corporate human factors guidelines that had been developed previously as the result of extensive

Figure 5 DB2I section of the ISPF/PDF panel tree



testing and research. They were especially important when we chose designs for our panels. The panels were carefully reviewed by many people to optimize their human factors. The panels were made as simple, yet as informative and flexible, as possible. This consideration required many compromises. For example, in the interest of simplicity and keeping the number of panels low, we decided that we would not support all the capabilities of the DSN command processor. Although some DSN subcommands have parameters that are lists of potentially "infinite" length, DB2I handles only short versions of these lists. Lists are generally easy to process in linear syntax but awkward on entry panels. The shortened list capability kept our overall panel structure significantly smaller and simpler.

An on-line Help facility. Perhaps the single most important thing that can be done to help users is to provide an on-line Help facility. It should be accessible from whichever environment is currently executing. Since TSO Attach runs interactively either as part of ISPF (DB2I), or just in TSO (DSN), we needed help for both environments. For DB2I, there are almost 500 ISPF Help panels. There are Help panels that correspond to particular tasks being done, and panels that contain information like SQL syntax and the meanings of SQL return codes. For DSN users, we use the TSO Help facility. This Help is less extensive than the ISPF Help facility, but again, the presumption was that the DSN users would already know command syntax. Otherwise, they would be using the DB2I panels. Of course, all this information (DSN and DB2I) is contained in the various reference manuals that accompany DB2.[9,10]

User-initiated program interruptions. Some of our human factors effort has had a more technical flavor. For example, in order to allow users to interrupt program execution when DB2 is processing an SQL request, we had to design the attach code in what might appear to be an odd structure. Rather than running under a single task (TCB), we use a two-task structure. When DSN starts, it performs some initialization, and then attaches a second load module. Without this second task, it would not have been possible to interrupt the execution of the TSO Attach TCB when it is executing in the DB2 address spaces, without losing our connection to DB2. This property is fundamental to MVS.

Asynchronous communication with DB2. Another significant technical problem we had to solve was to provide a communications channel from DB2 to the attach code. Under normal circumstances, the attach code makes requests to DB2, DB2 performs the requested service, then returns control to the application program or the attachment facility. The communication direction is from TSO Attach to DB2. However, there are situations in which DB2 may need to communicate in the opposite direction. For example, if the operator issues the -STOP DB2 command with the QUIESCE option, DB2 has an unanticipated need for TSO users to conclude their DB2 sessions as soon as conveniently possible. Good human factors required us to ask users to end their sessions, rather than just having us terminate their sessions. This option permits users to finish their current tasks and then to end their sessions. If the users cooperate, and there is no urgent need to terminate the TSO sessions quickly, the QUIESCE option will eventually make the system quiescent. However, there are some situations in which only limited delay is tolerable, or in which users ignore requests to terminate their sessions. In these cases the DB2 operator will need a stronger command: the -STOP DB2 command with the FORCE option. To implement these STOP options, we used a combination of MVS WAIT and POST macros to permit asynchronous communication from DB2 to TSO Attach. The important point here is that for good human factors, we had to build a two-way communications channel between TSO Attach and DB2.

One of the basic ideas that appears in various forms throughout the attach facility is centralization.

TSO commands from DSN. Another way we made the product easier to use was by allowing users to issue TSO commands from within the DSN command processor. Whenever DSN reads in a new command, it searches its list of known commands. If it finds a match, it does the work requested by the command. If no match occurs, it attempts to attach the command. If a TSO command processor by that name is found,[11] it will execute. Otherwise, an error message is issued. This feature lets users do most of their work under the umbrella of the TSO Attach package; it is not necessary to be continually changing environments in order to complete a complex job.

Ease of development and maintenance

TSO Attach had to be written in a limited amount of time by a limited number of people. There was little time for wasted or duplicated effort. We had to design a high-quality package that could be developed and maintained with a minimum of effort. To accomplish this task, we adapted various ideas and practices that have made our work more efficient.

