12.6 THE MECHANICS OF STRUCTURED ANALYSIS

In the previous section, we discussed basic and extended notation for structured analysis. To be used effectively in software requirements analysis, this notation must be combined with a set of heuristics that enable a software engineer to derive a good analysis model. To illustrate the use of these heuristics, an adapted version of the Hatley and Pirbhai [HAT87] extensions to the basic structured analysis notation will be used throughout the remainder of this chapter.

In the sections that follow, we examine each of the steps that should be applied to develop complete and accurate models using structured analysis. Through this discussion, the notation introduced in Section 12.4 will be used, and other notational forms, alluded to earlier, will be presented in some detail.

12.6.1 Creating an Entity/Relationship Diagram

The entity/relationship diagram enables a software engineer to fully specify the data objects that are input and output from a system, the attributes that define the properties of these objects, and their relationships. Like most elements of the analysis model, the ERD is constructed in an iterative manner. The following approach is taken:

1. During requirements elicitation, customers are asked to list the “things” that the application or business process addresses. These “things” evolve into a list of input and output data objects as well as external entities that produce or consume information.

2. Taking the objects one at a time, the analyst and customer define whether or not a connection (unnamed at this stage) exists between the data object and other objects.

3. Wherever a connection exists, the analyst and the customer create one or more object/relationship pairs.

4. For each object/relationship pair, cardinality and modality are explored.

5. Steps 2 through 4 are continued iteratively until all object/relationships have been defined. It is common to discover omissions as this process continues. New objects and relationships will invariably be added as the number of iterations grows.

6. The attributes of each entity are defined.

7. An entity relationship diagram is formalized and reviewed.

8. Steps 1 through 7 are repeated until data modeling is complete.

To illustrate the use of these basic guidelines, the SafeHome security system example, discussed in Chapter 11, will be used. Referring back to the processing narrative
for SafeHome (Section 11.3.3), the following (partial) list of “things” are relevant to the problem:

- homeowner
- control panel
- sensors
- security system
- monitoring service

Taking these “things” one at a time, connections are explored. To accomplish this, each object is drawn and lines connecting the objects are noted. For example, referring to Figure 12.18, a direct connection exists between homeowner and control panel, security system, and monitoring service. A single connection exists between sensor and security system, and so forth.

Once all connections have been defined, one or more object/relationship pairs are identified for each connection. For example, the connection between sensor and security system is determined to have the following object/relationship pairs:

- security system monitors sensor
- security system enables/disables sensor
- security system tests sensor
- security system programs sensor

Each of these object/relationship pairs is analyzed to determine cardinality and modality. For example, considering the object/relationship pair security system monitors sensor, the cardinality between security system and sensor is one to many. The modality is one occurrence of security system (mandatory) and at least one occurrence of sensor (mandatory). Using the ERD notation introduced in Section 12.3, the
connecting line between security system and sensor would be modified as shown in Figure 12.19. Similar analysis would be applied to all other data objects.

Each object is studied to determine its attributes. Since we are considering the software that must support SafeHome, the attributes should focus on data that must be stored to enable the system to operate. For example, the sensor object might have the following attributes: sensor type, internal identification number, zone location, and alarm level.

### 12.6.2 Creating a Data Flow Model

The data flow diagram enables the software engineer to develop models of the information domain and functional domain at the same time. As the DFD is refined into greater levels of detail, the analyst performs an implicit functional decomposition of the system, thereby accomplishing the fourth operational analysis principle for function. At the same time, the DFD refinement results in a corresponding refinement of data as it moves through the processes that embody the application.

A few simple guidelines can aid immeasurably during derivation of a data flow diagram: (1) the level 0 data flow diagram should depict the software/system as a single bubble; (2) primary input and output should be carefully noted; (3) refinement should begin by isolating candidate processes, data objects, and stores to be represented at the next level; (4) all arrows and bubbles should be labeled with meaningful names; (5) information flow continuity must be maintained from level to level, and (6) one bubble at a time should be refined. There is a natural tendency to overcomplicate the data flow diagram. This occurs when the analyst attempts to show too much detail too early or represents procedural aspects of the software in lieu of information flow.

