

Stemate Course

Kai Baukus

Stemate/SDL

W.-P. de Roever

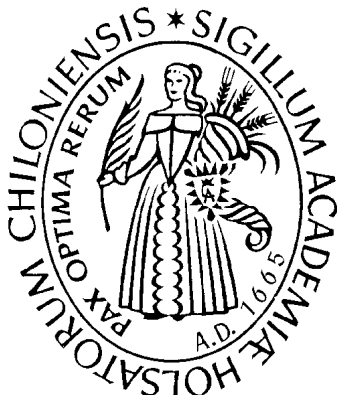
D. Hogrefe

K. Baukus

H. Neukirchen

CAU Kiel

MU Lübeck



The STATEMATE *Toolset*

The notion of a reactive system and the language Statecharts were introduced in the last session. We explained the rationale behind the design decisions of Statecharts in relation to the specific nature of reactive systems.

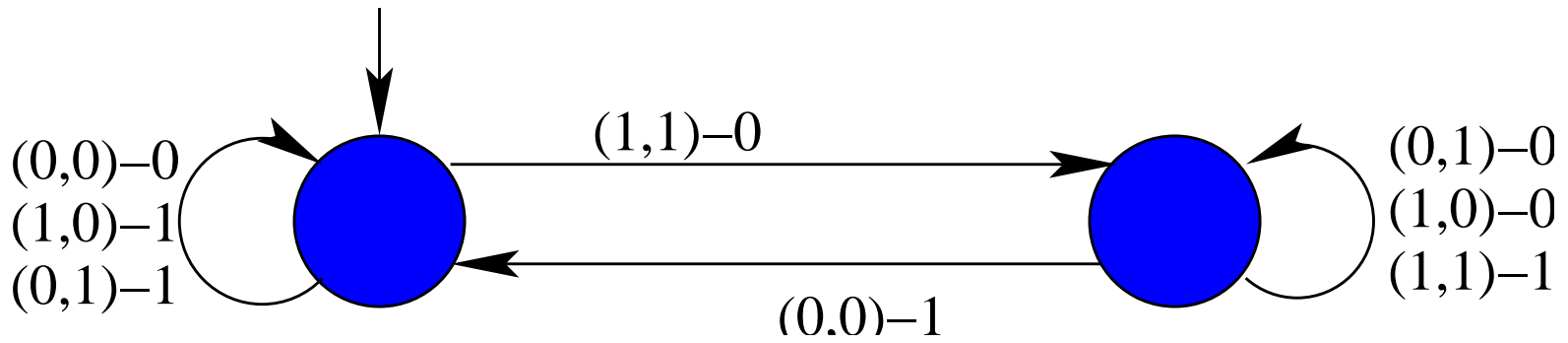
This time, the role of models in a system development life cycle is described. We introduce three languages to characterize reactive systems from different views. This results in a brief description of the STATEMATE toolset

[HP98] Modeling Reactive Systems with Statecharts: The STATEMATE Approach, D. Harel, M, Politi. McGraw-Hill, 1998.



Last Session

Serial Addition: Mealy Machine



Statecharts = Mealy Machines
+ depth
+ orthogonality
+ broadcast
+ data

- The elementary unit of observation in a reactive system is the **event**
- The **environment** sends events to the system to **trigger computations**, the system reacts to the environment by sending, or **generating**, events.
- Events are also means of **communication** between parts of a system.

Time (cont'd)

- Because one wants to specify reactive systems at the highest level of abstraction in a discrete fashion, events are discrete signals, occurring at a point in time.
- Events have no duration; they are generated from one state to another. Hence, transitions have a discrete uninterruptable nature and all time is spent in states.

Reaction Time

- We know that transitions have no duration, but when do they take place, relative to the trigger? And:

HOW LONG DOES IT TAKE THE SYSTEM TO COMPUTE A REACTION UPON AN EXTERNAL EVENT?

- For transformational systems this is easy — the only important distinction is between finite and infinite values (corresponding to a final state or no final state)
- For reactive systems this is not enough:
We have to know when an output occurs relative to the events in the input sequence (see Brock-Ackermann paradox) \implies
One has to determine the reaction time of a sequence.

Reaction Time

Summary : We want the execution time associated to reactions to have following properties:

- It should be accurate, but not depending on the actual implementation.
- It should be as short as possible, to avoid artificial delays.
- It should be abstract in the sense that the timing behavior must be orthogonal to the functional behavior.



Only choice that meets all wishes is **zero reaction time**.

Berry's synchrony hypothesis

This choice, that the reaction time between a trigger and its event is zero, is called **Berry's synchrony hypothesis**.
Is this implementable? No, a real computation takes time.
But in actual implementation this means:

The reaction comes before the next input arrives,
or, so to say,

Reactions are not infinitely fast but fast enough.

Grandfather Paradoxon

If **a** is absent, i.e., $\neg a$ holds as condition, transition t_1 is taken, i.e., **b** is generated, and hence t_2 , i.e., **b/a** is taken, generating **a** within the same time unit, i.e., in zero time, hence transition t_1 should not be taken.

But that means that event **b** is not generated, and hence event **a** is not generated, so transition t_1 should be taken, etc.

This is called the “Grandfather paradox”.

It’s solution is to order event occurrences causally, with later events not influencing earlier events: $\neg a \leq b \leq a$

Note here: this causal order has nothing to do with the passage of time; it merely refers to causal chains within one time step.

