

TRAINING & REFERENCE

murach's
CICS
for the COBOL
programmer

Raul Menendez
Doug Lowe

Chapter 20
(Excerpt)



Mike Murach & Associates

Toll-free: 1-800-221-5528 • Fax: 559-440-0963

murachbooks@murach.com • www.murach.com

Copyright © 2004 Mike Murach & Associates. All rights reserved.

How to design, code, and test a modular CICS program

Traditionally, CICS programs were written so that a single source program performed all of the required functions. Those are the types of programs you've seen so far in this book. Today, however, it's more and more common to divide a program into two program modules: one that performs all of the functions for interacting with the user, and one that performs all of the business operations. That way, the two programs can run on different systems in a distributed environment. In addition, the programs can be written in different languages, and they can run on different platforms.

In this excerpt from chapter 20 of *Murach's CICS for the COBOL Programmer*, you'll get an overview of how this modular design technique works. Then, the rest of the chapter in the book itself explains how to design, code, and test modular programs. To find out more about this book or to get a copy for yourself or your company, please go to www.murach.com.

Program design considerations	550
Traditional program design techniques	550
Modular program design techniques	552
How to identify the presentation logic and the business logic	554
How to design a modular CICS program	556
The program overview for the customer maintenance program	556
How to create an event/response chart	558
How to create a structure chart for the presentation logic	560
The complete structure chart for the presentation logic	562
How to create a structure chart for the business logic	564
How to code and test a modular CICS program	566
How to code the communication area	566
How to code the top-level module of the business logic program	568
How to test the program	570
The customer maintenance program	572
The COBOL code for the presentation logic	572
The COBOL code for the business logic	584
Perspective	592

Program design considerations

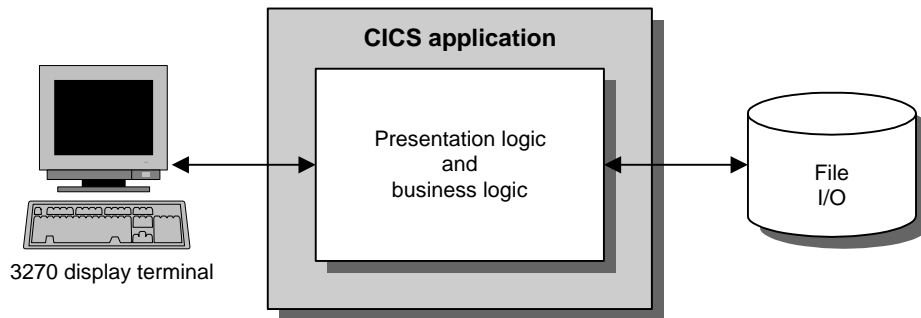
Before you can design a modular program, you need to understand the concept behind this design technique. The easiest way to explain this concept is to compare it to the traditional design concept. That's what I'll do in the next two topics. Then, I'll explain how you can separate the logic of a program into two program modules.

Traditional program design techniques

The diagram in figure 20-1 illustrates how a program works using traditional program design. As you can see, both the *presentation logic*—the logic that handles the program's interaction with the user—and the *business logic*—the logic that handles the processing of the data that's received from the user—are contained in a single program module. The module works in conjunction with BMS and CICS terminal control to interact with the user at a terminal. And it works in conjunction with CICS file control to access data on disk.

The biggest advantage of this design is that, because all of the code is contained in a single module, it's easy to implement. The biggest disadvantage is that it is inflexible. Because the entire program runs under the control of CICS, it can use only commands and statements that CICS recognizes. In contrast, if you separate the presentation logic from the business logic, the business logic can still be written to run under CICS, but the presentation logic can be written in another language to run in another environment.

Traditional program design



Description

- Traditional CICS programs consist of a single program module that handles both the interaction with the user at a terminal (the *presentation logic*) and the processing of data (the *business logic*).
- Both the presentation logic and the business logic are implemented using COBOL in conjunction with CICS commands, and the user interface is implemented using BMS.
- Although programs designed using this technique are easy to implement, they are inflexible because they run entirely under the control of CICS.

