6.1- THE ELEMENTS ANALYSIS MODEL

The analysis model must achieve three primary objectives:

(1) to describe what the customer requires.

(2) to establish a basis for the creation of a software design.

(3) to define a set of requirements that can be validated once the software is built.

To accomplish these objectives, the analysis model derived during structured analysis takes the form illustrated in Figure 6.1. At the core of the model lies the ***data dictionary***—a repository that contains descriptions of all data objects consumed or produced by the software. Three different diagrams surround the core. The ***entity relation diagram*** (**ERD**) depicts relationships between data objects. The ERD is the notation that is used to conduct the data modeling activity. The attributes of each data object

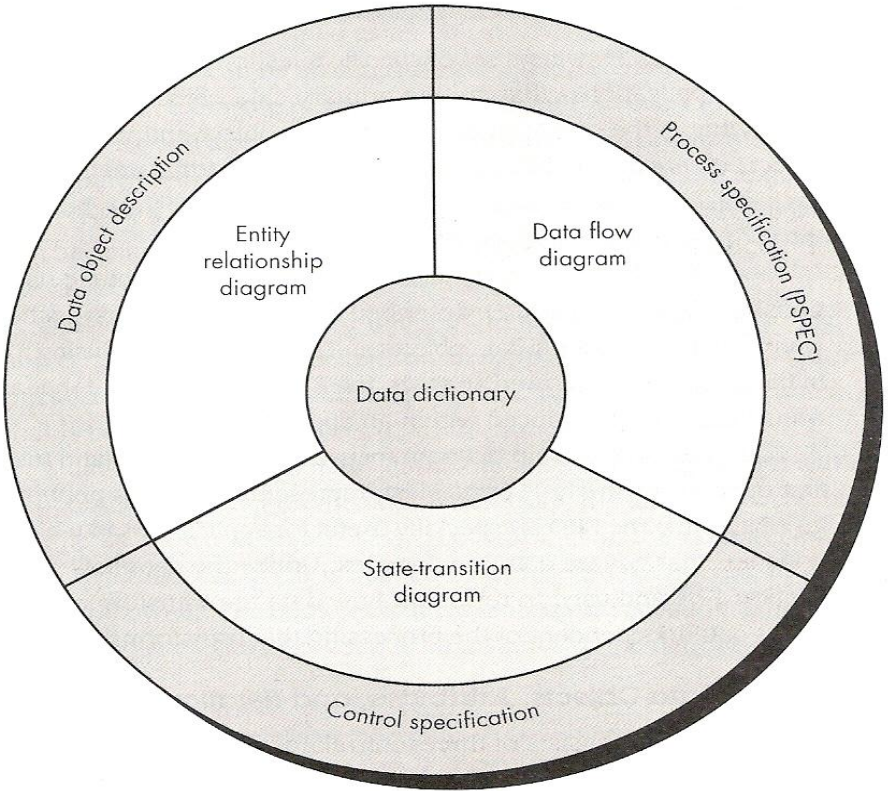

Figure 6.1: The structure of the analysis model

noted in the ERD can be described using a data object description. The ***data flow diagram*** (**DFD**) serves two purposes:

(1) to provide an indication of how data are transformed as they move through the system and

(2) to depict the functions (and sub-functions) that transform the data flow.

The DFD provides additional information that is used during the analysis of the information domain and serves as a basis for the modeling of function. A description of each function presented in the DFD is contained in a ***process specification*** (**PSPEC**).

The *state transition diagram* (STD) indicates how the system behaves as a consequence of external events. To accomplish this, the STD represents the various modes of behavior (called slates) of the system and the manner in which transitions are made from state to state. The STD serves as the basis for behavioral modeling. Additional information about the control a specs of the software is contained in the *control specification* (CSPEC).The analysis model encompasses each of the diagrams, specifications,

descriptions, and the dictionary noted in Figure 6.1. A more detailed discussion of these elements of the analysis model is presented in the sections that follow.

6.2- DATA MODELING

Data modeling answers a set of specific questions that are relevant to any data processing application. What are the primary data objects to be processed by the system? What is the composition of each data object and what attributes describe the object? Where do the objects currently reside? What are the relationships between each object and other objects? What are the relationships between the objects and the processes that transform them?

To answer these questions, data modeling methods make use of the entity relationship diagram. The ERD, described in detail later in this section, enables a software engineer to identify data objects and their relationships using a graphical notation. In the context of structured analysis, the ERD defines all data that are entered, stored, transformed, and produced within an application.

The entity relationship diagram focuses solely on data (and therefore satisfies the first operational analysis principles), representing a data network that exists for a given system. The ERD is especially useful for applications in which data and the relationships that govern data are complex. Unlike the data flow diagram (discussed in next section and used to represent how data are transformed), data modeling considers data independent of the processing that transforms the data.

6.2.1 Data Objects, Attributes, and Relationships

The data model consists of three interrelated pieces of information: the data object, the attributes that describe the data object, and the relationships that connect data objects to one another.

