Cross System Product application generator: Application design

by M. E. Dewell

This paper describes some techniques that can be used for Cross System Product/Application Development (CSP/AD) application design. CSP/AD is an application development tool for professional programmers. A well-designed application is obtained by using proven principles of structured analysis, structured design, and structured programming. An understanding of these principles and the application definition constructs provided by Cross System Product/Application Development is necessary for the CSP/AD application designer. Application design for CSP/AD is accomplished by using a combination of techniques for data design, application design, and application program design. For each of these design techniques there exist formal, accepted practices, and methodologies that may be used. These techniques are described, and methods that have proven successful for designing CSP/AD applications are presented.

Cross System Product/Application Development (CSP/AD) is a product that provides interactive definition and test, including batch or interactive generation of application programs. In this paper, an *application* consists of one or more closely related programs that support a functional area of an enterprise, e.g., a payroll or inventory control system. An *application program* is the application definition and generation unit for CSP/AD. A CSP/AD application definition is functionally equivalent to a third-generation language source program. See References 1 to 3 for more information on the facilities of CSP/AD.

CSP/AD had its beginning in 1978 and has emphasized the advantages of a structured modular program design. A history of the product is discussed in Reference 4. Structured design is enabled in CSP/AD by a number of application definition constructs that are presented in this paper.

The reader should be familiar with structured analysis and design methods, and data normalization principles. This paper does not present, teach, or favor any particular method. Structured analysis and design methods are adequately and elaborately presented in other publications that are available to the reader. (See References 5 to 9.)

The CSP/AD application designer may use many standard structured analysis and design methods. For a structured analysis method, see Gane and Sarson (Reference 5); for structured design and module design, including good module coupling and cohesion, see Yourdon and Constantine (Reference 6); for structured programming, see Linger, et al. (Ref-

erence 7); and for data normalization, see Codd (References 8 and 9).

CSP/AD is one of the participating products in the AD/Cycle™ architecture and AD/Cycle tools strategy. Another paper in this issue of the *IBM Systems Journal* discusses these subjects.[10]

CSP/AD provides a common and consistent end-user interface conforming to the Systems Application Architecture™ (SAA™) Common User Access (CUA) architecture, with its cooperative application definition facility implementation on a programmable workstation.

The external source format interface of CSP/AD is the means used to inform CSP/AD of enterprise analysis and design information that is provided by AD/Cycle tools.

CSP/AD supports Systems Application Architecture by providing the SAA application generator interface components of the SAA Common Programming Interface (CPI).[11] CSP/AD applications that conform to the SAA application generator interface are portable across all the SAA execution environments.

The design challenge

The ease of use of the CSP/AD definition facility encourages developers to create applications with little effort or no prior design experience. It is easy to iteratively define, prototype, and redefine applications. Unfortunately this can result in poorly designed applications.

CSP/AD enables application reusability if the application designer and the application program designer have reusability as a goal. Reusability usually results in high development productivity. In order to obtain reusability, the designer must understand and apply good design principles. A well-designed application uses sound structured analysis and design methods, meets the application user's requirements with high quality, and is easy to maintain.

This paper describes the requirements for designing applications that are developed with CSP/AD. First, the design of data is briefly described and a method is suggested. Next, some application design methods are described and the CSP/AD support is defined. Finally, the specific CSP/AD application program design requirements are presented.

Data design

Data are defined as *records* (data structures) in CSP/AD. Records can be defined for flat files (sequential, indexed, or relative record access), hierarchical database segments, and relational database tables. Records are composed of elements called *data items.*

CSP/AD supports both local and global data item definitions.