Solution

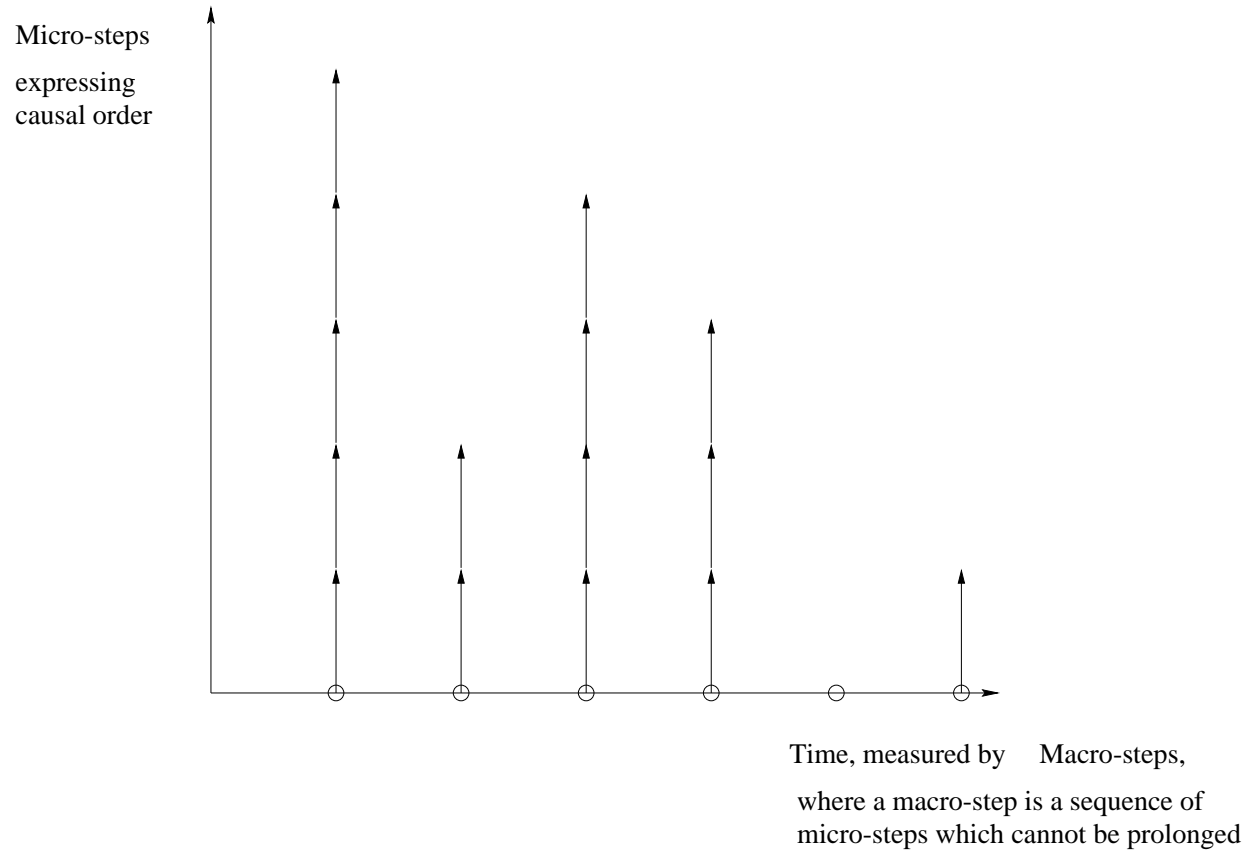
Introduce two levels of time

- **Macro steps**, for counting time, (these are observable) time steps, and
- **Micro steps**, which describe the causal chain within reactions.
Every macro-step is then divided in an arbitrary but finite number of micro-steps.

This sequence of micro-steps has only an operational meaning.

The New Semantics

This leads to a semantics of the following form:



Problems with New Semantics

The problem with macro-steps is that they lead to a globally inconsistent semantics.

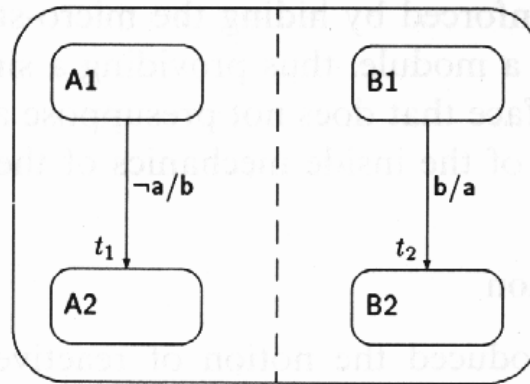


Fig. 8.

$$S_1 \xRightarrow{\emptyset}^b S_2 \xRightarrow{b}^a S_3$$

Here absence of triggers generates presence of triggers, which violates their absence within the same step.



Systems Life Cycle

Specification in a systems life cycle

- Identify several phases in the development life cycle of a system
- **Classic waterfall model:** requirements analysis, specification, design, implementation, testing, and maintenance.
- Other approaches center around prototyping, incremental development, reusable software, or automated synthesis.

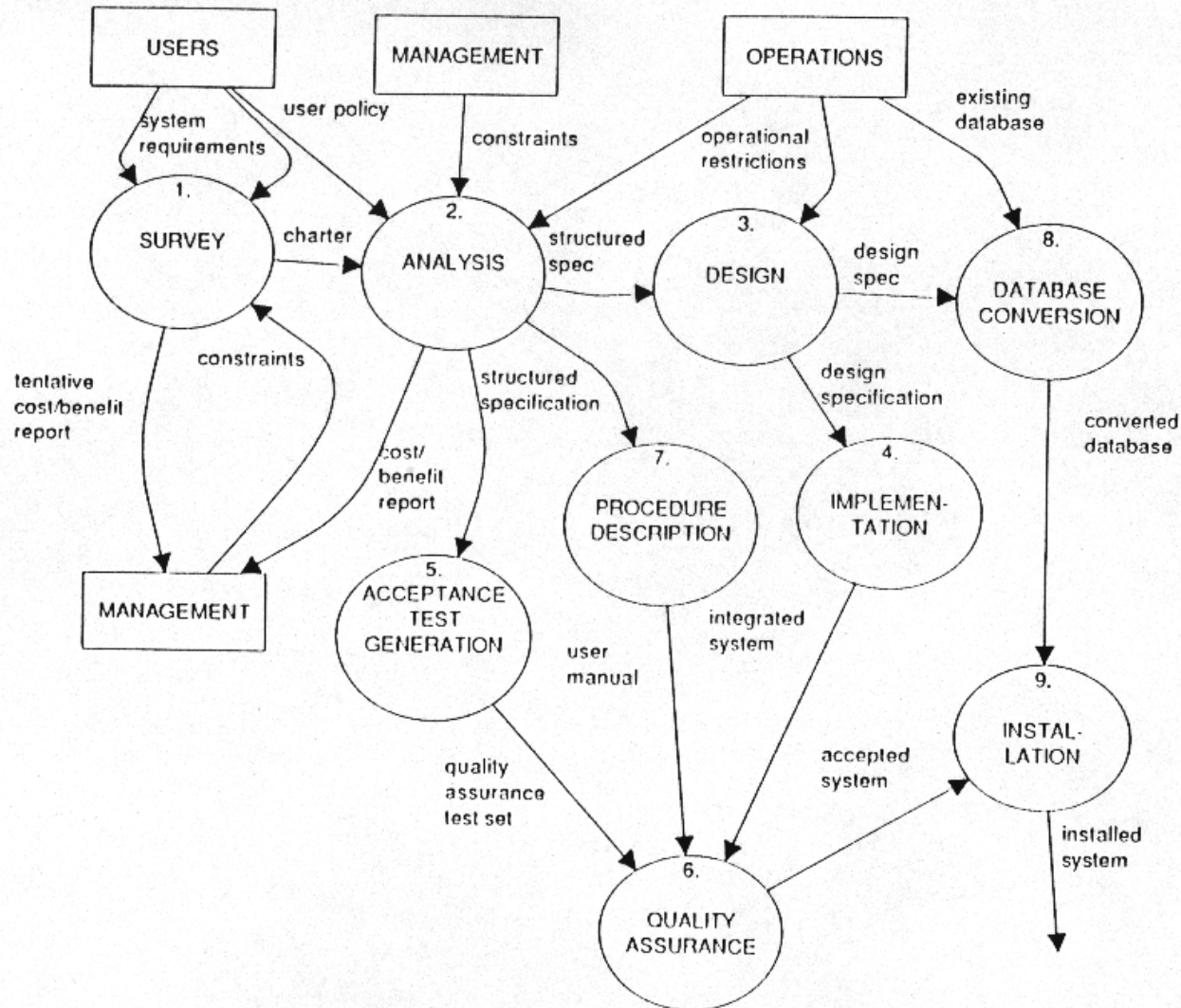
Requirements Analysis

- Most proposals contain a requirements analysis phase. Specification errors and misconceptions should be discovered in that early phase.
- Correcting errors in later stages is extremely expensive.
- Special languages are therefore used in the requirements analysis phase to specify a model of the system, and special techniques are used to analyze it extensively.

System's life cycle

MODERN STRUCTURED ANALYSIS

THE PROJECT LIFE CYCLE 89



System Model

- A good model is important for all participants in the system's development.
- Having a clear and executable model the functionality and behavior can be approved before investigating heavily in the implementation stages.
- The specification team uses modeling as the main medium for expressing ideas.

Methodology

A methodology provides guidelines for performing the processes that comprise the various phases. Concentrating on the modeling and analysis phase, a methodology consists of the following components:

- The methodology's underlying approach and the concepts it uses.
- The notation used, that is, the modeling languages with their syntax and semantics.
- The process prescribed by the methodology, that is, which activities have to be carried out to apply the methodology and in what order.
- The computerized tools that can be used to help in the process.

