CHAPTER 5

PROCESS MODELING

A process model describes business processes—the activities that people do. Process models are developed for the as-is system and/or the to-be system. This chapter describes data flow diagramming, one of the most commonly used process modeling techniques.

OBJECTIVES

- Explain the rules and style guidelines for data flow diagrams.
- Describe the process used to create data flow diagrams.
- Create data flow diagrams.

CHAPTER OUTLINE

Introduction
Data Flow Diagrams
  Reading Data Flow Diagrams
  Elements of Data Flow Diagrams
  Using Data Flow Diagrams to Define Business Processes
  Process Descriptions
Creating Data Flow Diagrams
  Creating the Context Diagram
  Creating Data Flow Diagram Fragments
  Creating the Level 0 Data Flow Diagram
  Creating Level 1 Data Flow Diagrams (and Below)
  Validating Data Flow Diagrams
Applying the Concepts at Tune Source
  Creating the Context Diagram
  Creating Data Flow Diagram Fragments
  Creating the Level 0 Data Flow Diagram
  Creating Level 1 Data Flow Diagrams (and Below)
  Validating Data Flow Diagrams
Summary

INTRODUCTION

Chapters 3 and 4 discussed several requirements elicitation activities, such as interviewing and JAD, and how to clarify those requirements by developing more detailed use cases. In this chapter, we discuss how the requirements definition and use cases may be further clarified through a process model. You may have heard the expression "A picture is worth a 1,000 words." A *process model* is a graphical way of representing how a business system should operate. It illustrates the processes or activities that are performed and how data move among them. A process model can be used to document the current system (i.e., as-is system) or the new system being developed (i.e., to-be system), whether computerized or not.

Process models have been a part of structured systems analysis and design techniques for many years. Today, with use cases gaining favor due to their ability to clarify user requirements in an understandable way, you may see that organizations place less emphasis on process modeling than in the past. An organization that strives to create fully dressed use cases as depicted in Figure 4-1, for example, may not find that process models add much to their understanding of the system under development. Other organizations, however, especially those that create more casual use cases, may find process modeling to be a beneficial part of their analysis phase deliverables. We find that graphically depicting the system that will be developed in a set of well-organized diagrams is a very useful approach. Remember that our goal is to be able to employ an array of tools and techniques that will help us understand and clarify what the new system must do before the new system is actually constructed.

There are many different process modeling techniques in use today. In this chapter, we focus on one of the most commonly used techniques:[1] data flow diagramming. Data flow diagramming is a technique that diagrams the business processes and the data that pass among them. In this chapter, we first describe the basic syntax rules and illustrate how they can be used to draw simple one-page data flow diagrams (DFDs). Then we describe how to create more complex multipage diagrams.

Although the name *data flow diagram* (DFD) implies a focus on data, this is not the case. The focus is mainly on the processes or activities that are performed. Data modeling, discussed in the next chapter, presents how the data created and used by processes are organized. Process modeling—and creating DFDs in particular—is one of the most important skills needed by systems analysts.

In this chapter, we focus on *logical process models*, which are models that describe processes, without suggesting how they are conducted. When reading a logical process model, you will not be able to tell whether a process is computerized or manual, whether a piece of information is collected by paper form or via the Web, or whether information is placed in a filing cabinet or a large database. These physical details are defined during the design phase when these logical models are refined into *physical models*, which provide information that is needed to ultimately build the system. (See Chapter 10.) By focusing on logical processes first, analysts can focus on how the business should run, without being distracted by implementation details.

[1] Another commonly used process modeling technique is IDEF0. IDEF0 is used extensively throughout the U.S. Government. For more information about IDEF0, see FIPS 183: *Integration Definition for Function Modeling (IDEF0)*, Federal Information Processing Standards Publications, Washington, DC: U.S. Department of Commerce, 1993.

In this chapter, we first explain how to read DFDs and describe their basic syntax. Then we describe the process used to build DFDs that draws information from the use cases and from additional requirements information gathered from the users.

DATA FLOW DIAGRAMS

Reading Data Flow Diagrams

Figure 5-1 shows a DFD for the event we introduced in Chapter 4, that of a Lawn Chemical Applicator (LCA) requesting a lawn chemical. By examining the DFD, an analyst can understand the process by which LCAs request lawn chemicals. Take a moment to examine the diagram before reading on. How much do you understand? Before continuing, you may want to review this event's use case in the previous chapter (Figure 4-1) and the functional requirements (Figure 4-4).

Most people from Western cultures start reading diagrams from left to right, top to bottom. So, whenever possible, this is where most analysts try to make the DFD begin. The first item on the left side of Figure 5-1 is the "Lawn Chemical Applicator (LCA)" external entity, which is a rectangle that represents individual employees who must request the chemicals they will use for their lawn care assignments. This symbol has three arrows pointing away from it to rounded rectangular symbols. These arrows represent data flows and show that the external entity (LCA) provides three "bundles" of data to processes that use the data. Now look again at Figure 4-1 and you will see that these same data bundles are listed as Major Inputs in the use case, with the source listed as the LCA. Also, there are several arrows arriving at the LCA external entity from the rounded rectangles, representing bundles of data that the processes produce to flow back to the LCA. These data bundles are listed under Major Outputs in the use case (Figure 4-1), with the destination listed as the LCA.

Now look at the arrow that flows in to the "Determine Chemical Approval Status" process from the right side. In order to determine if the chemical requested is approved for usage, the process has to retrieve some information from storage. The open-ended rectangle labeled "Lawn Chemical Supply" is called a data store, and it represents a collection of stored data. The "Determine Chemical Approval Status" process uses an identifier for the chemical requested to find the requested chemical and to determine whether it is an approved chemical or a disapproved chemical. Notice that the "List of approved chemicals" is listed as a Major Input on the use case (Figure 4-1), with the source listed as the Lawn Chemicals Supply data store. Now, still referring to Figure 4-1, notice that every Major Input listed in the use case flows in to a process from an external entity or stored data (noted by the source). Also notice that every Major Output listed in the use case flows out to a destination (an external entity or data storage) on the data flow diagram.

Now look more closely at the Major Steps Performed section of the use case. You can see that a number of steps are listed in the use case. On the data flow diagram, these steps have been organized into five major processes, each performing one main component of the interactions detailed on the use case. On the DFD (Figure 5-1), as you follow the arrows starting with "Chemical needed" from the LCA to the "Determine Chemical Approval Status" process, imagine the LCA specifying the chemical he needs for a job. The system looks up the chemical and responds with a message verifying it as an approved chemical or informing the

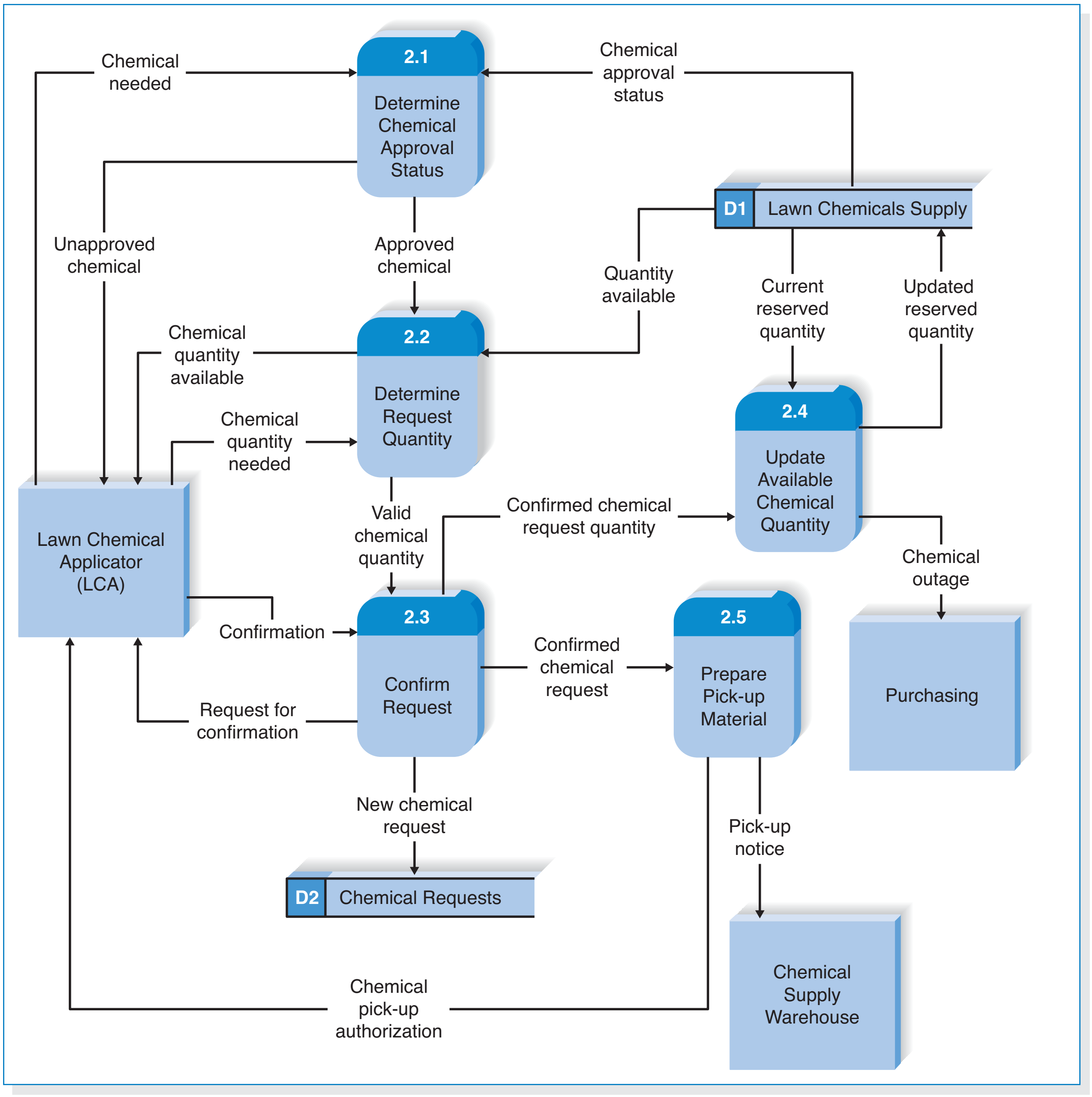


FIGURE 5-1
Request a Chemical Level 1 DFD

LCA that the chemical cannot be used. For an approved chemical, the system looks up how much of the chemical is available and informs the LCA. The LCA indicates the quantity he wants. Now look at the description on the use case (Figure 4-1) for steps 1-4 and notice how the use case describes those processes in words. Notice also how the "Information for Steps" section of the use case lists the data elements that are either used or produced by each step, corresponding to the inflows and outflows from the process symbols (2.1, 2.2) on the data flow diagram (Figure 5-1).

Look at the other three process symbols in the DFD and examine the flows into and out of each process. On the basis of the data flowing in and flowing out, try to understand what the process is doing. Check your understanding by looking at the Major Steps Performed and Information for Steps in the use case.

You probably recognized that the "Confirm Request" process (2.3) receives the LCA's chemical request confirmation and creates and stores a new Chemical Request. You can also see that two additional processes are performed by the system. The quantity of the chemical available is modified by marking the quantity requested as "reserved" (and no longer available to another LCA). Finally, the pick-up authorization is provided to the LCA and the Chemical Supply Warehouse is notified of the approved pick-up. You can see by the flows from process 2.3 to processes 2.4 and 2.5 that sometimes a process sends a data flow directly to another process.

The use case in Figure 4-1 and the data flow diagram in Figure 5-1 are purposefully related. A well-constructed use case makes developing a data flow diagram quite straightforward, although the analyst will have to make some decisions about how much detail to depict in the DFD. The steps outlined in a use case can be organized into logical processes on a DFD. The Major Inputs and Major Outputs listed on the use case provide a list of the sources and destinations, respectively, of the inflows and outflows of the processes. The Information for Steps section shows the data flowing in or produced by each step of the use case, and these correspond to the data flows that enter or leave each process on the data flow diagram.

Elements of Data Flow Diagrams

Now that you have had a glimpse of a DFD, we will present the language of DFDs, which includes a set of symbols, naming conventions, and syntax rules. There are four symbols in the DFD language (processes, data flows, data stores, and external entities), each of which is represented by a different graphic symbol. There are two commonly used styles of symbols, one set developed by Chris Gane and Trish Sarson and the other by Tom DeMarco and Ed Yourdon[2] (Figure 5-2). Neither is better than the other; some organizations use the Gane and Sarson style of symbols, and others use the DeMarco/Yourdon style. We will use the Gane and Sarson style in this book.

Process A *process* is an activity or a function that is performed for some specific business reason. Processes can be manual or computerized. Every process should be named starting with a verb and ending with a noun (e.g., "Determine request quantity"). Names should be short, yet contain enough information so that the reader can easily understand exactly what they do. In general, each process performs only one activity, so most system analysts avoid using the word "and" in process names because it suggests that the process performs several activities. In addition, every process must have at least one input data flow and at least one output data flow.

Figure 5-2 shows the basic elements of a process and how they are usually named in CASE tools. Every process has a unique identification number, a name, and a description, all of which are noted in the CASE repository. Descriptions clearly and

[2] See Chris Gane and Trish Sarson, *Structured Systems Analysis: Tools and Techniques*, Englewood Cliffs, NJ: Prentice Hall, 1979; Tom DeMarco, *Structured Analysis and System Specification*, Englewood Cliffs, NJ: Prentice-Hall, 1979; and E. Yourdon and Larry L. Constantine, *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*, Englewood Cliffs, NJ: Prentice-Hall, 1979.

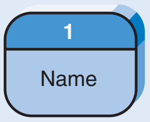
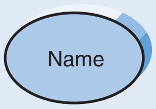
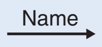
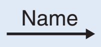
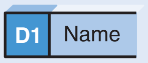
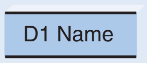
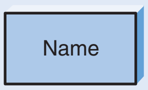
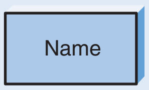
Data Flow Diagram Element	Typical Computer-Aided Software Engineering Fields	Gane and Sarson Symbol	DeMarco and Yourdon Symbol
Every *process* has a number a name (verb phase) a description at least one output data flow at least one input data flow	Label (name) Type (process) Description (what is it) Process number Process description (structured English) Notes		
Every *data flow* has a name (a noun) a description one or more connections to a process	Label (name) Type (flow) Description Alias (another name) Composition (description of data elements) Notes		
Every *data store* has a number a name (a noun) a description one or more input data flows one or more output data flows	Label (name) Type (store) Description Alias (another name) Composition (description of data elements) Notes		
Every *external entity* has a name (a noun) a description	Label (name) Type (entity) Description Alias (another name) Entity description Notes		

FIGURE 5-2
Data Flow Diagram Elements

precisely describe the steps and details of the processes; ultimately, they are used to guide the programmers who need to computerize the processes (or the writers of policy manuals for noncomputerized processes). The process descriptions become more detailed as information is learned about the process through the analysis phase. Many process descriptions are written as simple text statements about what happens. More complex processes use more formal techniques such as structured English, decision tables, or decision trees, which are discussed in a later section.

Data Flow A *data flow* is a single piece of data (e.g., quantity available) (sometimes called a data element), or a logical collection of several pieces of information (e.g., new chemical request). Every data flow should be named with a noun. The description of a data flow lists exactly what data elements the flow contains. For example, the pick-up notice data flow can list the LCA name, chemical, and quantity requested as its data elements.

Data flows are the glue that holds the processes together. One end of every data flow will always come from or go to a process, with the arrow showing the direction into or out of the process. Data flows show what inputs go into each process and what outputs each process produces. Every process must create at least one output data flow, because if there is no output, the process does not do anything. Likewise, each process has at least one input data flow, because it is difficult, if not impossible, to produce an output with no input.

Data Store A *data store* is a collection of data that is stored in some way (which is determined later when creating the physical model). Every data store is named with a noun and is assigned an identification number and a description. Data stores form the starting point for the data model (discussed in the next chapter) and are the principal link between the process model and the data model.