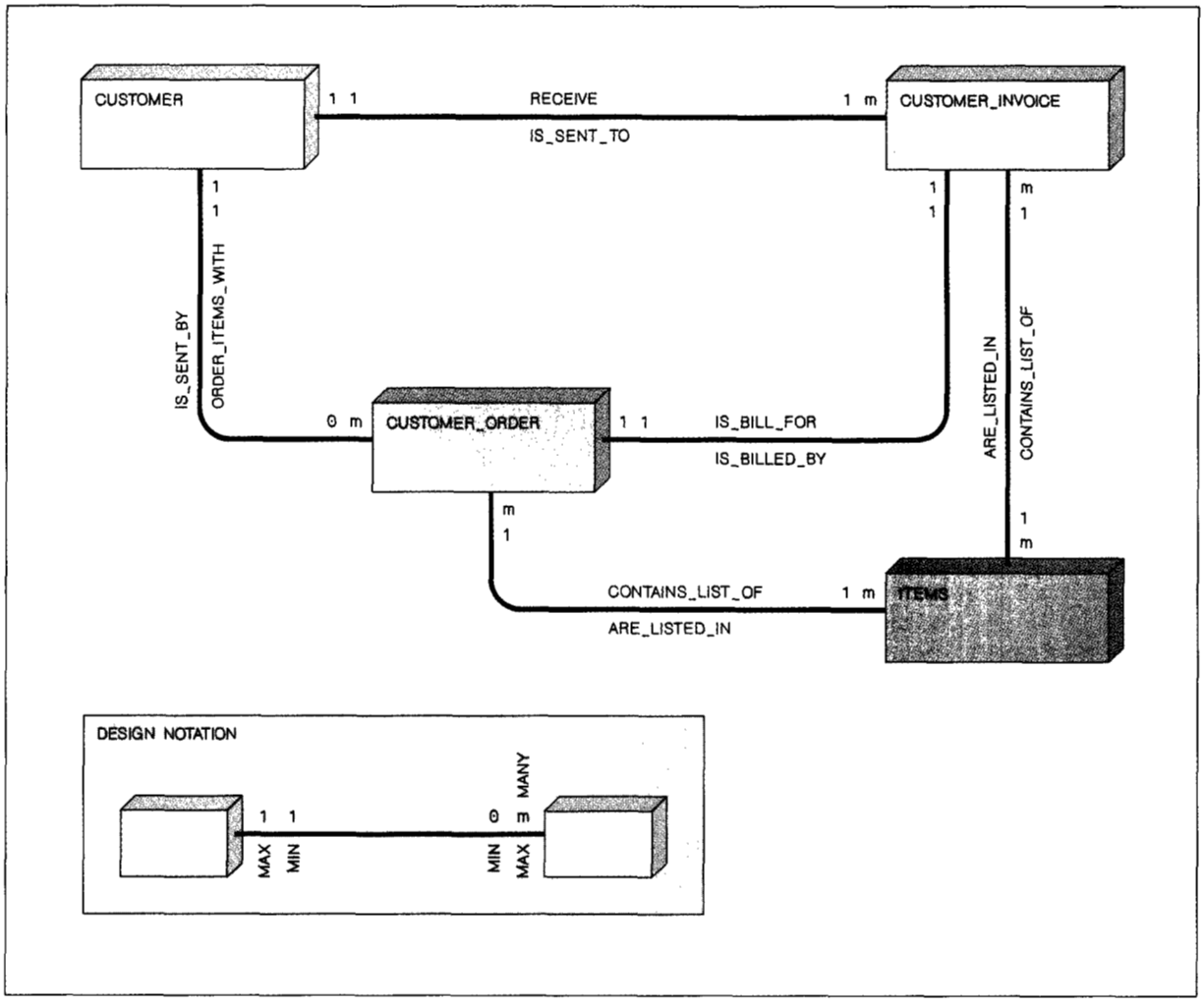
CSP/AD supports both local and global data item definitions. The developer can have the benefit of globally available data item characteristics (data type, length, decimal positions, and description) for data items that must be consistent in all of their CSP/AD application programs or data item characteristics that are used only locally in the defined record.

Although data structures with substructures and arrays are supported, it is recommended that the data design be targeted for relations. A *relation* is a relational term for a normalized two-dimensional table (rows and columns) of data elements with the following characteristics:

- There are no duplicate rows.
- The rows are not ordered.
- The columns are not ordered.
- All elements are single valued.