Reactive systems

The Statecharts language is especially effective for reactive systems.

A typical reactive system exhibits the following distinctive characteristics:

- It continuously interacts with its environment, using inputs and outputs that are either continuous in time or discrete.
- It must be able to respond to interrupts, i.e., high-priority events.
- Its operation and reaction often reflect stringent time requirements.
- It is very often based on interacting processes that operate in parallel.

Examples

On-line interactive systems: e.g., automatic teller machines, flight reservation systems

Computer-embedded systems: avionics, automotive, and telecommunication systems

Control systems: such as chemical and manufacturing systems.

The early warning system

A system model constitutes a tangible representation of the system's conceptual and physical properties and serves as a vehicle for the specifier and designer to capture their thoughts.

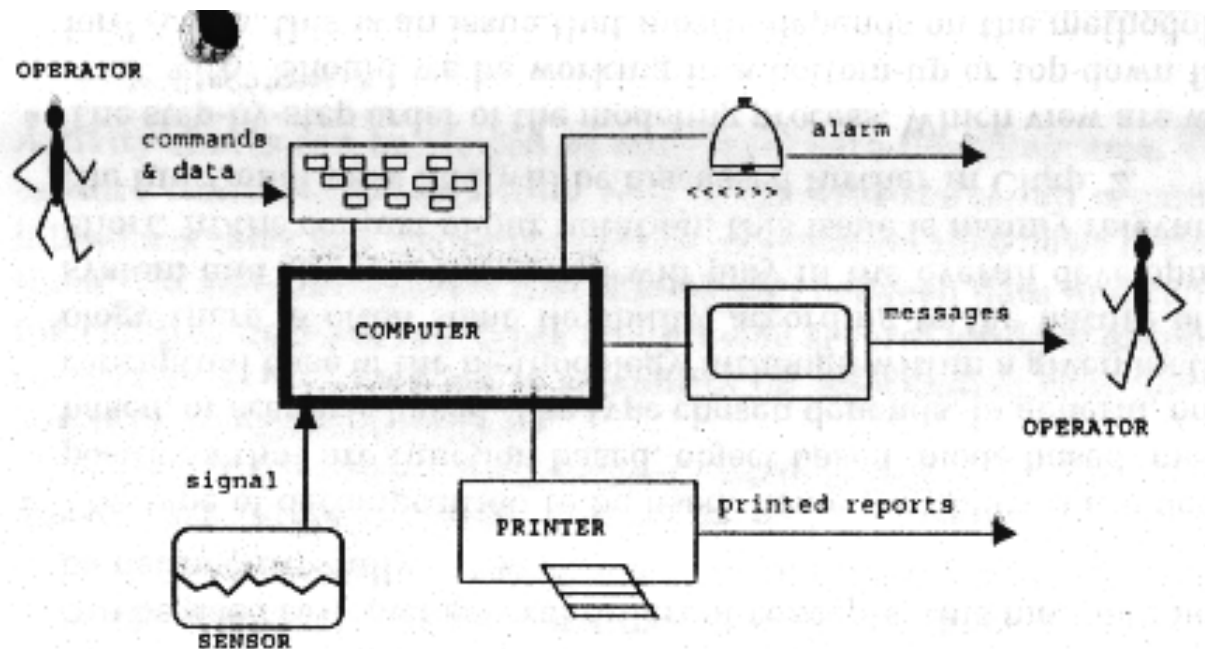


Figure 1.1 The early warning system (EWS).

Characteristics of models

Beside for communication, systems models should also be used for inspection and analysis.

When the model reflects some preexisting descriptions, such as requirements written in natural language, it is useful to keep track of how the components of the developing model are derived from the earlier descriptions.

The modeling languages used in STATEMATE have been designed with several important properties in mind:

- to be intuitive and clear
- to be precise
- to be comprehensive
- to be fully executable

How to achieve these properties?

- To achieve clarity, elements of the model are represented graphically whenever possible.
- For precision, all languages features have rigorous mathematical semantics
- Comprehension comes from the fact that the languages have the full expressive power needed to model all relevant issues, including the what, the when, and the how.
- For executability, the behavioral semantics is detailed and rigorous enough to enable the model to be executed (or be used to generate code).

Modeling Views

Building a model can be considered as a transition from ideas and informal descriptions to concrete descriptions that use concepts and predefined terminology.

Here, the descriptions used to capture the system specification are organized into three views: the **functional**, the **behavioral**, and the **structural**

Illustration

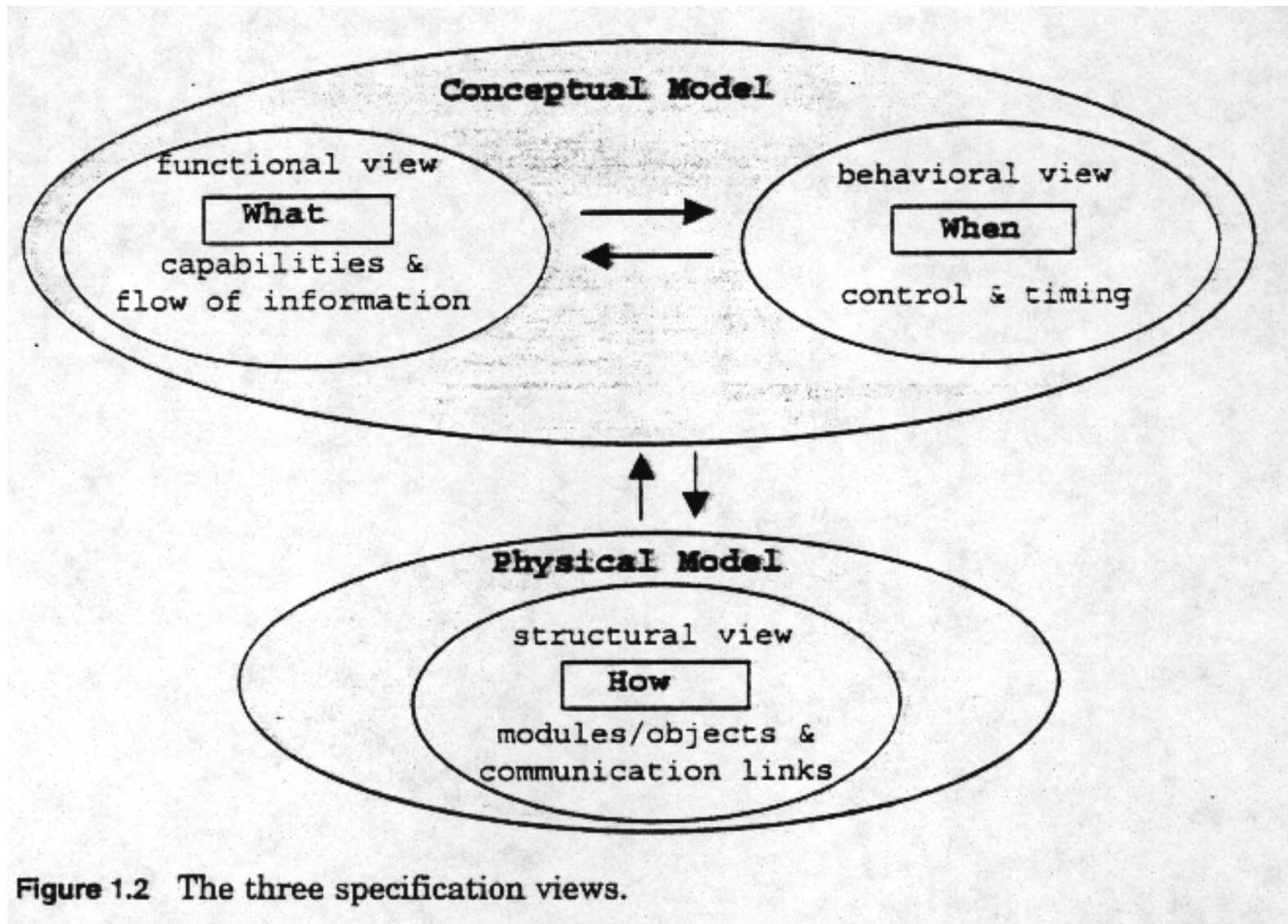


Figure 1.2 The three specification views.