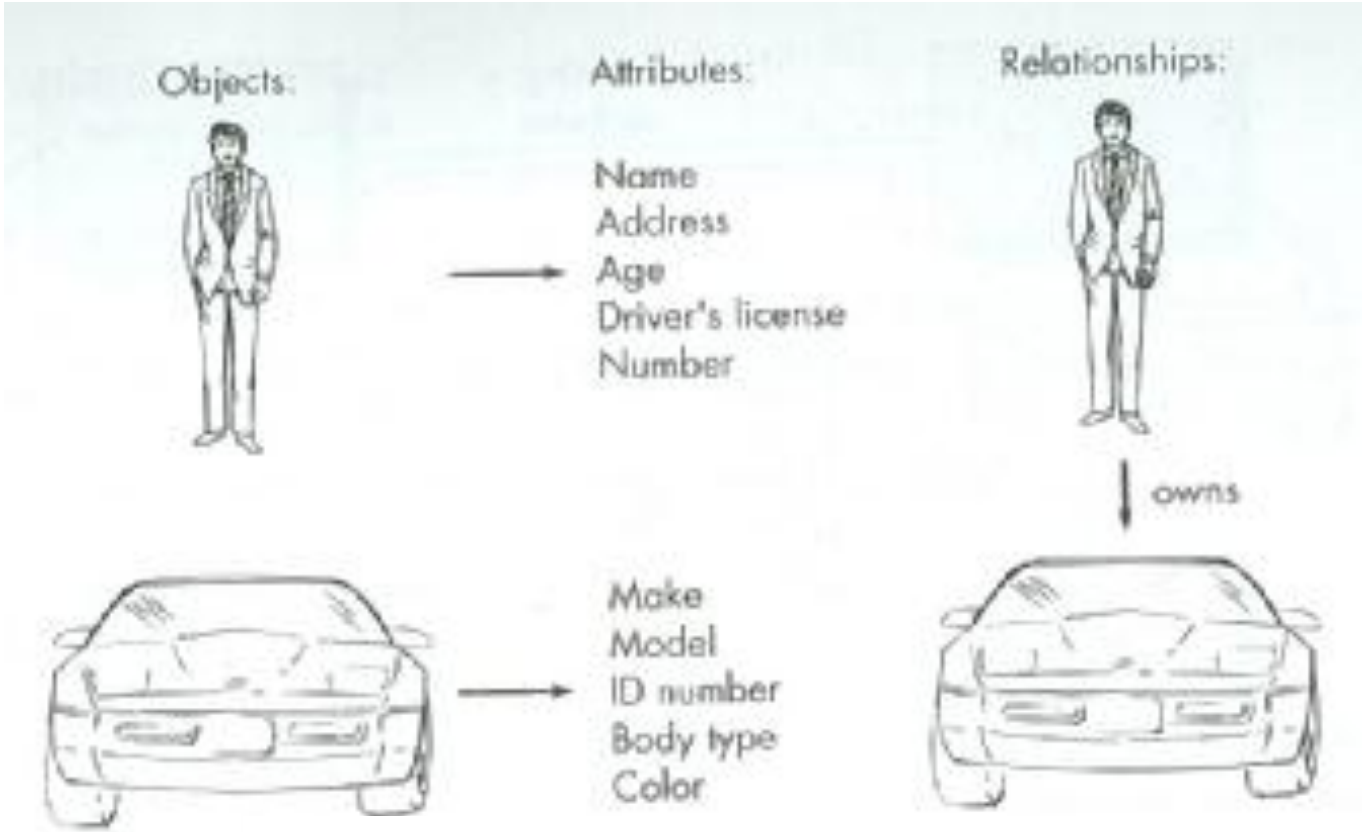

Figure 6.2: Data objects, attributes and relationship

Data objects: A data object is a representation of almost any composite information that must be understood by software. By composite information, we mean something that has a number of different properties or attributes. Therefore, width (a single value) would not be a valid data object, but dimensions (incorporating height, width, and depth) could be defined as an object.

A data object can be an external entity (e.g., anything that produces or consumes information), a thing (e.g., a report or a display), an occurrence (e.g., a telephone call) or event (e.g., an alarm), a role (e.g., salesperson), an organizational unit (e.g., accounting department), a place (e.g., a warehouse), or a structure (e.g., a file). For example, a person or a car (Figure 6.2) can be viewed as a data object in the sense that either can be defined in terms of a set of attributes. The data object description incorporates the data object and all of its attributes.

Data objects (represented in bold) are related to one another. For example, **person** can *own* **car**, where the relationship own connotes a specific connection" between **person** and **car**. The relationships are always defined by the context of the problem that is being analyzed.

A data object encapsulates data only—there is no reference within a data object to operations that act on the data. Therefore, the data object can be represented as a table as shown in Figure 6.3. The headings in the table reflect attributes of the object. In this case, a car is defined in terms of make, model, ID number, body type, color and owner. The body of the table represents specific instances of the data object. For example, a Chevy Corvette is an instance of the data object **car**.

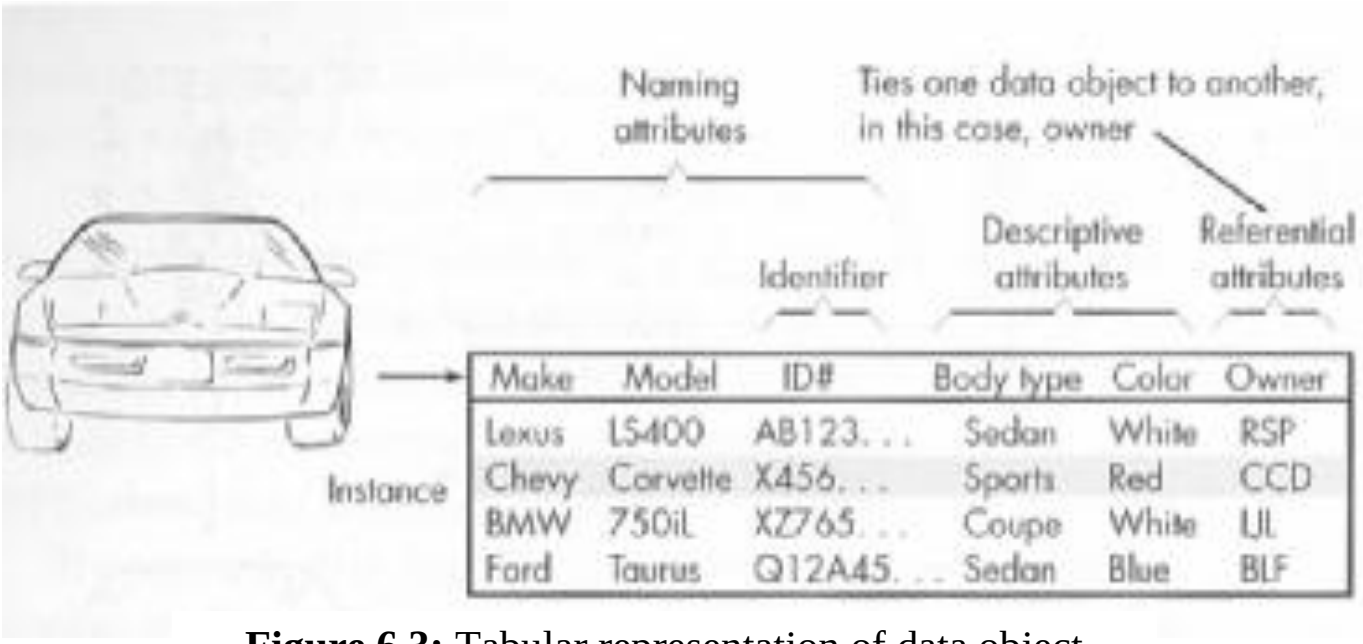

Figure 6.3: Tabular representation of data object

Attributes: Attributes define the properties of a data object and take on one of three different characteristics. They can be used to:

(1) name an instance of the data object
(2) describe the instance
(3) make reference to another instance in another table.

In addition, one or more of the attributes must be defined as an identifier—that is, the identifier attribute becomes a 'key' when we want to find an instance of the data object. In some cases, values for the identifier(s) are unique, although this is not a requirement. Referring to the data object **car**, a reasonable identifier might be the ID number.