Again considering the SafeHome product, a level 0 DFD for the system is shown in Figure 12.20. The primary external entities (boxes) produce information for use by the system and consume information generated by the system. The labeled arrows represent data objects or data object type hierarchies. For example, user commands and data encompasses all configuration commands, all activation/deactivation...
commands, all miscellaneous interactions, and all data that are entered to qualify or expand a command.

The level 0 DFD is now expanded into a level 1 model. But how do we proceed? A simple, yet effective approach is to perform a "grammatical parse" on the processing narrative that describes the context level bubble. That is, we isolate all nouns (and noun phrases) and verbs (and verb phrases) in the SafeHome narrative originally presented in Chapter 11. To illustrate, we again reproduce the processing narrative underlining the first occurrence of all nouns and italicizing the first occurrence of all verbs.

SafeHome software enables the homeowner to configure the security system when it is installed, monitors all sensors connected to the security system, and interacts with the homeowner through a keypad and function keys contained in the SafeHome control panel shown in Figure 11.2.

During installation, the SafeHome control panel is used to "program" and configure the system. Each sensor is assigned a number and type, a master password is programmed for arming and disarming the system, and telephone numbers are input for dialing when a sensor event occurs.

When a sensor event is recognized, the software invokes an audible alarm attached to the system. After a delay time that is specified by the homeowner during system configuration activities, the software dials a telephone number of a monitoring service, provides information about the location, reporting the nature of the event that has been detected. The telephone number will be redialed every 20 seconds until telephone connection is obtained.

All interaction with SafeHome is managed by a user-interaction subsystem that reads input provided through the keypad and function keys, displays prompting messages on the LCD display, displays system status information on the LCD display. Keyboard interaction takes the following form . . .

Referring to the "grammatical parse," a pattern begins to emerge. All verbs are SafeHome processes; that is, they may ultimately be represented as bubbles in a sub-

---

3 It should be noted that nouns and verbs that are synonyms or have no direct bearing on the modeling process are omitted.
sequent DFD. All nouns are either external entities (boxes), data or control objects (arrows), or data stores (double lines). Note further that nouns and verbs can be attached to one another (e.g., sensor is assigned number and type). Therefore, by performing a grammatical parse on the processing narrative for a bubble at any DFD level, we can generate much useful information about how to proceed with the refinement to the next level. Using this information, a level 1 DFD is shown in Figure 12.21.

The context level process shown in Figure 12.20 has been expanded into six processes derived from an examination of the grammatical parse. Similarly, the information flow between processes at level 1 has been derived from the parse.

It should be noted that information flow continuity is maintained between levels 0 and 1. Elaboration of the content of inputs and output at DFD levels 0 and 1 is postponed until Section 12.7.

The processes represented at DFD level 1 can be further refined into lower levels. For example, the process monitor sensors can be refined into a level 2 DFD as shown in Figure 12.22. Note once again that information flow continuity has been maintained between levels.
The refinement of DFDs continues until each bubble performs a simple function. That is, until the process represented by the bubble performs a function that would be easily implemented as a program component. In Chapter 13, we discuss a concept, called cohesion, that can be used to assess the simplicity of a given function. For now, we strive to refine DFDs until each bubble is “single-minded.”

12.6.3 Creating a Control Flow Model

For many types of data processing applications, the data model and the data flow diagram are all that is necessary to obtain meaningful insight into software requirements. As we have already noted, however, a large class of applications are “driven” by events rather than data; produce control information rather than reports or displays, and process information with heavy concern for time and performance. Such applications require the use of control flow modeling in addition to data flow modeling.