Data flows coming out of a data store indicate that information is retrieved from the data store. Looking at Figure 5-1, you can see that process 2.1 (Determine Chemical Approval Status) retrieves the Chemical Approval Status data flow from the Lawn Chemicals Supply data store. Similarly, Process 2.2 (Determine Request Quantity) retrieves the Quantity Available data flow from the Lawn Chemicals Supply data store. Data flows going into a data store indicate that information is added to the data store. For example, process 2.3 adds a New Chemical Request data flow to the Chemical Requests data store. Finally, data flows going both into and out of a data store indicate that information in the data store is changed (e.g., by retrieving data from a data store, changing it, and storing it back). In Figure 5-1, we can see that process 2.4 (Update Available Chemical Quantity) retrieves the current reserved quantity from the Lawn Chemical Supply data store, modifies it, and writes the updated data back into the data store.

All data stores must have at least one input data flow (or else they never contain any data), unless they are created and maintained by another information system or on another page of the DFD. Likewise, they have at least one output data flow on some page of the DFD. (Why store data if you never use it?) In cases in which the same process both stores data and retrieves data from a data store, there is a temptation to draw one data flow with an arrow on both ends. This practice is incorrect, however. The data flow that stores data and the data flow that retrieves data should always be shown as two separate data flows.

External Entity An *external entity* is a person, organization, organization unit, or system that is external to the system, but interacts with it (e.g., customer, clearinghouse, government organization, accounting system). The external entity typically corresponds to the primary actor identified in the use case. External entities provide data to the system or receive data from the system, and serve to establish the system boundaries. Every external entity has a name and a description. The key point to remember about an external entity is that it is external to the system, but may or may not be part of the organization. People who use the information from the system to perform other processes or who decide what information goes into the system are documented as external entities (e.g., managers, staff).

Using Data Flow Diagrams to Define Business Processes

Most business processes are too complex to be explained in one DFD. Most process models are therefore composed of a set of DFDs. The first DFD provides a summary of the overall system, with additional DFDs providing more and more detail about each part of the overall business process. Thus, one important principle in process modeling with DFDs is the decomposition of the business process into a hierarchy of DFDs, with each level down the hierarchy representing less scope but more detail. Figure 5-3 shows how one business process can be decomposed into several levels of DFDs.

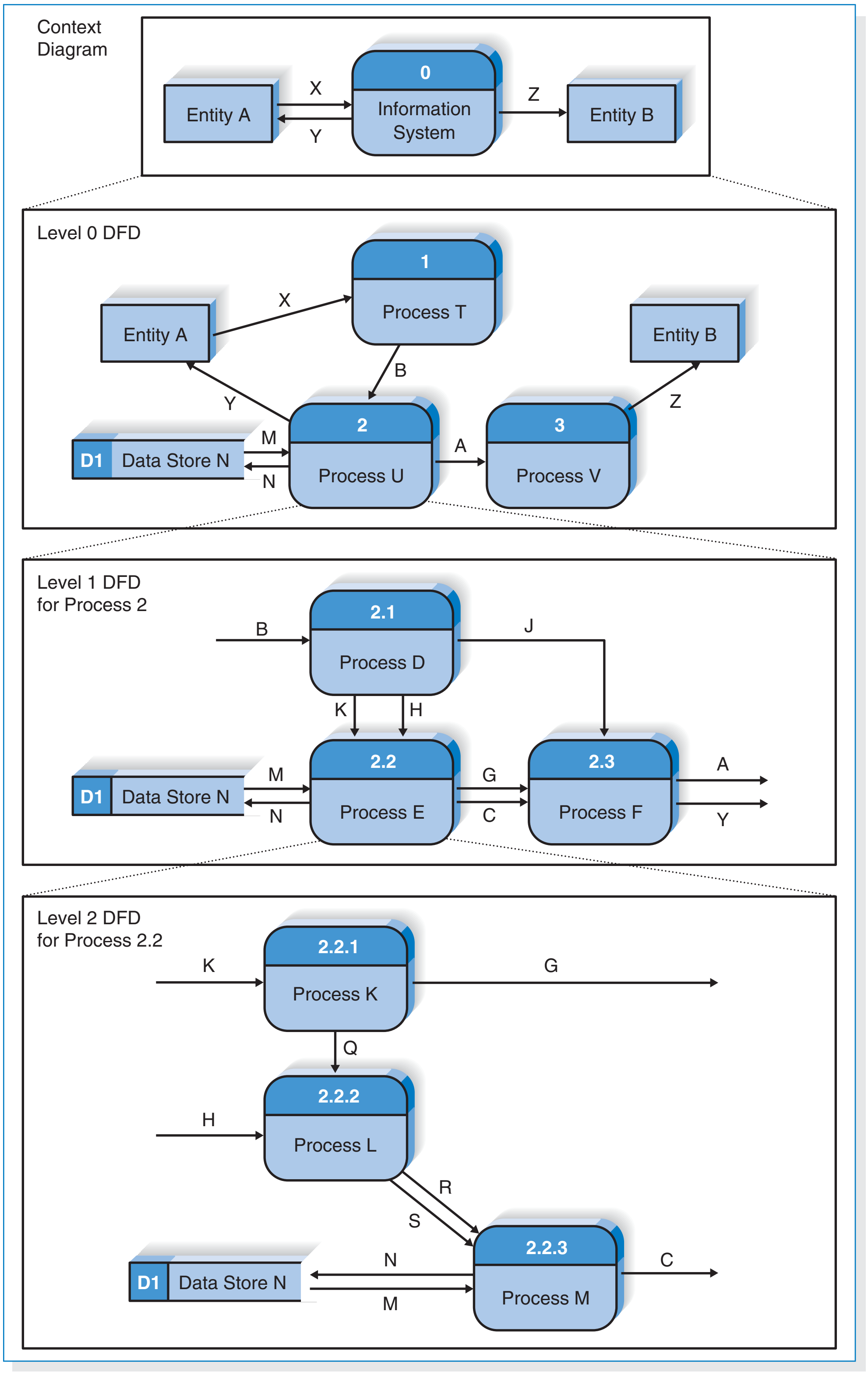


FIGURE 5-3
Relationships among Levels of Data Flow Diagrams (DFDs)

Context Diagram The first DFD in every business process model, whether a manual system or a computerized system, is the *context diagram* (see Figure 5-3). As the name suggests, the context diagram shows the entire system in context with its environment. All process models have one context diagram.

The context diagram shows the overall business process as just one process (i.e., the system itself) and shows the data flows to and from external entities. Data stores usually are not included on the context diagram, unless they are "owned" by systems or processes other than the one being documented. For example, an information system used by the university library that records who has borrowed books would likely check the registrar's student information database to see whether a student is currently registered at the university. In this context diagram, the registrar's student information data store could be shown on the context diagram because it is external to the library system, but used by it. Many organizations, however, would show this as an external entity called "Registrar's Student Information System," not as a data store.

Level 0 Diagram The next DFD is called the *level 0 diagram* or *level 0 DFD.* (See Figure 5-3.) The level 0 diagram shows all the processes at the first level of numbering (i.e., processes numbered 1 through 3), the data stores, external entities, and data flows among them. The purpose of the level 0 DFD is to show all the major high-level processes of the system and how they are interrelated. All process models have one and only one level 0 DFD.

Another key principle in creating sets of DFDs is *balancing.* Balancing means ensuring that all information presented in a DFD at one level is accurately represented in the next-level DFD. This doesn't mean that the information is identical, but that it is shown appropriately. There is a subtle difference in meaning between these two words that will become apparent shortly, but for the moment, let's compare the context diagram with the level 0 DFD in Figure 5-3 to see how the two are balanced. In this case, we see that the external entities (A, B) are identical between the two diagrams and that the data flows to and from the external entities in the context diagram (X, Y, Z) also appear in the level 0 DFD. The level 0 DFD replaces the context diagram's single process (always numbered 0) with three processes (1, 2, 3), adds a data store (D1), and includes two additional data flows that were not in the context diagram (data flow B from process 1 to process 2; data flow A from process 2 to process 3).

These three processes and two data flows are contained within process 0. They were not shown on the context diagram because they are the internal components of process 0. The context diagram deliberately hides some of the system's complexity in order to make it easier for the reader to understand. Only after the reader understands the context diagram does the analyst "open up" process 0 to display its internal operations by decomposing the context diagram into the level 0 DFD, which shows more detail about the processes and data flows inside the system.

Level 1 Diagrams In the same way that the context diagram deliberately hides some of the system's complexity, so, too, does the level 0 DFD. The level 0 DFD shows only how the major high-level processes in the system interact. Each process on the level 0 DFD can be decomposed into a more explicit DFD, called a *level 1 diagram*, or *level 1 DFD*, which shows how it operates in greater detail. The DFD illustrated in Figure 5-1 is a level 1 DFD.

In general, all process models have as many level 1 diagrams as there are processes on the level 0 diagram; every process in the level 0 DFD would be *decomposed* into its own level 1 DFD, so the level 0 DFD in Figure 5-3 would have three level 1 DFDs (one for process 1, one for process 2, one for process 3). For simplicity, we have chosen to show only one level 1 DFD in this figure, the DFD for process 2. The processes in level 1 DFDs are numbered on the basis of the process being decomposed. In this example, we are decomposing process 2, so the processes in this level 1 DFD are numbered 2.1, 2.2, and 2.3.

Processes 2.1, 2.2, and 2.3 are the *children* of process 2, and process 2 is the *parent* of processes 2.1, 2.2, and 2.3. These three children processes wholly and completely make up process 2. The set of children and the parent are identical; they are simply different ways of looking at the same thing. When a parent process is decomposed into children, its children must completely perform all of its functions, in the same way that cutting up a pie produces a set of slices that wholly and completely make up the pie. Even though the slices may not be the same size, the set of slices is identical to the entire pie; nothing is omitted by slicing the pie.

Once again, it is very important to ensure that the level 0 and level 1 DFDs are balanced. The level 0 DFD shows that process 2 accesses data store D1, has two input data flows (B, M), and has three output data flows (A, N, and Y). A check of the level 1 DFD shows the same data store and data flows. Once again, we see that five new data flows have been added (C, G, H, J, K) at this level. These data flows are contained within process 2 and therefore are not documented in the level 0 DFD. Only when we decompose or open up process 2 via the level 1 DFD do we see that they exist.

The level 1 DFD shows more precisely which process uses the input data flow B (process 2.1) and which produces the output data flows A and Y (process 2.3). Note, however, that the level 1 DFD does not show where these data flows come from or go to. To find the source of data flow B, for example, we have to move up to the level 0 DFD, which shows data flow B coming from external entity B. Likewise, if we follow the data flow from A up to the level 0 DFD, we see that it goes to process 3, but we still do not know exactly which process within process 3 uses it (e.g., process 3.1, 3.2). To determine the exact source, we would have to examine the level 1 DFD for process 3.

This example shows one downside to the decomposition of DFDs across multiple pages. To find the exact source and destination of data flows, one often must follow the data flow across several DFDs on different pages. Several alternatives to this approach to decomposing DFDs have been proposed, but none is as commonly used as the "traditional" approach. The most common alternative is to show the source and destination of data flows to and from external entities (as well as data stores) at the lower level DFDs. The fact that most data flows are to or from data stores and external entities, rather than processes on other DFD pages, can significantly simplify the reading of multiple page DFDs. We believe this to be a better approach, so when we teach our courses, we show external entities on all DFDs, including level 1 DFDs and below.

Level 2 Diagrams The bottom of Figure 5-3 shows the next level of decomposition: a *level 2 diagram*, or *level 2 DFD*, for process 2.2. This DFD shows that process 2.2 is decomposed into three processes (2.2.1, 2.2.2, and 2.2.3). The level 1 diagram for process 2.2 shows interactions with data store D1, which we see in the level 2 DFD as occurring in process 2.2.3. Likewise, the level 2 DFD for 2.2 shows two input data flows (H, K) and two output data flows (C, G), which we also see on the level 2 diagram, along with several new data flows (Q, R, S). The two DFDs are therefore balanced.

It is sometimes difficult to remember which DFD level is which. It may help to remember that the level numbers refer to the number of decimal points in the process numbers on the DFD. A level 0 DFD has process numbers with no decimal points (e.g., 1, 2), whereas a level 1 DFD has process numbers with one decimal point (e.g., 2.3, 5.1), a level 2 DFD has numbers with two decimal points (e.g.,1.2.5, 3.3.2), and so on.

Alternative Data Flows Suppose that a process produces two different data flows under different circumstances. For example, a quality-control process could produce a quality-approved widget or a defective widget, or our search for a chemical could find it is approved or not approved for use. How do we show these alternative paths in the DFD? The answer is that we show both data flows and use the process description to explain that they are alternatives. Nothing on the DFD itself shows that the data flows are mutually exclusive. For example, process 2.1 on the level 1 DFD produces three output data flows (H, J, K). Without reading the text description of process 2.1, we do not know whether these are produced simultaneously or whether they are mutually exclusive.

Process Descriptions

The purpose of the process descriptions is to explain what the process does and provide additional information that the DFD does not provide. As we move through the SDLC, we gradually move from the general text descriptions of requirements into more and more precise descriptions that are eventually translated into very precise programming languages. In most cases, a process is straightforward enough that the requirements definition, a use case, and a DFD with a simple text description together provide sufficient detail to support the activities in the design phase. Sometimes, however, the process is sufficiently complex that it can benefit from a more detailed process description that explains the logic which occurs inside the process. Three techniques are commonly used to describe more complex processing logic: structured English, decision trees, and decision tables. Very complex processes may use a combination of structured English and either decision trees or decision tables.

Structured English uses short sentences to describe the work that a process performs. *Decision trees* display decision logic (IF statements) as a set of nodes (questions) and branches (answers). *Decision tables* represent complex policy decisions as rules that link various conditions with actions. Since these techniques are commonly discussed in programming texts, we will not elaborate on them here. They are useful to the systems analyst in conveying the proper understanding of what goes on "inside" a process.

CREATING DATA FLOW DIAGRAMS

Data flow diagrams start with the information in the use cases and the requirements definition. Although the use cases are created by the users and project team working together, the DFDs typically are created by the project team and then reviewed by the users. Generally speaking, the set of DFDs that make up the process model simply integrates the individual use cases (and adds in any processes in the requirements definition not selected as use cases). The project team takes the use cases and rewrites them as DFDs. However, because DFDs have formal rules about symbols and syntax that use cases do not, the project team sometimes has to revise some of the information in the use cases to make them conform to the DFD rules. The most common types of changes are to the names of the use cases that become processes and the inputs and outputs that become data flows. The second most common type of change is to combine several small inputs and outputs in the use cases into larger data flows in the DFDs (e.g., combining three separate inputs, such as "customer name," "customer address," and "customer phone number," into one data flow, such as "customer information").

