Chapter

2

The Functional View
Activity-Charts

This chapter deals with the language of Activity-charts, which is used to depict the functional view of the system under development. We describe how the functionality of a system is specified by a hierarchy of functional components, called *activities,* what kind of information is exchanged between these activities and manipulated by them, how this information flows, how it is stored, and so on.

Many of the concepts and notions represented in this view are quite well known, and are not specific to our approach. They are used in other notations and methods, perhaps with small variations. In fact, activity-charts can be viewed as a variant of hierarchical data flow diagrams, but they embody many enhancements and use some special terms and notations.

2.1 Functional Description of a System

The functional description of a system specifies the system's capabilities. It details the functional components, or activities, that the system is capable of carrying out and how these components communicate through the flow of information among them. It does so in the context of the system's environment, that is, it defines the environment with which the system interacts and the interface between the two.

The functional view does not address the physical and implementational aspects of the system. As for the dynamic and behavioral issues, it attempts to separate them from the functional description whenever possible, but, as we shall see, there is a close relationship between functionality and behavior. For example, the functional view is appropriate for telling whether a medical diagnosis system can monitor a patient's blood pressure and, if so, where it would get its input data and which functions would have access to the output data.

However, to deal with such issues as the conditions under which the monitoring is started and the question of whether it can be carried out parallel to temperature monitoring, the behavioral view must be considered as well as its connections with the functional view. These crucial parts of modeling the system are described in Chaps. 4 to 8.

The structural view, which deals with sensors, processors, monitors, software modules, and so on, is described in Chaps. 9 and 10.

2.1.1 Functional decomposition

The main method for describing the functionality of a system in our approach is that of functional decomposition, by which the system is viewed as a collection of interconnected functional components (or *activities,* as they are called in our terminology), organized into a hierarchy. Thus, each of the activities may be decomposed into its subactivities repeatedly until the system has been specified in terms of *basic activities,* which are those that the specifiers have decided require no further decomposition. Basic activities are specified using alternative means, such as textual description, formal or informal, or code in a programming language. The intended meaning of the functional decomposition is that the capabilities of the parent activity are distributed between its subactivities. The order in which these subactivities are performed and the conditions that cause their activation or deactivation are not explicitly represented in the functional view and are usually specified in the behavioral view, as discussed in later chapters.

Note that the term *functional decomposition* is usually identified with the Structured Analysis methodology (DeMarco 1978), in which the functional components of a system are functions in the mathematical sense of the word. Here, we use this term in a broader meaning, where the main idea is to decompose the functionality of the entire system into activities, the functional components, which may very well be reactive in nature and which together capture the whole picture.

The activities themselves can represent different concepts used in conventional modeling techniques. They can be objects, processes, functions, use cases, logical machines, or any other kind of functionally distinct entity.

Which one is selected depends on the modeler's preference, but it is recommended to try to stick with a common type of functional component, based on a single conceptual approach or methodology. To some extent, this selection dictates the nature of the interface and communication between the activities as well as some of the behavioral aspects.

In the following subsections we discuss two types of decomposition: *function-based decomposition,* in which the activities are system functions and *object-based decomposition,* in which they are objects. Both styles are illustrated by the EWS example of Chap. 1.

2.1.2 Function-based decomposition

In function-based decomposition, the activities are (possibly reactive) functions. To illustrate it, we consider the EWS example. We start with a narrative that describes its functionality and reorganize it into the following list of requirements:

- The EWS receives a signal from an external sensor.
- It samples and processes the signal continuously, producing some result.
- It checks whether the value of the result is within a specified range that is set by the operator.
- If the value is out of range, the system issues a warning message on the operator display and posts an alarm.
- If the operator does not respond within a given time interval, the system prints a fault message on a printing facility and stops monitoring the signal.

As the first step of our functional description of the EWS, we identify the various functions that are called for by these textual requirements:

SET_UP	Receives the range limits from the operator.
PROCESS_SIGNAL	Reads the "raw" signal from the sensor and performs some processing to yield a value that is to be compared to the range limits.
COMPARE	Compares the value of the processed signal with the range limits.
DISPLAY_FAULT	Issues a warning message on the operator display and posts an alarm.
PRINT_FAULT	Prints a fault message on the printing facility.

Notice that the description of the activities also contains information about the data they handle. An activity may transform its input information into output information to be consumed by other functions that can be either internal or external to the system. For example, the activity PROCESS_SIGNAL transforms its input, the raw signal, into a value that is checked by the COMPARE function. (The signal processing can be a simple conversion of an analog signal into a digital representation at a fixed rate. Of course, it could also be a more complex transformation, such as computing the average value over some time interval.)

In the function-based decomposition approach the interface of an activity is described in terms of input and output signals, both data and control. Also, the model will usually present the source activity of input information and the target activity of output information.

2.1.3 Object-based decomposition

In an object-based approach, the decomposition is defined by the entities on which operations are performed or, alternatively, is based on the active agents, or the active components, of the system (these are called *logical machines* in the ROOM methodology; see Selic et al. 1994). In our approach, the interface between objects consists of the events and messages that cause the internal operations to take place and sometimes the data that is used in these operations, just as in function-based decomposition. This is somewhat different from object-oriented design (OOD) paradigms, in which an object's interface consists of its operations.

To illustrate, we decompose the functionality of the EWS system into the following components, using encapsulation guidelines that are often presented in object-oriented methods. When applicable, a component is characterized by its subject and associated operations:

`SIGNAL_PROCESSOR`	Handles the signal from the sensor. It reads the signal, processes the read value, and checks the processed signal against the legal range.
`FAULT_HANDLER`	Consists of all functionality related to fault situations. It handles a fault occurrence by issuing the alarm, printing the fault report, and resetting the fault situation.
`RANGE`	Handles the range limits against which the processed signal is compared. It reads the range limits provided by the operator, validates the read values, stores the current legal range, and makes its values and status available to the other objects.
`MMI_HANDLER`	Takes care of all interaction with the operator (i.e., the human-machine interface). It accepts commands and data from the operator and displays messages and other information.
`CONTROLLER`	Controls the behavior of the entire system.