The graphical notation required to create a control flow diagram was presented in Section 12.4.4. To review the approach for creating a CFD, a data flow model is “stripped” of all data flow arrows. Events and control items (dashed arrows) are then added to the diagram and a “window” (a vertical bar) into the control specification is shown. But how are events selected?
We have already noted that an event or control item is implemented as a Boolean value (e.g., true or false, on or off, 1 or 0) or a discrete list of conditions (empty, jammed, full). To select potential candidate events, the following guidelines are suggested:

- List all sensors that are ‘read’ by the software.
- List all interrupt conditions.
- List all ‘switches’ that are actuated by an operator.
- List all data conditions.
- Recalling the noun/verb parse that was applied to the processing narrative, review all ‘control items’ as possible CSPEC inputs/outputs.
- Describe the behavior of a system by identifying its states; identify how each state is reached; and define the transitions between states.
- Focus on possible omissions—a very common error in specifying control; for example, ask: ‘Is there any other way I can get to this state or exit from it?’

A level 1 CFD for SafeHome software is illustrated in Figure 12.23. Among the events and control items noted are sensor event (i.e., a sensor has been tripped), blink flag (a signal to blink the LCD display), and start/stop switch (a signal to turn the system on or off). When the event flows into the CSPEC window from the outside world, it implies that the CSPEC will activate one or more of the processes shown in the CFD. When a control item emanates from a process and flows into the CSPEC window, control and activation of some other process or an outside entity is implied.

12.6.4 The Control Specification

The control specification (CSPEC) represents the behavior of the system (at the level from which it has been referenced) in two different ways. The CSPEC contains a state transition diagram that is a sequential specification of behavior. It can also contain a program activation table—a combinatorial specification of behavior. The underlying attributes of the CSPEC were introduced in Section 12.4.4. It is now time to consider an example of this important modeling notation for structured analysis.

Figure 12.24 depicts a state transition diagram for the level 1 control flow model for SafeHome. The labeled transition arrows indicate how the system responds to events as it traverses the four states defined at this level. By studying the STD, a software engineer can determine the behavior of the system and, more important, can ascertain whether there are ‘holes’ in the specified behavior. For example, the STD (Figure 12.24) indicates that the only transition from the reading user input state occurs when the start/stop switch is encountered and a transition to the monitoring system status state occurs. Yet, there appears to be no way, other than the occurrence of sensor event, that will allow the system to return to reading user input. This is an
FIGURE 12.23 Level 1 CFD for SafeHome

FIGURE 12.24 State transition diagram for SafeHome
error in specification and would, we hope, be uncovered during review and corrected. Examine the STD to determine whether there are any other anomalies.

A somewhat different mode of behavioral representation is the process activation table. The PAT represents information contained in the STD in the context of processes, not states. That is, the table indicates which processes (bubbles) in the flow model will be invoked when an event occurs. The PAT can be used as a guide for a designer who must build an executive that controls the processes represented at this level. A PAT for the level 1 flow model of SafeHome software is shown in Figure 12.25.

The CSPEC describes the behavior of the system, but it gives us no information about the inner working of the processes that are activated as a result of this behavior. The modeling notation that provides this information is discussed in the next section.

12.6.5 The Process Specification

The process specification (PSPEC) is used to describe all flow model processes that appear at the final level of refinement. The content of the process specification can include narrative text, a program design language (PDL) description of the process algorithm, mathematical equations, tables, diagrams, or charts. By providing a PSPEC to accompany each bubble in the flow model, the software engineer creates a “mini-spec” that can serve as a first step in the creation of the Software Requirements Specification and as a guide for design of the software component that will implement the process.

<table>
<thead>
<tr>
<th>Input events</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensor event</td>
<td>0 0</td>
</tr>
<tr>
<td>Blink flag</td>
<td>0 0</td>
</tr>
<tr>
<td>Start/stop switch</td>
<td>0 0</td>
</tr>
<tr>
<td>Display action status</td>
<td>0 0</td>
</tr>
<tr>
<td>Complete</td>
<td>0 0</td>
</tr>
<tr>
<td>In progress</td>
<td>0 0</td>
</tr>
<tr>
<td>Time out</td>
<td>0 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Alarm signal</td>
<td>0 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Process activation</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Monitor and control system</td>
<td>0 1</td>
</tr>
<tr>
<td>Activate/deactivate system</td>
<td>0 1</td>
</tr>
<tr>
<td>Display messages and status</td>
<td>1 0</td>
</tr>
<tr>
<td>Interact with user</td>
<td>1 0</td>
</tr>
</tbody>
</table>
To illustrate the use of the PSPEC, consider the process password transform represented in the flow model for SafeHome (Figure 12.21). The PSPEC for this function might take the form:

**PSPEC: process password**

The process password transform performs all password validation for the SafeHome system. Process password receives a four-digit password from the interact with user function. The password is first compared to the master password stored within the system. If the master password matches, \(<\text{valid id message} = \text{true}\>\) is passed to the message and status display function. If the master password does not match, the four digits are compared to a table of secondary passwords (these may be assigned to house guests and/or workers who require entry to the home when the owner is not present). If the password matches an entry within the table, \(<\text{valid id message} = \text{true}\>\) is passed to the message and status display function. If there is no match, \(<\text{valid id message} = \text{false}\>\) is passed to the message and status display function.

If additional algorithmic detail is desired at this stage, a program design language representation may also be included as part of the PSPEC. However, many believe that the PDL version should be postponed until component design commences.

### 12.7 THE DATA DICTIONARY

The analysis model encompasses representations of data objects, function, and control. In each representation data objects and/or control items play a role. Therefore, it is necessary to provide an organized approach for representing the characteristics of each data object and control item. This is accomplished with the data dictionary.

The data dictionary has been proposed as a quasi-formal grammar for describing the content of objects defined during structured analysis. This important modeling notation has been defined in the following manner [YOU89]:

The data dictionary is an organized listing of all data elements that are pertinent to the system, with precise, rigorous definitions so that both user and system analyst will have a common understanding of inputs, outputs, components of stores and [even] intermediate calculations.

Today, the data dictionary is always implemented as part of a CASE "structured analysis and design tool." Although the format of dictionaries varies from tool to tool, most contain the following information:

- **Name**—the primary name of the data or control item, the data store or an external entity.
- **Alias**—other names used for the first entry.
- **Where-used/how-used**—a listing of the processes that use the data or control item and how it is used (e.g., input to the process, output from the process, as a store, as an external entity.
• **Content description**—a notation for representing content.

• **Supplementary information**—other information about data types, preset values (if known), restrictions or limitations, and so forth.

Once a data object or control item name and its aliases are entered into the data dictionary, consistency in naming can be enforced. That is, if an analysis team member decides to name a newly derived data item *xyz*, but *xyz* is already in the dictionary, the CASE tool supporting the dictionary posts a warning to indicate duplicate names. This improves the consistency of the analysis model and helps to reduce errors.

“Where-used/how-used” information is recorded automatically from the flow models. When a dictionary entry is created, the CASE tool scans DFDs and CFDs to determine which processes use the data or control information and how it is used. Although this may appear unimportant, it is actually one of the most important benefits of the dictionary. During analysis there is an almost continuous stream of changes. For large projects, it is often quite difficult to determine the impact of a change. Many a software engineer has asked, “Where is this data object used? What else will have to change if we modify it? What will the overall impact of the change be?” Because the data dictionary can be treated as a database, the analyst can ask “where used/how used’ questions, and get answers to these queries.

The notation used to develop a content description is noted in the following table:

<table>
<thead>
<tr>
<th>Data Construct</th>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence</td>
<td>=</td>
<td>is composed of</td>
</tr>
<tr>
<td>Selection</td>
<td>[ ]</td>
<td>either-or</td>
</tr>
<tr>
<td>Repetition</td>
<td>{ }</td>
<td>* repetitions of</td>
</tr>
<tr>
<td></td>
<td></td>
<td>optional data</td>
</tr>
<tr>
<td></td>
<td>* ... *</td>
<td>delimits comments</td>
</tr>
</tbody>
</table>

The notation enables a software engineer to represent composite data in one of the three fundamental ways that it can be constructed:

1. As a sequence of data items.
2. As a selection from among a set of data items.
3. As a repeated grouping of data items. Each data item entry that is represented as part of a sequence, selection, or repetition may itself be another composite data item that needs further refinement within the dictionary.