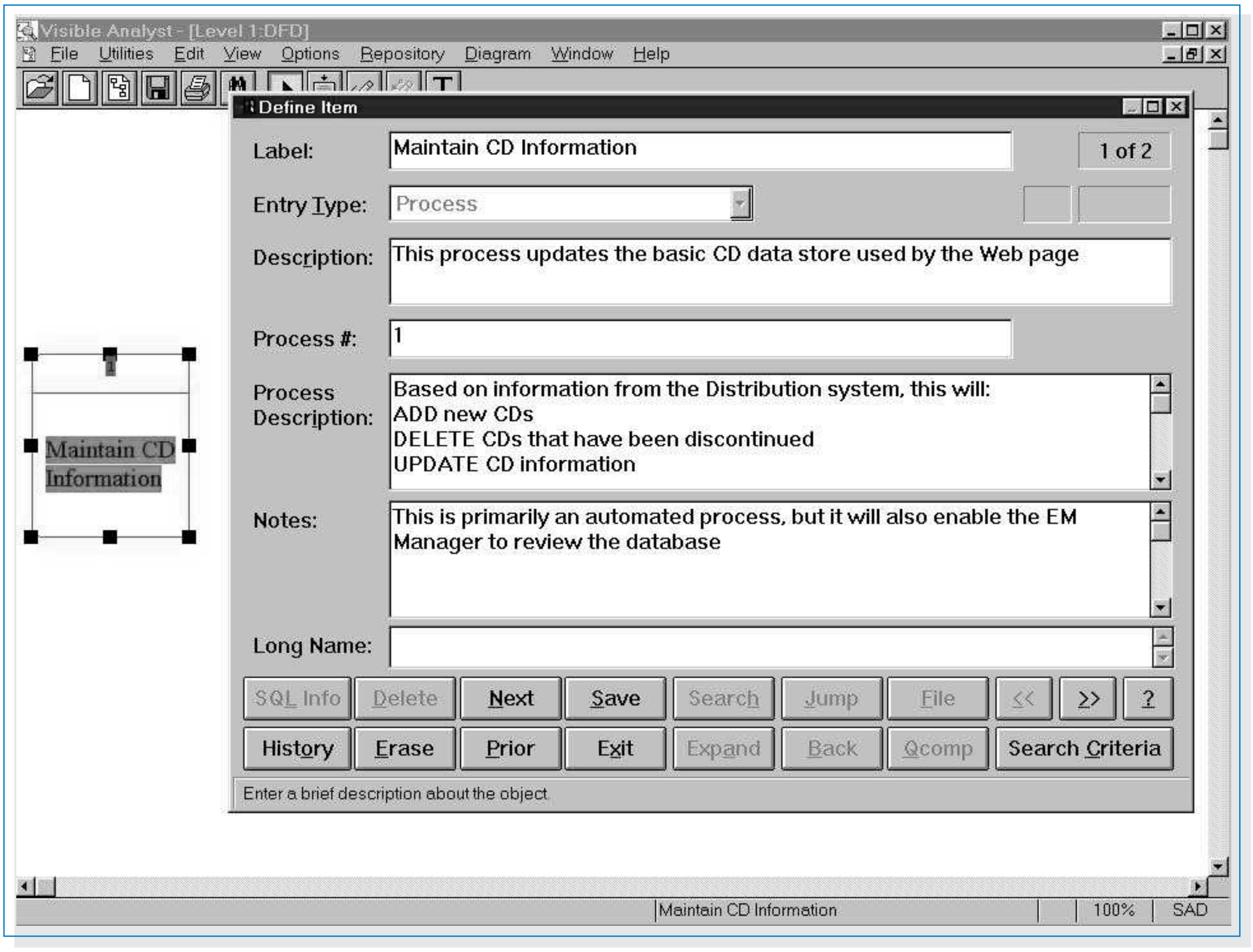
Project teams usually use process modeling tools or CASE tools to draw process models. Simple tools such as Visio contain DFD symbol sets and enable easy creation and modification of diagrams. Other process modeling tools such as BPWin understand the DFD and can perform simple syntax checking to make sure that the DFD is at least somewhat correct. A full CASE tool, such as Visible Analyst Workbench, provides many capabilities in addition to process modeling (e.g., data modeling as discussed in the next chapter). CASE tools tend to be complex, and while they are valuable for large and complex projects, they often cost more than they add for simple projects. Figure 5-4 shows a sample screen from the Visible Analyst CASE tool.

Building a process model that has many levels of DFDs usually entails several steps. Some analysts prefer to begin process modeling by focusing first on the level 0 diagram. We have found it useful to first build the context diagram showing all the external entities and the data flows that originate from or terminate in them. Second, the team creates a DFD fragment for each use case that shows how the use case exchanges data flows with the external entities and data stores. Third, these DFD fragments are organized into a level 0 DFD. Fourth, the team develops level 1 DFDs, based on the steps within each use case, to better explain how they operate. In some cases, these level 1 DFDs are further decomposed into level 2 DFDs, level 3 DFDs, level 4 DFDs, and so on. Fifth, the team validates the set of DFDs to make sure that they are complete and correct.

In the following sections, process modeling is illustrated with the Holiday Travel Vehicles information system.

Creating the Context Diagram

The context diagram defines how the business process or computer system interacts with its environment—primarily the external entities. To create the context diagram, you simply draw one process symbol for the business process or system being modeled (numbered 0 and named for the process or system). You read through the use cases and add the inputs and outputs listed on the form, as well as their sources and destinations. Usually, all the inputs and outputs will come from or go to external entities such as a person, organization, or other information system. If any inputs and outputs connect directly to data stores in an external system, it is best practice to create an external entity which is named for the system that owns the data store. None of the data stores inside the process/system that are created by the process or system itself are included in the context diagram, because they are "inside" the system. Because there are sometimes so many inputs and outputs, we often combine several small data flows into larger data flows.



Label: Maintain CD Information — 1 of 2

Entry Type: Process

Description: This process updates the basic CD data store used by the Web page

Process #: 1

Process Description: Based on information from the Distribution system, this will:
ADD new CDs
DELETE CDs that have been discontinued
UPDATE CD information

Notes: This is primarily an automated process, but it will also enable the EM Manager to review the database

Long Name:

Enter a brief description about the object.

Maintain CD Information

FIGURE 5-4
Entering Data Flow Diagram Processes in a Computer-Aided Software Engineering Tool

Figure 5-5 shows the context diagram for the Holiday Travel Vehicles system focusing on vehicle sales. Take a moment to review this system as described in Chapter 4 and review the use cases in Figure 4-11. You can see from the Major Inputs and Outputs sections in the Figure 4-11 use cases that the system has many interactions with the salesperson external entity. We have simplified these inflows and outflows to just three primary data flows on the context diagram. If we had included each small data flow, the context diagram would become too cluttered. The smaller data flows will become evident as we decompose the context diagram into more detailed levels. Notice that we have established three external entities to represent parts of the Holiday Travel Vehicle organization which receive information from or supply information to this system. Salespeople provide key inputs to the system, and shop work orders flow to the shop manager. The company owner or manager provides information to the system.

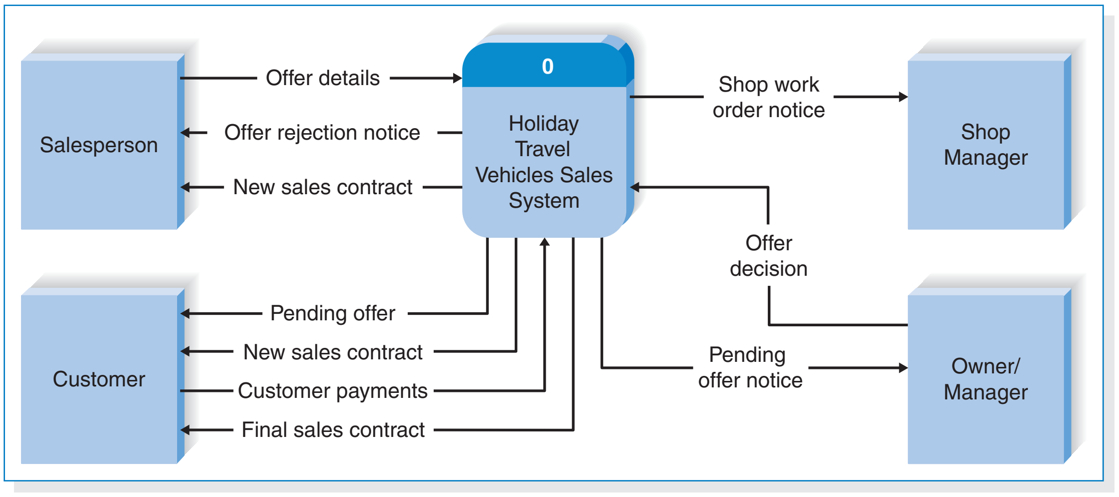


FIGURE 5-5
Holiday Travel Vehicles Sales System Context Diagram

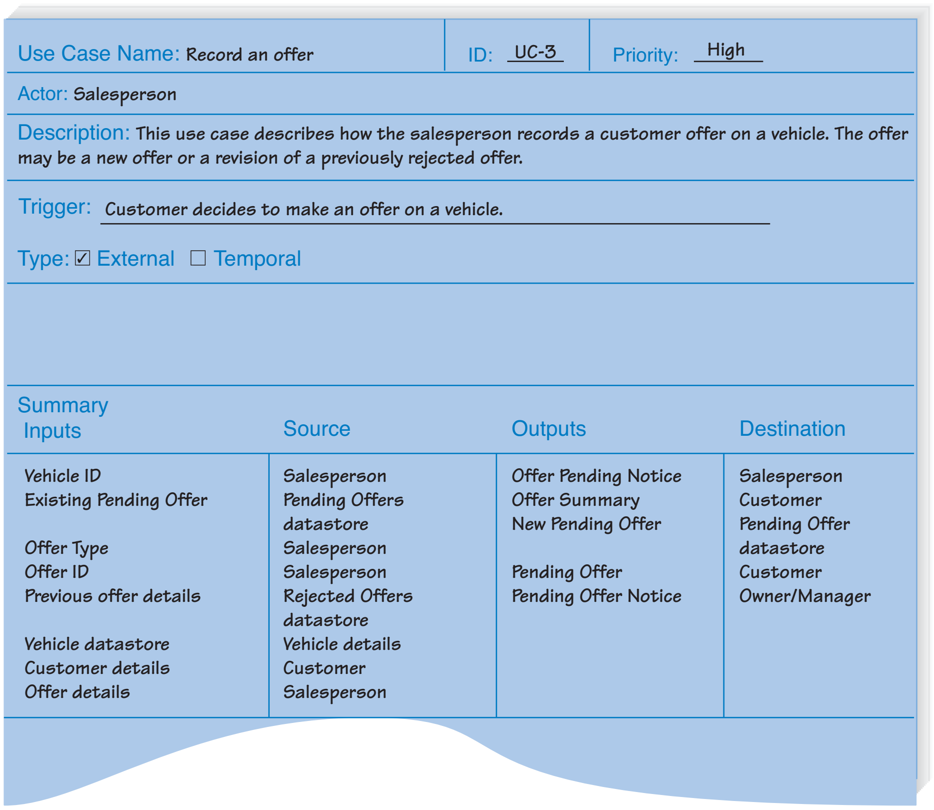
Creating Data Flow Diagram Fragments

A *DFD fragment* is one part of a DFD that eventually will be combined with other DFD fragments to form a DFD diagram. In this step, each use case is converted into one DFD fragment. You start by taking each use case and drawing a DFD fragment, using the information given on the top of the use case: the name, ID number, and major inputs and outputs. The information about the major steps that make up each use case is ignored at this point; it will be used in a later step. Figure 5-6 shows a use case and the DFD fragment that was created from it.

Once again, some subtle, but important changes are often made in converting the use case into a DFD. The two most common changes are modifications to the process names and the addition of data flows. There were no formal rules for use case names, but there are formal rules for naming processes on the DFD. All process names must be a verb phrase—they must start with a verb and include a noun. (See Figure 5-2.) Not all of our use case names are structured in this way, so we sometimes need to change them. It is also important to have a consistent *viewpoint* when naming processes. For example, the DFD in Figure 5-6 is written from the viewpoint of the dealership, not of the customer. All the process names and descriptions are written as activities that the staff performs. It is traditional to design the processes from the viewpoint of the organization running the system, so this sometimes requires some additional changes in names.

The second common change is the addition of data flows. Use cases are written to describe how the system and user interact. Typically, they do not describe how the system obtains data, so the use case often omits data flows read from a data store. When creating DFD fragments, it is important to make sure that any information given to the user is obtained from a data store. The easiest way to do this is to first create the DFD fragment with the major inputs and outputs listed on the use case and then verify that all outputs have sufficient inputs to create them.

There are no formal rules covering the *layout* of processes, data flows, data stores, and external entities within a DFD. Most systems analysts try to put the process in the middle of the DFD fragment, with the major inputs starting from the



Use Case Name: Record an offer	ID: UC-3	Priority: High	
Actor: Salesperson			
Description: This use case describes how the salesperson records a customer offer on a vehicle. The offer may be a new offer or a revision of a previously rejected offer.			
Trigger: Customer decides to make an offer on a vehicle.			
Type: ☑ External ☐ Temporal			

Summary Inputs	Source	Outputs	Destination
Vehicle ID	Salesperson	Offer Pending Notice	Salesperson
Existing Pending Offer	Pending Offers datastore	Offer Summary	Customer
Offer Type	Salesperson	New Pending Offer	Pending Offer datastore
Offer ID	Salesperson	Pending Offer	Customer
Previous offer details	Rejected Offers datastore	Pending Offer Notice	Owner/Manager
Vehicle datastore	Vehicle details		
Customer details	Customer		
Offer details	Salesperson		

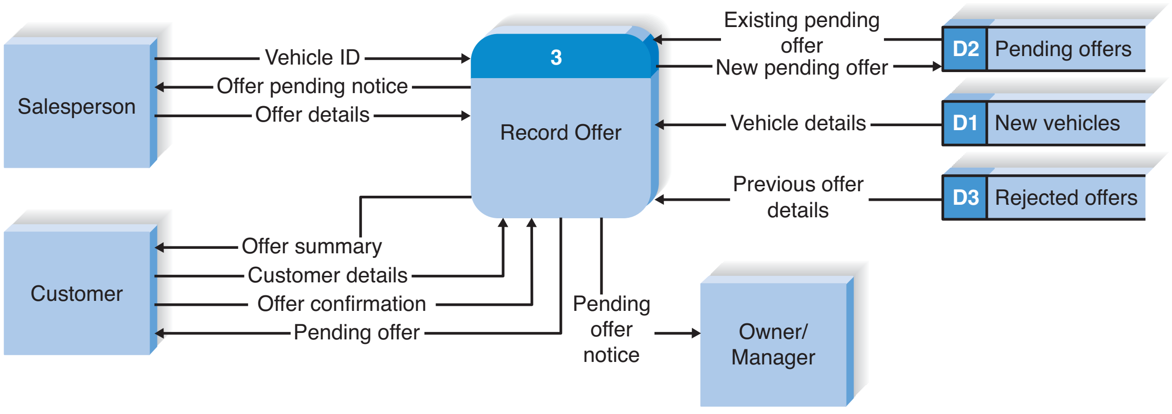


FIGURE 5-6
Holiday Travel Vehicles Process 3 (Record Offer) DFD Fragment

left side or top entering the process and outputs leaving from the right or the bottom. Data stores are often written below the process.

Take a moment and draw a DFD fragment for the two other use cases shown in Figure 4-11 (Evaluate Offer and Take Delivery of Vehicle). We have included possible ways of drawing these fragments in Figure 5-7. (Don't look until you've attempted the drawings on your own!)