This decomposition is not overly detailed, and some of the components can be further decomposed into lower-level objects that help them accomplish their goals.

2.1.4 System context

One of the first decisions that should be made when developing a system involves its boundaries, or *context.* We must determine which entities are part of the system's environment—these can be other systems, functions, or objects (depending on the decomposition approach)—and how they communicate with the system itself.

In both approaches to the preceding EWS description some of the inputs come from outside the system and some of the outputs are sent outside. For example, in the function-based decomposition, the raw signal consumed by `PROCESS_SIGNAL` comes from the `SENSOR`, which is not part of the specified system but belongs to the environment. Similarly, the printed message produced by `PRINT_FAULT` is sent to the `OPERATOR`, which is also external to the EWS. In the object-based

decomposition, the interaction with the environment is handled by the MMI_HANDLER that interfaces with the OPERATOR, and by the SIGNAL_PROCESSOR that reads the signal from the SENSOR.

As a result, we may now decide that the EWS's environment consists of two external entities, or systems: the (presumably human) OPERATOR and the SENSOR (see Fig. 2.1).

The system context is sometimes given as part of the requirements, before the beginning of the specification process. However, it is often the responsibility of whoever carries out the functional description to determine the best way to set up the system boundaries.

For example, we could have defined the specification boundaries of the EWS differently because they were not given as part of the textual description, removing the printing facility from the system itself and turning it into an external entity.

2.1.5 The decomposition process

Some specification methodologies that are based on functional decomposition provide guidelines for how the subfunctions ought to be defined and the order in which the functional description should be prepared. According to one of these methodologies, the analyst should start by describing the system's context, that is, the environment entities and the information flowing between them and the system itself. The process is then continued in a top-down manner, proceeding from the description of the entire system, to the description of its subfunctions, to their subfunctions, etc. Alternatively, a bottom-up approach may be adopted, whereby the basic, lowest-level functions are specified first and used as building blocks to construct higher-level functions. We shall not address such methodological issues of order and process here but concentrate on the way the concepts relevant to the functional view of specification can be expressed in our languages.

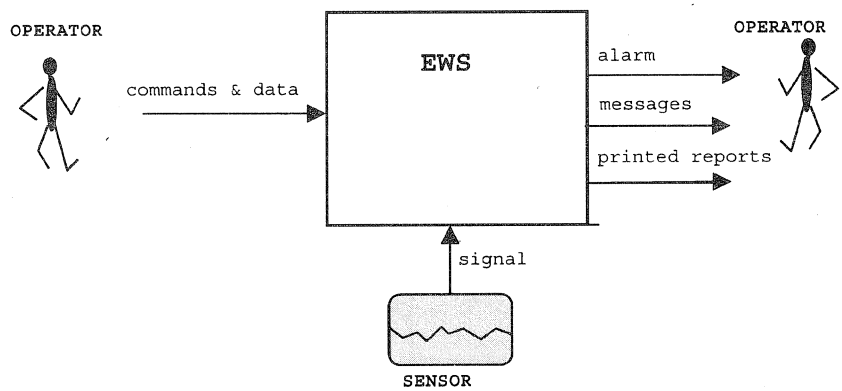

Figure 2.1 The context of the EWS.

The functional view is specified in our approach by *Activity-charts,* together with a *Data Dictionary* that may contain additional information about the elements appearing in the charts. The following sections describe the details of the Activity-charts language. Almost all our examples will use function-based decomposition, although the same language constructs can be used for other approaches, such as the object-based one.

2.2 Activities and Their Representation

2.2.1 The hierarchy of activities

The activities in an activity-chart are depicted as rectangular or rectilinear solid-line boxes, and the subactivity relationship is depicted by box encapsulation. An activity's name appears inside its box. Figure 2.2 shows one level of the decomposition of the EWS system.

The overall activity of the system has been named `EWS_ACTIVITIES`. In function-based decomposition, it is useful to use verbs for names of activities, with or without a qualifying noun, as we have done for the subactivities in Fig. 2.2. This helps convey the purpose of the functions the activities perform. In other decomposition approaches, some other naming policy may be more appropriate. In any case, names must follow the rules of legal element name, that is, they start with an alphabetic character, and consist of alphanumeric characters and underscores. See App. A.1.

We may further decompose subactivities into subsubactivities on lower levels, and the new activities may be drawn inside their *parent activities* in the same chart. See Fig. 2.3, in which `SET_UP` is decomposed into three subsubactivities. We use the terms *descendants* and *ancestors* to denote subactivities and parent activities, respectively, on any level of nesting. Activities that have no descendants are termed *basic,* while those that do are called *nonbasic.* Two activities with a common parent may not have the same name, but subactivities of different parents may be named identically.

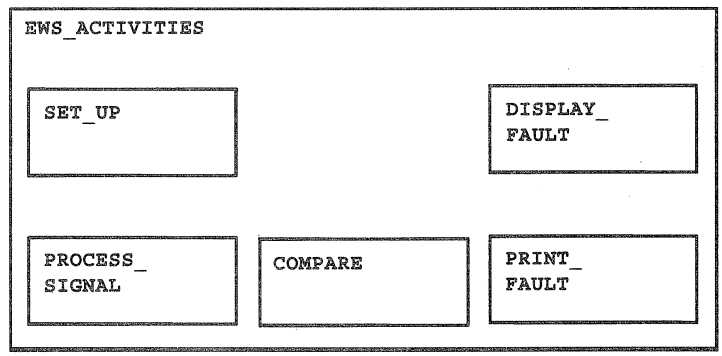

Figure 2.2 First-level decomposition of an activity.