Shared and centralized code. One of the basic ideas that appears in various forms throughout the attach facility is centralization. DSN was implemented as a single, centralized processor. One advantage this structure has over a group of command processors is that a single processor can provide common services that are needed by each of the basic functions. In other words, we avoided the need to duplicate code that would have done the same job in different command processors. The DB2 connection services are a good example. Talking to DB2 directly from TSO, without an attachment facility, is not such an easy thing. A complex connection protocol must be rigorously followed. It is necessary to carefully manage a number of control blocks and parameter lists. Timing is important; just the right DB2 service requests must be made at just the right time. As it is, approximately 15 percent of the TSO Attach code is devoted to managing the connection. All the DSN subcommands share this code.

However, because of a technical problem, SPUFI had to establish its own connection to DB2. In fact, SPUFI could be described as a separate attachment facility. It can run without any notion of DSN. Thus, a fairly large piece of code is duplicated in SPUFI and DSN. This aspect is probably the least satisfactory of all the aspects of the TSO Attach design. Unfortunately, there was no alternative.

Another area where centralization cut our coding and maintenance effort was attention handling. We needed an attention routine to let users interrupt the execution of their programs. The attention routine we coded serves all the subcommands of the DSN command processor. Error and abnormal end processing is also conveniently handled by a centralized approach. A single ESTAE and a single ESTAI exit

routine are shared by all the DSN subcommands. These exit routines make it possible to trap and handle abnormal ends, whether caused by the application or by the system.

Similarly, our message-generation subsystem benefits by centralization. By grouping all the message text into a small number of data sets (that contain nothing but message text), we made it much easier to translate our messages into foreign languages. Foreign language translation of decentralized messages would have required recompilation of all the TSO Attach modules. Such recompilation would have been especially undesirable since a basic part of the DB2 maintenance strategy was *not* to ship source code, let alone PLS compilers.[12]

Inclusion of a tracing facility made the development and maintenance effort much easier.

A basic implementation strategy we adopted at the very beginning was to take advantage of whatever services were provided as part of the TSO environment. This strategy minimized duplicate coding and maintenance effort. It led us to use the TSO scanning, parsing, dynamic allocation, and message-writing services rather than writing our own. TSO Test was our main debugging facility. The CLIST and batch job submission facilities enabled us to automate much of our extensive testing effort. Most of our compilation was done in background mode, so we could keep working in foreground while we waited for compiles to complete. All in all, TSO services helped to reduce our overall development effort. We also expect that they will reduce the maintenance effort.

ISPF Dialog Management Services makes it easy to create, to display, to read from, and to write to ISPF panels. Panel images are essentially constructed as they are to appear. Panel logic initializes and controls the entry of values into fields on the panels. A high-level command language makes it easy to display and read from the panels. ISPF variable services permit easy initialization and recall of variable values from previous sessions. The ISPF profile and shared variable support provide a mechanism for defining installation defaults. Finally, ISPF EDIT and BROWSE provide a powerful file processing capability, permitting easy implementation of the SPUFI input and output data set processing facilities. Thus, the ISPF environment provided numerous services that reduced the effort required to implement DB2I, our full-screen interface. In particular, we did not have to design and write our own screen handler and dialog manager.

Tracing and debugging facilities. The inclusion of a tracing facility made the development and maintenance effort much easier. TSO Attach has an extensive tracing facility. We provide four different types of tracing: one for DSN, one for the CLISTs, and two for SPUFI. These tracing mechanisms make it very easy for us to pinpoint the general area of a problem. Frequently, they lead us to the exact problem. They often enable us to fix a problem without resorting to a storage dump.

When tracing, DSN normally writes its trace messages to the terminal. However, by allocating a DSNTRACE data set, the trace stream can be collected in a data set or sent to a printer. DSN trace messages include a module identifier and four other substitutable tokens. One of the tokens explains what the others mean.

One special class of DSN trace messages has been especially useful—these are the "wall messages." DSN emits one of these messages just before it makes each DB2 service request and just after control returns from DB2. Figure 6 shows part of a sample DSN trace stream that includes the wall messages that appear before and after a TERMINATE thread call. The wall messages have been exceedingly useful for debugging problems when it was not clear if the problem was in TSO Attach or DB2 code. We had to make this decision frequently. Many users started by invoking the TSO Attach code; then if something went wrong (and in the early phases of testing, things went wrong), they were much inclined to blame us, even though the problem was in the underlying DB2 code. The walls drew a clear line between our problems and theirs. They helped us avoid many fruitless hours of trying to fix problems in DB2 by "debugging" TSO Attach code.