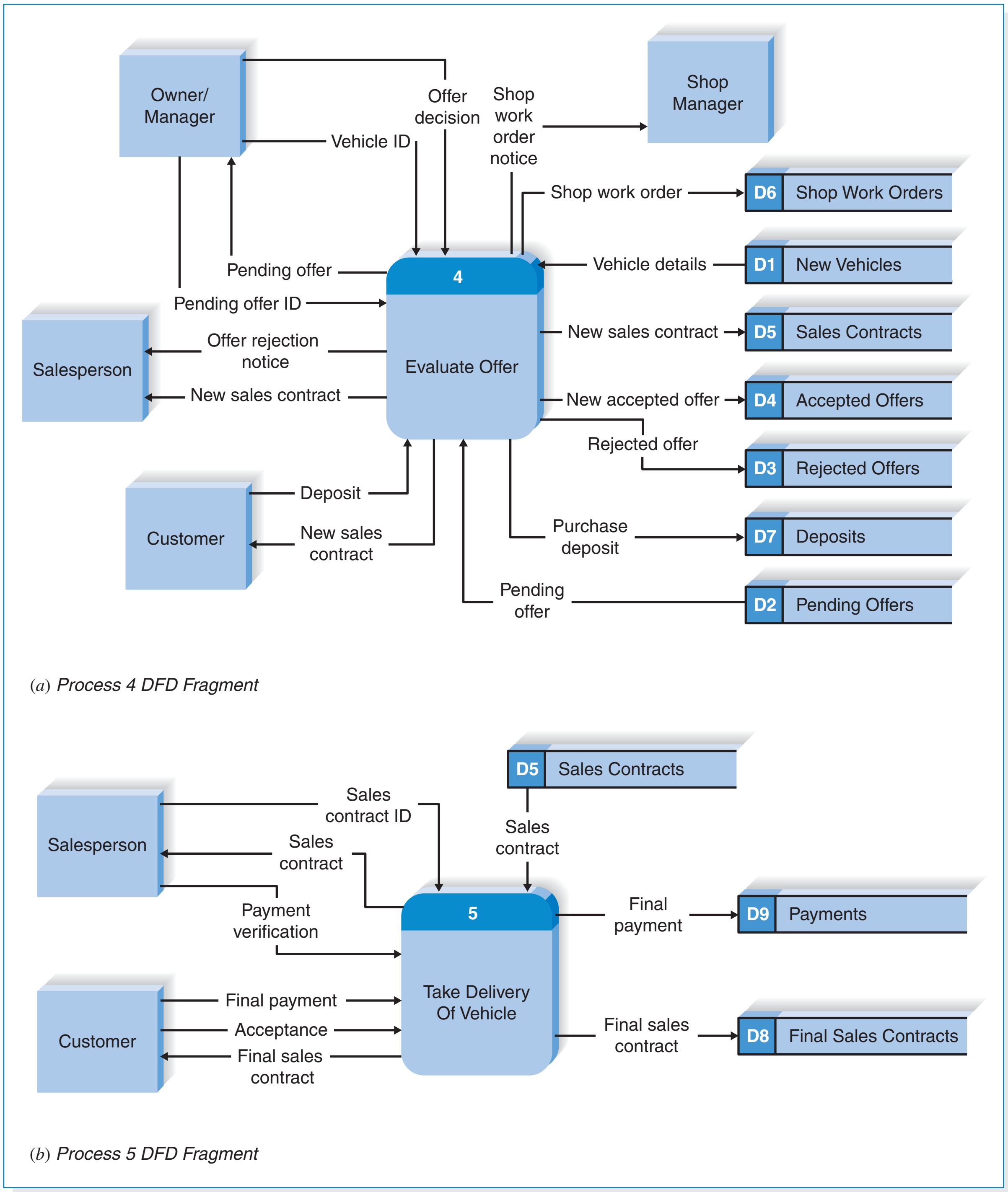


FIGURE 5-7
Additional DFD Fragments for Holiday Travel Vehicles

Creating the Level 0 Data Flow Diagram

Once you have the set of DFD fragments (one for each of the major use cases), you combine them into one DFD drawing that becomes the level 0 DFD. As mentioned earlier, there are no formal layout rules for DFDs. Most systems analysts try to put the process that is first chronologically in the upper left corner of the diagram and work their way from top to bottom, left to right (e.g., Figure 5-1). Generally speaking, most analysts try to reduce the number of times that data flow lines cross or to ensure that when they do cross, they cross at right angles so that there is less confusion. (Many give one line a little "hump" to imply that one data flow jumps over the other without touching it.) Minimizing the number of data flows that cross is challenging.

Iteration is the cornerstone of good DFD design. Even experienced analysts seldom draw a DFD perfectly the first time. In most cases, they draw it once to understand the pattern of processes, data flows, data stores, and external entities and then draw it a second time on a fresh sheet of paper (or in a fresh file) to make it easier to understand and to reduce the number of data flows that cross. Often, a DFD is drawn many times before it is finished.

Figure 5-8 combines the DFD fragments in Figures 5-6 and 5-7. Take a moment to examine Figure 5-8 and find the DFD fragments from Figures 5-6 and 5-7 contained within it.

Creating Level 1 Data Flow Diagrams (and Below)

The team now begins to create lower-level DFDs for each process in the level 0 DFD that needs a level 1 DFD. Each one of the use cases is turned into its own DFD. The process for creating the level 1 DFDs is to take the steps as written on the use cases and convert them into a DFD in much the same way as for the level 0 DFD. Usually, each major step in the use case becomes a process on the level 1 DFD, with the inputs and outputs becoming the input and output data flows. Once again, however, sometimes subtle changes are required to go from the informal descriptions in the use case to the more formal process model, such as adding input data flows that were not included in the use case. And because the analysts are now starting to think more deeply about how the processes will be supported by an information system, they sometimes slightly change the use case steps to make the process easier to use.

In some approaches to creating DFDs, no source and destination are given on the level 1 DFD (and lower) for the inputs that come and go between external entities (or other processes outside of this process). But the source and destination of data flows for data stores and data flows that go to processes within this DFD are included (i.e., from one step to another in the same use case, such as "Confirmed Chemical Request" from process 2.3 to 2.5 in Figure 5-1). This is because the information is redundant; you can see the destination of data flows by reading the level 0 DFD.

The problem with these approaches is that in order to really understand the level 1 DFD, you must refer back to the level 0 DFD. For small systems that only have one or two level 1 DFDs, this is not a major problem. But for large systems that have many levels of DFDs, the problem grows; in order to understand the destination of a data flow on a level 3 DFD, you have to read the level 2 DFD, the level 1 DFD, and the level 0 DFD—and if the destination is to another activity, then you have to trace down in the lower-level DFDs in the other process.

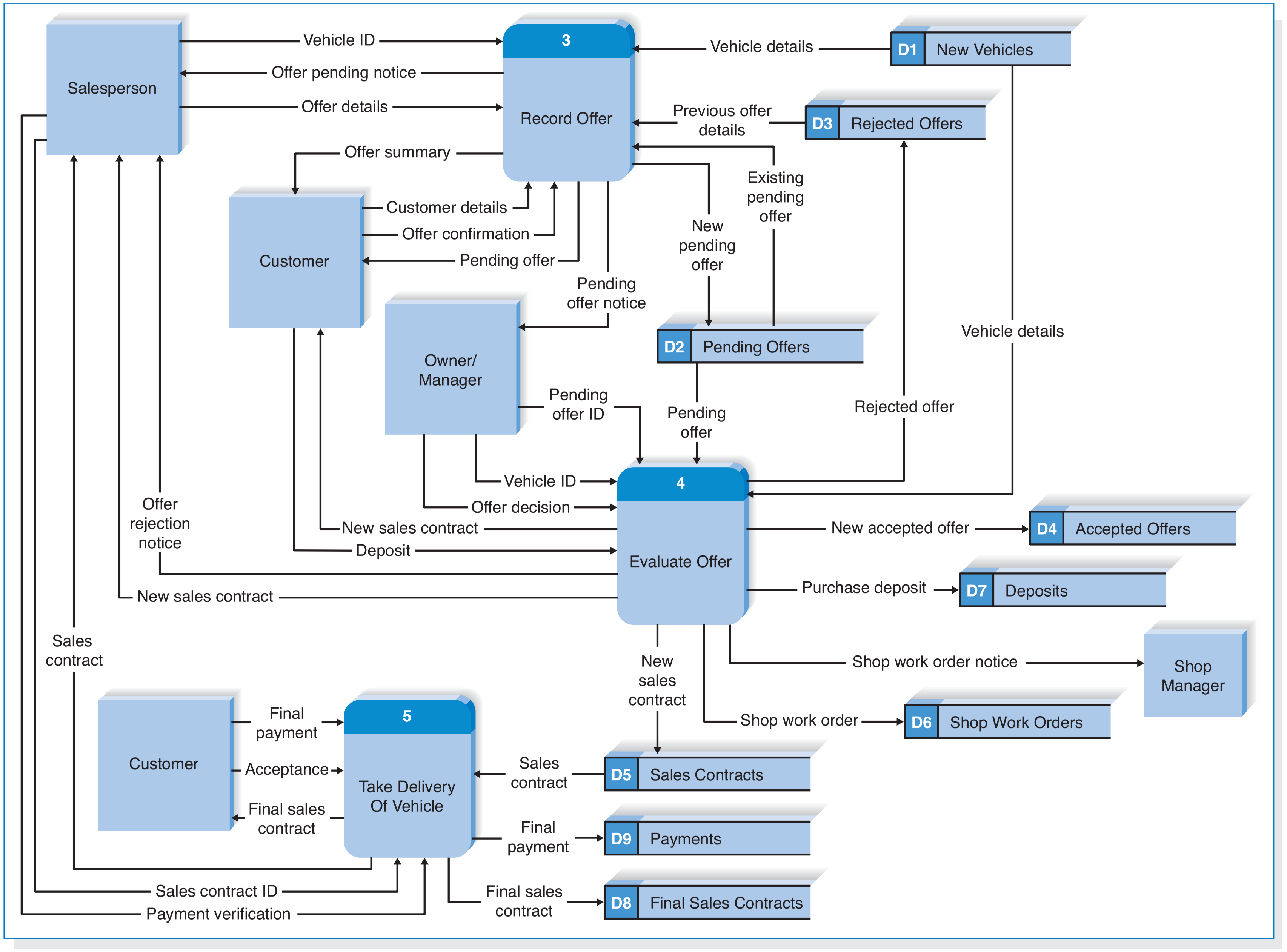


FIGURE 5-8
Holiday Travel Vehicles Level 0 DFD

We believe that including external entities in level 1 and lower DFDs dramatically simplifies the readability of DFDs, with very little downside. In our work in several dozen projects with the U.S. Department of Defense, several other federal agencies, and the military of two other countries, we came to understand the value of this approach and converted the powers that be to our viewpoint. Because DFDs are not completely standardized, each organization uses them slightly differently. So, the ultimate decision of whether or not to include external entities on level 1 DFDs is yours—or your instructor's! In this book, we will include them.

Ideally, we try to keep the data stores in the same general position on the page in the level 1 DFD as they were in the level 0 DFD, but this is not always possible. We try to draw input data flows arriving from the left edge of the page and output data flows leaving from the right edge. For example, see the level 1 DFD in Figure 5-1.

One of the most challenging design questions is when to decompose a level 1 DFD into lower levels. The decomposition of DFDs can be taken to almost any level, so for example, we could decompose process 1.2 on the level 1 DFD into processes 1.2.1, 1.2.2, 1.2.3, and so on in the level 2 DFD. This can be repeated to any level of detail, so one could have level 4 or even level 5 DFDs.

There is no simple answer to the "ideal" level of decomposition, because it depends on the complexity of the system or business process being modeled. In general, you decompose a process into a lower-level DFD whenever that process is sufficiently complex that additional decomposition can help explain the process. Most experts believe that there should be at least three, and no more than seven to nine, processes on every DFD, so if you begin to decompose a process and end up with only two processes on the lower-level DFD, you probably don't need to decompose it. There seems little point in decomposing a process and creating another lower-level DFD for only two processes; you are better off simply showing two processes on the original higher level DFD. Likewise, a DFD with more than nine processes becomes difficult for users to read and understand, because it is very complex and crowded. Some of these processes should be combined and explained on a lower-level DFD.

One guideline for reaching the ideal level of decomposition is to decompose until you can provide a detailed description of the process in no more than one page of process descriptions: structured English, decision trees, or decision tables. Another helpful rule of thumb is that each lowest level process should be no more complex than what can be realized in about 25–50 lines of code.

In Figures 5-9, 5-10, and 5-12, we have provided level 1 DFDs for the vehicles sales processes we have focused on for the Holiday Travel Vehicle system. As we describe each figure, take a moment to back at the Major Steps section of their respective use cases in Figure 4-11.

Figure 5-9 depicts the level 1 DFD for the process of Recording an Order (process 3). The salesperson first checks for any existing Pending Offers for the vehicle (3.1). If a pending offer is found, the salesperson is notified and the process ends. Otherwise, the salesperson is specified if the offer is a new offer or a revision of a previous (rejected) offer (3.2). If it is a revised offer, the previous offer is retrieved from the Rejected Offers data store (3.3). If it is a new offer, information about the vehicle and the customer is obtained (3.4). The salesperson completes the specific details of the offer (3.5), and the offer must be confirmed by the customer (3.6). Once confirmed, the Pending Offer is stored, a copy is given to the customer, and the Owner/Manager is notified of the new Pending Offer (3.7). As you look at the use case for this event, you will see that the steps outlined have been captured in these

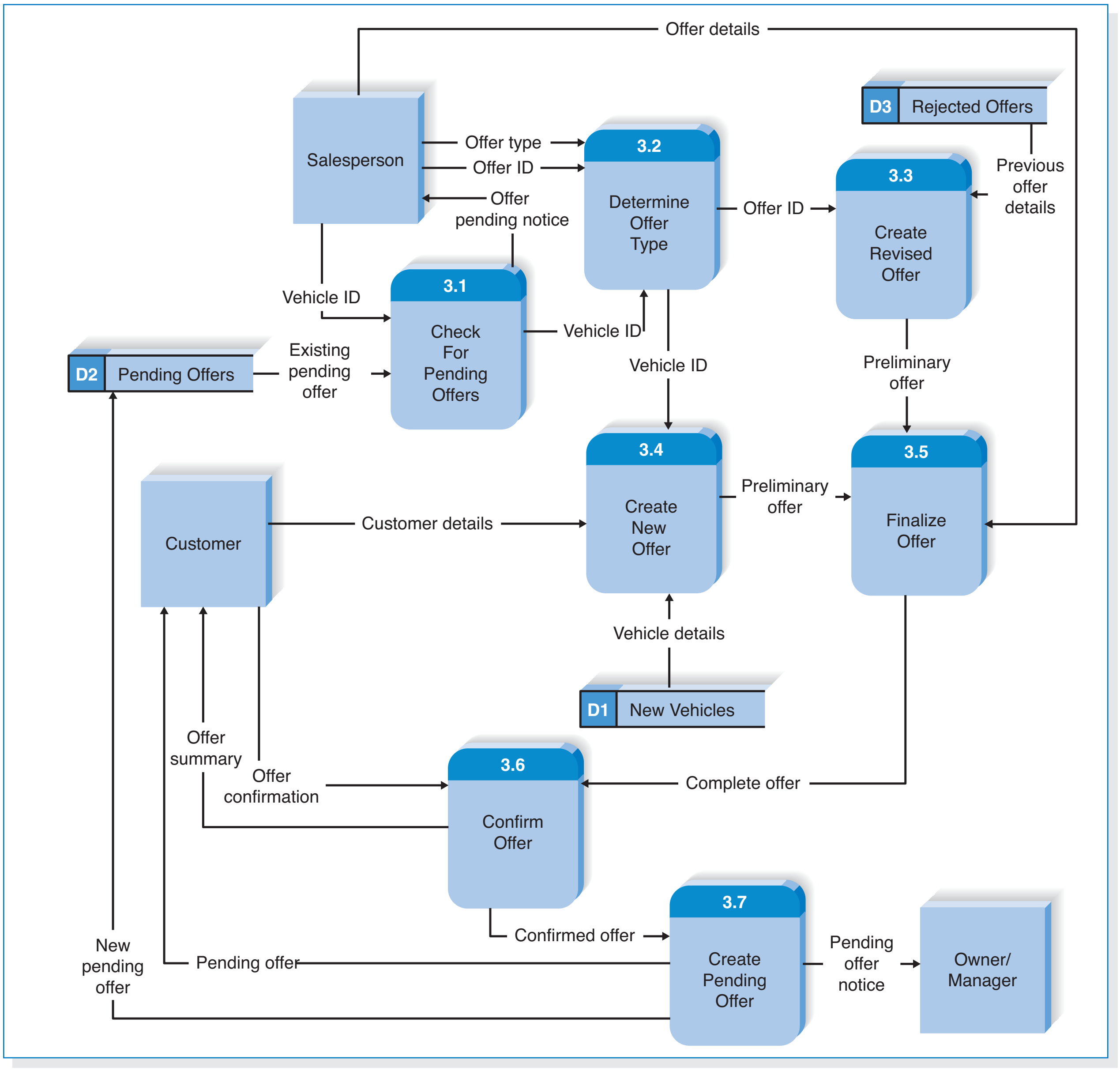


FIGURE 5-9
Holiday Travel Vehicles Process 3 (Record Offer) Level 1 DFD

seven processes in Figure 5-9. Also notice that we have attempted to organize the use case steps into processes that focus on one primary task.