The set of attributes that is appropriate for a given data object is determined through an understanding of the problem context. The attributes for **car** might serve well for an application that would be used by a Department of Motor Vehicles, but these attributes would be useless for an automobile company that needs manufacturing control software. In the latter case, the attributes for car might also include ID number, body type and color, but many additional attributes (e.g., interior code, drive train type, trim package designator, transmission type) would have to be added to make **car** a meaningful object in the manufacturing control context.

Relationships: Data objects are connected to one another in different ways. Consider two data objects, book and bookstore. These objects can be represented using the simple notation illustrated in Figure

6.4a. A connection is established between book and bookstore because the two objects are related. But what are the relationships? To determine the answer, we must understand the role of books and bookstores within the context of the software to be built. We can define a set of object/relationship pairs that define the relevant relationships. For example,

- A bookstore orders books.
- A bookstore displays books.
- A bookstore stocks books.
- A bookstore sells books.
- A bookstore returns books.

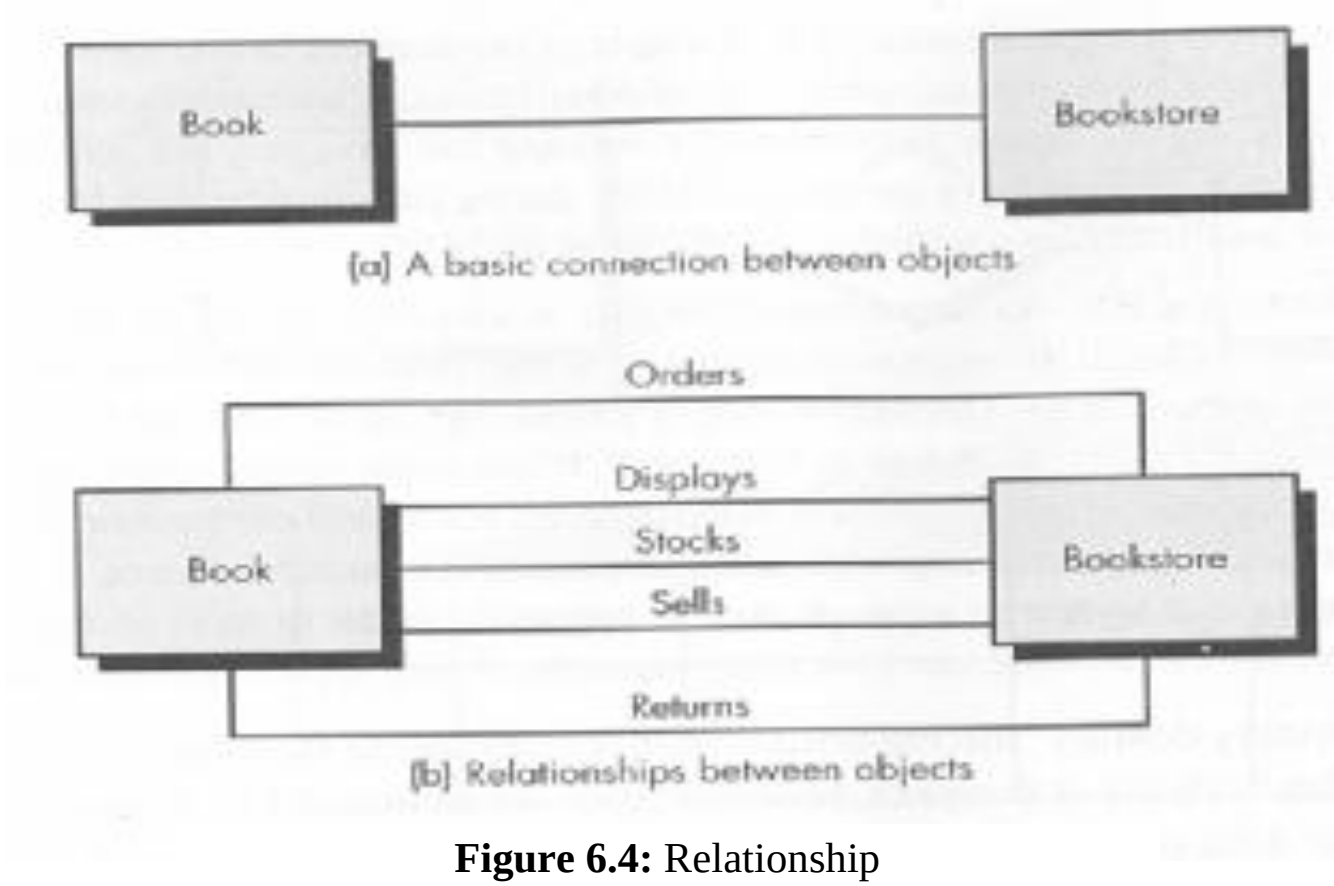

Figure 6.4: Relationship

The relationships orders, displays, stocks, sells, and returns define the relevant connections between **book** and **bookstore**. Figure 6.4b illustrates these object/relationship pairs graphically.

It is important to note that object/relationship pairs are bidirectional. That is, can be read in either direction. A bookstore orders books or books are ordered bookstore.

6.2.2 Cardinality and Modality

The elements of data modeling—data objects, attributes, and relationships— provide the basis for understanding the information domain of a problem. However, additional information related to these basic elements must also be understood.

We have defined a set of objects and represented the object/relationship pairs bind them. But a simple pair that states: **object x** relates to **object Y** does not provide enough information for software engineering purposes. We must understand how many occurrences of **object X** are related to how many occurrences of **object Y**. This leads to a data modeling concept called ***cardinality.***

Cardinality: The data model must be capable of representing the number of a occurrences objects in a given relationship. Tillmann defines the cardinality object/relationship pair in the following manner:

Cardinality is the specification of the number of occurrences or one [object] that can be related to the number of occurrences of another [object]. Cardinality is usually expressed as simply 'one' or 'many'. *For example*, a husband can have only one wife (in most cultures), while a parent can have many children. Taking into consideration all combinations of 'one' and 'many' two [objects] can be related as

• **One-to-one** (1:1)—An occurrence of [object] 'A' can relate to one and only one occurrence of [object] 'B' and an occurrence of 'B' can relate to only one occurrence of 'A'.

• **One-to-many** (1:N)—One occurrence of [object] 'A' can relate to one or many occurrences of [object] 'B', but an occurrence of 'B' can relate to only one occurrence of 'A,' *For example*, a mother can have many children, but a child can have only one mother.

• **Many-to-many** (M:N)—An occurrence of [object] 'A' can relate to one or more occurrences of 'B', while an occurrence of 'B' can relate to one or more occurrences of 'A'. *For example*, an uncle can have many nephews, while a nephew can have many uncles.