The DB2I CLISTs trace by using the CONTROL statement provided in the CLIST language. This option

Figure 6 Sample DSN trace stream showing "wall messages"

```
DSNET201 DSNECP13 CIBPLNID :TSOAP1 X'00000000' X'00000000'
DSNET201 DSNECP13 CIBCORID :H443722 X'00000000' X'00000000'
DSNET201 DSNECP13 ZINDOUBT :NO X'00000000' X'00000000'
DSNET201 DSNECP13 R6=, CIBCTRTN= X'0013F7D8' X'E2E8D5C3'
DSNET201 DSNECP13 EXIT DSNECP13 X'00000000' X'0013FD38'
DSNET201 DSNECP28 AFTER DSNECP13, CIBCTRTN=, CIBCTFRB= X'E2E8D5C3' X'0013FD38'
DSNET201 DSNECP28 BEFORE SQL CALL NUMBER ONE  CIBFRMLI=, DSNETRAP= X'8015F698' X'8015F698'
DSNET201 DSNECP28 HERE COMES THE FRB <<<<<<<<<<<<<<<< CIBRFRB= X'8013FD38' X'00000000'
DSNET201 DSNECP28 FRBRAL(PTR),     FRBRALE(BIN15), FRBFVLE(BIN15) X'00000000' X'00030001'
DSNET201 DSNECP28 FRBPARM(PTR),    FRBPCNT(BIN15) X'001671E0' X'00000000'
DSNET201 DSNECP28 FRBRC1(BIN15),   FRBRC2(CHAR4) X'00000000' X'00000000'
DSNET201 DSNECP28 FRBFBACK:                                                X'00000000' X'00000000'
DSNET201 DSNECP28 FRBFBACK PTR(31),FRBRHPC X'00000000' X'00000000'
DSNET201 DSNECP28 FRBQUAL(BIN15), FRBRSV1(BIN15) X'00000001' X'00000000'
DSNET201 DSNECP28 ADDR(DSNAPRH)=, ADDR(DSNETRAP)= X'8014B198' X'8015F698'
DSNET201 DSNETRAP BEFORE SQL CALL=======================FRB,R1== X'8013FD38' X'0013F808'
DSNET201 DSNETRAP AFTER SQL CALL========================RC1,FBACK== X'00000000' X'00000000'
DSNET201 DSNECP28 HERE COMES THE FRB <<<<<<<<<<<<<<<< CIBRFRB= X'8013FD38' X'00000000'
DSNET201 DSNECP28 FRBRAL(PTR),     FRBRALE(BIN15), FRBFVLE(BIN15) X'00000000' X'00030001'
DSNET201 DSNECP28 FRBPARM(PTR),    FRBPCNT(BIN15) X'001671E0' X'00000000'
DSNET201 DSNECP28 FRBRC1(BIN15),   FRBRC2(CHAR4) X'00000000' X'00000000'
DSNET201 DSNECP28 FRBFBACK:                                                X'00000000' X'00000000'
DSNET201 DSNECP28 FRBFBACK PTR(31),FRBRHPC X'00000000' X'00000000'
DSNET201 DSNECP28 FRBQUAL(BIN15), FRBRSV1(BIN15) X'00000001' X'00000000'
DSNET201 DSNECP28 ADDR(DSNAPRH)=, ADDR(DSNETRAP)= X'8014B198' X'8015F698'
DSNET201 DSNECP28 EXIT DSNECP28, RETURNING TO DSNELI    FRBRC1,FRBRC2 X'00000000' X'00000000'
DSNET201 DSNETRAP BEFORE SQL CALL=======================FRB,R1== X'8013FD38' X'0013F808'
DSNET201 DSNETRAP AFTER SQL CALL========================RC1,FBACK== X'00000000' X'00000000'
DSNET201 DSNETRAP BEFORE SQL CALL=======================FRB,R1== X'8013FD38' X'0013F808'
DSNET201 DSNETRAP AFTER SQL CALL========================RC1,FBACK== X'00000000' X'00000000'
DSNET201 DSNECP16 AFTER LINK GOOD, R15=, R1= X'00000000' X'0013FB70'
DSNET201 DSNECP18 ENTER DSNECP18, CIBTRMOP= X'E2E8D5C3' X'00000000'
DSNET201 DSNECP18 CIBTRMOP NOT BLANK, CIBTRMOP= X'E2E8D5C3' X'00000000'
DSNET201 DSNECP18 FRB FIELDS FOLLOW (CIBFRB): X'0013FD38' X'00000000'
DSNET201 DSNECP18 FRBRAL(PTR),     FRBRALE(BIN15), FRBFVLE(BIN15) X'00000000' X'00010001'
DSNET201 DSNECP18 FRBPARM(PTR),    FRBPCNT(BIN15) X'0013F360' X'00000000'
DSNET201 DSNECP18 FRBRC1(BIN15),   FRBRC2(CHAR4) X'00000000' X'00000000'
DSNET201 DSNECP18 FRBFBACK(PTR),   FRBRHPC(BIN32) X'00000000' X'00000000'
DSNET201 DSNECP18 FRBQUAL(BIN15), FRBRSV1(BIN15) X'00000001' X'00000000'
DSNET201 DSNECP18 BEFORE TERMINATE DB2 CALL===============TRMOP= X'E2E8D5C3' X'00000000'
DSNET201 DSNECP18 AFTER TERMINATE DB2 CALL=======================  X'00000000' X'00000000'
DSNET201 DSNECP18 FRB FIELDS FOLLOW (CIBFRB): X'0013FD38' X'00000000'
DSNET201 DSNECP18 FRBRAL(PTR),     FRBRALE(BIN15), FRBFVLE(BIN15) X'00000000' X'00010001'
```