The Three Views

Functional view : The functional view captures the “what”. It describes the system’s functions, processes, or objects, also called activities, thus pinning down its capabilities. This view includes the inputs and outputs of the activities.

Behavioral view : The behavioral view captures the “when”. It describes the system’s behavior over time, including the dynamics of activities, their control and timing behavior, the states and modes of the system, and the conditions and events that cause modes to change and other occurrences to take place.

Structural view : The structural view captures the “how”. It describes the subsystems, modules, or objects constituting the real system and the communication between them.

Connection

While the two former views provide the conceptual model of the system, the structural view is considered to be its physical model.

The main connection between the conceptual and physical models is captured by specifying the modules of the structural view that are responsible for implementing the activities in the functional view.

Modeling Heuristics

Modeling heuristics are guidelines for how the notation should be used to model the system.

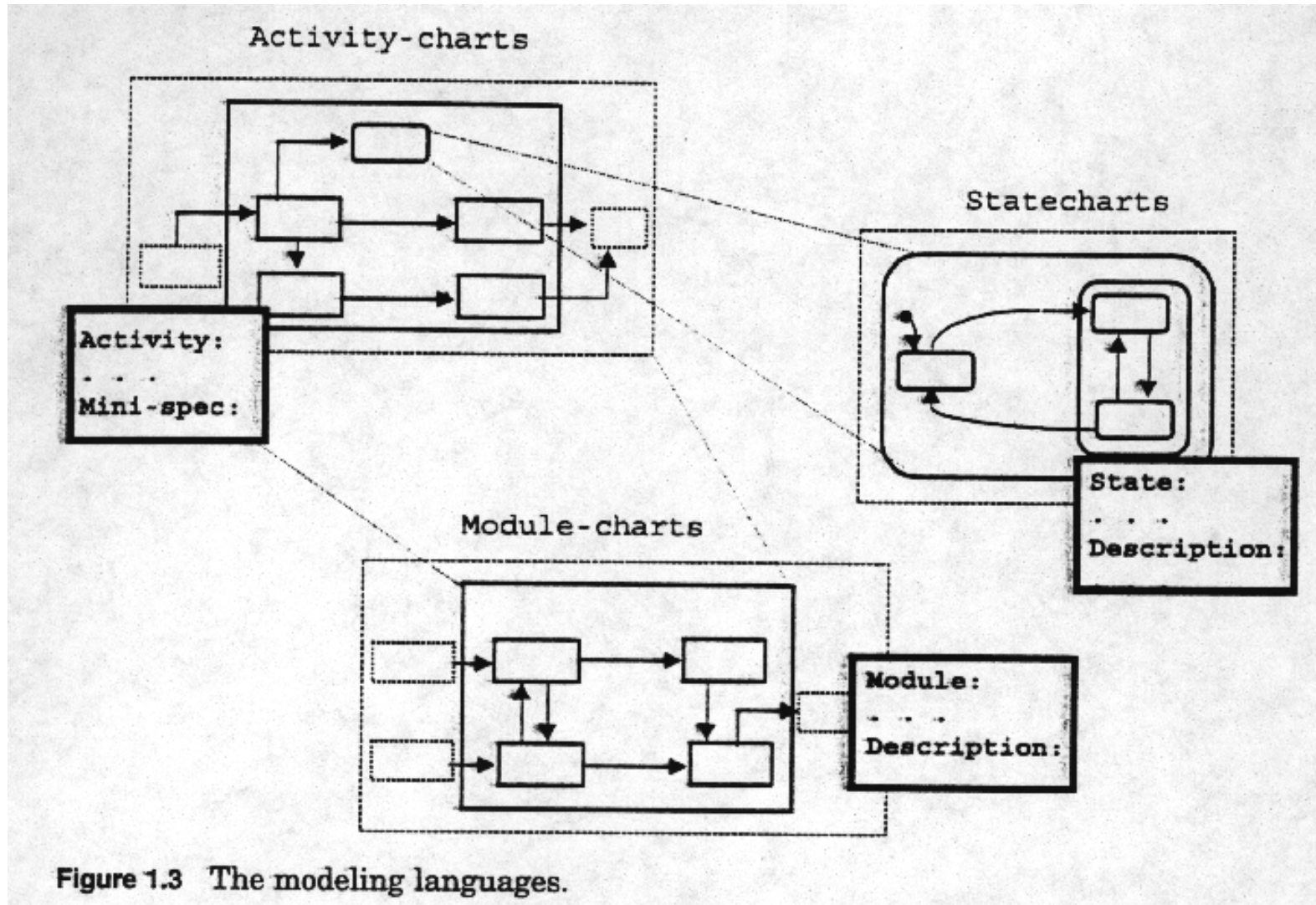
- The mapping between the methodology's concepts and the elements allowed in the notation.
- The type of decomposition to be used: e.g., function based, object based, mode based, module based, or scenario based.
- The step-by-step order of the modeling process: bottom-up or top-down

The Modeling Languages

The three views of a system model are described in our approach using three graphical languages.

- **Activity-charts** for the functional view,
- **Statecharts** for the behavioral view,
- and **Module-charts** for the structural view.
- Additional non-graphical information related to the views themselves and their inter-connections is provided in a **Data Dictionary**

Illustration



Activity-charts

Activity-charts can be viewed as multilevel data-flow diagrams. They capture functions, or activities, as well as data-stores, all organized into hierarchies and connected via the information that flows between them.