The data may be analyzed and the design refined until all the relations are normalized. *Normalized* is a relational term that refers to data that have no repeating element groups, i.e., none of the elements are sets (this is *first normal form*). This allows and encourages good functional modular design of application programs—that is, related functions are confined in a single unit. The definition of processing for a single relation should be defined in one process or process hierarchy. It should not include processing for unrelated functions or relations. Detailed knowledge about the internal processing is not necessary to access the function. Well-defined interfaces are provided for access to the function. This is a good

Figure 1 Entity-relationship diagram data design

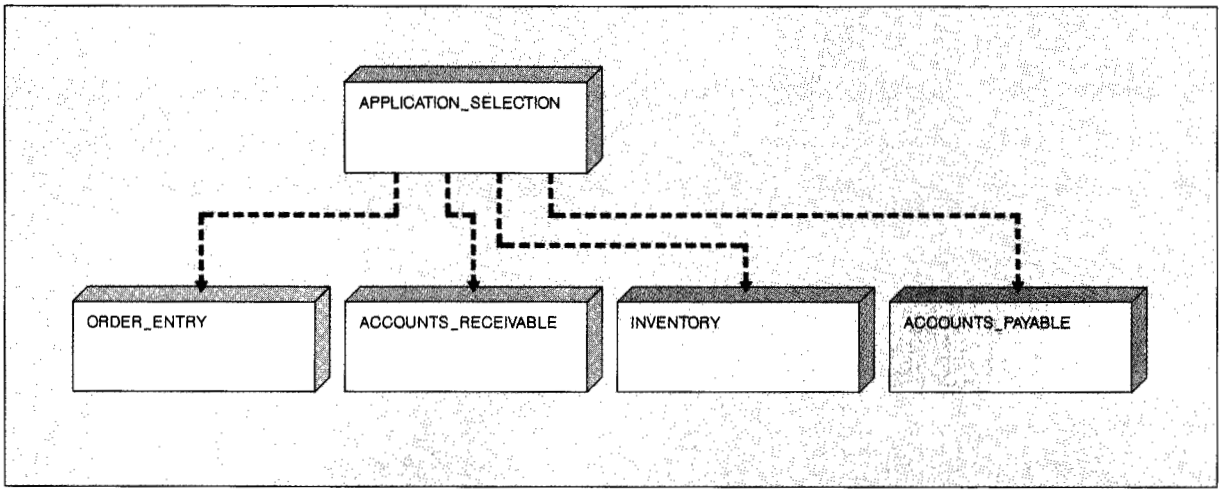


modular structure which allows reusability and simplifies maintenance. These modular units are iterated to process sets of the relations, and combined with other functional modules to create structured application programs.

Entity-relationship (ER) diagraming is a useful technique for data analysis and design.[12] See the example in Figure 1. Relational database design methods promote the use of ER. This ER design can directly reflect the user's view of data because the entities may define items that the user recognizes. An *entity* is defined as a person, place, thing, or event, such as CUSTOMER and CUSTOMER_INVOICE. The user also recognizes the *relationships* that exist between items; for example, CUSTOMER_INVOICE IS_SENT_TO CUSTOMER. In the example, the rectangles represent entities, the labeled lines connecting the entities are relationships. The notations on the line adjacent to an entity specify the minimum and maximum number of entities of that type that can participate in the stated relationship with one of the entities at the opposite end of the line. For example, a CUSTOMER_INVOICE IS_SENT_TO one and only one CUSTOMER but a CUSTOMER may receive one or many CUSTOMER_INVOICES.

The ER diagram can be viewed as a definition of data structures or relations (tables). The entities and re-

Figure 2 Application program transfer design example



lationships can be logical relations whose attributes (columns) are used in process and function design. The relation and its attributes can be associated with screens, reports, and parameters in the function design. When used for database design, the relations may become Data Language/One (DL/I) segments or DATABASE 2™ (DB2™) tables. These are represented by CSP/AD records.

Application design

The application programs may communicate information through a database or file, or through parameters passed from calling programs or working storage received from transferring programs. CSP/AD supports application program transfer and application program call designs.

Application program transfer. The application program transfer design includes application control and data transfer. An application program transfer passes control to another program without returning control to the transferring program. The transferring program is terminated. During the transfer, data may be passed.