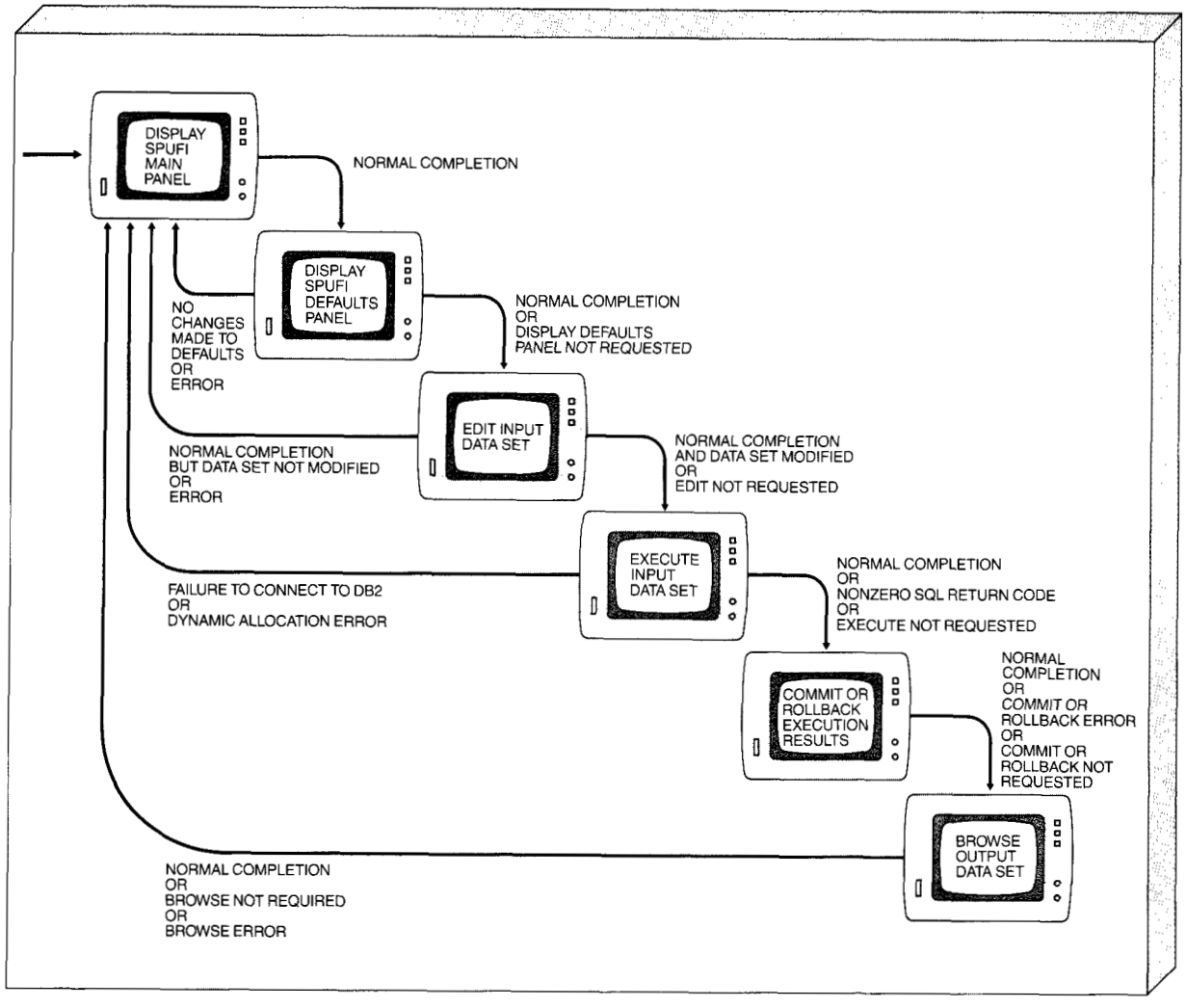
WALL MESSAGES

causes all the CLIST statements to be echoed to the terminal as they execute. Actually, we see two copies of each line. The first shows the line before variable substitution has occurred, the second shows the line after the symbolic variables have been replaced by their values.

SPUFI has two tracing mechanisms. We call one SPUFI tracing, the other, SPUFI debugging. Either or both can be selected. In SPUFI tracing, an ISPF Services call is used to write trace information to the ISPF LOG. Entry to and exit from each SPUFI module, as well as significant internal events, are traced. In SPUFI debugging, ISPF Service calls display SPUFI debug panels. SPUFI debug panels display the contents of interesting storage areas at predefined points in time. From a single debug panel, the scope of future panel displays can be expanded or contracted. Also, any of the debug panels may be requested from any other debug panel. Thus, storage areas normally viewed at earlier or later points in time can be redisplayed when desired. In addition, most storage areas can be changed by typing over values displayed on the debug panels. This is a very powerful debugging mechanism.

Resilient program design. The design of the main SPUFI control module is a further instance of our attempt to ease the burden of development and maintenance. This module implements a finite-state machine.[13] A state exists for displaying and modifying the SPUFI processing panel and for each SPUFI processing option selectable from that panel. Hence, additional states exist for displaying and changing defaults, editing the input data set, executing SQL statements, committing execution results, and browsing the output data set. Figure 7 presents a simplified version of the state diagram.

Figure 7 Simplified SPUFI state diagram



The finite-state machine model guided the implementation of each state as a well-defined independent functional entity. Each state can be selected and executed separately. This independence allows a modularization of SPUFI function, which makes the implementation easy to understand and maintain.