Cardinality defines "the maximum number of objects that can participate in a relationship". It does not, however, provide an indication of whether or not a particular data object must participate in the relationship. To specify this information, the data model adds modality to the object/relationship pair.

Modality. The modality of a relationship is 0 if there is no explicit need for the relationship to occur or the relationship is optional. The modality is 1 if an occurrence of the relationship is mandatory. To illustrate, consider software that is used by a local telephone company to process requests for held service. A customer indicates that there is a problem. If the problem is diagnosed as relative[y simple, a single repair action occurs. However, if the problem is complex, multiple repair actions may be required. Figure 6.5 illustrates the relationship, cardinality, and modality between the data objects **customer** and **repair action**.

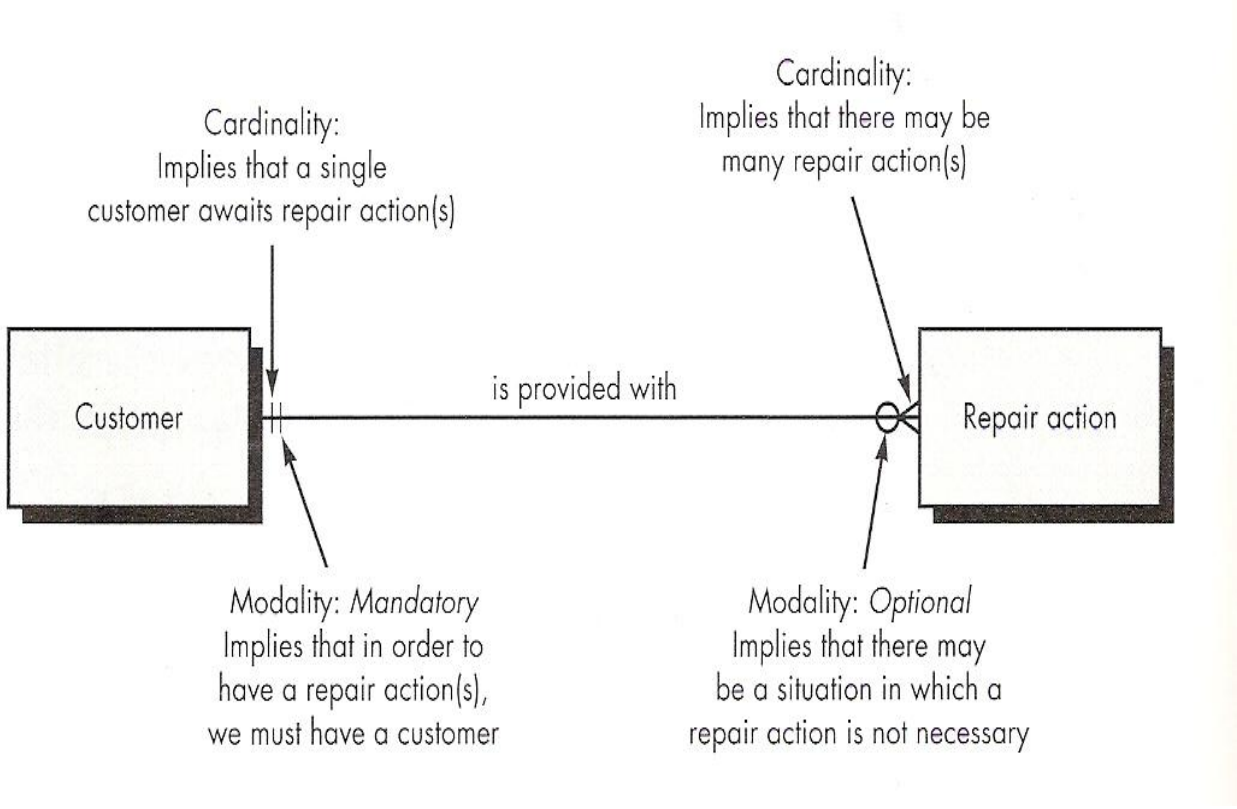

Figure 6.5: Cardinality and modality

Referring to the figure, a **one-to-many** cardinality relationship is established. That is, a single customer can be provided with zero or many repair actions. The symbols on the relationship connection closest to the data object rectangles indicate cardinality. The vertical bar indicates one and the three-pronged fork

indicates many. Modality is indicated by the symbols that are further away from the data object rectangles. The second vertical bar on the left indicates that there must be a customer for a repair action to occur. The circle on the right indicates that there may be no repair action required for the type of problem reported by the customer.

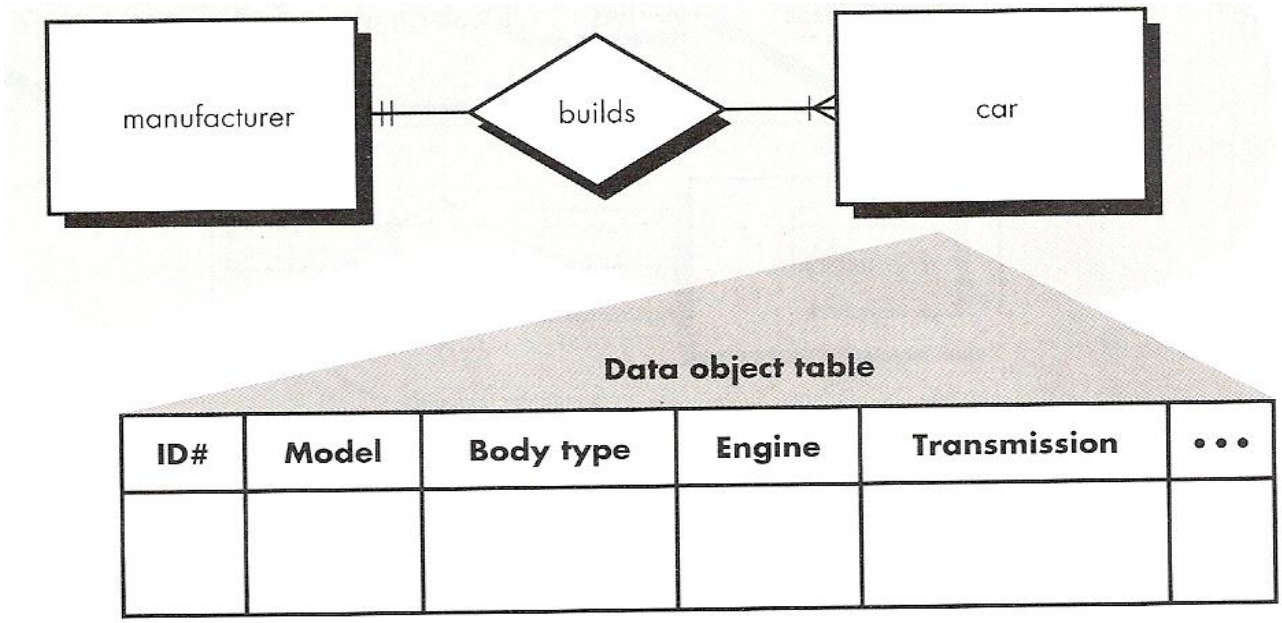

Figure 6.6: A simple ERD and data object table

6.2.3 Entity/Relationship Diagrams

The object/relationship pair (discussed in Section 6.2.1) is the cornerstone of the data model. These pairs can be represented graphically using the entity/relationship diagram. The ERD was originally proposed by Peter Chen for the design of relational database systems and has been extended by others. A set of primary components are identified for the ERD: data objects, attributes, relationships, and various type indicators. The primary purpose of the ERD is to represent data objects and their relationships.