Figure 20-1 Traditional program design techniques

Modular program design techniques

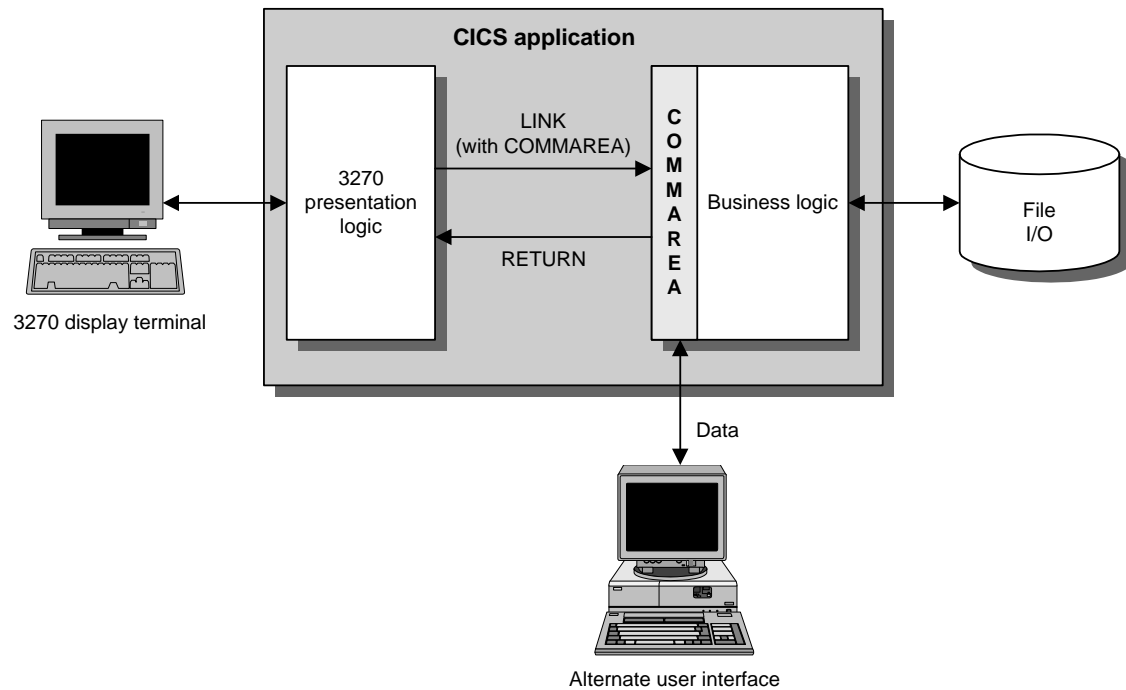
The diagram in figure 20-2 illustrates how modular CICS programs are being designed. Here, the 3270 presentation logic and the business logic are coded in separate modules that interact with each other. Notice that only the module that contains the presentation logic can interact with the user, and only the module that contains the business logic can access data.

The biggest advantage of this design technique is that you don't have to implement the user interface using BMS and CICS terminal control, and you don't have to implement the presentation logic in COBOL and CICS. Instead, you can develop an alternate user interface that handles all the presentation logic and terminal I/O using modern languages and tools like Visual Basic and Java. That's ideal for a client/server environment where the clients are PCs and the server is an OS/390 system. In that case, a tool like Visual Basic can be used to develop a more robust and user-friendly interface than can be developed in CICS, and it can be done more quickly and easily. This design technique is also critical to the development of Web-based applications, since the presentation logic for these applications can't be implemented in CICS.

But even shops that are still developing programs entirely in COBOL and CICS can benefit from modular program design. For example, programs that will run in a distributed environment like the ones described in chapter 19 can be designed so that the presentation logic will run on one CICS system, and the business logic will run on another. In addition, modular design can be used to prepare for client/server or Web-based applications in the future. That way, when the time comes, the presentation logic can be rewritten in the appropriate language, while the business logic can continue to run without change.