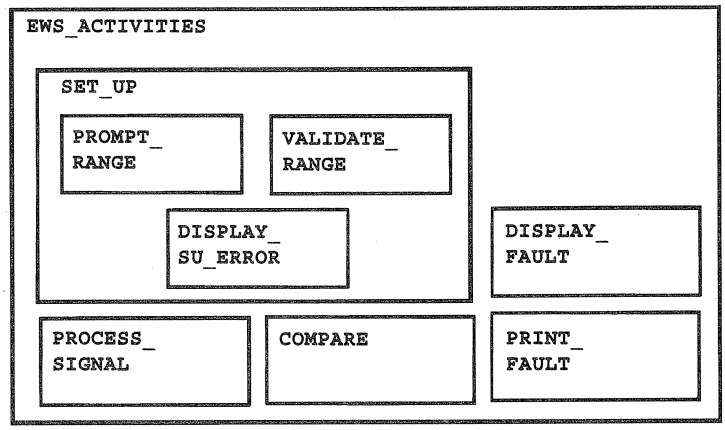

Figure 2.3 Multi-level decomposition of an activity.

All the activities appearing in the preceding examples are referred to as *internal regular activities,* to distinguish them from other types of activities participating in the functional description, which are discussed later.

Like many of the elements in our languages, some of the information related to activities is represented nongraphically. Each activity has a corresponding item in the Data Dictionary, which may contain additional information about it, such as textual descriptions, attributes, and relationships with other elements. Parts of the activity's Data Dictionary item are used to complete the description of its functionality, as discussed in Sec. 2.4.

2.2.2 The context of an activity

The functional description of a system may consist of multiple activity-charts linked together. Each such chart focuses on a portion of the system's functionality. It may describe the functionality of the entire system or that of some of its subsystems, or it may concentrate on some specific capability, object, or process being defined as a functional component in the higher-level decomposition. In each case, it is important to delineate the borders of the described portion, separating it from its environment, and to represent the flow of information between the two.

Each activity-chart contains one top-level box, with solid-line edges. This box represents the *top-level activity* of the chart, and its borderline separates this activity (and its internal description) from its environment. The components that constitute the environment are always referred to as *external activities* of the considered chart, although they may correspond to physical modules, humans, or activities or datastores that are internal to other activity-charts in the overall model. Of course, they may also be real environment entities, external to the

entire system under description. This issue will become clearer in later chapters, where the relations between charts in a full model are described.

External activities are depicted as boxes with dashed-line edges, which are located outside the top-level activity. They have the same names as the modules, the humans, the other activities (external, internal or control), or the data-stores that they represent in other parts of the specification.

For example, the environment of the EWS, as presented in Fig. 2.1, consists of two components, the OPERATOR and the SENSOR. They are drawn as external activities in the activity-chart of Fig. 2.5 that describes the overall functionality of the EWS.

Several external boxes in an activity-chart may bear the same name, in which case they are considered as representing the same external activity and are merely duplicated to help keep the chart uncluttered. Thus, for example, a flow-line (see Sec. 2.3.1) that represents the flow of information between an internal activity and an external one can be drawn to connect to the closest occurrence of the latter activity. When the identity of a particular external component is unknown or is irrelevant, it may be represented by an unnamed external activity box.

External activities are beyond the scope of the chart and are therefore not decomposed further into subactivities. Later we shall see that representing information flow between them is not allowed either.

2.3 Flow of Information between Activities

2.3.1 Flow-lines

To complete the functional view of the system, we complement the description of the activities themselves with the identification of inputs and outputs and the *flow of information* among subactivities.

We use the word *flow* to capture the communication and the transfer of information between activities. This flow of information can serve as a means not only to transfer data but to post commands and to synchronize by exchanging control signals. As in data-flow diagrams, we use labeled arrows for the visual representation of this flow. We refer to these connections as *a-flow-lines* (for *activity-chart flow-lines*), or just *flow-lines* for short.

The *label* on a flow-line denotes either a single information element that flows along the line (i.e., a *data-item,* a *condition,* or an *event*) or a group of such elements. We call a grouping of several information elements an *information-flow.* The flowing elements are used to specify communication according to the general specification approach that is adopted by the modeler. In particular, in the functional decomposition method they correspond to data and control flow.

A flow-line originates from its *source activity,* which is the activity that produces the information elements described in the flow-line's label, and it leads to its *target activity,* which is the one that consumes those elements. The communicating activities may belong to different levels in a multilevel activity-chart (see Fig. 2.4), but both cannot be external.

Referring to Fig. 2.4, we say that Y flows from A1 to B1, and U flows from A1 to A2. We also say that Y is an output of A and an input of B because the flow-line labeled with Y exits A and enters B, crossing their respective borderlines.

One of the graphical features present in all of our languages is that an arrow can be connected to a nonbasic box. In general, this means that the arrow is relevant to all the subboxes contained within the box in question. (See the discussion of this feature in the general setting of *higraphs* in Harel 1988.) In Activity-charts, this feature can take the form of a flow-line that leads to the edge of a nonbasic activity A but does not cross it. The arrow is taken to represent flow of information to all A's descendants. For example, the signal Z in Fig. 2.4 is accessible to both A1 and A2. Similarly, an arrow departing from the borderline of a nonbasic activity denotes the possibility that the corresponding information is produced by any of the descendant activities. For example, the arrow on the right-hand side of Fig. 2.4, emanating from B and labeled by V, can represent a global variable that is modified by the two activities B1 and B2, but it is used only by B2. Note that this convention enables us to replace several flow-lines from or to subactivities by one arrow from or to the parent, thus better representing the modeled flow. We also use this convention in cases where most of the subactivities consume or produce the information, but we do not want to specify exactly which ones they are.