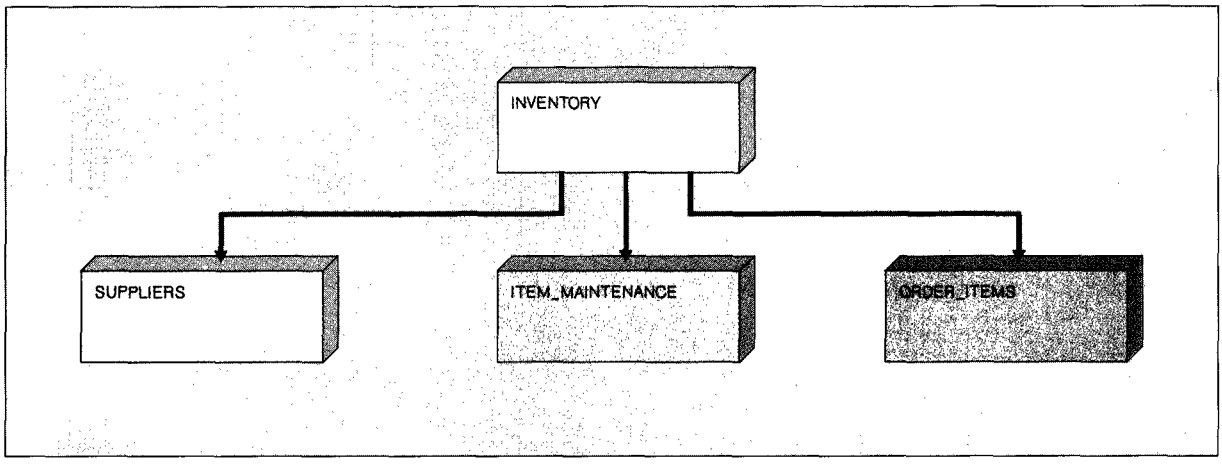
A module structure chart may be used for designing application program networks. Figure 2 is an example using a module structure chart for an application program transfer design. A module, or application program, is represented by a rectangle with a module name in the rectangle. A connection between modules is represented by an arrow between the rectangles. The dotted lines indicate an asynchronous call (transfer), the arrows indicate the direction of program control flow. These connections may be only potential connections that occur based on the satisfaction of some conditions. Multiple references in one module to another are usually not represented. The names of the data or parameters that are shared between the modules may be indicated on the chart. The APPLICATION_SELECTION program is the controlling module. The INVENTORY program has design interfaces that are shown in Figure 3.

Application programs use the CSP/AD asynchronous transfer statements (XFER and DXFR) to implement the design. Data may be passed to the other program during the transfer as an argument of the transfer statement. The argument that is transferred may be defined as a working storage record (containing one or more data structures) or may be defined as a map (containing variable data fields).

CSP/AD applications may execute as segmented mode transactions in Customer Information Control System for Virtual Storage (CICS/VS) and Information Management System for Virtual Storage (IMS/VS). In CICS/VS, this execution mode is called pseudoconversational. In IMS/VS, it is called conversational or nonconversational, depending upon whether the scratch pad area is used or not.

Application program call. An application program call design also involves control and data passing, but control is passed and returned, and data may be passed and returned. An application program call

Figure 3 Application program call design example



passes control to another program and receives control again when the called program returns (terminates). When the program calls involve more than one level, the resulting design forms a call hierarchy.

The module structure chart may be used to design a call hierarchy. Figure 3 is an example of an application program call design. In this example, the solid connecting lines indicate a synchronous call (control is returned). Application programs use the CSP/AD synchronous call statement (CALL) to implement the design. Data may be received by the called program and returned to the calling program. These data are defined in the called program as parameters and in the calling program as arguments on the call statement. The design requirements for parameters (arguments) are: (1) There may be single elements or data structures and (2) arguments must match parameters in number, order, structure, data type, and length.

