Chapter 4: *Tools of Modern Systems Analysis*

"Nature has . . . some sort of arithmetical-geometrical coordinate system, because nature has all kinds of models. What we experience of nature is in models, and all of nature's models are so beautiful. It struck me that nature's system must be a real beauty, because in chemistry we find that the associations are always in beautiful whole numbers — there are no fractions."

— R. Buckminster Fuller
"In the Outlaw Area: profile by Calvin Tompkins," the *New Yorker*, January 8, 1966

In this chapter, you will learn:

1. What a systems analyst uses modeling tools for;
2. The nature and components of a dataflow diagram;
3. The components of a data dictionary;
4. How to model stored data and data relationships;
5. The components of a process specification;
6. How to model the time-dependent behavior of a system; and
7. How to model the structure of a computer program.

Much of the work that you will do as a systems analyst involves modeling the system that the user wants. As we will see in this chapter, and in more detail in Part II, there are many different kinds of models that we can build — just as there are many different models that an architect can build of a proposed new house. The systems analysis models discussed in this book are, for the most part, *paper* models of the system-to-be; that is, abstract representations of what will eventually become a combination of computer hardware and computer software.

The term model may sound rather formal and frightening to you, but it represents a concept that you have used for much of your life. Consider the following kinds of models:

* *Maps*: two-dimensional models of the world we live in.
* *Globes*: three-dimensional models of the world we live in.
* *Flowcharts*: schematic representations of the decisions and sequence of activities for carrying out some procedure.
* *Architect's drawings*: schematic representations of a building or a bridge.
* *Musical scores*: graphic/text representations of the musical notes and tempo of a piece of music.

Though you may not know how to read the architectural model shown in Figure 4.1, the concept of such a model ought not to frighten you; it is not too hard to imagine that you could learn how to read and understand such a model, even if you never intended to create one yourself. Similarly, you probably don't yet know how to read many of the models used by systems analysts, but you will know how to read them and create them by the end of this book. The users you work with will certainly be able to read the models (with a little initial assistance) and they may even be able to create them.

Why should we build models? Why not just build the system itself? The answer is that we can construct models in such a way as to highlight, or emphasize, certain critical features of a system, while simultaneously de-emphasizing other aspects of the system. This allows us to communicate with the user in a focused way, without being distracted by issues and system features that are irrelevant to us. And if we learn that our understanding of the user's requirements was incorrect (or that the user changed his mind about his requirements), we can change the model *or throw it away and build a new model, if necessary*. The

alternative is to carry on some initial discussions with the user and then build the entire system; the risk, of course, is that the final product may be unacceptable, and it may be extraordinarily expensive to change at that point. **[1]**

* Thus, the systems analyst uses modeling tools to:
* Focus on important system features while downplaying less important features.
* Discuss changes and corrections to the user's requirements with low cost and minimal risk.
* Verify that the systems analyst correctly understands the user's environment and has documented it in such a way that the systems designers and programmers can build the system.