To illustrate the use of the data dictionary, we return to the level 2 DFD for the *monitor system* process for *SafeHome*, shown in Figure 12.22. Referring to the figure, the data item *telephone number* is specified as input. But what exactly is a telephone number? It could be a 7-digit local number, a 4-digit extension, or a 25-digit
long distance carrier sequence. The data dictionary provides us with a precise definition of \textit{telephone number} for the DFD in question. In addition it indicates where and how this data item is used and any supplementary information that is relevant to it. The data dictionary entry begins as follows:

\begin{verbatim}
name: telephone number
aliases: none
where used/how used: assess against set-up (output)
    dial phone (input)
description:
    telephone number = [local number|long distance number]
    local number = prefix + access number
    long distance number = 1 + area code + local number
    area code = [800 | 888 | 561]
    prefix = *a three digit number that never starts with 0 or 1*
    access number = * any four number string *
\end{verbatim}

The content description is expanded until all composite data items have been represented as elementary items (items that require no further expansion) or until all composite items are represented in terms that would be well-known and unambiguous to all readers. It is also important to note that a specification of elementary data often restricts a system. For example, the definition of area code indicates that only three area codes (two toll-free and one in South Florida) are valid for this system.

The data dictionary defines information items unambiguously. Although we might assume that the telephone number represented by the DFD in Figure 12.22 could accommodate a 25-digit long distance carrier access number, the data dictionary content description tells us that such numbers are not part of the data that may be used.

For large computer-based systems, the data dictionary grows rapidly in size and complexity. In fact, it is extremely difficult to maintain a dictionary manually. For this reason, CASE tools should be used.

\section*{12.8 OTHER CLASSICAL ANALYSIS METHODS}

Over the years, many other worthwhile software requirements analysis methods have been used throughout the industry. While all follow the operational analysis principles discussed in Chapter 11, each uses a different notation and a unique set of heuristics for deriving the analysis model. An overview of three important analysis methods:

- \textit{Data Structured Systems Development} (DSSD) [WAR81], [ORR81]
- \textit{Jackson System Development} (JSD) [JAC83]
- \textit{Structured Analysis and Design Technique} (SADT) [ROS77], [ROS85]
is presented within the SEPA Web site for those readers interested in a broader view of analysis modeling.

## 12.9 SUMMARY

Structured analysis, a widely used method of requirements modeling, relies on data modeling and flow modeling to create the basis for a comprehensive analysis model. Using entity-relationship diagrams, the software engineer creates a representation of all data objects that are important for the system. Data and control flow diagrams are used as a basis for representing the transformation of data and control. At the same time, these models are used to create a functional model of the software and to provide a mechanism for partitioning function. A behavioral model is created using the state transition diagram, and data content is developed with a data dictionary. Process and control specifications provide additional elaboration of detail.

The original notation for structured analysis was developed for conventional data processing applications, but extensions have made the method applicable to real-time systems. Structured analysis is supported by an array of CASE tools that assist in the creation of each element of the model and also help to ensure consistency and correctness.

## REFERENCES


### PROBLEMS AND POINTS TO PONDER

12.1. Acquire at least three of the references discussed in Section 12.1 and write a brief paper that outlines how the perception of structured analysis has changed over time. As a concluding section, suggest ways that you think the method will change in the future.

12.2. You have been asked to build one of the following systems:
   a. A network-based course registration system for your university.
   b. A Web-based order-processing system for a computer store.
   c. A simple invoicing system for a small business.
   d. Software that replaces a Rolodex and is built into a wireless phone.
   e. An automated cookbook that is built into an electric range or microwave.

Select the system that is of interest to you and develop an entity/relationship diagram that describes data objects, relationships, and attributes.