Rudimentary ERD notation has already been introduced in the Section 6.2. Data objects are represented by a labeled rectangle. Relationships are indicated with a labeled line connecting objects. In some variations of the ERD, the connecting line contains a diamond that is labeled with the relationship. Connections between data objects and relationships are established using a variety of special symbols that indicate cardinality and modality (Section 6.2.2).

The relationship between the data objects **car** and **manufacturer** would be represented as shown in Figure 6.6. One manufacturer builds one or many cars. Given the context implied by the ERD, the specification of the data object **car** (data object table in Figure 6.6) would be radically different from the earlier specification (Figure 6.3). By examining the symbols at the end of the connection line between objects, it can be seen that the modality of both occurrences is mandatory (the vertical lines).

Expanding the model, we represent a grossly oversimplified ERD (Figure 6.7) of the distribution element of the automobile business. New data objects, **shipper** and **dealership**, are introduced. In addition, new relationships—*transports, contracts, licenses, and stocks*—indicate how the data objects shown in the figure associate with one another. Tables [or each of the data objects contained in the ERD would have to be developed according to the rules introduced earlier in this chapter.

In addition to the basic ERD notation introduced in Figures 6.6 and 6.7, the analyst can represent *data object type hierarchies*. In many instances, a data object may actually represent a class or category of information. *For example*, the data object **car** can be categorized as domestic, European, or Asian. The ERD notation shown in Figure 6.8 represents this categorization in the form of a hierarchy.

ERD notation also provides a mechanism that represents the associativity between objects. An *associative data object* is represented as shown in Figure 6.9. In the figure, each of the data objects that model the individual subsystems is associated with the data object **car**.