Two types of flow-lines are allowed in Activity-charts: *data flow-lines,* drawn as solid arrows, and *control flow-lines,* drawn as dashed arrows. Typically, control flow-lines carry information or signals that are used in making control decisions (e.g., commands or synchronization messages) while data flow-lines carry information that is used in computations and data-processing operations. The different line types

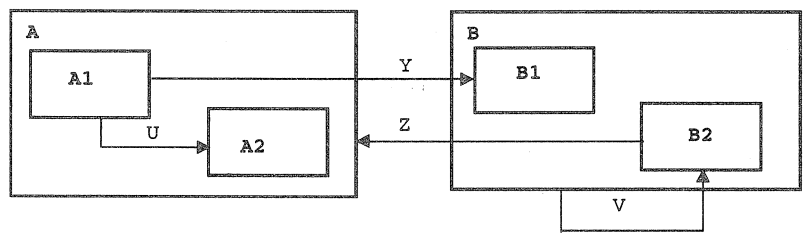

Figure 2.4 Flow of information among levels.

are intended to make this distinction visually. There are no clear criteria for deciding whether the flow of a given information element should be represented by a control flow-line or a data flow-line, but very often the source or target of a control flow-line will be the control activity that makes control decisions, as described in the next section.

Looking ahead to Fig. 2.5 for a moment, we see an illustration of the interface of the activity EWS_ACTIVITIES with its environment, and the flow of information between its subactivities. The figure illustrates both data and control flow-lines. The SIGNAL flowing along the data flow-line from the SENSOR to PROCESS_SIGNAL is used in data processing, while the OPERATOR's COMMANDS, flowing along the control flow-line, are used to decide control issues, such as which activities will be activated.

As hinted before, flow-lines in an activity-chart do not, by themselves, represent any specific method of transferring the information between the activities they connect, nor do they enforce or imply any timing specifications. Flow-lines may represent a variety of means for information transfer, such as parameter passing to procedures or global variables in software programs, messages transferred along transmission lines in distributed systems or through queues between tasks in real-time software applications, as well as signals flowing along physical links in hardware systems. They can also be used to represent the flow of tangible matter or energy.

The flow itself can be continuous or discrete in time, and the target and source activities are not necessarily active at the time of writing or reading the transferred data. Only an event appearing on a flow-line implies some timing constraints, because it is an information element with a specific time-related behavior. A special kind of element

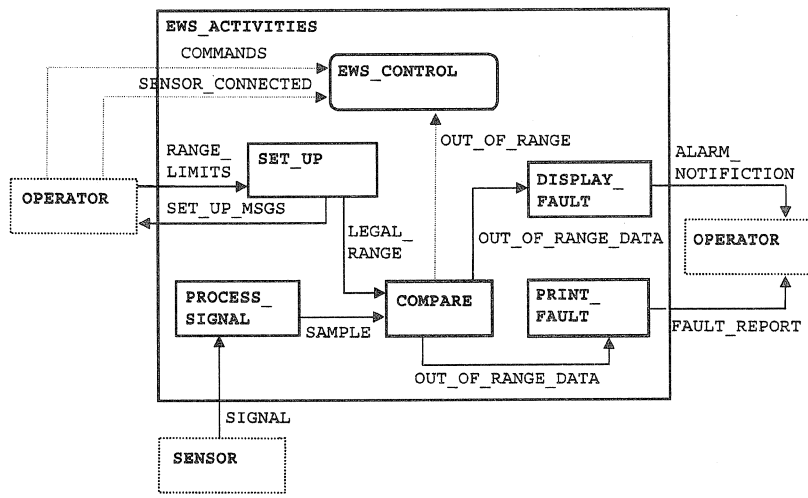

Figure 2.5 EWS_ACTIVITIES, its environment, and flow of information.

that is discussed later on, called a *data-store,* can be used to depict the presence of persistent data for lengthy periods, but a regular flow-line can serve the same purpose. Nevertheless, we emphasize that the dynamic aspects of the actual data transfer are not described in the activity-chart but in the statecharts or minispecs associated with the relevant activities, as explained in Chap. 8.

2.3.2 Flowing elements

We have already said that the information that flows between activities and is processed by them is an essential component of the functional view of a system. Three types of information elements may flow between activities: *events, conditions,* and *data-items.* The differences are in their domains of values and their timing characteristics. Any of them can appear as the label of a flow-line.

Events are instantaneous signals used for synchronization purposes. They indicate that something has happened. In the EWS example, the activity `COMPARE` issues the event `OUT_OF_RANGE` to indicate that the tested value has been determined to be out of the expected range.

Conditions are persistent signals that may be either true or false. For example, the `OPERATOR` in the EWS model sets the condition signal `SENSOR_CONNECTED`, whose truth value indicates whether or not the `SENSOR` is connected to the system—an essential prerequisite to activating the signal processing.

Data-items may hold values of various types and structures, like variables in programming languages. They can be of basic types, such as integer, real, bit, string, and so on, or of grouped types, such as records or unions. They can also be arrays or queues. In the EWS, the `SIGNAL` that comes from the `SENSOR` to be processed by the `PROCESS_SIGNAL` activity is of a numeric type (integer or real), while the `LEGAL_RANGE`, to which the processed value is compared, is a record consisting of two numeric fields: `HIGH_LIMIT` and `LOW_LIMIT`.

Figure 2.5 illustrates how these elements appear in the activity-chart labeling the flow-lines. The top-level activity, `EWS_ACTIVITIES`, is surrounded by external activities, and the figure represents both the interface with the environment and the internal flow of information. Among other subactivities of `EWS_ACTIVITIES`, the figure shows the activity `EWS_CONTROL`, which is a special type of activity—a control activity—that will be discussed in Sec. 2.4.1.

All three types of information elements—events, conditions, and data-items—can be organized in an array structure. The flow of information between activities can consist of an entire array, denoted by its name, with no indexing notation. When an activity deals with individual components of an array we can label the flow-line with the component identification. A typical case is shown in Fig. 2.6, in which each of three similar activities, `A1`, `A2`, and `A3`, takes care of one

component of an array V, and produces a corresponding component of an array W. Similarly, a flow-line can be labeled by a portion of an array, such as V(1..8), or by a record or a union component, such as R.X.