Unfortunately, separating the presentation and business logic can complicate the code that's required to implement a program. You'll have a better idea of why that is when you see the code for the customer maintenance program that's written using this technique later in this chapter. For now, just realize that the interface that allows the two modules to interact with each other is critical. For a program whose presentation logic and business logic are both written using COBOL and CICS, that interface is implemented using a LINK command in the presentation logic program that invokes the business logic program and passes data to it through the communication area. If the presentation logic is written in another language, however, additional facilities are required to implement the interface. You learned about some of those facilities in chapter 19, and you'll learn about others in chapter 21. Even with this complication, though, I believe that you should code the presentation and business logic in separate modules for all new CICS programs you develop because of the added flexibility it gives you.

Modular program design



Description

- Modular CICS programs separate the presentation logic from the business logic. That way, the presentation logic can be implemented using something other than COBOL and CICS, and the user interface can be implemented using something other than BMS.
- With this technique, only the presentation logic can perform terminal I/O, and only the business logic can perform file I/O.
- Because the business logic is independent of the presentation logic, it can be used with presentation logic that's written in any language and that can run on any platform, as long as the appropriate interface is provided between the two.
- The interface for a program whose presentation and business logic are both developed using COBOL and CICS is implemented using a LINK command and the communication area. If the presentation logic is developed using another language, more sophisticated facilities are required.
- The technique of separating the presentation and business logic facilitates the development of client/server applications and is critical to the development of Web-based applications.

Figure 20-2 Modular program design techniques

How to identify the presentation logic and the business logic

One of the tricks of separating the presentation and business logic is deciding what code goes in what program. To help you with that, figure 20-3 presents some guidelines you can follow.

To start, the presentation logic should always include the high-level logic for the program. That makes sense if you remember that a pseudo-conversational program performs functions based on user actions, and only the module that contains the presentation logic can interact with the user. Because the presentation logic drives the execution of the program, this program can be referred to as the *driver program*. If this program is written in CICS, it's the program that CICS will start when the user presses an attention key at the terminal, and it's the program that will end after it sends information to the terminal. That means that this program must also maintain the required information in the communication area between program executions.

In addition to controlling the execution of the program and interacting with the user, the presentation logic should handle any basic editing of the data the user enters. For example, it might check to be sure that a field that should contain a quantity is numeric or that a date entered by the user contains a valid month, day, and year. If it determines that the data should be processed further, it can then link to the business logic to perform that processing.

The business logic can do additional editing based on business rules, like checking that the date entered for an invoice isn't in the future or that a zip code is valid for the associated state. The business logic should also perform all calculations and handle all file I/O. It's also responsible for maintaining data integrity. That means it must ensure that two users can't update the same data at the same time, and that related updates are committed or rolled back as necessary. In short, the business logic should handle all processing that isn't directly related to interacting with the user.

Processing typically done by presentation logic

- Because CICS programs are driven by user events, the presentation logic should contain the high-level logic for the program. The presentation logic can be referred to as the *driver program* since it drives the execution of the program.
- The presentation logic must determine what processing is done based on user input and the context in which it's received.
- The presentation logic should handle all screen interactions with the user.
- The presentation logic should handle preliminary editing of the input that's entered by the user. That includes making sure that required entries are made, that entries are numeric when required, and that a valid key has been pressed.
- The presentation logic should link to the business logic whenever business processing is required.
- The presentation logic should maintain a copy of any data it requires between executions in a pseudo-conversational session. It should also maintain a copy of any data that's required by the business logic between its executions.

Processing typically done by business logic

- The processing done by the business logic should be based on information sent to it by the presentation logic.
- The business logic should handle all file processing.
- The business logic should ensure that data integrity is maintained.
- The business logic should handle all editing of data that depends on business rules. That includes checking that a value falls within a particular range, making sure that a record with a corresponding key value does or does not exist, and making sure that related fields agree with one another.
- The business logic should perform all calculations.

Note

- The presentation logic should *never* include any file I/O statements, and the business logic should *never* include any terminal I/O statements.