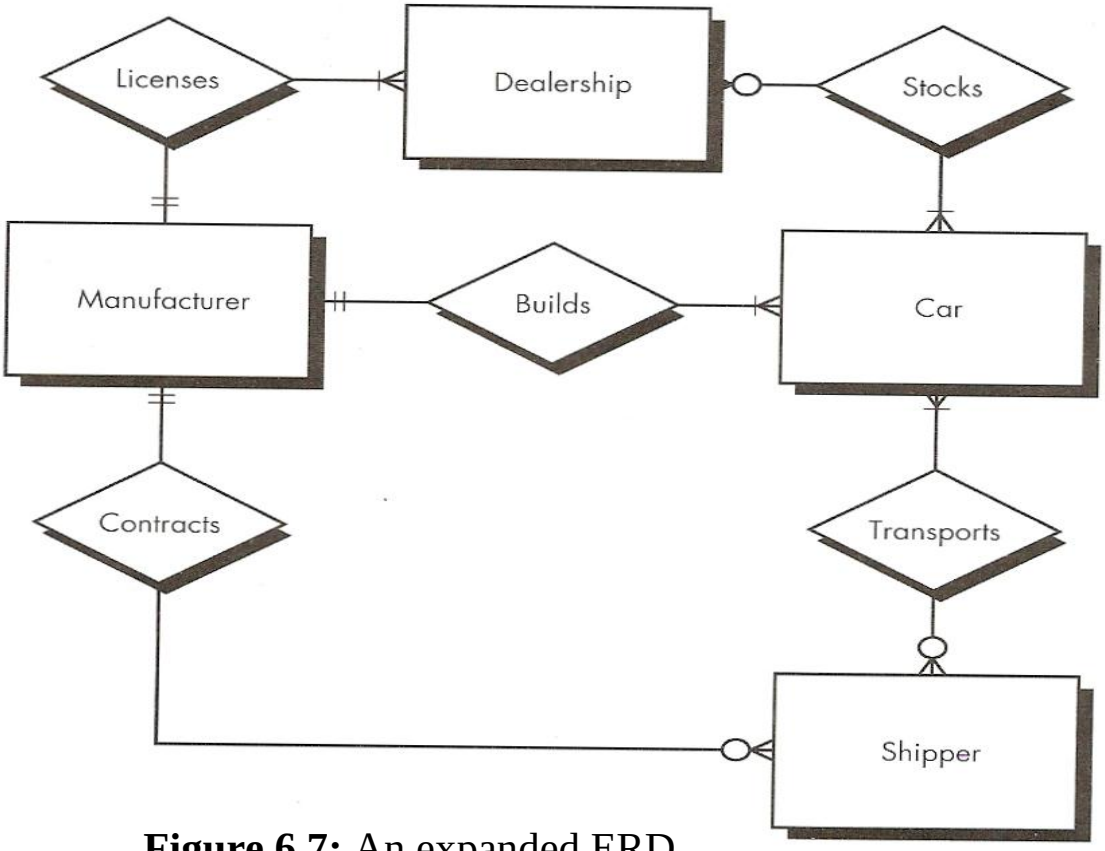

Figure 6.7: An expanded ERD

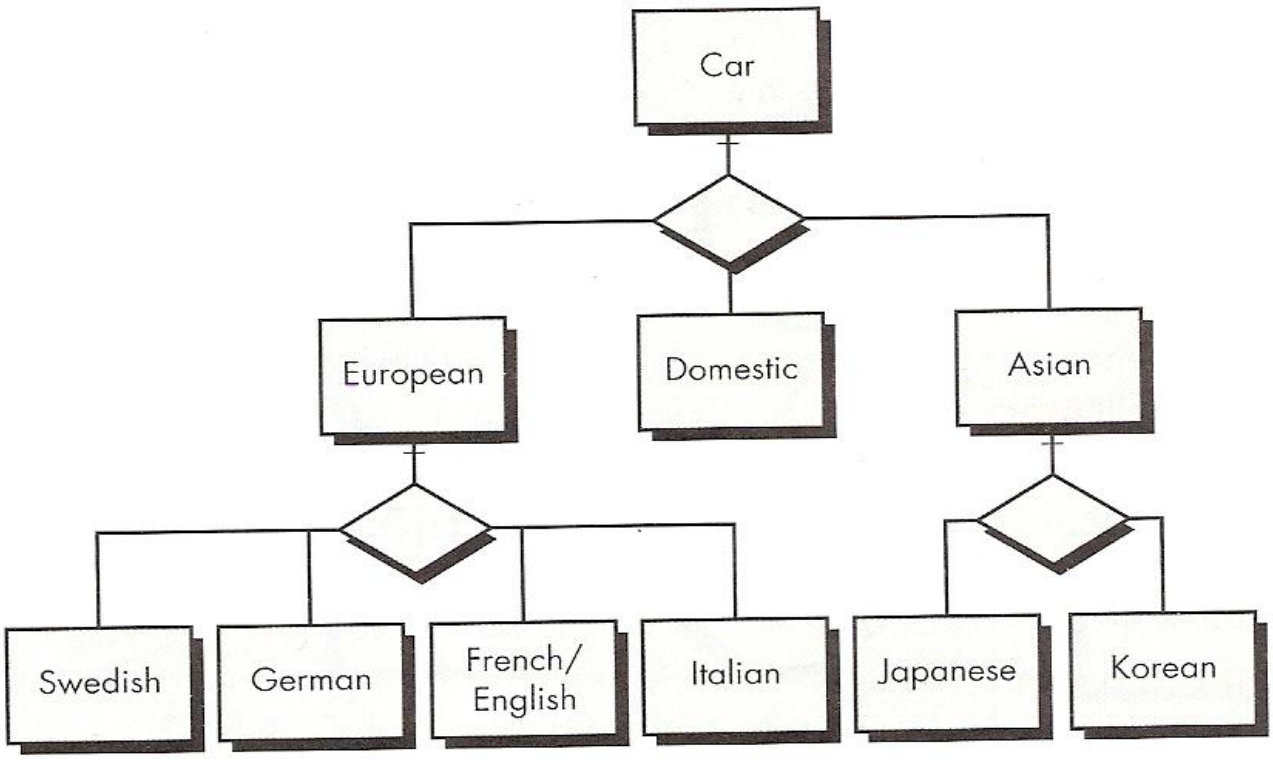

Figure 6.8: Data object-type hierarchies

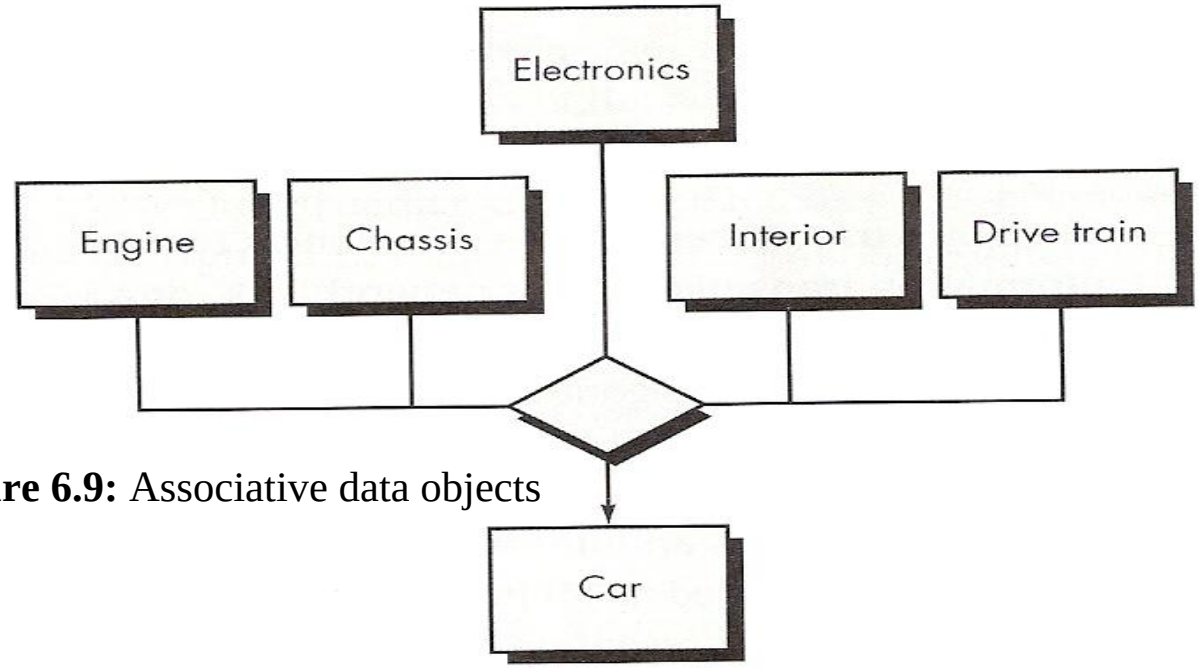

Figure 6.9: Associative data objects

Data modeling and the entity relationship diagram provide the analyst with a concise notation for examining data within the context of a software application. In most cases, the data modeling approach is used to create one piece of the analysis model, but it can also be used for database design and to support any other requirements analysis methods.

6.3 FUNCTIONAL MODELING AND INFORMATION FLOW

Information is transformed as it flows through a computer-based system. The system accepts input in a variety of forms; applies hardware, software, and human elements to transform it; and produces output in