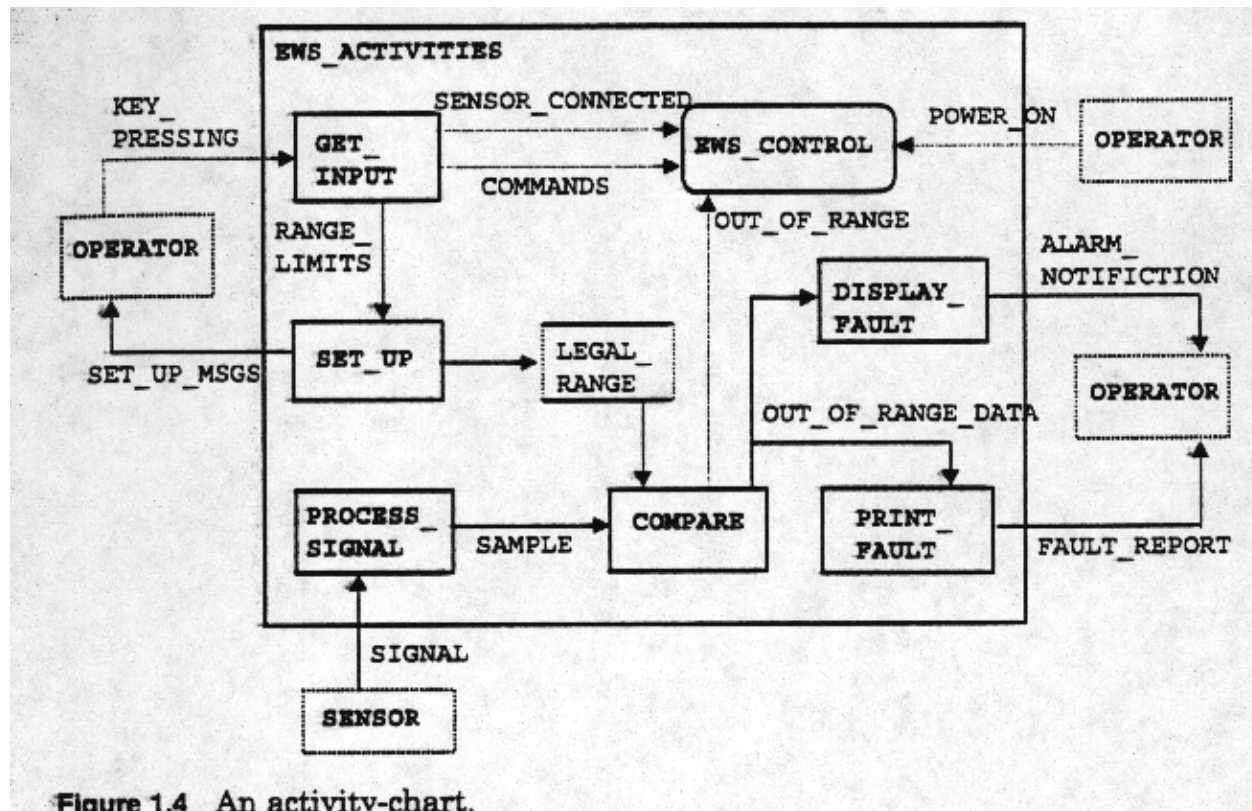
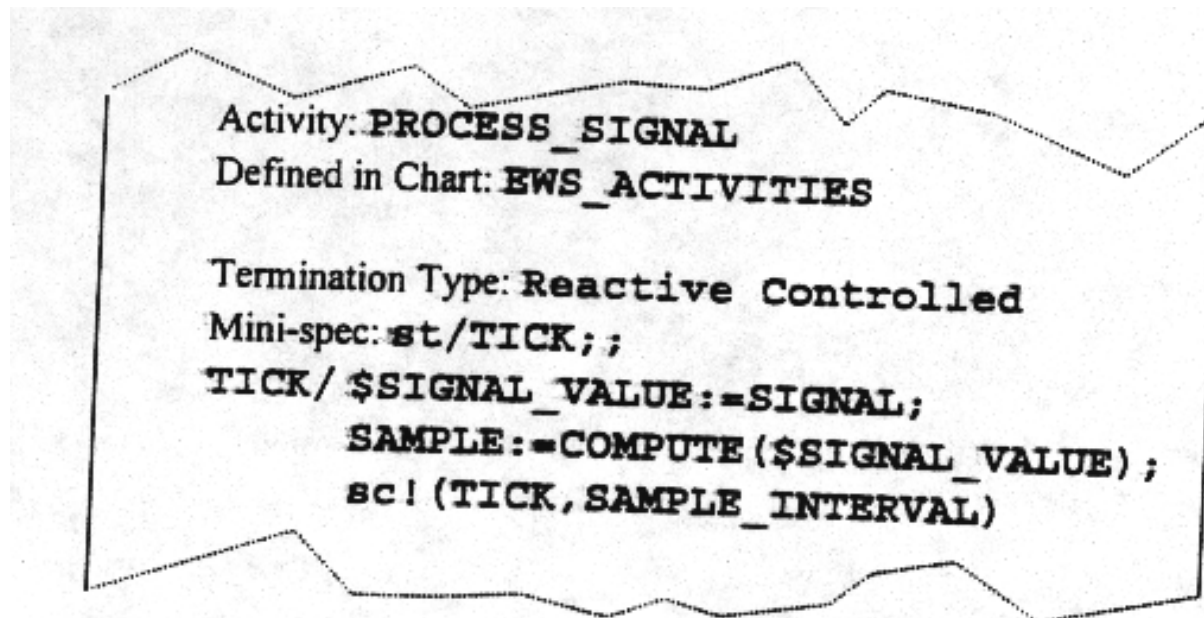


Figure 1.4 An activity-chart.

Non-graphical information

In addition to the graphical information, each element in the described has an entry in the Data Dictionary, which may contain non-graphical information about the element. For example, the activity entry contains fields called mini-spec and long description, in which it is possible to provide formal and informal textual descriptions of the activities workings.



The diagram shows a data dictionary entry for an activity. The entry is enclosed in a rectangular box with a jagged, hand-drawn border. The text inside the box is as follows:

```
Activity: PROCESS_SIGNAL
Defined in Chart: EWS_ACTIVITIES

Termination Type: Reactive Controlled
Mini-spec: st/TICK;;
TICK/ $SIGNAL_VALUE:=SIGNAL;
      SAMPLE:=COMPUTE($SIGNAL_VALUE);
      sc!(TICK, SAMPLE_INTERVAL)
```

Figure 1.5. An activity entry in the Data Dictionary.

Statecharts

Statecharts constitute an extensive generalization of state-transition diagrams. They allow for multilevel states decomposed in an and/or fashion, and thus support economical specification of concurrency and encapsulation. They incorporate a broadcast communication mechanism, timeout and delay operators for specifying synchronization and timing information, and a means for specifying transitions that depend on the history of the system's behavior.