Figure 5-10 describes the process of Evaluating an Offer. We have taken a slightly different approach to creating this level 1 DFD. The Owner/Manager is able to look at the Pending Offer and details about the vehicle and enters his decision to accept or reject the offer (4.1). For rejected offers, the Owner/Manager enters the reason for the rejection, the rejected offer is recorded, and the salesperson is notified of the offer rejection (4.2). For accepted offers, a variety of tasks are

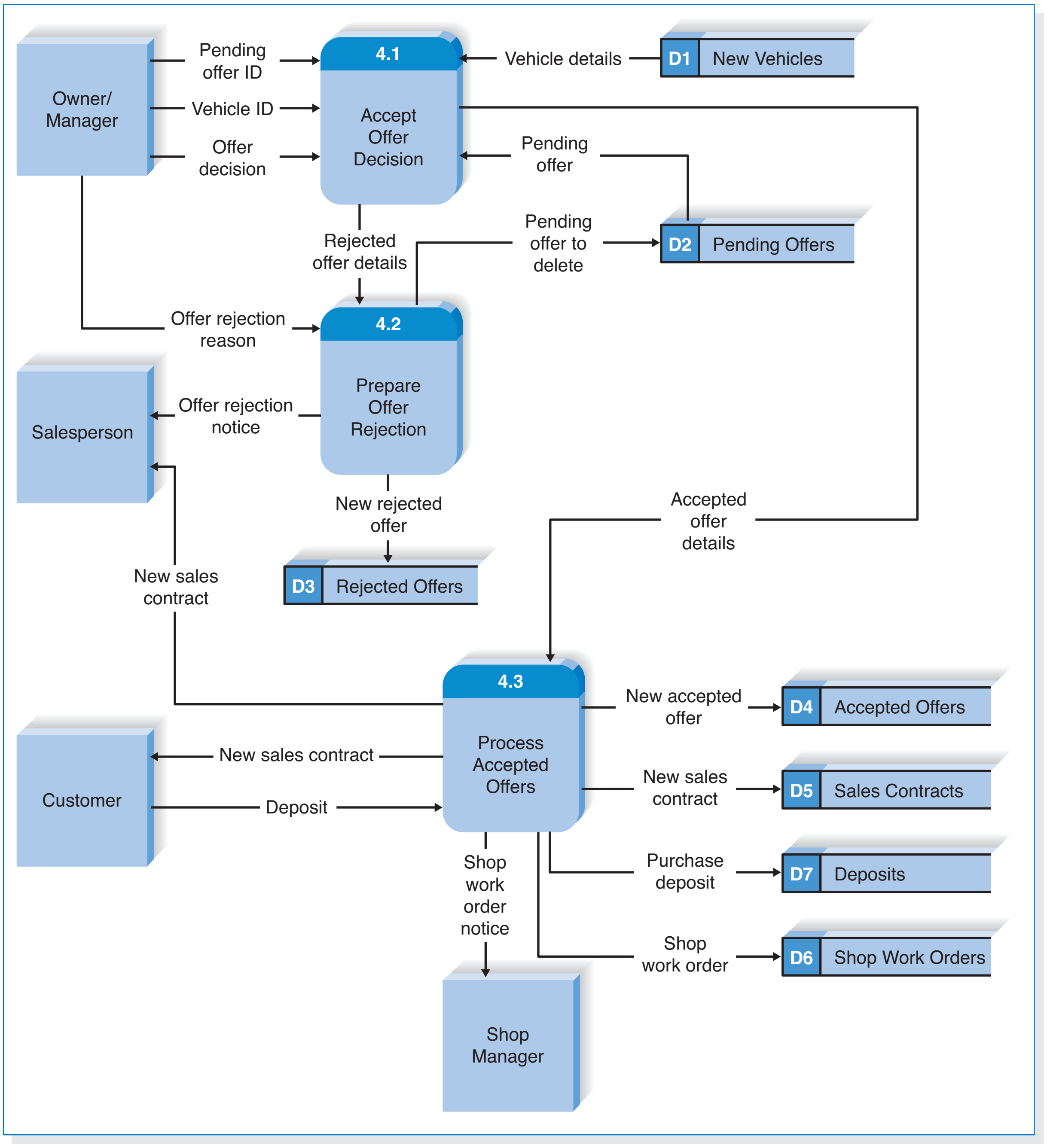


FIGURE 5-10
Holiday Travel Vehicles Process 4 (Evaluate Offer) Level 1 DFD

performed in order to complete the accepted offer (4.3). We have purposely collected these tasks into process 4.3 so that we can demonstrate the process of "exploding" that process into a level 2 DFD.

Process 4.3 in Figure 5-10 is clearly a process that is performing many tasks. When a process has numerous inflows and outflows, it is a good candidate for

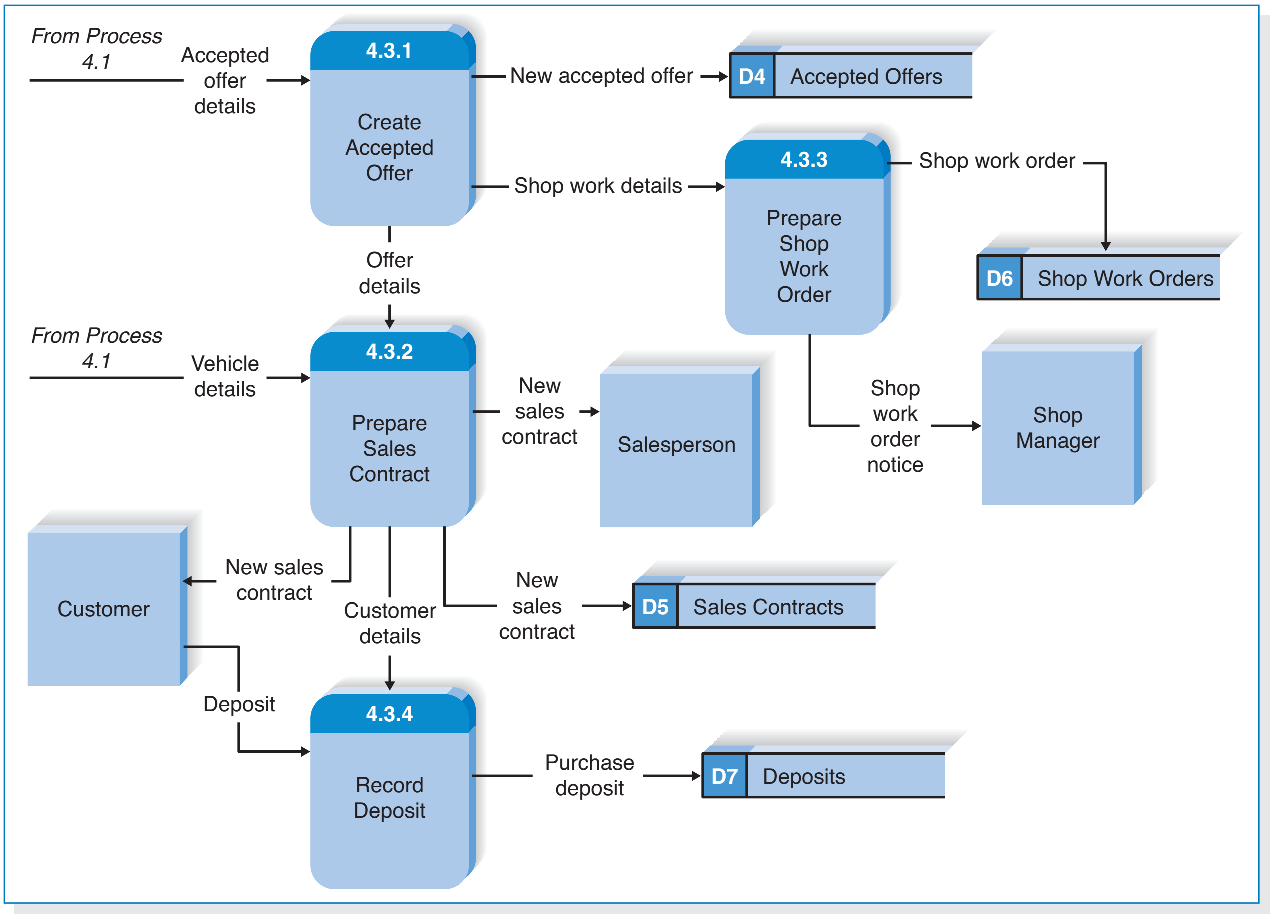


FIGURE 5-11
Holiday Travel Vehicles Process 4.3 (Process Accepted Offers) Level 2 DFD

decomposition into a lower-level DFD. As we mentioned previously, analysts often purposely "hide" details on a higher-level diagram to eliminate confusion and use a lower-level diagram to "reveal" the details. Figure 5-11 contains the level 2 DFD for process 4.3, Process Accepted Offers. As you can see by studying Figure 5-11, the parent process has been divided into four subprocesses, each focusing on one main task. Process 4.3.1 records the new Accepted Offer and provides data to two other processes. Process 4.3.3 uses the details from the accepted offer regarding options the customer wants to prepare a Shop Work Order and notify the shop manager. Process 4.3.2 uses the offer details and vehicle details to prepare the final sales contract. Finally, the customer's deposit is recorded (4.3.4).

Figure 5-12 depicts the third of our use cases from Figure 4-11 (Take Delivery of Vehicle). The salesperson obtains the Sales Contract (5.1). The salesperson verifies receipt of the customer's final payment, the customer signifies his/her acceptance of the vehicle, and the payment is recorded (5.2). Finally, the Final Sales Contract is recorded and a copy provided to the customer (5.3).

The process model is more likely to be drawn to the lowest level of detail for a to-be model if a traditional development process is used (i.e., not rapid

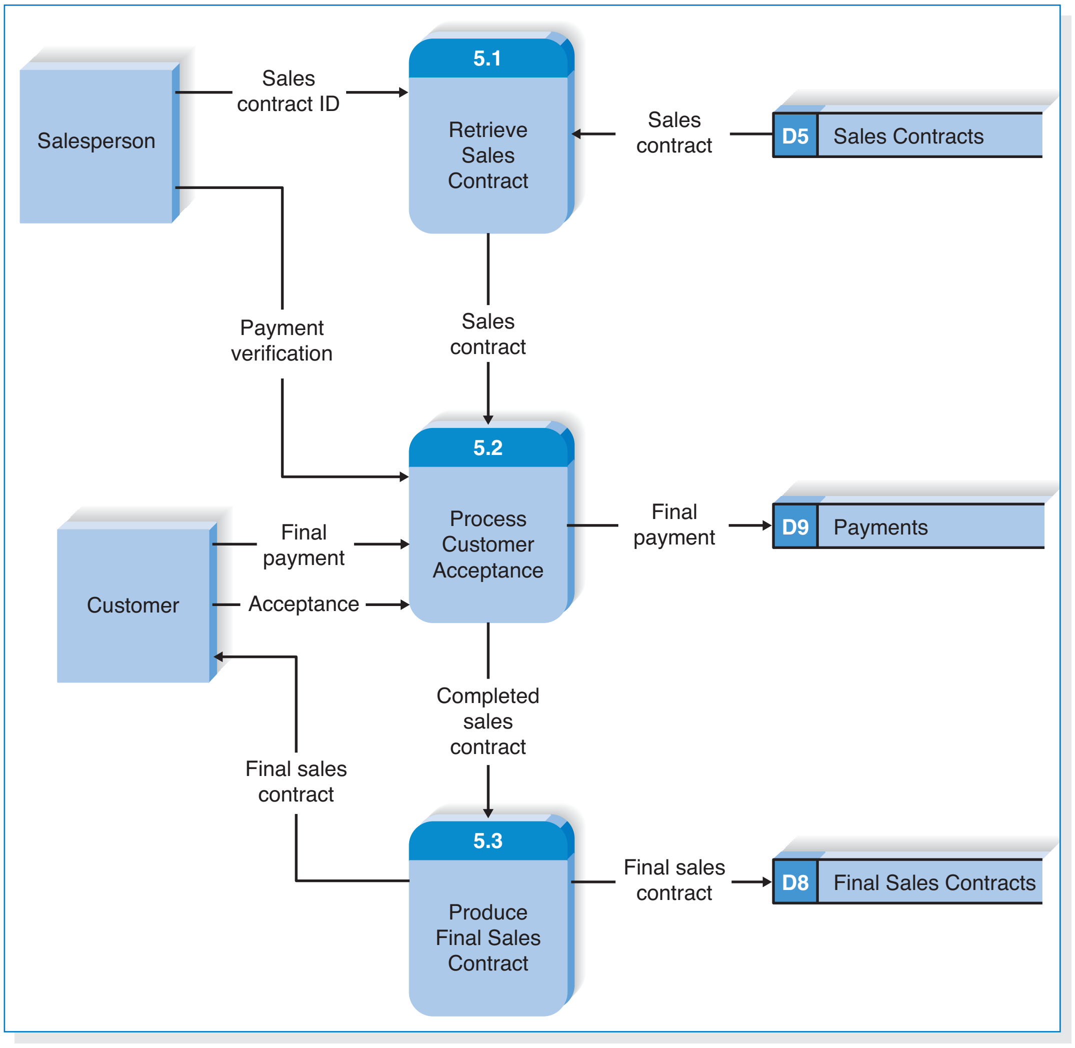


FIGURE 5-12
Holiday Travel Vehicles Process 5 (Take Delivery of Vehicle) Level 1 DFD

application development [RAD]; see Chapter 2) or if the system will be built by an external contractor. Without the complete level of detail, it may be hard to specify in a contract exactly what the system should do. If a RAD approach, which involves a lot of interaction with the users and, quite often, prototypes, is being used, we would be less likely to go to as low a level of detail, because the design will evolve through interaction with the users. In our experience, most systems go to only level 2 at most.

There is no requirement that all parts of the system must be decomposed to the same level of DFDs. Some parts of the system may be very complex and require many levels, whereas other parts of the system may be simpler and require fewer.

CONCEPTS IN ACTION

5-A U.S. Army and Marine Corps Battlefield Logistics

Shortly after the Gulf War in 1991 (Desert Storm), the U.S. Department of Defense realized that there were significant problems in its battlefield logistics systems that provided supplies to the troops at the division level and below. During the Gulf War, it had proved difficult for army and marine units fighting together to share supplies back and forth because their logistics computer systems would not easily communicate. The goal of the new system was to combine the army and marine corps logistics systems into one system to enable units to share supplies under battlefield conditions.

The army and marines built separate as-is process models of their existing logistics systems that had 165 processes for the army system and 76 processes for the marines. Both process models were developed over a 3-month time period and cost several million dollars to build, even though they were not intended to be comprehensive.

I helped them develop a model for the new integrated battlefield logistics system that would be used by both services (i.e., the to-be model). The initial process model contained 1,500 processes and went down to level 6 DFDs in many places. It took 3,300 pages to print. They realized that this model was too large to be useful. The project leader decided that level 4 DFDs was as far as the model would go, with additional information contained in the process descriptions. This reduced the model to 375 processes (800 pages) and made it far more useful.

Alan Dennis

Questions:

1. What are the advantages and disadvantages to setting a limit for the maximum depth for a DFD?
2. Is a level 4 DFD an appropriate limit?

Validating the Data Flow Diagrams

Once you have created a set of DFDs, it is important to check them for quality. Figure 5-13 provides a quick checklist for identifying the most common errors. There are two fundamentally different types of problems that can occur in DFDs: *syntax errors* and *semantics errors*. "Syntax," refers to the structure of the DFDs and whether the DFDs follow the rules of the DFD language. Syntax errors can be thought of as grammatical errors made by the analyst when he or she creates the DFD. "Semantics" refers to the meaning of the DFDs and whether they accurately describe the business process being modeled. Semantics errors can be thought of as misunderstandings by the analyst in collecting, analyzing, and reporting information about the system.