Information elements do not just appear along flow-lines. Their main use is in behavioral description. Using the Data Dictionary, one can define an information element that depends on the status or values of other elements. For example, we may define an event whose occurrence depends on the occurrence of other events, or a data-item whose value is expressed by values of other data-items. Information elements that have been defined in such a way cannot be used as labels on flow-lines.

We shall discuss the information elements in more detail in Chaps. 3 and 5.

2.3.3 Information-flows

The number of flow-lines in an activity-chart can be reduced by grouping information elements into an *information-flow,* which is used to label a common flow-line, thus helping a viewer to better comprehend the specification. The contents of the information-flow are defined in the Data Dictionary, associated with the name of the information-flow, as illustrated in Fig. 2.7. In the figure, the information-flow COMMANDS, labeling a flow-line from OPERATOR to the control activity, is a compact representation of three separate flow-lines, each of which is labeled by an individual component event. Using the three commands, SET_UP, EXECUTE, and RESET, the OPERATOR controls the operation of the EWS.

We should emphasize that because an information-flow is merely an abbreviation of several flow-lines, the elements it contains do not necessarily flow together. Also, an information-flow may be further decomposed into other information-flows or into concrete information elements (data-items, conditions, events, or array or record components).

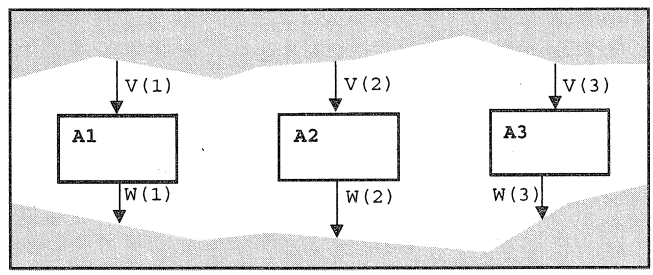

Figure 2.6 Array components labeling flow-lines.

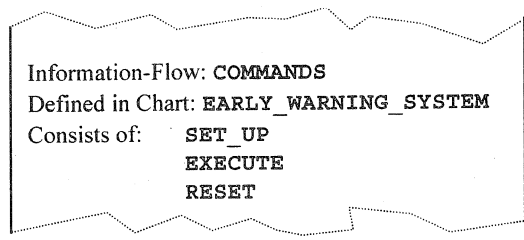

Figure 2.7 Information-flow COMMANDS in Data Dictionary.

Another way of using the information-flow feature is to consider it as the name of a link (or interface) between activities. This idea may be used as follows: at an initial stage, before getting into more detail, we can connect activity A1 to activity A2 by a flow-line labeled with some noncommitting information-flow, such as A1_TO_A2. The contents of this line may then become increasingly more concrete, by filling in more of its contents in the corresponding information-flow item in the Data Dictionary. Clearly, this can be carried out repeatedly for nested information-flows. In any case, we expect the contents of all information-flows to be eventually specified in full.

2.3.4 Data-stores

As mentioned earlier, there are no restrictions on the time that data reside on a flow-line. Data produced by the source activity are available to the target activity even when the source activity is no longer active. In this sense, a flow-line may be viewed as a kind of storage unit. Nevertheless, it is often more natural to incorporate an explicit *data-store* in the chart, which serves to represent information that is stored for later use. In addition, a data-store may be used to specify the aggregation of large volumes of data that accumulate continuously over time. Data-stores can be used to describe a buffer in computer memory, a message queue, a file on a disk, a database, or even a single variable. In object-based decomposition, a data-store can be used to encapsulate the object data.

Information is written into the data-store by one or more activities and can be read by other (possibly the same) activities. Thus, the data-store can be viewed as a "passive" activity, that is, one that does not change or produce information.

Data-stores are drawn as rectangular boxes with dashed vertical edges. The name of a data-store may be any legal name (see App. A.1), but it must be unique among its sibling activities and data-stores.

Data-stores are always basic; they cannot contain other data-stores or activities. The internal structure of a data-store may be defined by associating it with a data-item. To do this, a data-item is defined in the

Data Dictionary with the same name as the data-store. Any structure then given to this data-item is inherited by the data-store. For example, to specify that the data-store Q is a queue containing records of a certain type, say, MESSAGE, one defines the data-item Q in the Data Dictionary as a queue of the user-defined type MESSAGE, the structure of which is described separately.

In the EWS example, we might want to show that the record LEGAL_RANGE, composed of HIGH_LIMIT and LOW_LIMIT, is stored in a data-store by the SET_UP activity and consumed by COMPARE. To represent this, the flow-line labeled LEGAL_RANGE in Fig. 2.5 is replaced by a data-store LEGAL_RANGE that contains the appropriate record, and it is then connected to the source and target activities. The appropriate part of the resulting diagram is presented in Fig. 2.8. LEGAL_RANGE is defined as a record data-item in the Data Dictionary, as shown in Sec. 3.4.2.

Notice that the lines flowing to and from the data-store LEGAL_RANGE are not labeled. This is because we can name the data-store with the same name as the data-item flowing to or from it, in which case the labels on the corresponding flow-lines can be omitted. However, in general, a data-store's inputs and outputs can be any information elements, even when there is a data-item matched (by name) to this data-store. Data-stores can also store control elements to be used for control decisions, so control flow-lines can flow to and from data-stores, too. Nevertheless, it is meaningless to have an event, which is of transient nature, stored in or flowing to or from a data-store.

Data-stores cannot be drawn as part of the activity-chart's environment. The components of the environment are always drawn as external activities even when their functionality is that of storage.

Textual descriptions of data-stores, and the relationships they may have with other elements, are entered in the Data Dictionary.