Application program design

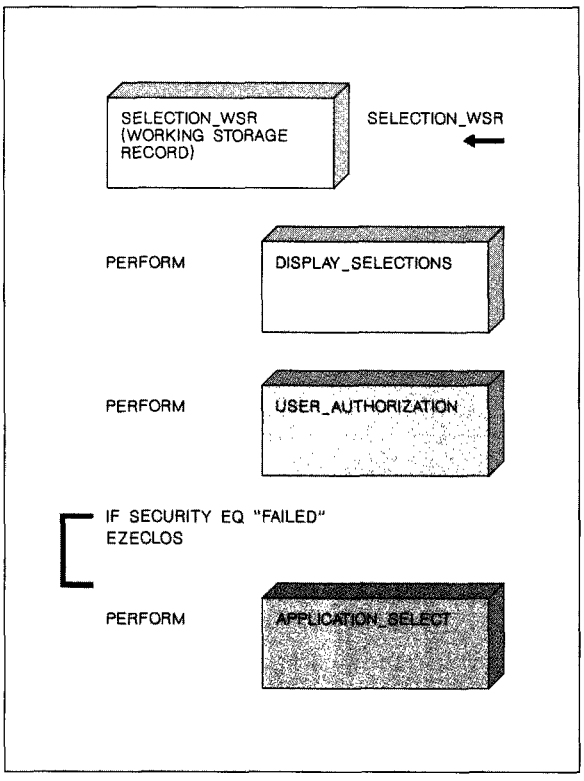
A key concept in software engineering using structured analysis and design methods is *cohesion*, also referred to as "modular strength," "binding," and "functionality." Cohesion is the degree of functional relatedness of processing elements within a single module (Chapter 7 of Reference 6). The CSP/AD application program design has a highly modular structure which facilitates design for high cohesion. The primary subordinate unit of a CSP/AD application program is a *process*. This is similar to a source program procedure or paragraph but is more restricted, as discussed in a following section on process design. There are no parameters or local variables for a process. The scope for access of data elements is global within an application program—that is, definition of variables that may be known only within a process is not supported by CSP/AD.

The CSP/AD application program structure consists of a top-level application program flow logic, and lower levels of processes invoked by the processes of the previous level, i.e., a process hierarchy. This CSP/AD application program structure may be designed using data flow diagrams, module structure charts, and action diagrams.

Program flow logic. The top level consists of a list of main processes and application flow processing statements defined for each main process. The main processes and associated application flow processing statements are the elements used to implement the top-level or main program design.

An action diagram may be used to design this application program logic. In Figure 4, the SELECTION_WSR (working storage record) definition is followed by the main processes, represented by the PERFORM and the labeled rectangle. The main processes are listed in default execution order. The first process in the list is always executed when the application is invoked. A set of application flow statements may be defined for execution following the completion of the execution of each main process. This is repre-

Figure 4 Action diagram example of program flow logic



sented in the figure by the bracketed CSP/AD conditional IF statement. The application flow processing statements determine which main process to execute next. Flow statements, i.e., conditional processing statements, are used to alter the default sequence of execution of the main processes. This allows considerable flexibility in the program design.

If a structured program design is preferred (and it should be), the developer must utilize structured program principles in this phase of the program design. A process sequence is good program structure for functions that are executed once in a specified order for each invocation of the program. For example, if it is assumed that security or user authorization is one of the main processes in the list, the flow statements may be used to bypass subsequent processes and terminate the application if the user does not have valid authorization. The principle of good structured programming described here is: Main process execution sequence should only be altered if the progression is forward in the list of main processes.

Process hierarchy. Each main process may be the root process of a process hierarchy. A main process may invoke other processes which in turn may invoke other processes. When each process below the main process in the hierarchy completes execu-

Figure 5 Data flow diagram example of application program design (VALIDATE_CUSTOMER)