In general, syntax errors are easier to find and fix than are semantics errors, because there are clear rules that can be used to identify them (e.g., a process must have a name). Most CASE tools have syntax checkers that will detect errors within one page of a DFD in much the same way that word processors have spelling checkers and grammar checkers. Finding syntax errors that span several pages of a DFD (e.g., from a level 1 to a level 2 DFD) is slightly more challenging, particularly for consistent viewpoint, decomposition, and balance. Some CASE tools can detect balance errors, but that is about all. In most cases, analysts must carefully and painstakingly review every process, external entity, data flow, and data store on all DFDs by hand to make sure that they have a consistent viewpoint and that the decomposition and balance are appropriate.

Each data store is required to have at least one input and one output on some page of the DFD. In Figure 5-10, data store D1, New Vehicles, has only outputs and several data stores have only inputs. This situation is not necessarily an error. The analyst should check elsewhere in the DFDs to find where data is written to data

Syntax	
Within DFD	
Process	• Every process has a unique name that is an action-oriented verb phrase, a number, and a description.
	• Every process has at least one input data flow.
	• Every process has at least one output data flow.
	• Output data flows usually have different names than input data flows because the process changes the input into a different output in some way.
	• There are between three and seven processes per DFD.
Data Flow	• Every data flow has a unique name that is a noun, and a description.
	• Every data flow connects to at least one process.
	• Data flows only in one direction (no two-headed arrows).
	• A minimum number of data flow lines cross.
Data Store	• Every data store has a unique name that is a noun, and a description.
	• Every data store has at least one input data flow (which means to add new data or change existing data in the data store) on some page of the DFD.
	• Every data store has at least one output data flow (which means to read data from the data store) on some page of the DFD.
External Entity	• Every external entity has a unique name that is a noun, and a description.
	• Every external entity has at least one input or output data flow.
Across DFDs	
Context diagram	• Every set of DFDs must have one context diagram.
Viewpoint	• There is a consistent viewpoint for the entire set of DFDs.
Decomposition	• Every process is wholly and completely described by the processes on its children DFDs.
Balance	• Every data flow, data store, and external entity on a higher level DFD is shown on the lower-level DFD that decomposes it.
Semantics	
Appropriate Representation	• User validation
	• Role-play processes
Consistent Decomposition	• Examine lowest-level DFDs
Consistent Terminology	• Examine names carefully

FIGURE 5-13
Data Flow Diagram Quality Checklist

store D2 or read from the other data stores. All data stores should have at least one inflow and one outflow, but the flows may not be on the same diagram, so check other parts of the system. Another issue that arises is when the data store is utilized by other systems. In that case, data may be added to or used by a separate system. This is perfectly fine, but the analyst should investigate to verify that the required flows in and out of data stores exist somewhere.

In our experience, the most common syntax error that novice analysts make in creating DFDs is violating the law of conservation of data.[3] The first part of the law states the following:

1. *Data at rest stays at rest until moved by a process.*

[3] This law was developed by Prof. Dale Goodhue at the University of Georgia.

In other words, data cannot move without a process. Data cannot go to or come from a data store or an external entity without having a process to push it or pull it.

The second part of the law states the following:

2. *Processes cannot consume or create data.*

In other words, data only enters or leaves the system by way of the external entities. A process cannot destroy input data; all processes must have outputs. Drawing a process without an output is sometimes called a "black hole" error. Likewise, a process cannot create new data; it can transform data from one form to another, but it cannot produce output data without inputs. Drawing a process without an input is sometimes called a "miracle" error (because output data miraculously appear). There is one exception to the part of the law requiring inputs, but it is so rare that most analysts never encounter it.[4] Figure 5-14 shows some common syntax errors.

Looking at Figure 5-14, we will discuss each error in turn. First, we can see data flow X drawn directly from Entity A to Entity B. Remember that a data flow must either originate or terminate at a process; therefore, a process is required. Second, we see data flow Z retrieved from Data Store P and sent to Entity B. Processes should exist to transform the data in some way, so we usually modify the data flow names to reflect the changes made in the process. Third, we see that Data Store P has outputs but has no inputs. This is not necessarily an error, but does deserve the analyst's investigation. We should make sure that a process that adds data to Data Store P exists somewhere in the diagrams of the entire process model. Fourth, we can see that Process F receives data but has no outputs. This is considered a black hole since data is received but nothing is produced. Fifth, Process D is shown producing a data flow but has no inputs. This is termed a miracle process. Sixth, we see a two-headed arrow depicting Data Flow G between Process E and Process F. Data flows should not be drawn this way, but should flow only in one direction. Seventh, Data Store H receives Data Flow H as an input, but has no outputs. This issue may not be an error, but should be followed up by the analyst to ensure that the data that is stored in Data Store H is used some place in the process model; otherwise, there is no reason to store it. Finally, we see that a process should be involved between Entity A and Data Store H.

Generally speaking, semantics errors cause the most problems in system development. Semantics errors are much harder to find and fix because doing so requires a good understanding of the business process. And even then, what may be identified as an error may actually be a misunderstanding by the person reviewing the model. There are three useful checks to help ensure that models are semantically correct. (See Figure 5-13.)

The first check to ensure that the model is an appropriate representation is to ask the users to validate the model in a walk-through (i.e., the model is presented to the users, and they examine it for accuracy). A more powerful technique is for the users to role-play the process from the DFDs in the same way in which they role-played the use case. The users pretend to execute the process exactly as it is described in the DFDs. They start at the first process and attempt to perform it by using only the inputs specified and producing only the outputs specified. Then they move to the second process, and so on.

[4] The exception is a temporal process that issues a trigger output based on an internal time clock. Whenever some predetermined period elapses, the process produces an output. The timekeeping process has no inputs because the clock is internal to the process.

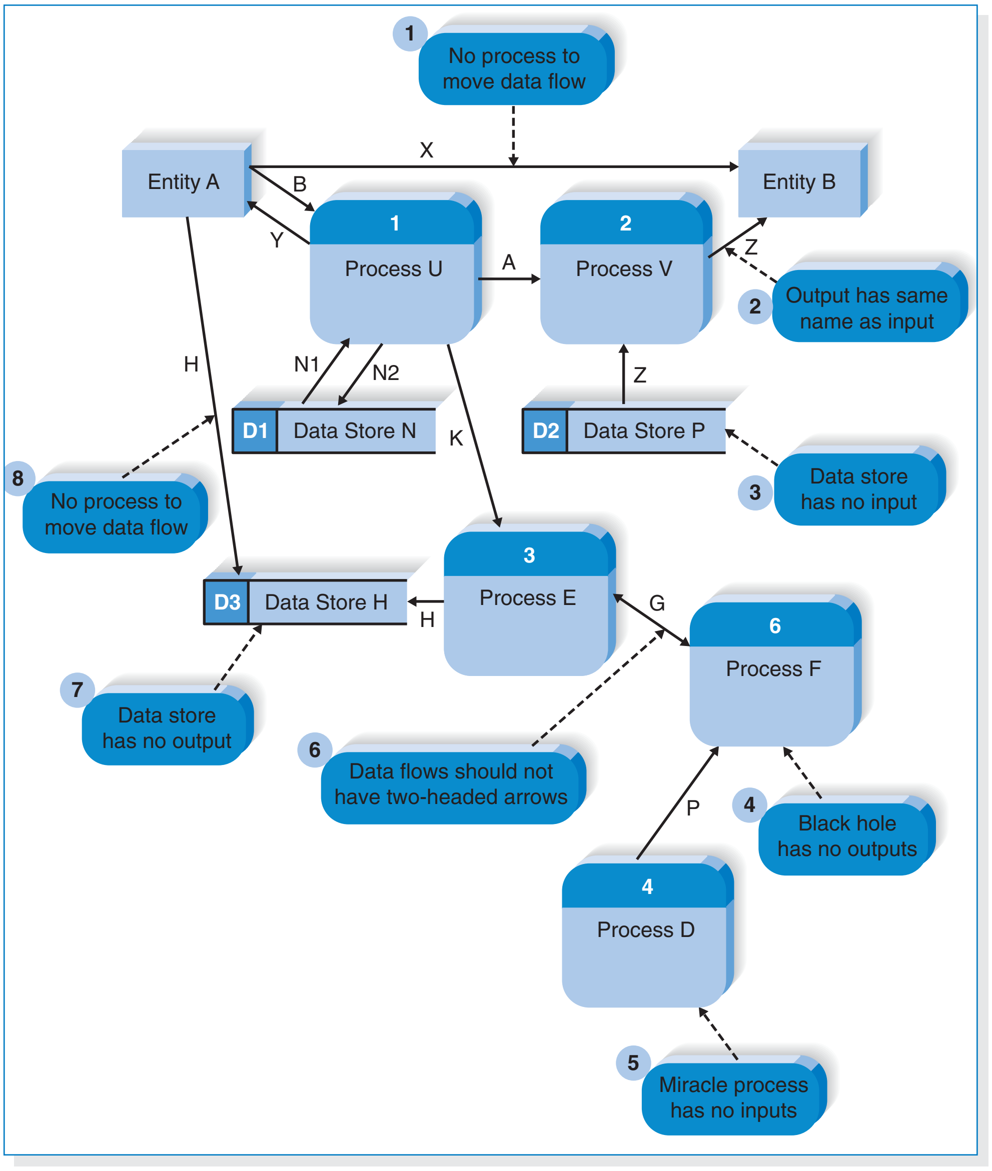

1
No process to move data flow
X
Entity A
Entity B
B
Y
1
Process U
A
2
Process V
Z
2
Output has same name as input
H
N1
N2
Z
D1 Data Store N
K
D2 Data Store P
3
Data store has no input
8
No process to move data flow
3
Process E
D3 Data Store H
H
G
6
Process F
7
Data store has no output
6
Data flows should not have two-headed arrows
P
4
Black hole has no outputs
4
Process D
5
Miracle process has no inputs

FIGURE 5-14
Some Common Errors

One of the most subtle forms of semantics error occurs when a process creates an output, but has insufficient inputs to create it. For example, in order to create water (H_2O), we need to have both hydrogen (H) and oxygen (O) present. The same is true of computer systems, in that the outputs of a process can be only combinations and transformations of its inputs. Suppose, for example, that we want to record an order; we need the customer name and mailing address and the quantities and prices for the items the customer is ordering. We need information from the customer data store (e.g., address) and information from the items data store (e.g., price). We cannot draw a process that produces an output order data flow without inputs from these two data stores. Role-playing with strict adherence to the inputs and outputs in a model is one of the best ways to catch this type of error.

A second semantics error check is to ensure consistent decomposition, which can be tested by examining the lowest-level processes in the DFDs. In most circumstances, all processes should be decomposed to the same level of detail—which is not the same as saying the same number of levels. For example, suppose that we were modeling the process of driving to work in the morning. One level of detail would be to say the following: (1) Enter car; (2) start car; (3) drive away. Another level of detail would be to say the following: (1) Unlock car; (2) sit in car; (3) buckle seat belt; and so on. Still another level would be to say the following: (1) Remove key from pocket; (2) insert key in door lock; (3) turn key; and so on. None of these is inherently better than another, but barring unusual circumstances, it is usually best to ensure that all processes at the very bottom of the model provide the same consistent level of detail.

Likewise, it is important to ensure that the terminology is consistent throughout the model. The same item may have different names in different parts of the organization, so one person's "sales order" may be another person's "customer order." Likewise, the same term may have different meanings; for example, "ship date" may mean one thing to the sales representative taking the order (e.g., promised date) and something else to the warehouse (e.g., the actual date shipped). Resolving these differences before the model is finalized is important in ensuring that everyone who reads the model or who uses the information system built from the model has a shared understanding.

APPLYING THE CONCEPTS AT TUNE SOURCE

Creating the Context Diagram

The project team began by creating the context diagram. They read through the summary area of the three major use cases in Figure 4-14 to find the major inputs and outputs.

The team noticed that the majority of data flow interactions are with the customers who are using the Web site to browse music selections and make download purchases. There will be an interaction with the payment clearinghouse entity that will handle payment verification and processing of purchases. Finally, although it is not obvious from the use cases, the marketing managers will be using sales information from the system to design and implement promotional campaigns. The team used the major inflows and outflows from the use cases and developed the context diagram shown in Figure 5-15.

Creating Data Flow Diagram Fragments

The next step was to create one DFD fragment for each use case. This was done by drawing the process in the middle of the page, making sure that the process number and name were appropriate, and connecting all the input and output data flows to it. Unlike the context diagram, the DFD fragment includes data flows to external entities and to internal data stores.

The completed DFD fragments are shown in Figure 5-16. Before looking at the figure, take a minute and draw them on your own. There are many good ways to draw these fragments. In fact, there are many "right" ways to create use cases and DFDs. Notice that on the DFD fragment for process 3 we have shown a dotted line inflow labeled "Time to determine promotions" into the process. Recall that we

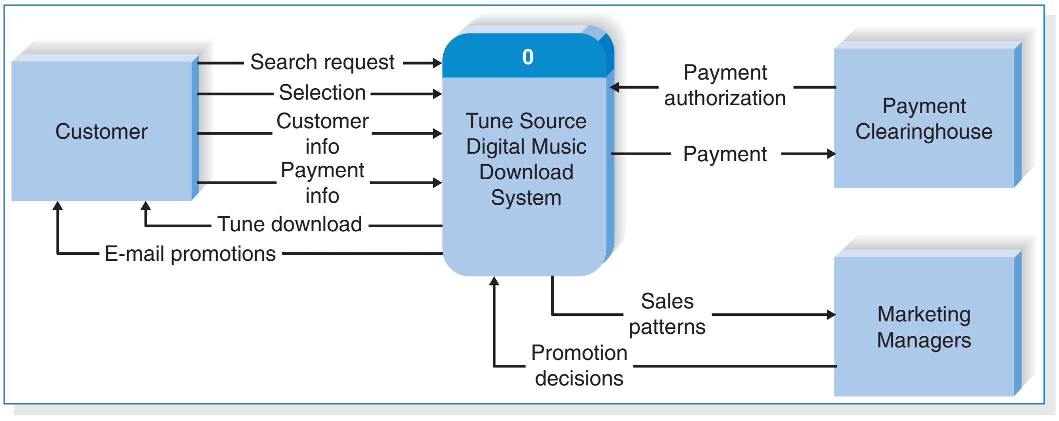


FIGURE 5-15
Tune Source Context Diagram

specified that the use case, Promote Tunes, was a temporal use case, triggered when it was time to update promotions and specials. The dotted line flow into process 3 in Figure 5-16 is sometimes referred to as a *control flow* and is commonly used to represent a time-based trigger for an event.

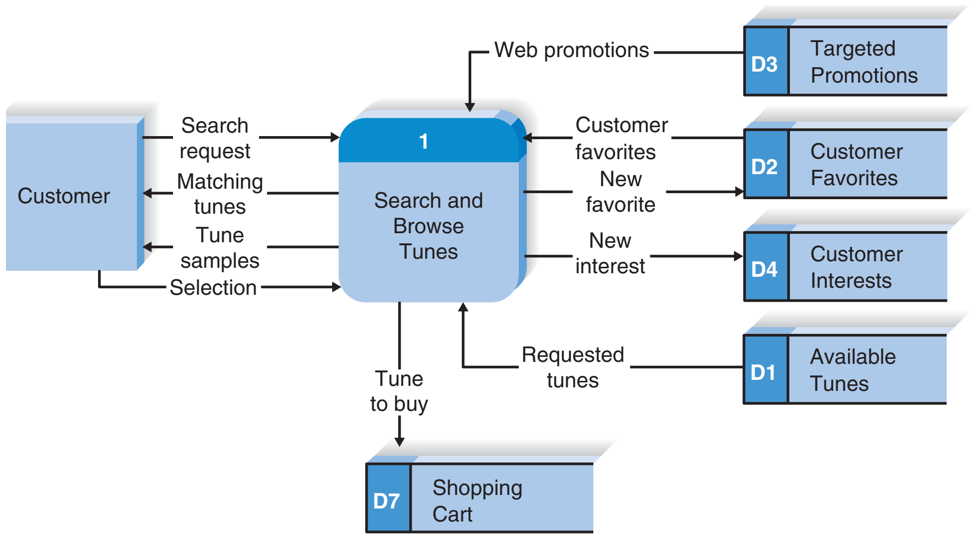
Creating the Level 0 Data Flow Diagram

The next step was to create the level 0 DFD by integrating the DFD fragments, which proved to be anticlimactic. The team simply took the DFD fragments and drew them together on one piece of paper. Although it sometimes is challenging to arrange all the DFD fragments on one piece of paper, it was primarily a mechanical exercise (Figure 5-17). Compare the level 0 diagram with the context diagram in Figure 5-15. Are the two DFDs balanced? Notice the additional detail contained in the level 0 diagram.