EWS-Statecharts

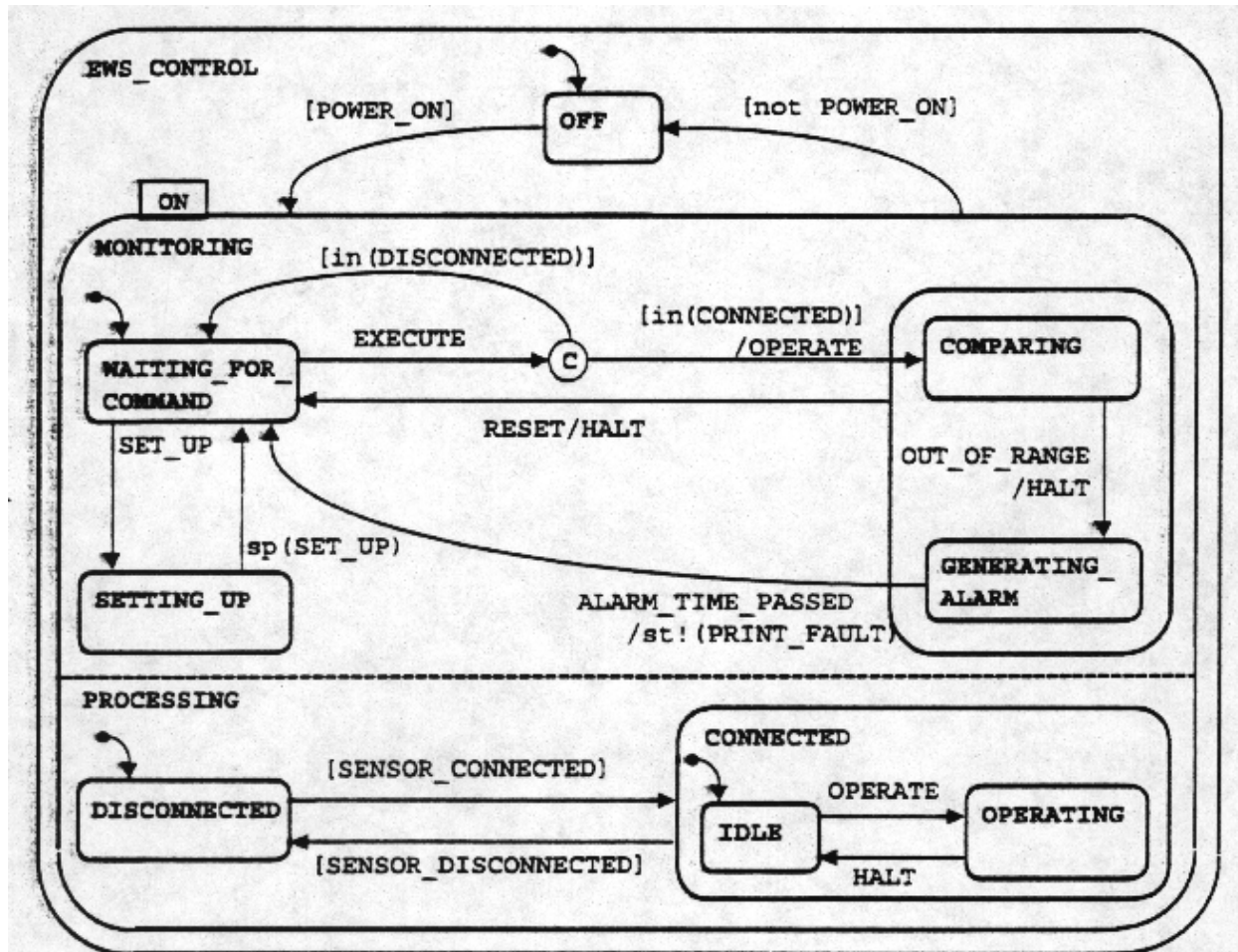
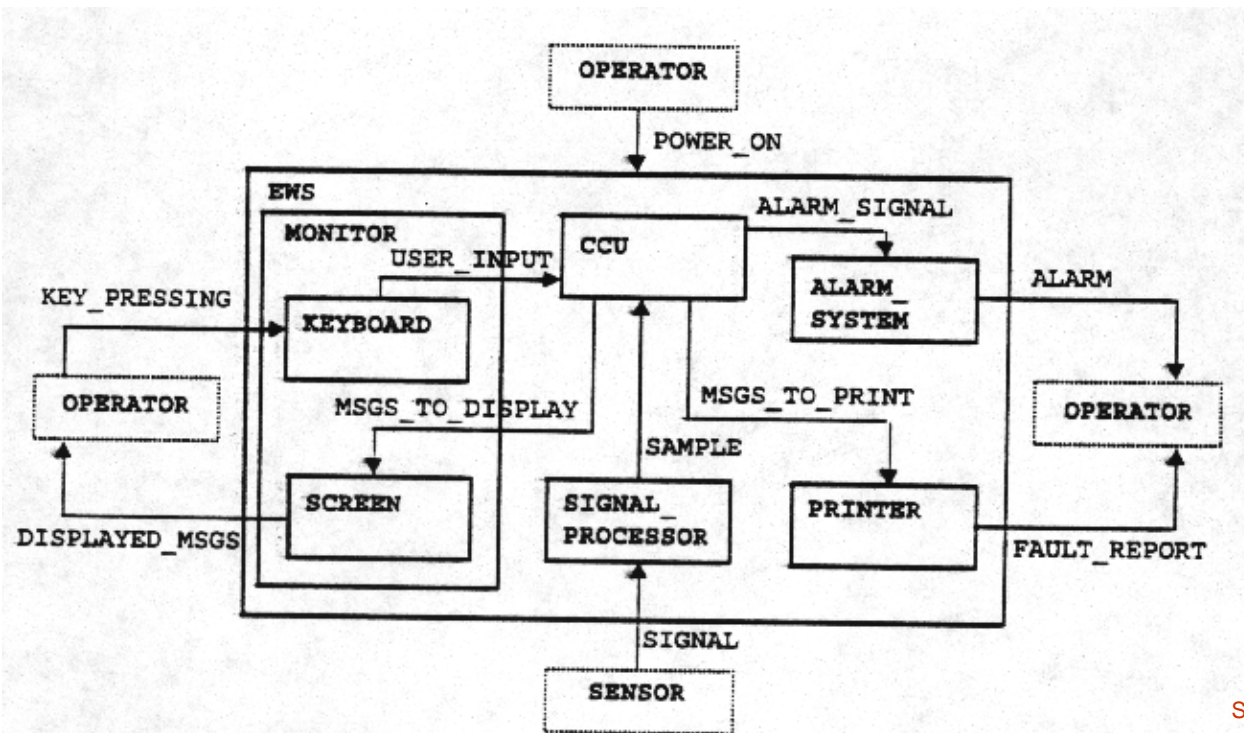


Figure 1.6 A statechart.

Module-charts

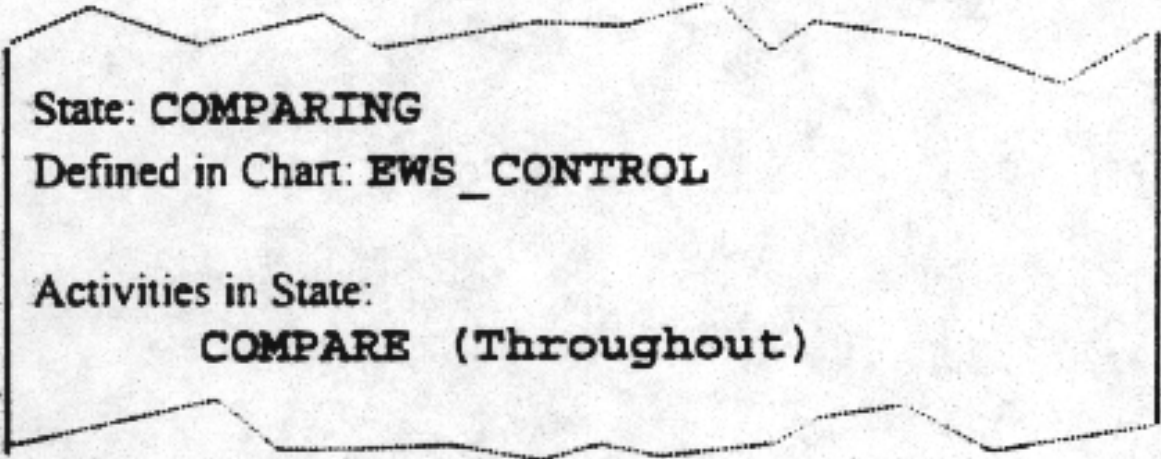
A module-chart can also be regarded as a certain kind of data-flow diagram or block diagram. Module-charts are used to describe the modules that constitute the implementation of the system, its division into hardware and software blocks and their inner components, and the communication between them.



Relationship between the languages

The relationship between the concepts of the three views are reflected in corresponding connections between the three modeling languages.

Most of these connections are provided in the Data Dictionary, and they tie the pieces together, thus yielding a complete model of the system under development.



The diagram shows a rectangular box with a wavy, irregular border. Inside the box, the text is as follows:

State: COMPARING
Defined in Chart: EWS_CONTROL

Activities in State:
COMPARE (Throughout)

Figure 1.8 Specifying an activity throughout a state.