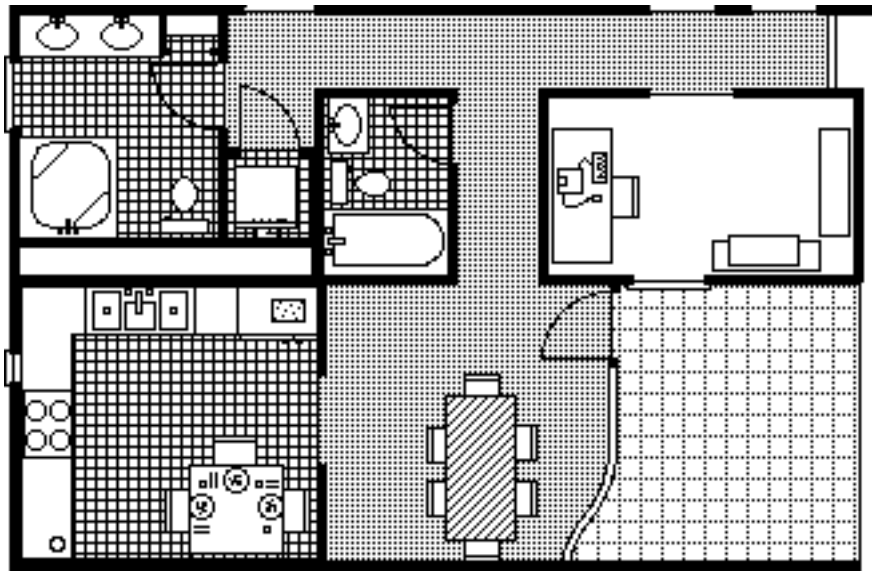

Figure 4.1: A typical architectural model

Not all modeling tools will accomplish these purposes: for example, a 500-page narrative description of the user's requirements (which is, in a crude sense, a model) could (1) thoroughly obscure all the system's features, (2) cost more to build than the system itself, and (3) fail to verify the system analyst's understanding of the user's true needs. In Chapter 8, we will explore in detail what characteristics a modeling tool must have in order to be useful to the systems analyst.

We will now introduce and briefly discuss three important systems modeling tools: the dataflow diagram, the entity-relationship diagram, and the state-transition diagram. The dataflow diagram illustrates the *functions* that the system must perform; the entity-relationship diagrams emphasize the *data relationships*, and the state-transition diagram focuses on the *time-dependent behavior* of the system. Chapters 9 to 16 explore these and other modeling tools in more detail. As we will see, all three of the major modeling tools consist of graphics (pictures) and supporting textual modeling tools. The graphics provide a vivid and easy-to-read way for the systems analyst to show the users the major *components* of the model, as well as the connections (or interfaces) between the components. The supporting textual modeling tools provide precise definitions of the *meaning* of the components and connections.

4.1 Modeling system functions: The dataflow diagram

An old adage in the systems development profession emphasizes that a data processing system involves both data and processing, and that one cannot build a successful system without considering both components. The processing aspect of a system is certainly an important aspect to model and to verify with the user. The modeling that we are carrying out can be described in a number of way:

* What functions must the system perform? What are the interactions between the functions?
* What transformations must the system carry out? What inputs are transformed into what outputs?
* What kind of work does the system do? Where does it get the information to do its work? Where does it deliver the results of its work?

The modeling tool that we use to describe the transformation of inputs into outputs is a *dataflow diagram*. A typical dataflow diagram is shown in Figure 4.2.