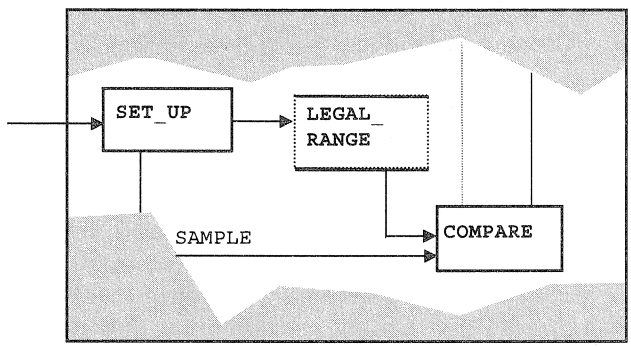

Figure 2.8 Data-store containing LEGAL_RANGE data.

2.4 Describing the Behavioral Functionality of Activities

We have seen that the functionality of the system is described by decomposing activities into subactivities and data-stores, and identifying the information that flows between them. This can be done repeatedly, until basic activities are reached, but it is not enough to present the full picture.

For *nonbasic* activities, which are decomposed into subactivities, we must provide information about the behavioral dynamics of the decomposition. In the methodology of Hatley and Pirbhai (1987) this issue is covered by what they call process activation tables. Other approaches deal with this differently. In our approach, we use the control activities for this (and more), as will be seen shortly. *Basic* activities are described by other means, which are specified via the Data Dictionary entry associated with the activity.

Describing the behavior of activities in our approach is a broad subject, and it is discussed in many of the later chapters. The present section should be viewed as an introduction.

2.4.1 Control activities

In many systems, the activities at each level of the functional decomposition perform their functions in a simple fashion. Some are continuously active, consuming their inputs and producing their outputs periodically. Others become active when their inputs arrive and stop when they have produced the outputs corresponding to these inputs. Sometimes, the behavior of activities follows more intricate patterns.

We address these aspects by introducing special *control activities,* which are drawn as subactivities of regular internal activities and whose function is to control their sibling activities. For example, as we shall see in Chap. 7, a control activity may explicitly start and stop its sibling activities. In the EWS model, `EWS_CONTROL` is responsible for determining the activation and deactivation of all the activities on the same level, that is, `SET_UP`, `PROCESS_SIGNAL`, `COMPARE`, and so on. (See Fig. 2.9.)

The control activity will typically receive signals from the siblings it controls or from other sources, make decisions based on them, and then, in turn, start and stop the activities it controls and produce signals that are consumed by its environment. In our example, the control activity `EWS_CONTROL` receives and reacts to the commands of the `OPERATOR` and to the `OUT_OF_RANGE` event generated by the `COMPARE` activity. (See Fig. 2.5.)

The control activity is depicted as a rectangle with rounded corners, and it cannot have subactivities. Rather, its specification is described in the language of *Statecharts,* the graphical language for

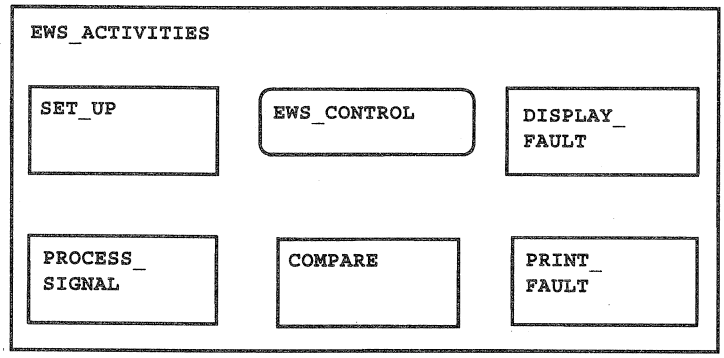

Figure 2.9 A control activity in an activity-chart.

modeling behavior. The control activity points to the statechart describing its behavior through its name, as explained in Chap. 7. The Statecharts language is described in Chap. 4, and the way a statechart controls the behavior of activities is discussed in Chap. 7.

Each activity may have at most one control activity. When an activity requires no further decomposition and its behavior can be conveniently described by a statechart alone, the control activity is its only subactivity. This situation is common in certain highly reactive systems. Like other elements, the control activity has an associated item in the Data Dictionary.

2.4.2 Activities in the Data Dictionary

As mentioned earlier, almost every element in our models has a corresponding entry in the Data Dictionary, in which various kinds of textual information about the element can be specified. Such additional information can be formal (i.e., possessing some semantics that is relevant to the model and its behavior) or informal. Some kinds of textual information are relevant to all types of elements, such as a one-line *short description* and an unlimited textual *long description.* These narrative additions, especially the long description, can be used to provide information about the element in an informal language, for the record. In addition, the general mechanism of an *attribute pair,* name and value, can be used to associate special characteristics with the element, as we shall see later on. The Data Dictionary can also used to associate a *synonym* with the element, usually a shorter name that is easier to incorporate into a detailed chart.

In the case of activities, the long description is very often used to add functional specification in a textual language that is not an integral part of our approach, such as an unstructured natural language. This additional information can be attached to basic or nonbasic activities alike.

On the other hand, for basic activities, our approach supports a number of formal executable textual descriptions that specify particular patterns of behavior. These are also associated with the activity in its Data Dictionary entry. The patterns are:

- A *reactive event-driven activity* is continuously "active" in an idle state and constantly waits for an event to occur and to cause it to perform some action. It then becomes idle again until the next event happens. An example of such an activity is a simple keyboard driver that accepts key press events and locally performs a very simple operation and/or transfers a command to some other activity. A reactive event-driven activity can be described by a *reactive minispec,* which is a list of reactions, each consisting of a trigger event and its implied action; see Fig. 2.10*a*. More complex reactive activities are described by statecharts, as we shall see, but simple event/action activities do not require a statechart, and they can be described by a reactive minispec.
- A *procedure-like activity,* when invoked, performs a sequence of operational statements and then stops. An example of such an activity is the VALIDATE_RANGE subactivity of the range SET_UP activity of the

```
Activity: PROCESS_SIGNAL
Defined in Chart: EWS_ACTIVITIES
Mini-spec: st/TICK;;
TICK/ $SIGNAL_VALUE:=SIGNAL;
      SAMPLE:=COMPUTE($SIGNAL_VALUE)
```

(a) Event-driven activity described by a mini-spec

```
Activity: VALIDATE_RANGE
Defined in Chart: SET_UP
Mini-spec: if (LOW_LIMIT < HIGH_LIMIT)
        then SUCCESS
        else FAILURE end if
```

(b) Procedure-like activity described by a mini-spec

```
Activity: COMPUTE_IN_RANGE
Defined in Chart: COMPARE
Combinational Assignments:
IN_RANGE:=(SAMPLE>LEGAL_RANGE.LOW_LIMIT)
     and (SAMPLE>LEGAL_RANGE.HIGH_LIMIT)
```

(c) Data-driven activity described by combinational assignments

Figure 2.10 Data Dictionary entries describing activities.