Handling large-scale systems

- The languages allow to split large hierarchical charts into separate ones:

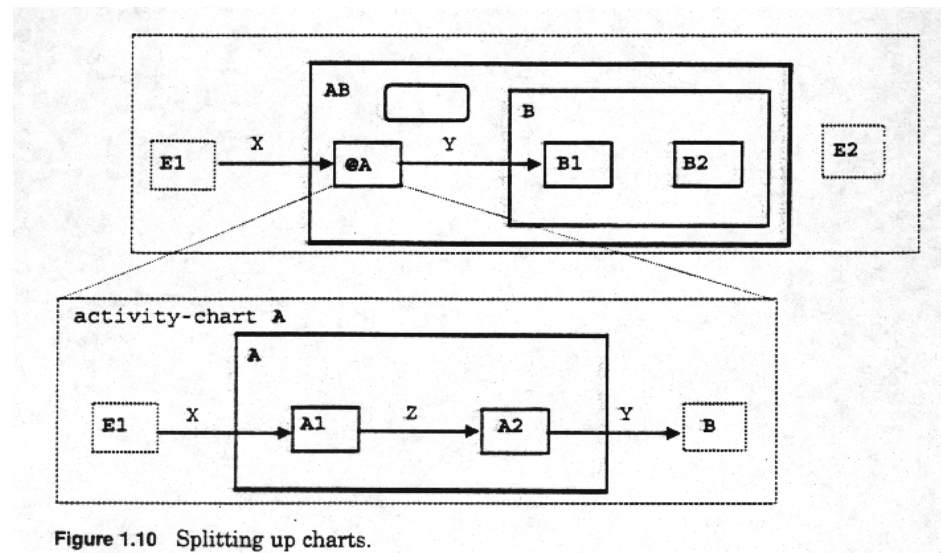


Figure 1.10 Splitting up charts.

- Also, coping with visibility and information hiding by setting scoping rules of elements in the model.
- Moreover, generic charts and user-defined types.

The STATEMATE *toolset*

STATEMATE has been constructed to “understand” the model and its dynamics. The user can then execute the specification by emulating the environment of the system under development and letting the model make dynamic progress in response.

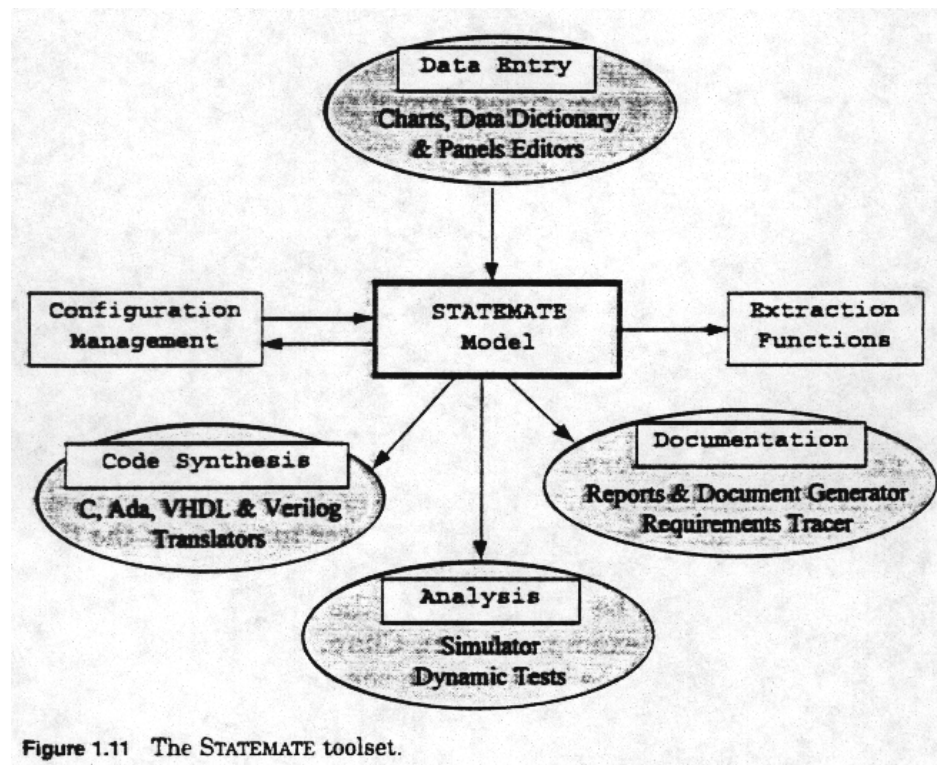


Figure 1.11 The STATEMATE toolset.



Activity-Charts

Describing the functional view of a system

Activity-charts are used to depict the functional view of a system under development (SUD), “what the SUD does”.

System's Specification

- a hierarchy of functional components, called **activities**,
- what kind of information is exchanged between these activities and is manipulated by them,
- how this information flows,
- how information is stored, and
- how activities are **started and terminated**, i.e., **controlled**, if necessary, and whether activities are **continuous**, or whether they **stop by themselves**.