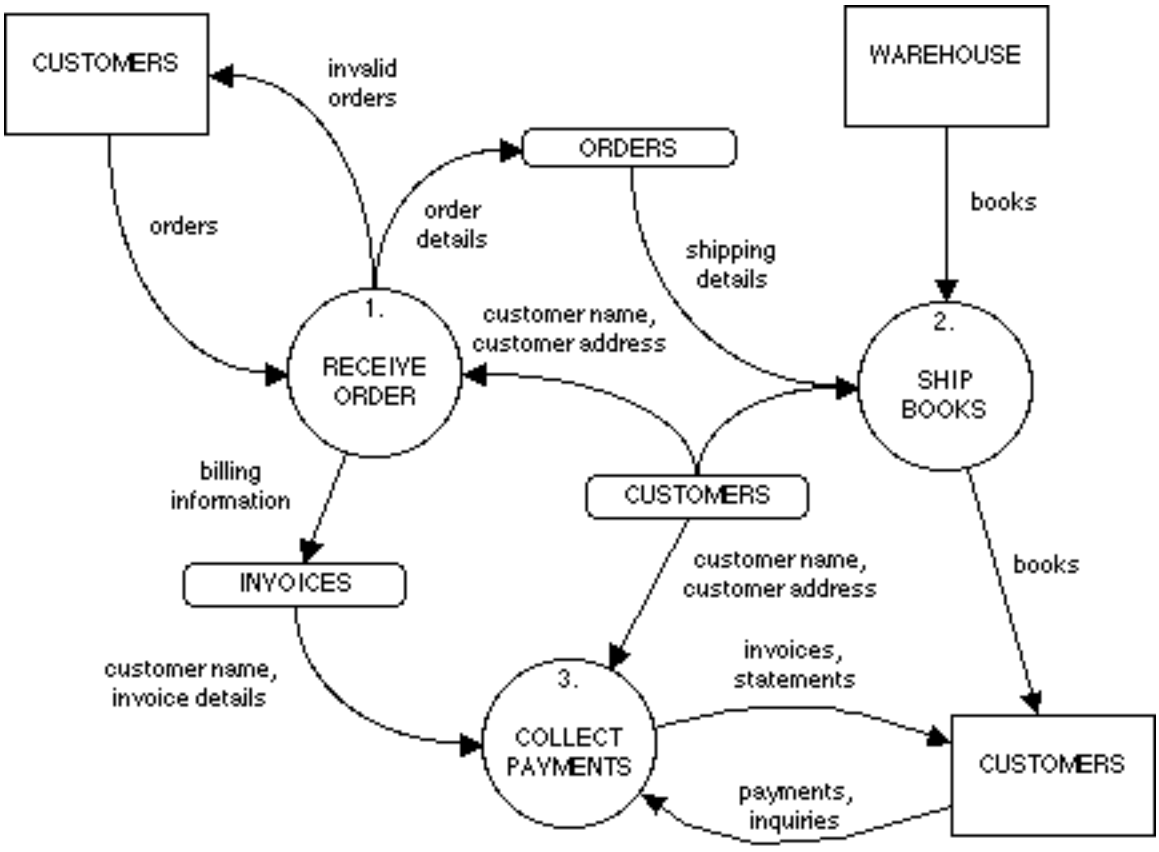

Figure 4.2: A typical dataflow diagram

Dataflow diagrams consist of processes, data stores, flows, and terminators:

* *Processes* are shown by the circles, or "bubbles," in the diagram. They represent the various individual functions that the system carries out. Functions transform inputs into outputs.

* *Flows* are shown by curved, directed arrows. They are the connections between the processes (system functions), and they represent the information that the processes require as input and/or the information they generate as output.

* *Data stores* are shown by two parallel lines, or by an ellipse. They show collections (aggregates) of data that the system must remember for a period of time. When the systems designers and programmers finish building the system, the stores will typically exist as files or databases.

* *Terminators* show the external entities with which the system communicates. Terminators are typically individuals, groups of people (e.g., another department or division within the organization), external computer systems, and external organizations.

The dataflow diagram is discussed in greater detail in Chapter 9. (In addition to processes, flows and stores, a dataflow diagram can also have *control flows*, *control processes*, and *control stores*. These are useful for modeling real-time systems and are discussed in more detail in Chapter 9.)

While the dataflow diagram provides a convenient overview of the major functional components of the system, it does not provide any detail about these components. To show the details of *what* information is transformed and *how* it is transformed, we need two supporting textual modeling tools: the data dictionary and the process specification. Figure 4.3 shows a typical *data dictionary* for the *dataflow diagram* that we saw in Figure 4.2. Similarly, Figure 4.4 shows a typical process specification for one process in the dataflow diagram that we saw in Figure 4.2.

```
name = courtesy-title + first-name + (middle-name) + last-name
courtesy-title = [Mr. | Miss | Mrs. | Ms. | Dr. | Prof. ]
first-name = {legal-character}
last-name = {legal-character}
legal-character = [A-Z | a-z | ' | - | |]
```

Figure 4.3: A typical data dictionary

1. IF the dollar amount of the invoice times the number of weeks overdue is greater than $10,000 **THEN:**
a. Give a photocopy of the invoice to the appropriate salesperson who is to call the customer.
b. Log on the back of the invoice that a copy has been given to the salesperson, with the date on which it was done.
c. Re-file the invoice in the file for examination two weeks from today.