Likewise, the transition from one state to another is based on the finite-state machine concept. The current state is driven to a particular next state by an input value. For example, suppose the user requests only a subset of the SPUFI processing options, namely, to execute a predefined input data set and to browse the results. The initial state is to display the SPUFI processing panel. The expected transitions to subsequent states follow the normal execution sequence (change the defaults, edit the input data set, execute the SQL statements, commit the execution results, and browse the output data set) of the selected processing options. In this case, the request to execute an input data set causes a transition to the state for executing SQL statements. For this example, the expected next transition would be the browse output state, followed by the standard last transition back to the display SPUFI processing panel state. Transitions between states are well-defined and crisply expressed.

This approach also makes it easier to handle both anticipated and unanticipated errors. Transitions to

states that display error responses and request additional or new information (e.g., back to the display SPUFI processing panel state) are easy to implement, and can be conceptualized separately from the details of a particular state's function. This approach also allows error processing to be developed in stages or enhanced, by merely adding additional transitions between states. SPUFI development profited from this design and implementation approach. Future maintenance efforts should benefit as well.

Summary

In building the TSO Attachment Facility, our basic design goal was to provide a work environment for many of the personnel associated with the use, maintenance, and control of a large and complex data base system. Other important design goals included ease of development, ease of maintenance, and ease of use.