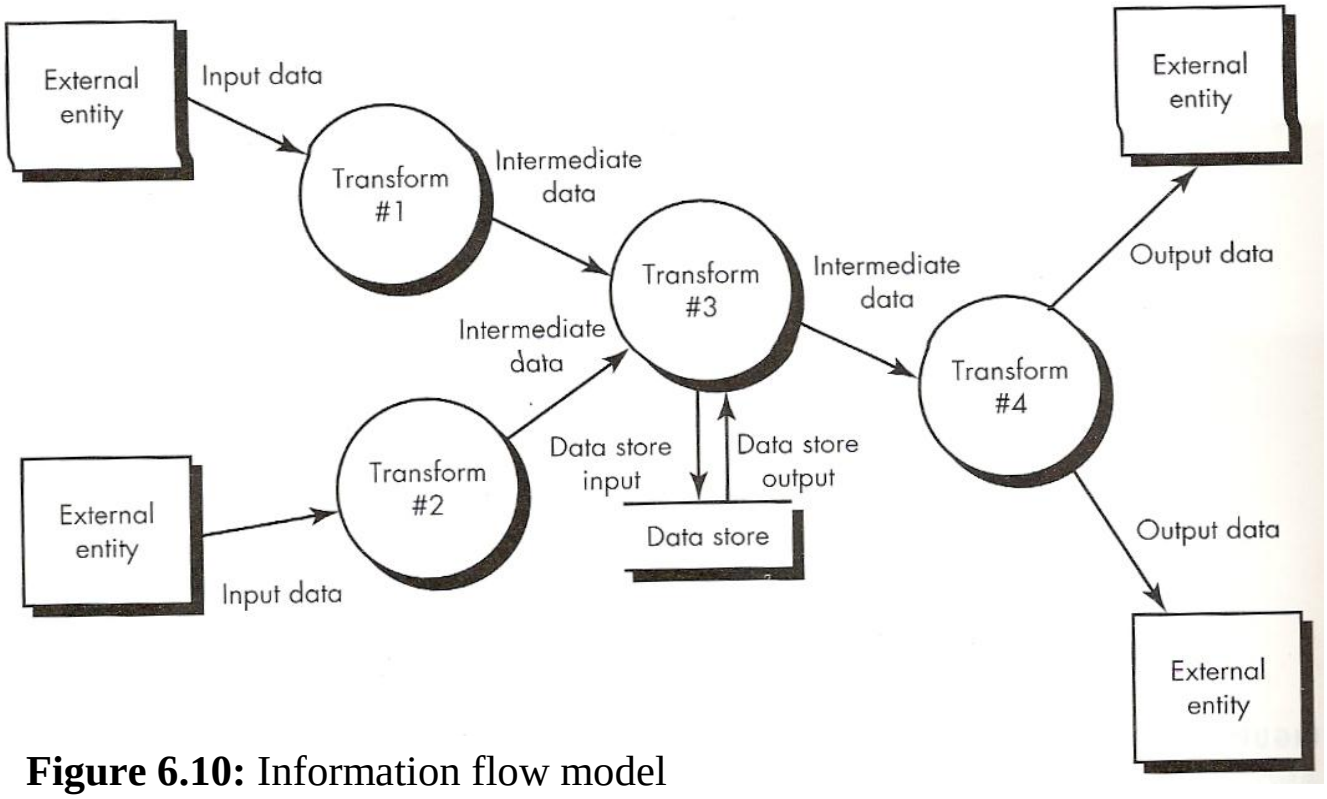

Figure 6.10: Information flow model

a variety of forms. Input may be a control signal transmitted by a transducer, a series of numbers typed by a human operator, a packet of information transmitted on a network link, or a voluminous data file

retrieved from secondary storage. The transform(s) may comprise a single logical comparison, a complex numerical algorithm, or a rule-inference approach of an expert system. Output may light a single LED or produce a 200-page report. In effect, we can create a flow model for any computer-based system, regardless of size and complexity.

Structured analysis began as an information flow modeling technique. A computer-based system is represented as an information transform as shown in Figure 6.10. A rectangle is used to represent an external entity; that is, a system element (e.g., hardware, a person, another program) or another system that produces information [or transformation by the software or receives information produced by the software. A circle (sometimes called a bubble) represents a process or transform that is applied to data (or control) and changes it in some way. An arrow represents one or more data items (data objects). All arrows on a data flow diagram should be labeled. The double line represents a data store—stored information that is used by the software. The simplicity of DED notation is one reason why structured analysis techniques are widely used.

It is important to note that no explicit indication of the sequence of processing or conditional logic is supplied by the diagram. Procedure or sequence maybe implicit in the diagram, but explicit logical detail is generally delayed until software design. It is important not to confuse a DFD with the flowchart.

Data Flow Diagrams

As information moves through software, it is modified by a series of transformations. A data flow diagram is a graphical representation that depicts information flow and the transforms that are applied as data move from input to output. The basic form of a data flow diagram also known as a data flow graph or a bubble chart, is illustrated in Figure 6.10.

The data flow diagram may be used to represent a system or software at any level of abstraction. In fact, DFDs maybe partitioned into levels that represent increasing information, flow and functional detail. Therefore, the DFD provides a mechanism for functional modeling as well as information flow modeling.

A level 0 DFD, also called a fundamental system model or a context model, represents the entire software element as a single bubble with input and output data indicated by incoming and outgoing arrows, respectively. Additional processes (bubbles) and information flow paths are represented as the level 0 DFD is partitioned to reveal more detail. For example, a level 1 DFD might contain five or six bubbles with interconnecting arrows. Each of the processes represented at level 1 is a subfunction of the overall system depicted in the context model.

As we noted earlier, each of the bubbles maybe refined or layered to depict more detail. Figure 6.11 illustrates this concept. A fundamental model for system F indicates the primary input is A, and ultimate output is B. We refine the F model into transforms f_1 to f_7. Note that information flow continuity must be maintained; that is, input

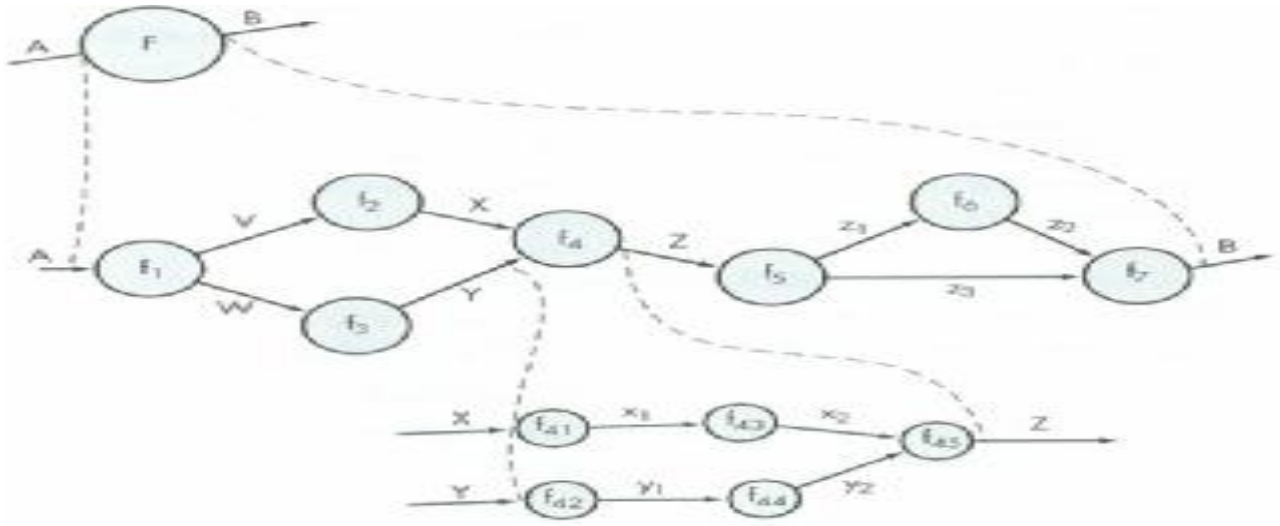

Figure 6.11: Information flow refinement

and output to each refinement must remain the same. This concept, sometimes called balancing, is essential for the development of consistent models. Further refinement of f_4 depicts detail in the form of transforms f_{43} to f_{45}. Again, the input (X, Y) and output (Z) remain unchanged.