2: OTHERWISE IF more than four overdue notices have been sent **THEN:**
a. Give a photocopy of the invoice to the appropriate salesperson to call the customer.
b. Log on the back of the invoice that a copy has been given to the salesperson, with the date on which it was done.
c. Re-file the invoice in the file to be examined one week from today.

3: OTHERWISE (the situation has not yet reached serious proportions):
a. Add 1 to the overdue notice count on the back of the invoice (if no such count has been recorded, write "overdue notice count = 1").
b. IF the invoice in the file is illegible **THEN** type a new one.
c. Send the customer a photocopy of the invoice, stamped "Nth notice: invoice overdue. Please remit immediately," where N is the value of the overdue notice count.
d. Log on the back of the invoice the date on which the Nth overdue notice was sent.
e. Re-file the invoice in the file for two weeks from today's date.

Figure 4.4: A typical process specification

There is much, *much* more to say about dataflow diagrams, data dictionaries, and process specifications; details are provided in Chapters 9 to 11. We will see, for example, that most complex systems are modeled with more than one dataflow diagram; indeed, there may be dozens or even hundreds of diagrams, arranged in hierarchical levels. And we will see that there are conventions for labeling and numbering the items in the diagram, as well as a number of guidelines and rules that help distinguish between good diagrams and poor diagrams.

4.2 Modeling stored data: The entity-relationship diagram

While the dataflow diagram is indeed a useful tool for modeling systems, it only emphasizes one major aspect of a system: its functions. The data store notation in the DFD show us the existence of one or more groups of stored data, but deliberately says very little about the details of the data.

All systems store and use information about the environment with which they interact; sometimes the information is minimal, but in most systems today it is quite complex. Not only do we want to know, in detail, what information is contained in each data store, we want to know what relationships exist *between* the data stores. This aspect of the system is not highlighted by the dataflow diagram, but is vividly portrayed by another modeling tool: the *entity-relationship* diagram. A typical entity-relationship diagram is shown in Figure 4.5.

The entity-relationship diagram consists of two major components:

1. *Object types* are shown by a rectangular box on the entity-relationship diagram. An object type represents a collection, or set, or objects (things) in the real world whose members play a role in the system being developed, can be identified uniquely, and can be described by one or more facts (attributes).

2. *Relationships* are shown by the diamond-shaped boxes on the diagram. A relationship represents a set of connections, or associations, between the object types connected by arrows to the relationship.