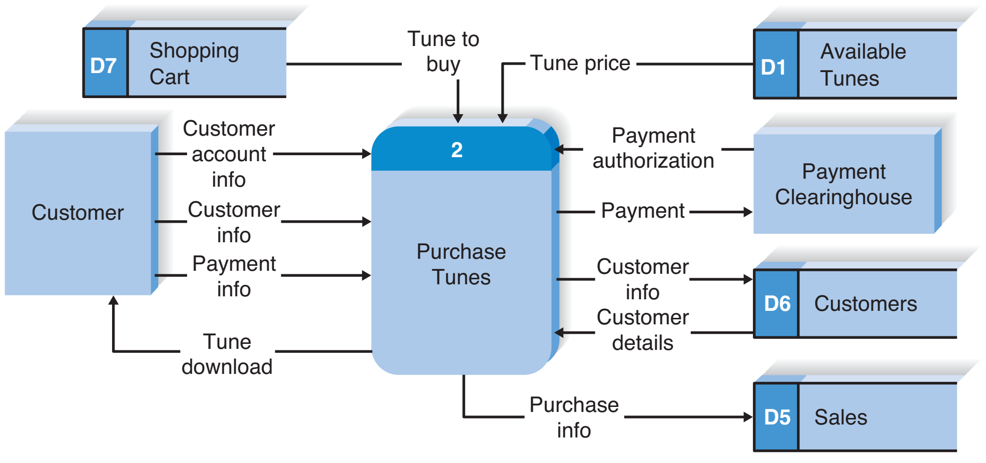
A careful review of the data stores in Figure 5-17 reveals that every one has both an inflow and an outflow, with one exception, D1:Available Tunes. D1:Available Tunes is read by two processes, but is not written to by any process shown. This violation of DFD syntax needs to be investigated by the team because it may be a serious oversight. As we will explain later, in this situation we need to create a process specifically for adding, modifying, and deleting data in D1:Available Tunes. "Administrative" processes such as this are often overlooked, as we initially focus on business requirements only, but will need to be added before the system is complete.

Creating Level 1 Data Flow Diagrams (and Below)

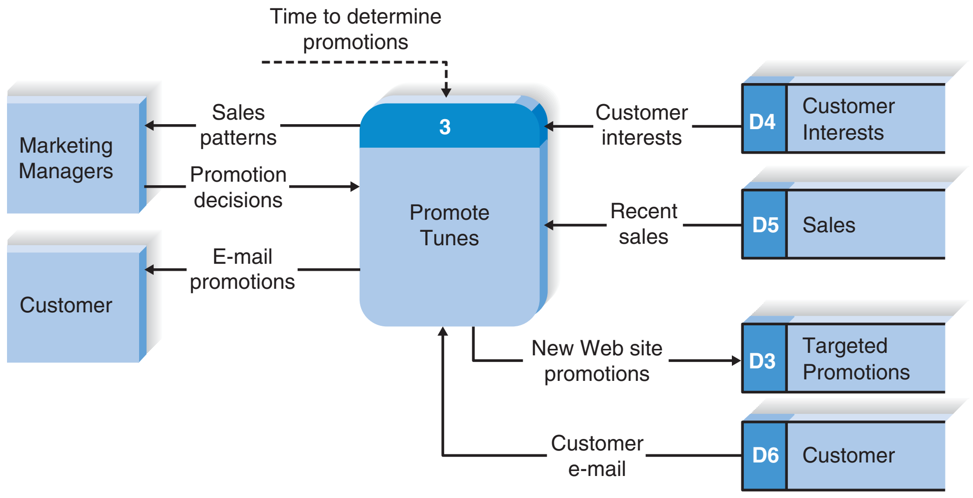
The next step was to create the level 1 DFDs for those processes that could benefit from them. The analysts started with the first use case (search and browse tunes) and started to draw a DFD for the individual steps it contained. The steps in the use case were straightforward, but as is common, the team had to choose names and numbers for the processes and to add input data flows from data stores not present



(a) Tune Source Process 1 DFD Fragment



(b) Tune Source Process 2 DFD Fragment



(c) Tune Source Process 3 DFD Fragment

FIGURE 5-16 Tune Source DFD Fragments

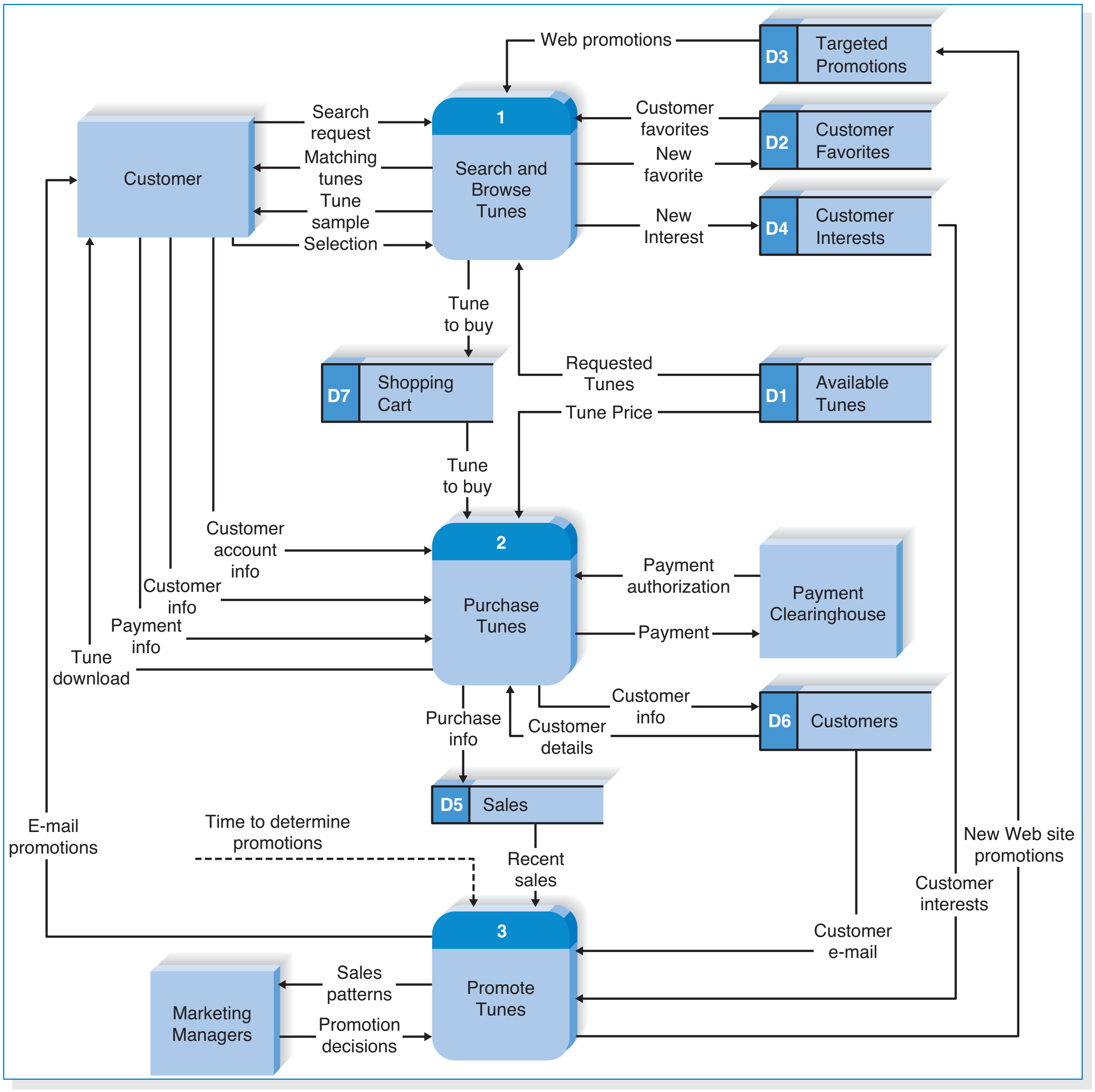


FIGURE 5-17
Tune Source Level 0 DFD

in the use case. The team also discovered the omission of a data flow, customer interests, from the use case. See Figure 5-18.

The team also developed level 1 diagrams for the other processes, using the major steps outlined in their respective use cases. Some adjustments were made from the steps as shown in the use cases, but the team followed the steps fairly closely. See Figures 5-19 and 5-20, and compare them with their use cases shown in Figure 4-14.

YOUR TURN

5-1 Campus Housing

Draw a context diagram, a level 0 DFD, and a set of level 1 DFDs (where needed) for the campus housing use cases that you developed for the Your Turn 4-1 box in Chapter 4.