Our design solution was to build the DSN command processor and the DB2I and SPUFI interfaces that complement it. These three parts support a rich command language, in both full-screen and linear syntax. They make it possible for users to tailor their work environment to suit themselves. The modular construction and shared code in TSO Attach minimized the amount of code we had to write. This construction, and the extensive debugging tools built into TSO Attach, make it easier to maintain. Much consideration was given to the design of command syntax, panels, and error messages to produce a product that would be easy to use. As a result, TSO Attach provides a productive user interface to DB2 that appears as a natural extension of the TSO/ISPF environment.

Acknowledgments

Aside from the authors, several other developers participated in the design and development of TSO Attach. We would like to acknowledge the contributions of T. G. Messinger, T. H. Sawyer, S. K. Fang, J. T. Heglar, and D. A. Weil. We also wish to acknowledge the management support and direction provided by H. L. Reeves, R. A. Reinsch, and H. Leonard.

Cited references and notes

1. M. M. Astrahan et al., "System R: Relational approach to database management," *ACM Transactions on Database Systems* **1**, No. 2, 97–137 (June 1967).
2. M. Stonebraker, E. Wong, P. Kreps, and G. Held, "The design and implementation of INGRES," *ACM Transactions on Database Systems* **1**, No. 3,189–222 (September 1976).
3. M. M. Zloof, "Query-by-Example," *Proceedings of the AFIPS 1975 National Computer Conference* **44**, 431–437, AFIPS Press, Montvale, NJ (May 1975).
4. N. McDonald and M. Stonebraker, "CUPID—The friendly query language," *Proceedings of the ACM–Pacific–75 Conference*, San Francisco, CA (April 1975), pp. 127–131.
5. M. Stonebraker and L. Rowe, *Observations On Data Manipulation Languages and Their Embedding in General Purpose Programming Languages*, Memorandum UCB/ERL M77/53, Electronics Research Laboratory, University of California, Berkeley, CA (July 1977).
6. DSN is the system prefix used in naming many of the objects associated with DB2. For example, all DB2 error messages begin with the string DSN.
7. *Query Management Facility: General Information*, GC26-4071, IBM Corporation (1983); available through IBM branch offices.
8. When the user enters an invalid parameter on a command in foreground, the TSO parsing service, which we use extensively, will ask him or her to reenter the invalid parameter. It can even suggest possible values to enter.
9. *IBM Database 2 Application Programming Guide for TSO Users*, SC26-4081, IBM Corporation (1983); available through IBM branch offices.
10. *IBM Database 2 Reference*, SC26-4078, IBM Corporation (1983); available through IBM branch offices.
11. TSO commands are normally implemented as TSO command processors. There is at least one load module, residing in either a system, project, or personal library, to process each command.
12. DB2 is mostly written in PLS. PLS is a PL/I-like proprietary language used by the IBM Corporation for product development. The language is not available to the public.
13. P. J. Denning, J. B. Dennis, and J. E. Qualitz, *Machines, Languages, and Computation*, Prentice-Hall, Inc., Englewood Cliffs, NJ (1978).

Reprint Order No. G321-5215.

Kenneth R. Hammond *IBM General Products Division, Santa Teresa Laboratory, P.O. Box 50020, San Jose, California 95150.* Mr. Hammond is a senior associate DB2 development programmer working at the Santa Teresa Laboratory. He began working on the TSO Attach Component in early 1980. Prior to this assignment, he attended the University of California at Berkeley, where he obtained the undergraduate degree in computer science. While attending college, he worked for IBM as a computer operator and computer programmer.

Melvin R. Zimowski *IBM General Products Division, Santa Teresa Laboratory, P.O. Box 50020, San Jose, California 95150.* Mr. Zimowski is a staff programmer in Advanced Database Development. He joined IBM in 1974 at Endicott, New York, where he participated in the development of final test systems for the logic and memory chips of the 4300 series of computers. He has worked with the Advanced Database group since 1980. In 1973, Mr. Zimowski completed courses of study for B.A. and M.A. degrees in mathematics at the State University of New York at Binghamton. He also received an M.S. degree in computer science from the University of California at Berkeley in 1980.