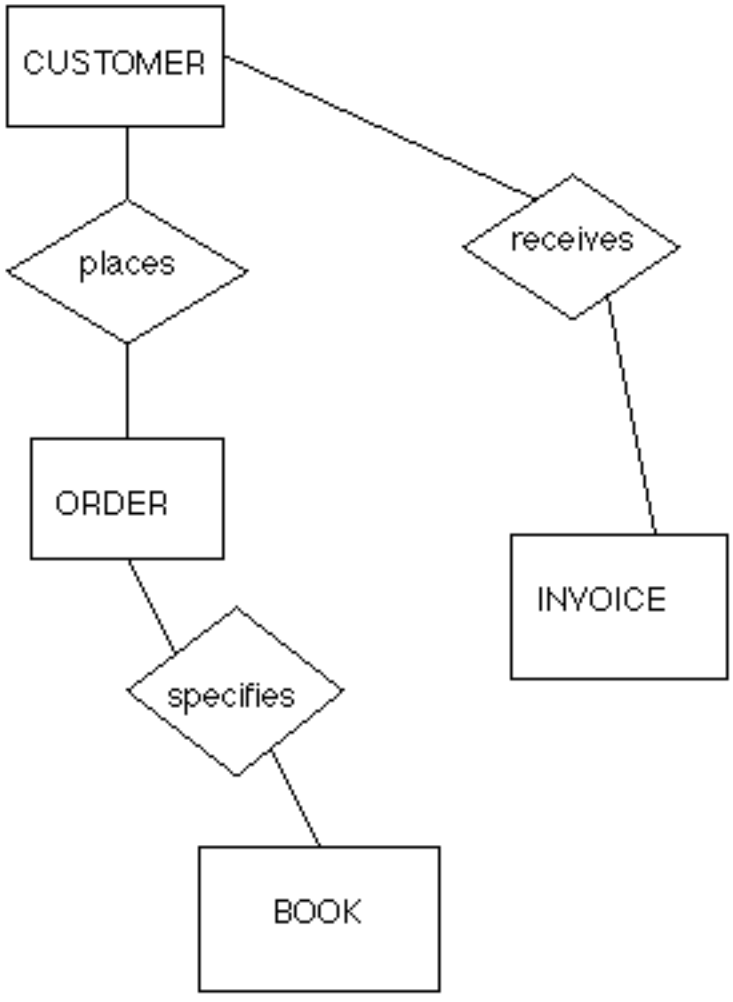

Figure 4.5: An entity-relationship diagram

As with the dataflow diagram, there is much more to say about the entity-relationship diagram; we will discuss it in much more detail in Chapter 12. We will see that there are certain specialized object types, as well as a number of guidelines for ensuring that the entity-relationship diagram is complete and consistent.

As with the dataflow diagram, we will find it necessary to augment the entity-relationship diagram with detailed textual information. The data dictionary, which we first saw in Figure 4.3, can also be used to maintain appropriate information about objects and relationships. This will be further explored in Chapters 10 and 12.

4.3 Modeling time-dependent behavior: The state-transition diagram

A third aspect of many complex systems is their time-dependent behavior, that is, the sequence in which data will be accessed and functions will be performed. For some business computer systems, this is not an important aspect to highlight, since the sequence is essentially trivial. Thus, in many batch computer systems (those which are neither on-line nor real-time), function N cannot carry out its work until it receives its required input; and its input is produced as an output of function N - 1; and so on.

However, many on-line systems and real-time systems, both in the business area and in the scientific/engineering area, have complex timing relationships that must be modeled just as carefully as the modeling of functions and data relationships. Many real-time systems, for example, must respond within a very brief period of time, perhaps only a few microseconds, to certain inputs that arrive from the external environment. And they must be prepared for various combinations and sequences of inputs to which appropriate responses must be made.

The modeling tool that we use to describe this aspect of a system's behavior is the *state-transition* diagram, sometimes abbreviated as STD. A typical diagram is shown in Figure 4.6; it models the behavior of a computer-controlled washing machine. In this diagram, the rectangular boxes represent states that the system can be in (i.e., recognizable "scenarios" or "situations"). Each state thus represents a period of time during which the system exhibits some observable behavior; the arrows connecting each rectangular box show the *state-change*, or transitions from one state to another. Associated with each state-change is one or more *conditions* (the events or circumstances that caused the change of state) and zero or more *actions* (the response, output, or activity that takes place as part of the change of state). We will examine the state transition diagram in more detail in Chapter 13.