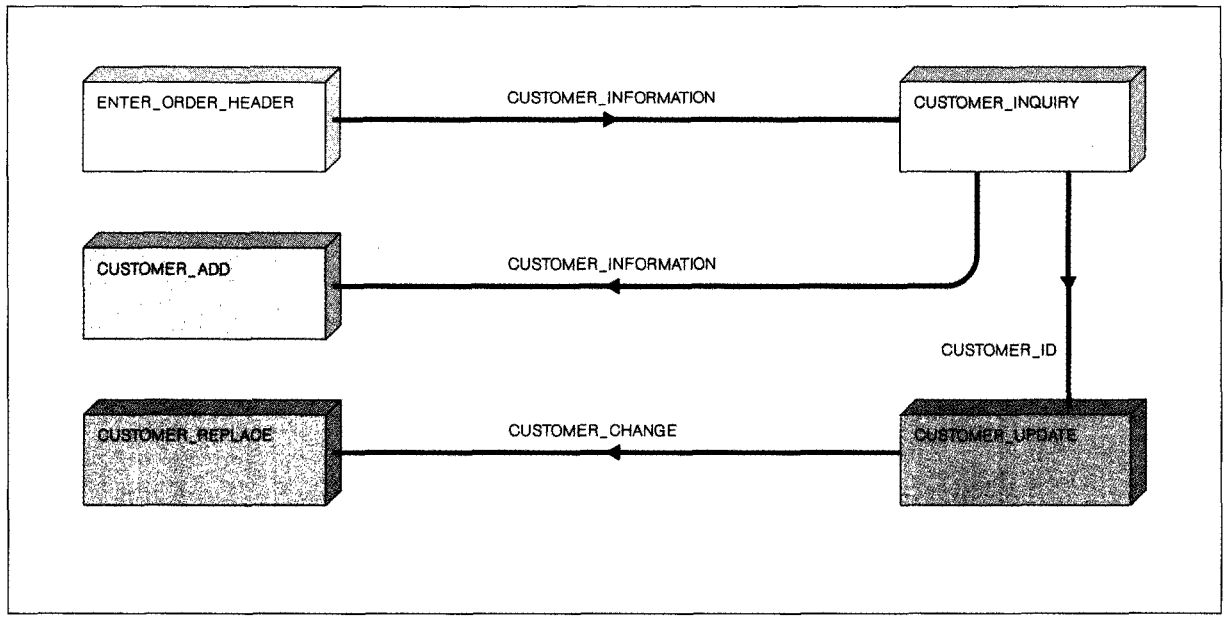
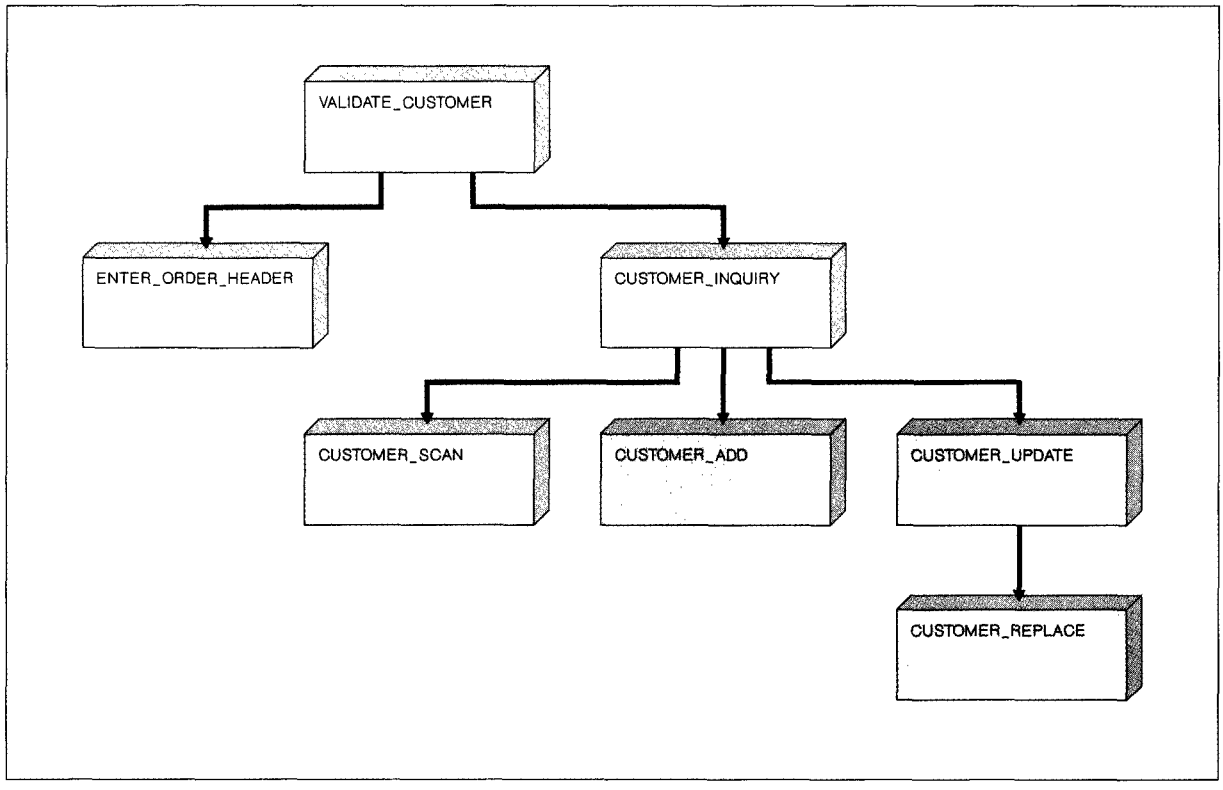