EWS. It is invoked when the user has inserted the range limits, and it checks the validity of the values, returning the check results. A procedure-like activity can be described by a *procedure-like minispec,* which is simply a list of actions; see Fig. 2.10*b*.

- A *data-driven activity* is also continuously "active," checking to detect any changes in the values of its inputs. When any of them changes value, the activity computes new output values and resumes its waiting. A logical gate in an integrated circuit is an example of a simple data-driven activity. In the EWS example, the COMPARE function has a subactivity COMPUTE_IN_RANGE, which is data-driven; it continuously monitors the processed signal and compares it to the legal range limits to calculate an IN_RANGE condition. (When this condition becomes false, the COMPARE function issues the OUT_OF_RANGE event.) A data-driven activity can be described by a collection of *combinational assignments,* which are ordinary-looking assignment statements that continuously compute the activity's outputs based on its inputs; see Fig. 2.10*c*.

Minispecs and combinational assignments are described in detail in Chap. 7.

2.5 Connectors and Compound Flow-Lines

Let us return to the technical mechanisms we provide for representing the flow of information between activities. Flow-lines in activity-charts can be combined using various types of connectors. The main motivation for this is to economize in the number of arrows, to reduce clutter, and to provide a clearer and more intuitive graphical representation. We refer to the resulting connected object, consisting of a number of flow-lines and connectors, as a *compound flow-line.* We now discuss the various types of connectors.

2.5.1 Joint connectors (fork and merge constructs)

A *fork* construct allows us to represent a single information element as flowing from one source to several targets. Instead of drawing separate lines departing from the source, we can draw a single departing line, which then splits up into separate arrows at a convenient place in the chart. For example, instead of drawing two separate lines emanating from COMPARE and labeled with OUT_OF_RANGE_DATA, as we did in Fig. 2.5, we can abbreviate as shown in Fig. 2.11.

Similarly, we can represent common information flowing from several sources to a single target by joining them at some convenient point before they reach their target. This is called a *merge construct,* and it

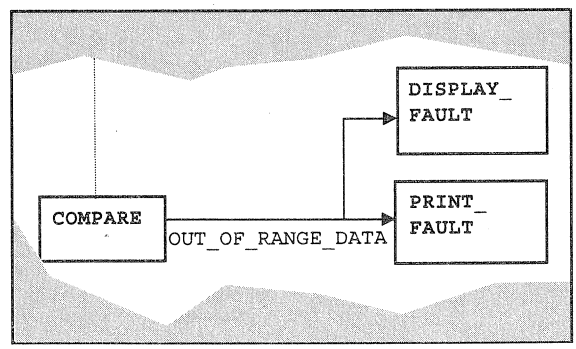

Figure 2.11 A joint connector (a fork construct).

indicates that the target may receive the information from either of several sources.

In both constructs, fork and merge, we refer to the connection point as a *joint connector.* The compound flow-line, consisting of the connected segments, may have several sources and several targets but only a single associated flowing element (which may actually be an information-flow consisting of several data elements). As for location, the flow element common to the entire construct can label any of the compound flow-line's segments.

2.5.2 Junction connectors

Another way of reducing the number of lengthy flow-lines in an activity-chart is to use a *junction connector.* Several flow-lines conveying different information elements may be connected using a junction connector to form a single flow-line that emanates from or enters a common box or connector.

Figure 2.12 illustrates several uses of junction connectors. Figure 2.12*a* contains three actual flows: `X` flows from `A1` to `B`, `Y` flows from `A2` to `B`, and `Z` flows from `A3` to `B`. Notice that the line segment from the junction connector to `B` is unlabeled because it is used only to connect the different flowing elements to the common target.

The case of a common source is similar. In Fig. 2.12*b,* the flow-line that carries the three flow elements from `A` to the junction connector is labeled `XYZ`. In the Data Dictionary we define the element `XYZ` to be an information-flow containing `X`, `Y`, and `Z` as components. Clustering flowing elements in this way and using the combined information-flow to label the common arrow is usually done when there is some logical relationship between the flowing elements; the additional name helps to clarify this relationship.

Figure 2.12*c* illustrates how a number of junction connectors may be combined. Nine potential routes exist from the activities on the left to