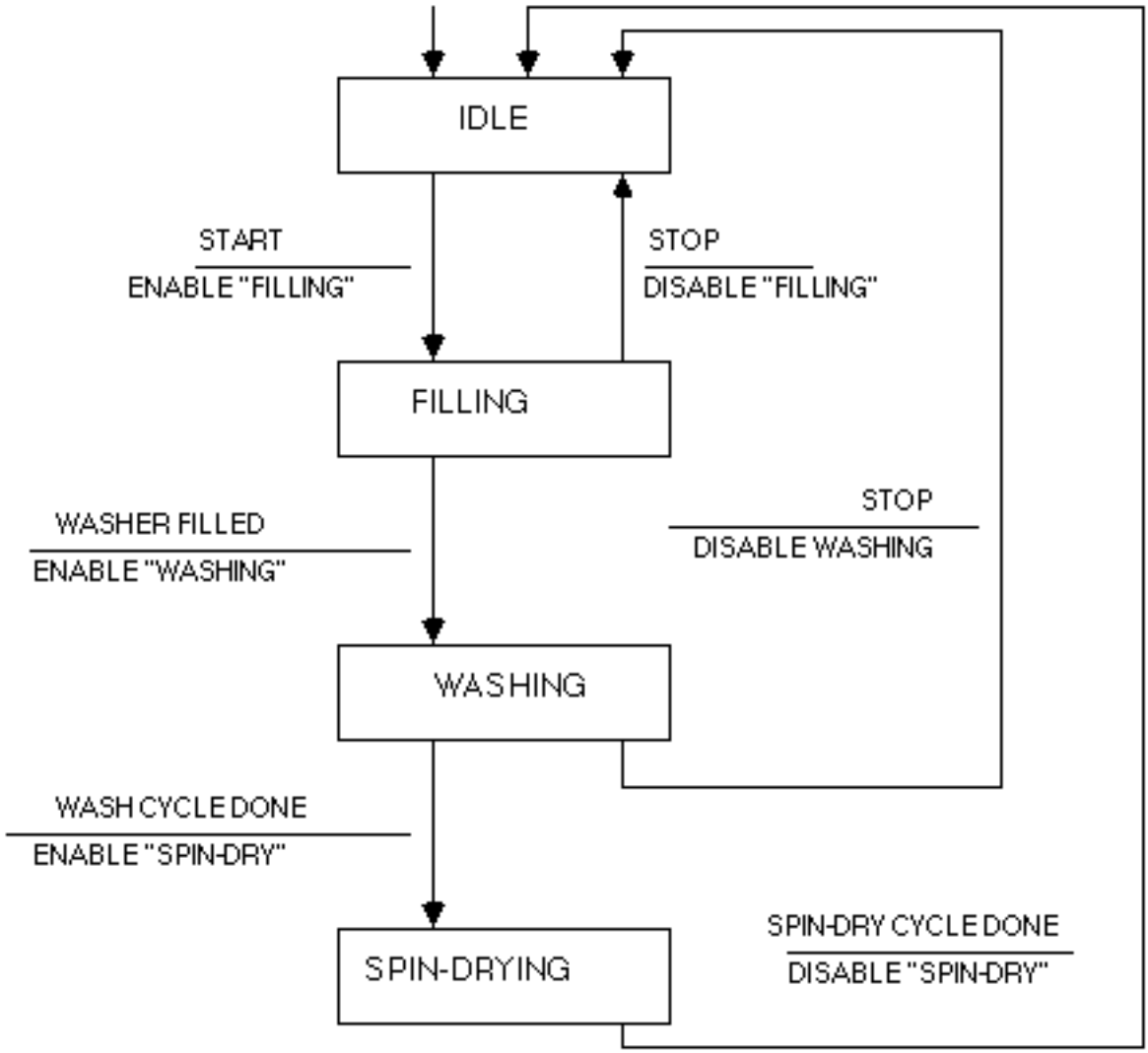

Figure 4.6: A state-transition diagram

4.4 Modeling program structure: The structure chart

Though you will not use it much as a systems analyst, you should be aware that many additional modeling tools are used during the development of a complex system. For example, the system designers will usually take the dataflow diagram, data dictionary, process specifications, entity-relationship diagrams, and state transition diagrams created by the systems analyst and use them to create a software architecture, that is, a hierarchy of modules (sometimes referred to as subroutines or procedures) to implement the system requirements. One common graphical modeling tool used to represent such a software hierarchy is a *structure chart*; a typical structure chart is shown in Figure 4.7. In this diagram, each rectangular box represents a module (e.g., a C or Pascal procedure, a COBOL paragraph or subprogram). The arrows connecting the boxes represent module invocations (e.g., subroutine calls or procedure calls). The diagram also shows the input parameters passed to each module that is invoked, and the output parameters returned by the module when it finishes its job and returns control to its caller.