Figure 6 Module structure chart example of application program design



tion, control returns to the invoking process. This process hierarchy structure is implemented with the CSP/AD synchronous process invocation statement (PERFORM).

The data flow diagram can be used to design the process hierarchy of the application program. Each of the main processes is refined through one or more levels of data flow diagrams. The lowest level contains processes that define a single function and may perform only one input/output operation, as described in the following section on process design. Figure 5 is a data flow diagram for the VALIDATE_CUSTOMER function. The rectangles are processes and the lines connecting the processes name and indicate the direction of flow for the data.

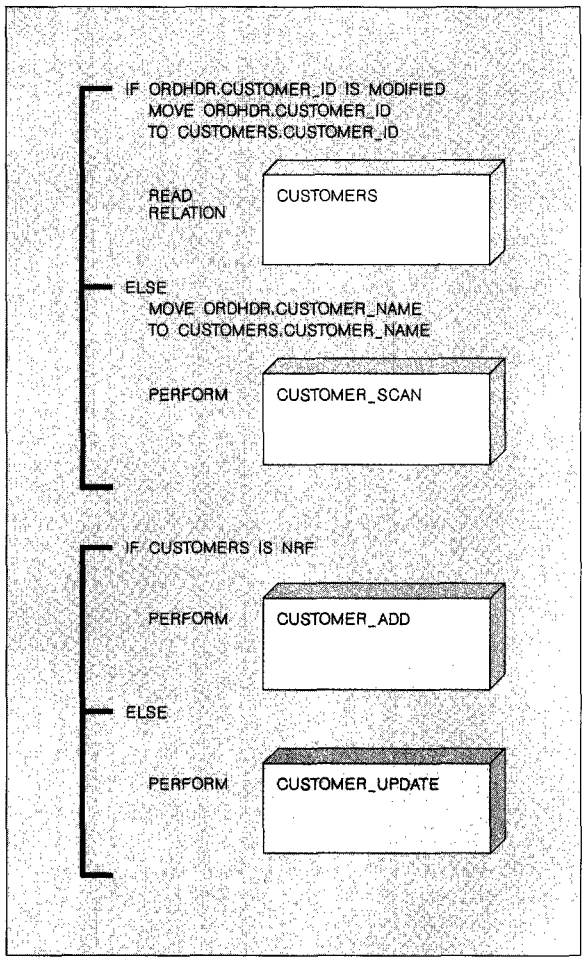
Each data flow diagram level may be transformed into a program structure chart. (See Chapter 10 of Reference 7.) Figure 6 represents the resulting process hierarchy design for the VALIDATE_CUSTOMER function.

Process design. A *process* is a sequence of CSP/AD statements consisting of three optional parts: before I/O processing, an I/O option, and after I/O processing. The action diagram, as shown in Figure 7, may be used to design the logic of a process. The READ RELATION is the I/O option in the figure.

A CSP/AD application program I/O operation is referred to as the *process option.* Process options are provided to display screens (DISPLAY and CONVERSE) which have a map definition as the object of the option, to print forms (DISPLAY) which also use a map definition as the object, and to perform database and file I/O (e.g., INQUIRY, ADD, UPDATE, REPLACE, DELETE) which have a record definition as the object.

If the design is a segmented transaction, each converse of a map divides the application into logical segments. Database locks and file positions are not maintained across the converse of a segmented application. The proper process design for data update follows:

Figure 7 Action diagram example of process design



- Read the database or file information (INQUIRY).
- Save the information in working storage.
- Display the information (CONVERSE).
- Reread the information with a lock (UPDATE).
- Compare the reread information with the saved information.
- If the information has not changed, update the information with the modifications from the displayed map and write the updated information (REPLACE).
- Otherwise, redisplay the changed information with an appropriate message to the user (CONVERSE).