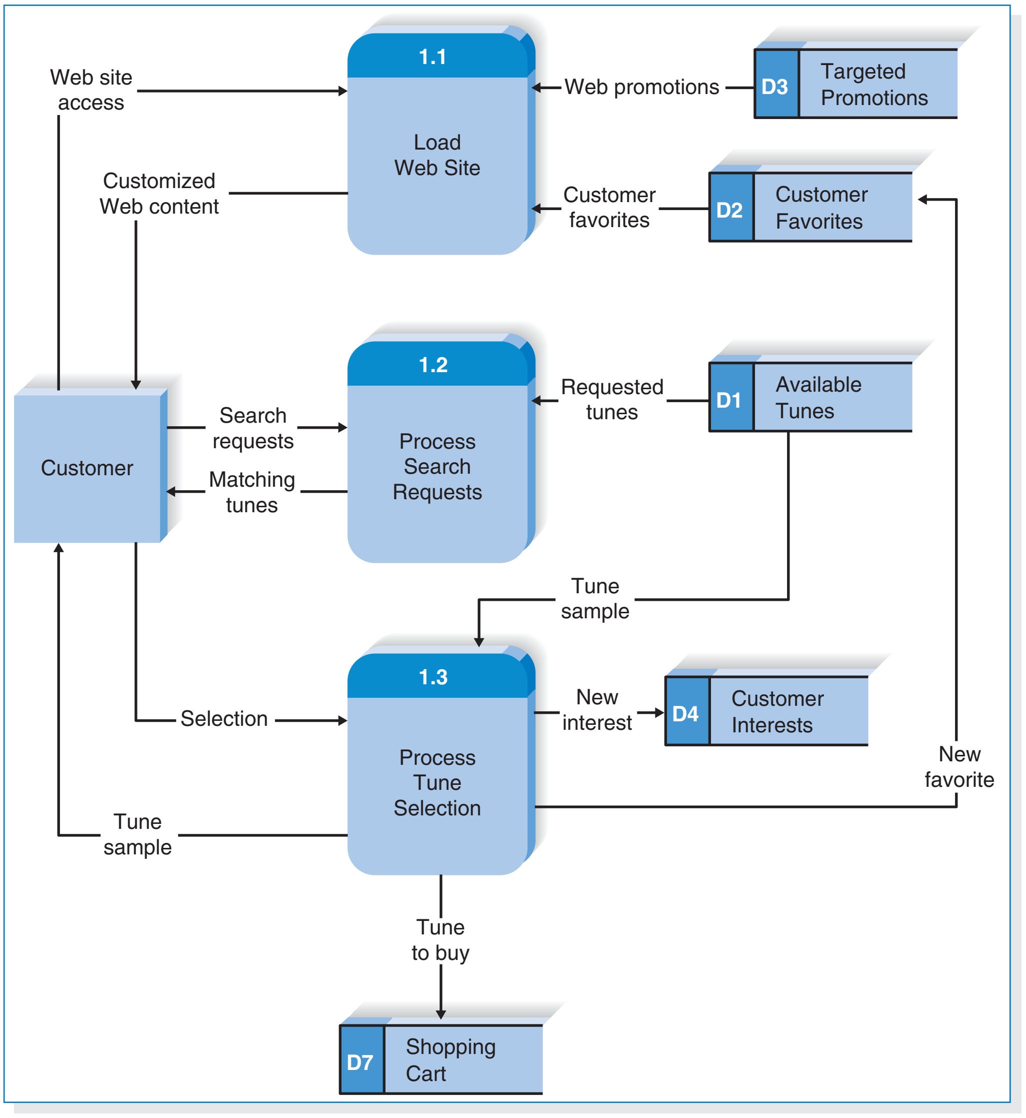


FIGURE 5-18
Level 1 DFD for Tune Source Process 1: Search and Browse Tunes

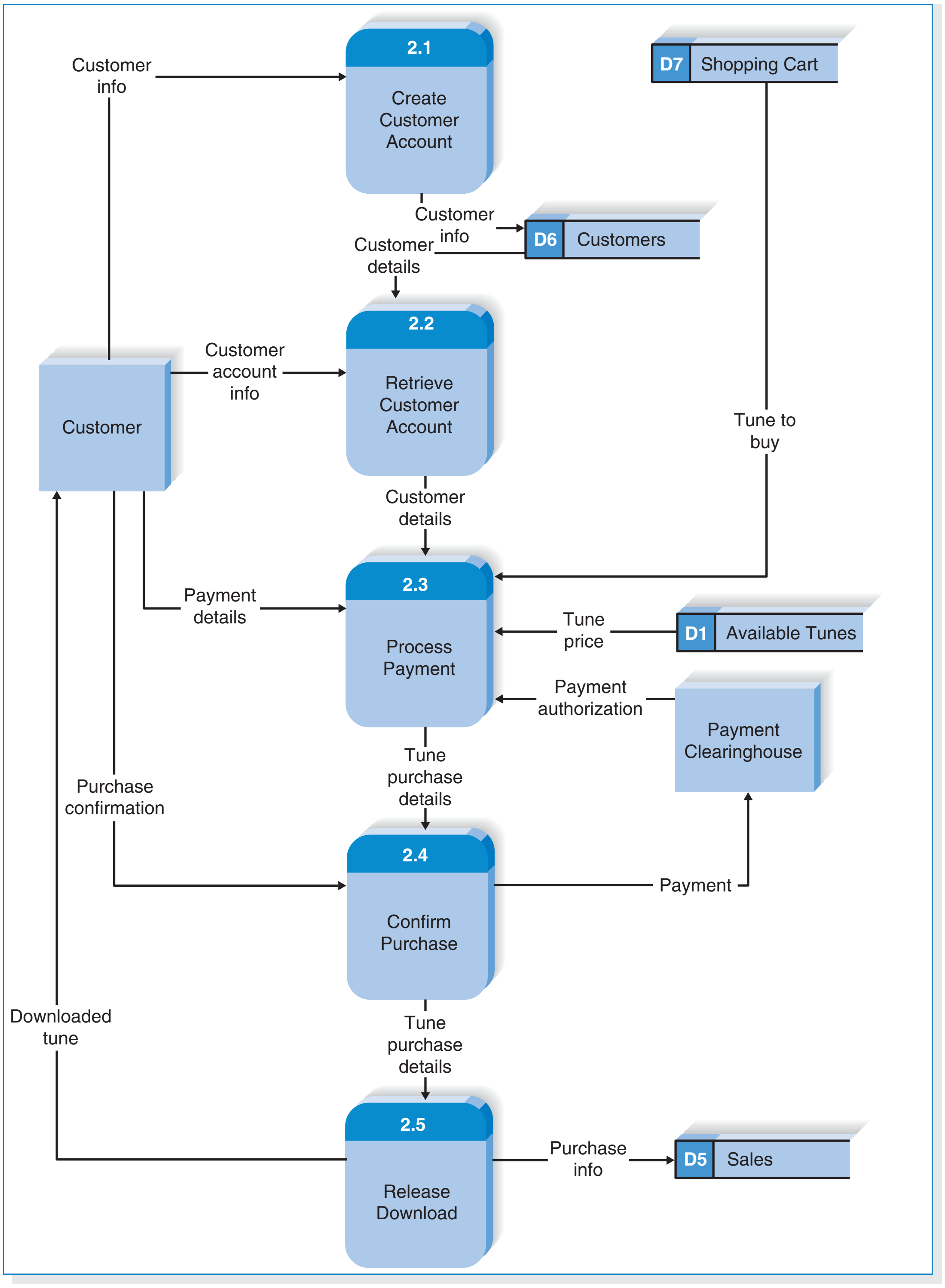


FIGURE 5-19
Level 1 DFD for Tune Source Process 2: Purchase Tunes

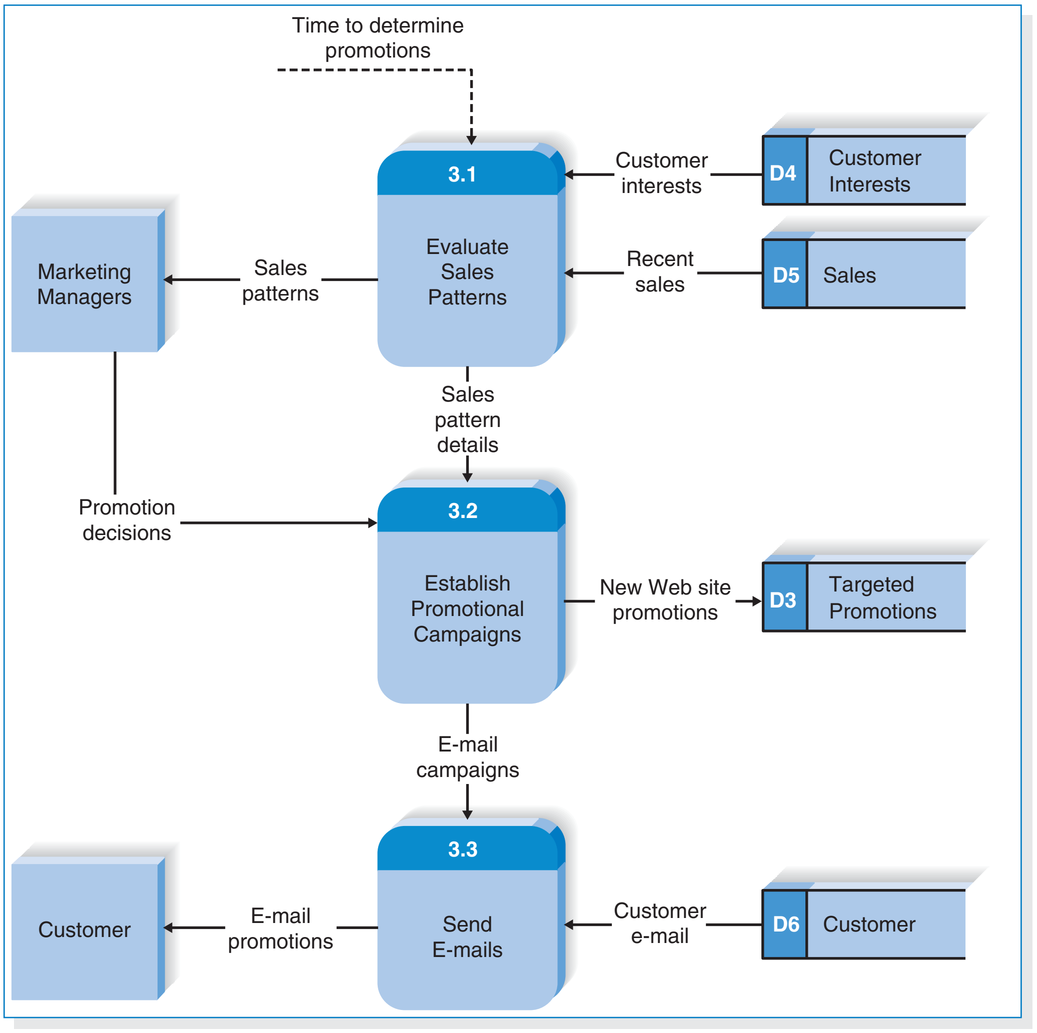


FIGURE 5-20
Level 1 DFD for Tune Source Process 3: Promote Tunes

As we specified in Figure 5-2, every data store should have one or more input data flows and one or more output data flows. A careful look at Figure 5-18, however, reveals that D1: Available Tunes has output data flows, but no input data flows. This data store is the main repository for the digital music library, so, clearly, it has a central role in our system. The team will need to be sure to create an administrative process for maintaining this data store: adding new information to it, modifying its existing contents, and removing information from it. These administrative tasks are sometimes omitted from the business-oriented tasks listed in the use cases, so it is up to the team to ensure that processes are included to add, modify, and delete the contents of data stores somewhere in the system. Simply checking the DFD syntax (all data stores must have at least one input data flow and at least one output data flow) helped discover this omission in the process model.

Although it would have been possible to decompose several of the processes on the level 1 DFDs into more detail on a level 2 diagram, the project team decided that that step was not necessary. Instead, they made sure that the process descriptions they created in the CASE repository for each of the processes was very detailed. This detail would be critical in developing the data models and designing the user interface and programs during the design phase.

Validating the Data Flow Diagrams

The final set of DFDs was validated by the project team and then by Carly and her marketing team in a final JAD meeting. There was general approval of the customer-facing processes (process 1 and process 2) and considerable discussion of the specific information that would be required to help the marketing team make its promotional campaign decisions (process 3). This information was recorded in the CASE repository so that it could be recalled during the development of the system's data model, the subject of our next chapter.

SUMMARY

Data Flow Diagram Syntax

Four symbols are used on data flow diagrams (processes, data flows, data stores, and external entities). A process is an activity that does something. Each process has a name (a verb phrase), a description, and a number that shows where it is in relation to other processes and to its children processes. Every process must have at least one output and usually has at least one input. A data flow is a piece of data or an object and has a name (a noun) and a description and either starts or ends at a process (or both). A data store is a manual or computer file, and it has a number, a name (a noun), and at least one input data flow and one output data flow (unless the data store is created by a process external to the data flow diagram [DFD]). An external entity is a person, organization, or system outside the scope of the system and has a name (a noun) and a description. Every set of DFDs starts with a context diagram and a level 0 DFD and has numerous level 1 DFDs, level 2 DFDs, and so on. Every element on the higher-level DFDs (i.e., data flows, data stores, and external entities) must appear on lower-level DFDs, or else they are not balanced.

Creating Data Flow Diagrams

The DFDs are created from the use cases. First, the team builds the context diagram that shows all the external entities and the data flows into and out of the system from them. Second, the team creates DFD fragments for each use case that show how the use case exchanges data flows with the external entities and data stores. Third, these DFD fragments are organized into a level 0 DFD. Fourth, the team develops level 1 DFDs on the basis of the steps within each use case to better explain how they operate. Fifth, the team validates the set of DFDs to make sure that they are complete and correct and contain no syntax or semantics errors. Analysts seldom create DFDs perfectly the first time, so iteration is important in ensuring that both single-page and multipage DFDs are clear and easy to read.

KEY TERMS

Action statement	Decomposition	Parent
Balancing	DFD fragment	Physical model
Bundle	External entity	Process model
Case statement	For statement	Process
Children	If statement	Semantics error
Context diagram	Iteration	Structured English
Data flow	Layout	Syntax error
Data flow diagram (DFD)	Level 0 DFD	Viewpoint
Data store	Level 1 DFD	While statement
Decision table	Level 2 DFD	
Decision tree	Logical process model	

QUESTIONS

1. What is a process model? What is a data flow diagram? Are the two related? If so, how?
2. Distinguish between logical process models and physical process models.
3. Define what is meant by a *process* in a process model. How should a process be named? What information about a process should be stored in the CASE repository?
4. Define what is meant by a *data flow* in a process model. How should a data flow be named? What information about a data flow should be stored in the CASE repository?
5. Define what is meant by a *data store* in a process model. How should a data store be named? What information about a data store should be stored in the CASE repository?
6. Define what is meant by an *external entity* in a process model. How should an external entity be named? What information about an external entity should be stored in the CASE repository?
7. Why is a process model typically composed of a set of DFDs? What is meant by decomposition of a business process?
8. Explain the relationship between a DFD context diagram and the DFD level 0 diagram.
9. Explain the relationship between a DFD level 0 diagram and DFD level 1 diagram(s).
10. Discuss how the analyst knows how to stop decomposing the process model into more and more levels of detail.
11. Suppose that a process on a DFD is numbered 4.3.2. What level diagram contains this process? What is this process's parent process?
12. Explain the use of structured English in process descriptions.
13. Why would one use a decision tree and/or decision table in a process description?
14. Explain the process of balancing a set of DFDs.
15. How are mutually exclusive data flows (i.e., alternative paths through a process) depicted in DFDs?
16. Discuss several ways to verify the correctness of a process model.
17. Identify three typical syntax errors commonly found in DFDs.
18. What is meant by a DFD semantic error? Provide an example.
19. Creating use cases when working with users is a recent development in systems analysis practice. Why is the trend today to employ use cases in user interviews or JAD sessions?
20. How can you make a DFD easier to understand? (Think first about how to make one difficult to understand.)
21. Suppose that your goal is to create a set of DFDs. How would you begin an interview with a knowledgeable user? How would you begin a JAD session?

EXERCISES

A. Draw a level 0 data flow diagram (DFD) for the process of buying glasses in Exercise A, Chapter 4.
B. Draw a level 0 data flow diagram (DFD) for the dentist office system in Exercise B, Chapter 4.
C. Draw a level 0 data flow diagram (DFD) for the university system in Exercise D, Chapter 4.
D. Draw a level 0 data flow diagram (DFD) for the real estate system in Exercise E, Chapter 4.
E. Draw a level 0 data flow diagram (DFD) for the video store system in Exercise F, Chapter 4.
F. Draw a level 0 data flow diagram (DFD) for the health club system in Exercise G, Chapter 4.
G. Draw a level 0 data flow diagram (DFD) for the Picnics R Us system in Exercise H, Chapter 4.
H. Draw a level 0 data flow diagram (DFD) for the Of-the-Month Club system in Exercise I, Chapter 4.
I. Draw a level 0 data flow diagram (DFD) for the university library system in Exercise J, Chapter 4.

MINICASES

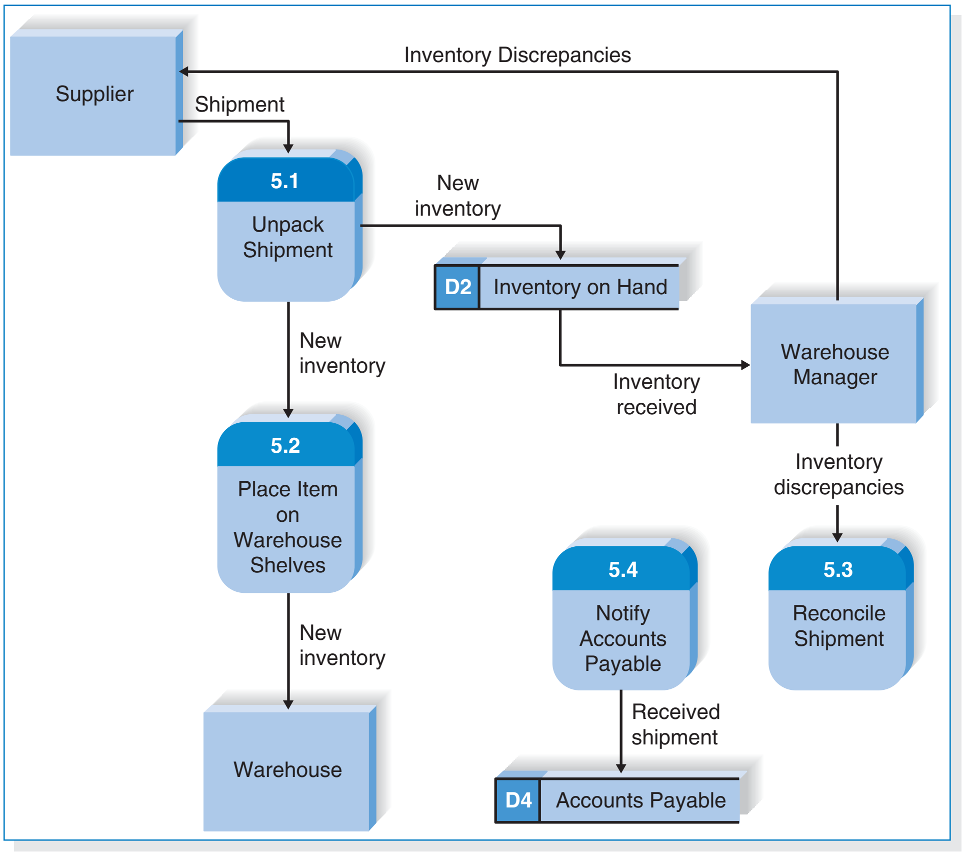
1. The Hatcher Company is in the process of developing a new inventory management system. One of the event handling processes in that system is Receive Supplier Shipments. The (inexperienced) systems analyst on the project has spent time in the warehouse observing this process and developed the following list of activities that are performed: getting the new order in the warehouse, unpacking the boxes, making sure that all the ordered items were actually received, putting the items on the correct shelves, dealing with the supplier to reconcile any discrepanices, adjusting the inventory quantities on hand, and passing along the shipment information to the accounts payable office. He also created the accompanying Level 1 data flow diagram for this process. Unfortunately, this DFD has numerous syntax and semantic errors. Identify the errors. Redraw the DFD to more correctly represent the Receive Supplier Shipments process.

2. Professional and Scientific Staff Management (PSSM) is a unique type of temporary staffing agency. Many organizations today hire highly skilled technical employees on a short-term, temporary basis to assist with special projects or to provide a needed technical skill. PSSM negotiates contracts with its client companies in which it agrees to provide temporary staff in specific job categories for a specified cost. For example, PSSM has a contract with an oil and gas exploration company, in which it agrees to supply geologists with at least a master's degree for $5000 per week. PSSM has contracts with a wide range of companies and can place almost any type of professional or scientific staff members, from computer programmers to geologists to astrophysicists.

When a PSSM client company determines that it will need a temporary professional or scientific employee, it issues a staffing request against the contract it had previously negotiated with PSSM. When a staffing request is received by PSSM's contract manager, the contract number referenced on the staffing request is entered into the contract database. Using information from the database, the contract manager reviews the terms and conditions of the contract and determines whether the staffing request is valid. The staffing request is valid if the contract has not expired, the type of professional or scientific employee requested is listed on the original contract, and the requested fee falls within the negotiated fee range. If the staffing request is not valid, the contract manager sends the staffing request back to the client with a letter stating why the staffing request cannot be filed, and a copy of the letter is filed. If the staffing request is valid, the contract manager enters the staffing request into the staffing request database, as an outstanding staffing request. The staffing request is then sent to the PSSM placement department.

In the placement department, the type of staff member, experience, and qualifications requested on the staffing request are checked against the database of available professional and scientific staff. If a qualified individual is found, he or she is marked "reserved" in the staff database. If a qualified individual cannot be found in the database or is not immediately available, the placement department creates a memo that explains the inability to meet the staffing request and attaches it to the staffing request. All staffing requests are then sent to the arrangements department.

In the arrangement department, the prospective temporary employee is contacted and asked to agree to the placement. After the placement details have been worked out and agreed to, the staff member is marked "placed" in the staff database. A copy of the staffing



Hatcher Company Inventory Management System Level 1 DFD

request and a bill for the placement fee is sent to the client. Finally, the staffing request, the "unable to fill" memo (if any), and a copy of the placement fee bill is sent to the contract manager. If the staffing request was filled, the contract manager closes the open staffing request in the staffing request database. If the staffing request could not be filled, the client is notified. The staffing request, placement fee bill, and "unable to fill" memo are then filed in the contract office.

a. Develop a use case for each of the major processes just described.
b. Create the context diagram for the system just described.
c. Create the DFD fragments for each of the four use cases outlined in part a, and then combine them into the level 0 DFD.
d. Create a level 1 DFD for the most complicated use case.