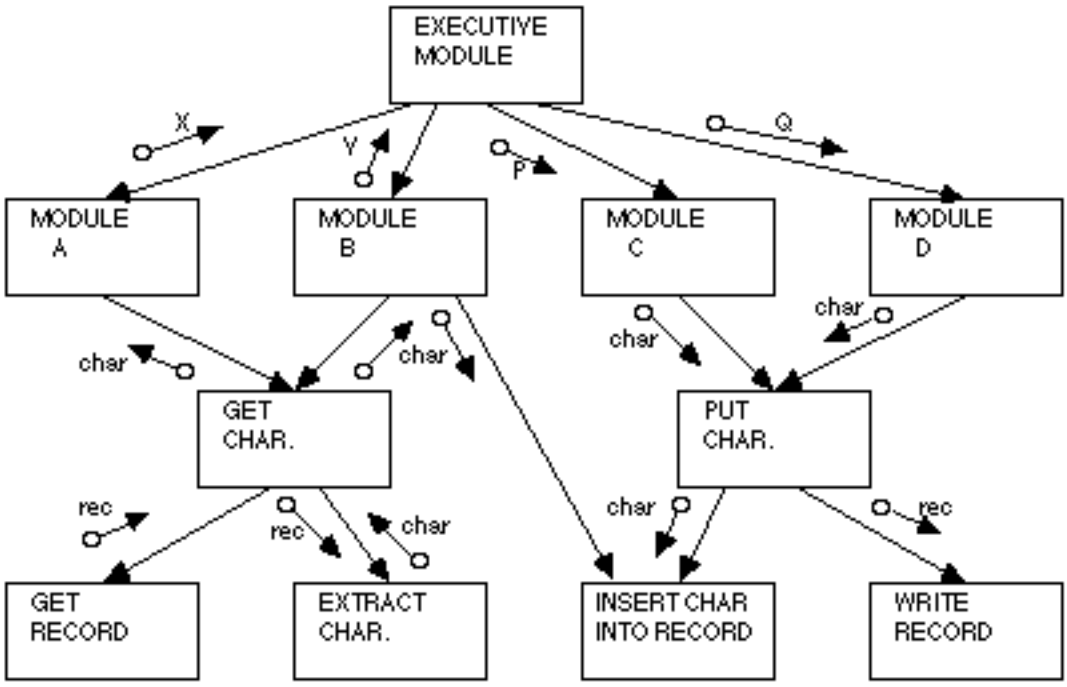

Figure 4.7: A structure chart

While the structure chart is an excellent tool for system designers, it is not the sort of model one would normally show to the user, because it models an aspect of the *implementation* of the system, rather than the underlying requirements **[2]**. We will discuss structure charts again in Chapter 22.

4.5 Relationships between the models

As you can see, each graphic model described in this chapter focuses on a different aspect of a system: the DFD illustrates the functions, the ERD highlights the data relationships, and the STD emphasizes the time-dependent behavior of the system. Because there is so much complexity in a typical system, it is helpful to study each of these aspects in isolation; on the other hand, these three views of the system must be consistent and compatible with one another.

In Chapter 14, we will examine a number of consistency rules that ensure that the three major system models, together with the detailed textual models are indeed consistent. For example, we will see that each store on the DFD must correspond to an object or a relationship in the ERD.

4.6 Summary

The models shown in this chapter are relatively simple and easy to read. Great care was taken to make them that way, for they are intended to be read and understood by users without a great deal of training or preparation. However, as we will see in Chapters 9 through 16, it requires a great deal of careful work to create the diagrams and ensure that they are complete, consistent, and accurate representations of the user requirements.

Footnotes

1. As we'll discuss in Chapter 5, another alternative is to build a *prototype* of the system, and then refine and elaborate that prototype as the user becomes more familiar with his "true" requirements. While this is indeed a popular approach in many organizations, it has one flaw: when the "final" prototype is finished, there is no model describing what it is supposed to do, or how it is organized internally. This makes life quite difficult for subsequent generations of maintenance programmers.