12.3. What is the difference between cardinality and modality?

12.4. Draw a context-level model (level 0 DFD) for one of the five systems that are listed in Problem 12.2. Write a context-level processing narrative for the system.

12.5. Using the context-level DFD developed in Problem 12.4, develop level 1 and level 2 data flow diagrams. Use a “grammatical parse” on the context-level processing narrative to get yourself started. Remember to specify all information flow by labeling all arrows between bubbles. Use meaningful names for each transform.

12.6. Develop a CFDs, CSPECs, PSPECs, and a data dictionary for the system you selected in Problem 12.2. Try to make your model as complete as possible.

12.7. Does the information flow continuity concept mean that, if one flow arrow appears as input at level 0, then one flow arrow must appear as input at subsequent levels? Discuss your answer.

12.8. Using the Ward and Mellor extensions, redraw the flow model contained in Figure 12.16. How will you accommodate the CSPEC that is implied in Figure 12.16? Ward and Mellor do not use this notation.
12.9. Using the Hatley and Pirbhai extensions, redraw the flow model contained in Figure 12.13. How will you accommodate the control process (dashed bubble) that is implied in Figure 12.13? Hatley and Pirbhai do not use this notation.

12.10. Describe an event flow in your own words.

12.11. Develop a complete flow model for the photocopier software discussed in Section 12.5. You may use either the Ward and Mellor or Hatley and Pirbhai method. Be certain to develop a detailed state transition diagram for the system.

12.12. Complete the processing narratives for the analysis model for SafeHome software shown in Figure 12.21. Describe the interaction mechanics between the user and the system. Will your additional information change the flow models for SafeHome presented in this chapter? If so, how?

12.13. The department of public works for a large city has decided to develop a Web-based pothole tracking and repair system (PHTRS). A description follows:

Citizens can log onto a Web site and report the location and severity of potholes. As potholes are reported they are logged within a “public works department repair system” and are assigned an identifying number, stored by street address, size (on a scale of 1 to 10), location (middle, curb, etc.), district (determined from street address), and repair priority (determined from the size of the pothole). Work order data are associated with each pothole and includes pothole location and size, repair crew identifying number, number of people on crew, equipment assigned, hours applied to repair, hole status (work in progress, repaired, temporary repair, not repaired), amount of filler material used and cost of repair (computed from hours applied, number of people, material and equipment used). Finally, a damage file is created to hold information about reported damage due to the pothole and includes citizen’s name, address, phone number, type of damage, dollar amount of damage. PHTRS is an on-line system; all queries are to be made interactively.

Using structured analysis notation, develop a complete analysis model for PHTRS.

12.14. Next generation software for a word-processing system is to be developed. Do a few hours of research on the application area and conduct a FAST meeting (Chapter 11) with your fellow students to develop requirements (your instructor will help you coordinate this). Build a requirements model of the system using structured analysis.

12.15. Software for a video game is to be developed. Proceed as in Problem 12.14.

12.16. Contact four or five vendors that sell CASE tools for structured analysis. Review their literature and write a brief paper that summarizes generic features that seem to distinguish one tool from another.

For an engineering emphasis [WAR85] and [HAT87] are the books of preference. However, Edwards (*Real-Time Structured Methods: Systems Analysis*, Wiley, 1993) also covers the analysis of real-time systems in considerable detail, presenting a number of useful examples drawn from actual applications.

Many variations on structured analysis have evolved over the last decade. Cutts (*Structured Systems Analysis and Design Methodology*, Van Nostrand-Reinhold, 1990) and Hares (*SSADM for the Advanced Practitioner*, Wiley, 1990) describe SSADM, a variation on structured analysis that is widely used in the United Kingdom and Europe.


A wide variety of information sources on structured analysis and related subjects is available on the Internet. An up-to-date list of World Wide Web references that are relevant to analysis concepts and methods can be found at the SEPA Web site:

http://www.mhhe.com/engcs/compsci/pressman/resources/reqm-analysis.mhtml