Good process design requires modular design with each process performing a single function. The advantages of decomposition of the design to this level are higher reusability, better maintainability, and extensibility. It also provides a means for estimating and measuring application development and maintenance by using function point analysis.

Summary

Recommended designs and methods for CSP/AD application programs include: data design using entity-relationship diagrams and relational principles, application program transfer and call designs using module structure charts, and structured application program design using data flow diagrams, module structure charts, and action diagram techniques.

An application must meet the user's requirements and have high quality and good maintainability. A well-designed application is obtained by using proven principles of structured analysis, structured design, and structured programming. An understanding of these principles, and the CSP/AD-supported application definition constructs, is necessary for the application designer.

Cited references

1. *Cross System Product/Application Development and Cross System Product/Application Execution, General Information,* GH23-0500, IBM Corporation; available through IBM branch offices.
2. *Cross System Product/Application Development User's Guide,* GH23-0501, IBM Corporation; available through IBM branch offices.
3. *Cross System Product/Application Development Sample Applications Guide,* GH23-6428, IBM Corporation; available through IBM branch offices.
4. W. K. Haynes, M. E. Dewell, and P. J. Herman, "The Cross System Product Application Generator: An Evolution," *IBM Systems Journal* **27,** No. 3, 384–390 (1988).
5. C. Gane and T. Sarson, *Structured Systems Analysis: Tools and Techniques,* Prentice-Hall, Englewood Cliffs, NJ (1979).
6. E. Yourdon and L. L. Constantine, *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design,* Prentice-Hall, Englewood Cliffs, NJ (1979).
7. R. C. Linger, H. D. Mills, and B. I. Witt, *Structured Programming: Theory and Practice,* Addison-Wesley Publishing Company, Inc., Reading, MA (1979).
8. E. F. Codd, "A Relational Model of Data for Large Shared Data Banks," *Communications of the ACM* **13,** No. 6, 377–387 (June 1970).
9. E. F. Codd, "Further Normalization of the Data Base Relational Model," *Courant Computer Science Symposia, Data Base Systems* (Volume 6), Prentice-Hall, Englewood Cliffs, NJ (1972).
10. V. J. Mercurio, B. F. Meyers, A. M. Nisbet, and G. Radin, "AD/Cycle Strategy and Architecture," *IBM Systems Journal* **29,** No. 2, 170–188 (1990, this issue).
11. *Systems Application Architecture Common Programming Interface Application Generator Reference,* SC26-4355, IBM Corporation; available through IBM branch offices.

12. P. P.-S. Chen, "The Entity-Relationship Model—Toward a Unified View of Data," *ACM Transactions on Database Systems* **1**, No. 1, 9–36 (March 1976).

M. Eugene Dewell *IBM Programming Systems, P.O. Box 60000, Cary, North Carolina 27512-9968.* Mr. Dewell is a senior programmer in the Advanced Design and Strategy department where he is involved with the design of the IBM Cross System Product application generator. He attended the University of Virginia where he received his B.S. degree in biology. Mr. Dewell joined IBM in 1967 as a systems engineer at the Indianapolis marketing branch office. In 1972 Mr. Dewell was the Systems Engineer Grand Award winner for the Midwestern Marketing Region. He has been a member of seven IBM systems engineer symposiums. In 1974 he moved to Raleigh to work in software development on a project that resulted in the extended telecommunications module (EXTM) product (a low-entry CICS-based communications product for support of IBM finance and point-of-sale terminals for the new synchronous data link control [SDLC] communications). Mr. Dewell was one of the original designers and developers of the current Cross System Product application generator. He has been continuously involved in design and development of various phases of Cross System Product since 1977. For the last five years he has focused on Cross System Product advanced design requirements. Mr. Dewell is a coauthor of "The Cross System Product Application Generator: An Evolution," which appeared in the *IBM Systems Journal* in 1988 (see Reference 4).

Reprint Order No. G321-5398.