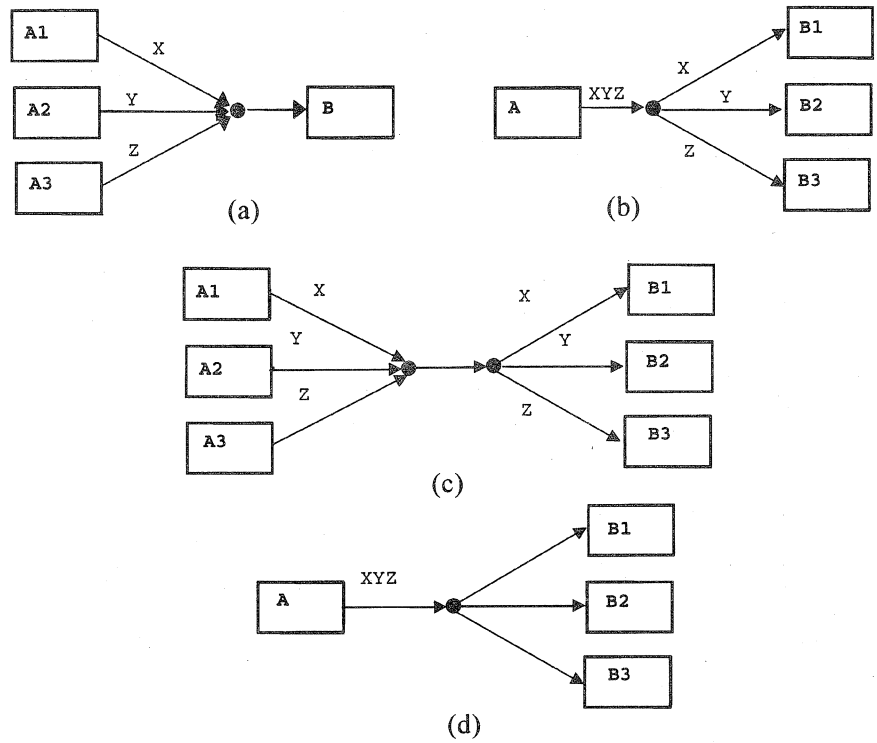

Figure 2.12 Junction connectors.

those on the right. However, the labeling used excludes six. The only three that represent actual flows are: X from A1 to B1, Y from A2 to B2, and Z from A3 to B3. A compound line with contradicting flow labels (such as the one composed of the segments labeled X and Y) is not considered a viable compound flow-line.

If we want to show more than one element flowing along a single line, the elements can be combined using an information-flow. Consider the example in Fig. 2.12*d*. It represents three compound flow-lines, each of which carries XYZ and has a single source and a single target. Notice that the same diagram drawn with a joint connector instead of a junction connector represents a single compound flow-line with one source and three targets. In this case the two are semantically equivalent, and although we used a junction connector, a joint connector might be preferred because it emphasizes that the same information is available to all three targets.

The junction connector is sometimes used with a record data-item and its components. In the EWS example, the COMPARE activity can be decomposed into two subactivities: one compares the processed signal SAMPLE with the HIGH_LIMIT field of LEGAL_RANGE, and the other

compares it with the LOW_LIMIT field, as shown in Fig. 2.13. The junction connector is used here to direct the fields of the record to two different target activities.

2.5.3 Diagram connectors

When the source of a flow-line is far from its target, we can avoid drawing a lengthy arrow by using a *diagram connector.* The arrow emanating from the source ends in a named connector, and its continuation emanates from a second connector with the same name, which is positioned closer to the target. The pair of identically named connectors are identified as the same logical entity, and the result has the same meaning as a junction connector connecting the two arrows. It is important to emphasize that the arrow segments are matched according to the names of the connectors and not according to the labels along the segments. As a consequence, the label can be omitted from one of the segments.

Any legal name (see App. A.1) may be used to label the diagram connectors, as can any integer number. Thus one can use names that indicate the identity of the target (as in Fig. 2.14), flowing signal names, or simply serial numbers.

To make life even easier, we allow more than two diagram connectors to have the same name and thus denote the same logical junction. Several arrows can then emanate from or enter a common diagram connector, but all arrows connected to the same occurrence of the connector must flow in the same direction.

2.5.4 Compound flow-lines

The various types of connectors presented earlier can be used to construct a variety of *compound flow-lines.* The compound flow-lines are really the logical flow-lines that depict the actual flow between activities

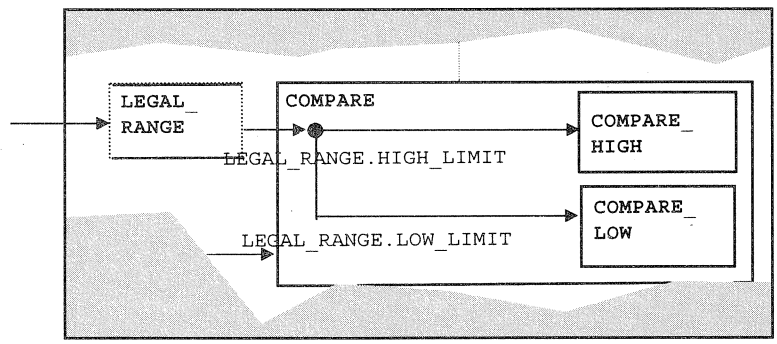

Figure 2.13 A junction connector with record fields.

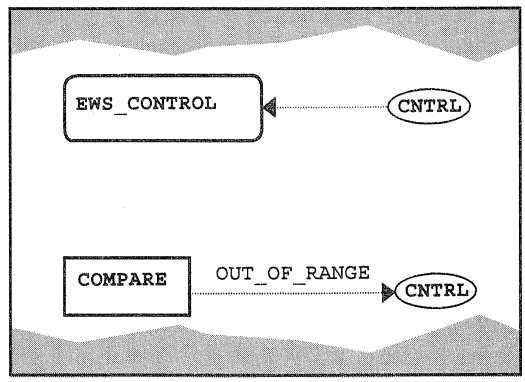

Figure 2.14 A diagram connector.

(or the other box-like entities in our other languages). When connectors are not used, a simple arrow that flows directly from one box to another depicts the actual flow, the logical flow-line consisting of a single segment. In Fig. 2.5, for example, no connectors are used, and all logical flow-lines are actually *simple flow-lines* (ones consisting of a single segment).

We have seen that a joint connector yields a single compound flow-line with multiple sources or multiple targets, while a junction connector produces multiple compound flow-lines. Diagram connectors are interpreted as junctions, and as such they can represent multiple compound flow-lines, although they can also be used in a way that results in a single flow-line.

The segments constituting a compound flow-line can be data flow-lines or control flow-lines. When both types appear in a single compound flow-line, the entire combination will be considered to be a control flow-line if the final segment that leads to the target is a control flow-line. This is because the type of flow is determined by the way the target uses the flowing information.