The basic notation used to develop a DFD is not in itself sufficient to describe requirements for software. For example, an arrow shown in a DFD represents a data object that is input to or output from a process. A data store represents some organized collection of data. But what is the content of the data implied by the arrow or depicted by the store? If the arrow (or the store) represents a collection of objects, what are they? These questions are answered by applying another component of the basic notation for structured analysis—the data dictionary. The use of the data dictionary is discussed later in this chapter. DFD graphical notation must be augmented with descriptive text. A process specification (PSPEC) can be used to specify the processing details implied by a bubble within a DFD. The process specification describes the input to a function, the algorithm that is applied to transform the input, and the output that is produced. In addition, the PSPEC indicates restrictions and limitations imposed on the process (function), performance characteristics that are relevant to the process, and design constraints that nay influence the way in which the process will be implemented.

6.4 BEHAVIORAL MODELING

Behavioral modeling is an operational principle for all requirements analysis methods. Yet, only extended version of structure analysis provide a notation for this type of modeling. The state transition diagram represents the behavior of a system by depicting its states and the events that cause the system to change state. In addition, the STD indicated what actions (e.g. progress activation) are taken as a consequence of particular event.

A state is any observable mode of behavior. *For example,* states for a monitoring and control system for pressure vessels describe in section 6.3 might be *monitoring state, alarm state, pressure release state,* and so on. Each of this state represents a mode of behavior of the system. A state transition diagram indicates how the system moves from state to state.

To illustrate the use of the Hatley and Pirbhai control and behavioral extensions, consider software embedded within an office photocopying machine. Data flow arrow has been lightly shaded for illustrative purpose, but in reality they are not shown as part of a control flow diagram.

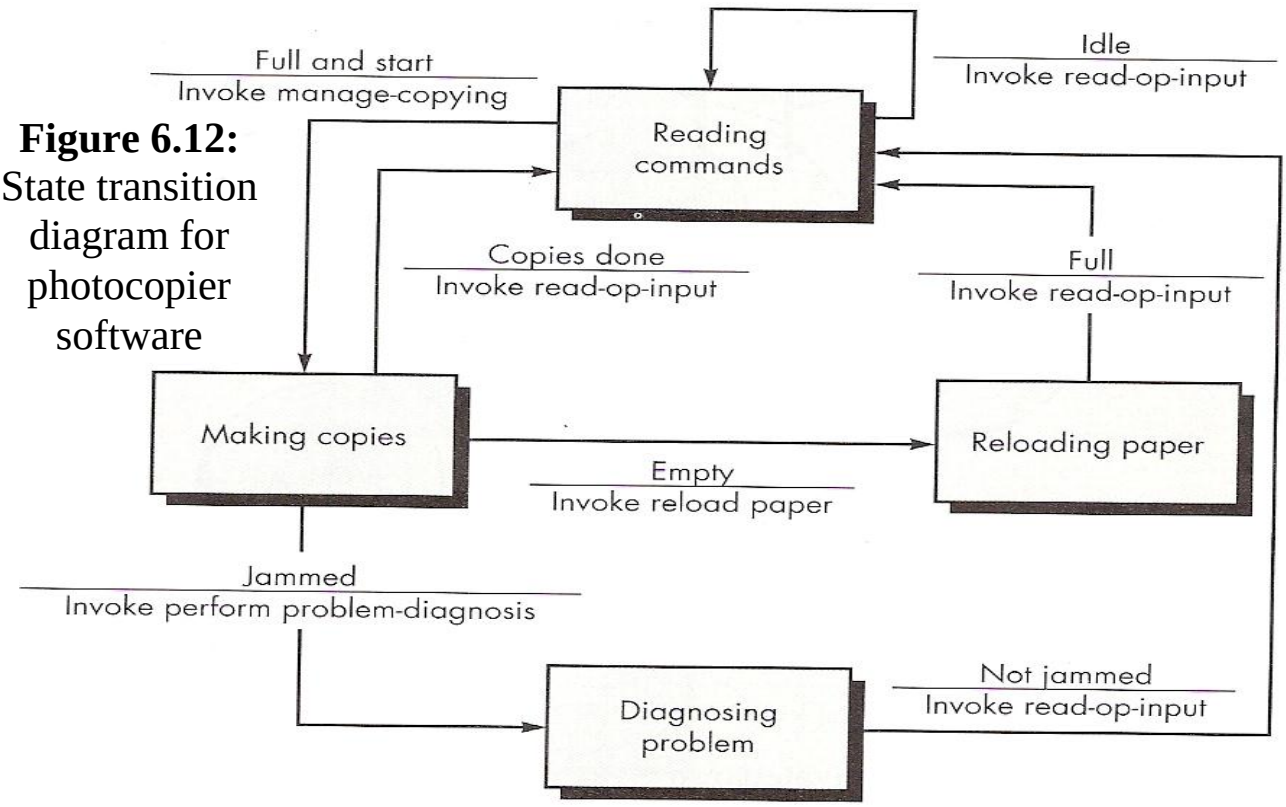

Figure 6.12: State transition diagram for photocopier software

Control flows are shown entering and exiting individual processes and the vertical bar representing the CSPEC window." For example, the **paper feed status and** start/stop events flow into the CSPEC bar. This implies that each of these events will cause some process represented in the CFD to he activated. If we were to examine the CSPEC internals, the start/stop event would be shown to activate/deactivate the *manage copying* process. Similarly, the jammed event (part of **paper feed status)** would activate *perform problem diagnosis.* It should be noted that all vertical bars within the CFD refer to the same CSPEC. An event flow can be input directly into a process as shown with **repro** fault. However, this flow does not activate the process but rather provides control information for the process algorithm.

A simplified state transition diagram for the photocopier software is shown in Figure 6.12. The rectangles represent system states and the arrows represent transitions between states. Each arrow is labeled with a ruled expression. The top value indicates the event(s) that cause the transition to occur. The bottom value indicates the action that occurs as a consequence of the event. Therefore, when the paper tray is full and the ***start*** button is pressed, the system moves from the *reading command/s* state to the *making copies* state. Note that states do not necessarily correspond to processes on a one-to-one basis. For example, the state *making copies* would encompass both the *manage copying* and *produce user displays* processes.