2. As we pointed out in Chapter 3, some users are more knowledgeable about computers than others, and some users take a more active role in a systems development project than others. If you are working with a user who is a full-time member of the project team, or perhaps even the project leader, and if the user is somewhat knowledgeable about systems design, there is no reason why you should not show him or her a structure chart. However, if the user is only interested in describing what the system has to do, he or she will probably not be interested in looking at a diagram describing the organization of COBOL subroutines that will implement those requirements.

Questions and Exercises

1. The introduction to Chapter 4 lists maps, globes, flowcharts, architectural drawings, and musical scores as examples of models. List three more examples of models in common use.

2. What is the dictionary definition of a model? Does it apply to information systems?

3. Why are models used in the development of information systems? List three reasons.

4. How would you respond if the user told you he thought models were a waste of time and that you should begin coding?

5. Do you think that a user's verbal description of his or her system requirements would be considered a model? Why or why not?

6. Why would it be useful to have more than one model for a system?

7. All the models discussed in Chapter 4 are paper models. Can you think of any other forms of models?

8. Most of the models discussed in Chapter 4 are graphical tools (i.e., pictures). What do you think are the advantages of using pictures as modeling tools?

9. Do you think graphical modeling tools are sufficient for representing the requirements of an information system? Why or why not?

10. Who should be responsible for creating the models needed to describe the requirements of an information system?

11. Should users be expected to create their own models? If so, under what circumstances?

12. What are the three main objectives of a dataflow diagram?

13. What are the four main components of a dataflow diagram?

14. Notice that the processes in a DFD are represented by circles and the terminators are represented by rectangles. Do you think this is significant? What if the processes were represented by rectangles and the terminators were represented by circles?

15. Notice that Figure 4.2 shows three different processes but does not indicate how many computers will be involved in the system. Do you think this is significant? What changes would be required if the project team decided to implement the system with one computer? With three computers?

16. Notice that Figure 4.2 shows several different dataflows between the processes, but does not indicate the physical medium that will be used to carry the data. Do you think this is significant? What if the system implementors decide to transmit data between the processes using telecommunication lines? What if they decide to transport the data from one process to another using magnetic tape?

17. What is the purpose of the data dictionary?

18. Who should be responsible for creating the data dictionary? Who should be responsible for keeping it up to date?

19. Figure 4.3 shows the data dictionary definition of a name. What do you think is the significance of the parentheses — ( ) — in that definition?

20. What is the purpose of the process specification?

Questions and Exercises (cont.)

21. How many process specifications should we expect to see in a complete specification of user requirements?

22. Who should be responsible for creating the process specification? Who should keep it up to date?

23. Note that the process specification shown in the example in the chapter looks somewhat like program coding; What do you think of the idea of using pseudocode to write process specifications? What do you think of the notion of using a real programming language — such as Ada, as many people have suggested — for program specifications? Why would a real programming language be good or bad?

24. What is the purpose of an entity-relationship diagram?

25. What are the major components of an ERD?

26. How many object types are shown in Figure 4.5?

27. How many relationships are shown in Figure 4.5?

28. Does the ERD show the reader any information about the functions being carried out by the system?

29. Does the DFD show the reader any information about the object types or relationships between object types in the system?

30. Where should we describe the details of object types and relationships shown on the ERD?

31. What is the purpose of the state transition diagram?

32. What are the components of an STD?

33. Are STDs useful for modeling batch computer systems? Why or why not?

34. What is the purpose of a structure chart?

35. What are the graphical components of a structure chart?

36. Should the user be expected to read and understand a structure chart? Should the user be expected to be able to create one?

37. Describe the relationship that exists between an ERD and a DFD.

38. Is there a relationship between a DFD and a structure chart? If so, what is it?