Hierarchical Data Flow Diagrams

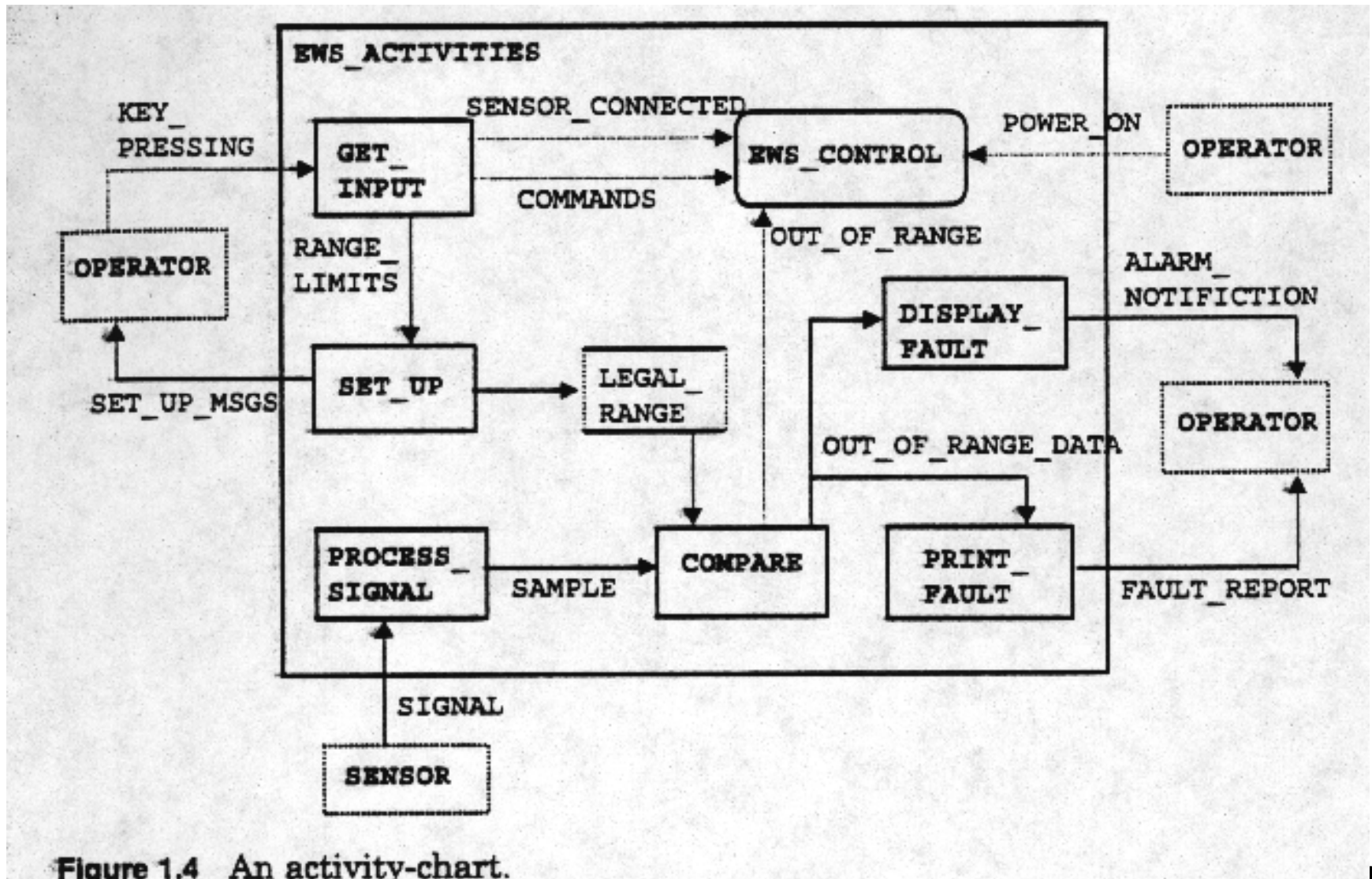
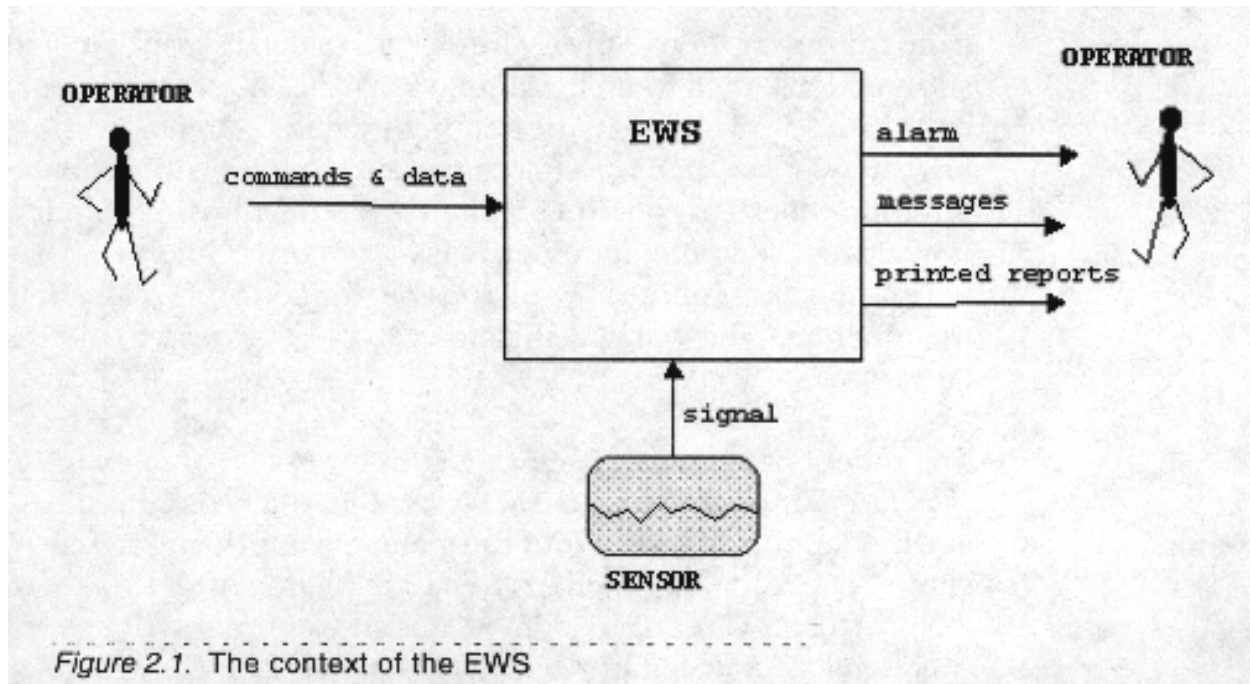


Figure 1.4 An activity-chart.

Functional decomposition of a System

The functional view of a system specifies the system's capabilities.

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:



Structural View

This functional view does not address the physical and implementation aspects of the system; the latter is done in its structural view, i.e., its module-chart:

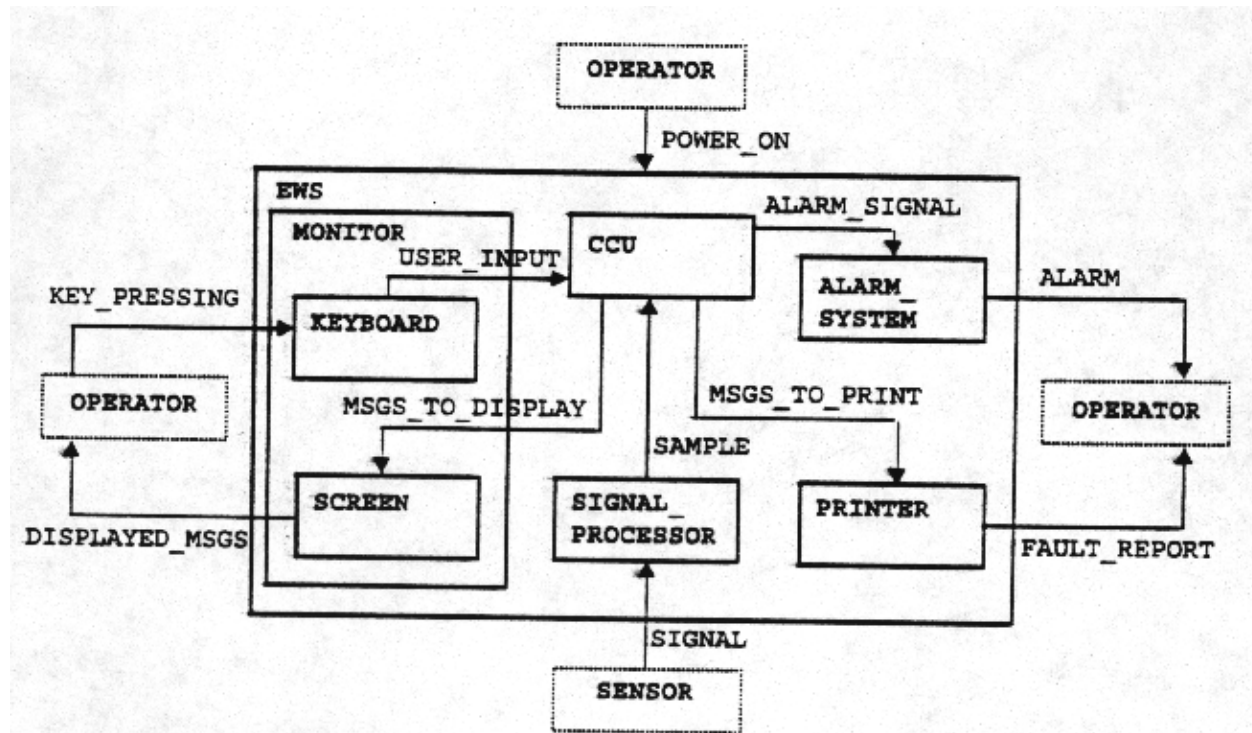


Figure 1.7 A module-chart.

Behavioral Aspects

Moreover it separates the dynamics and behavioral aspects of the SUD from its functional description. The former is done by its behavioral view, in its controlling Statecharts:

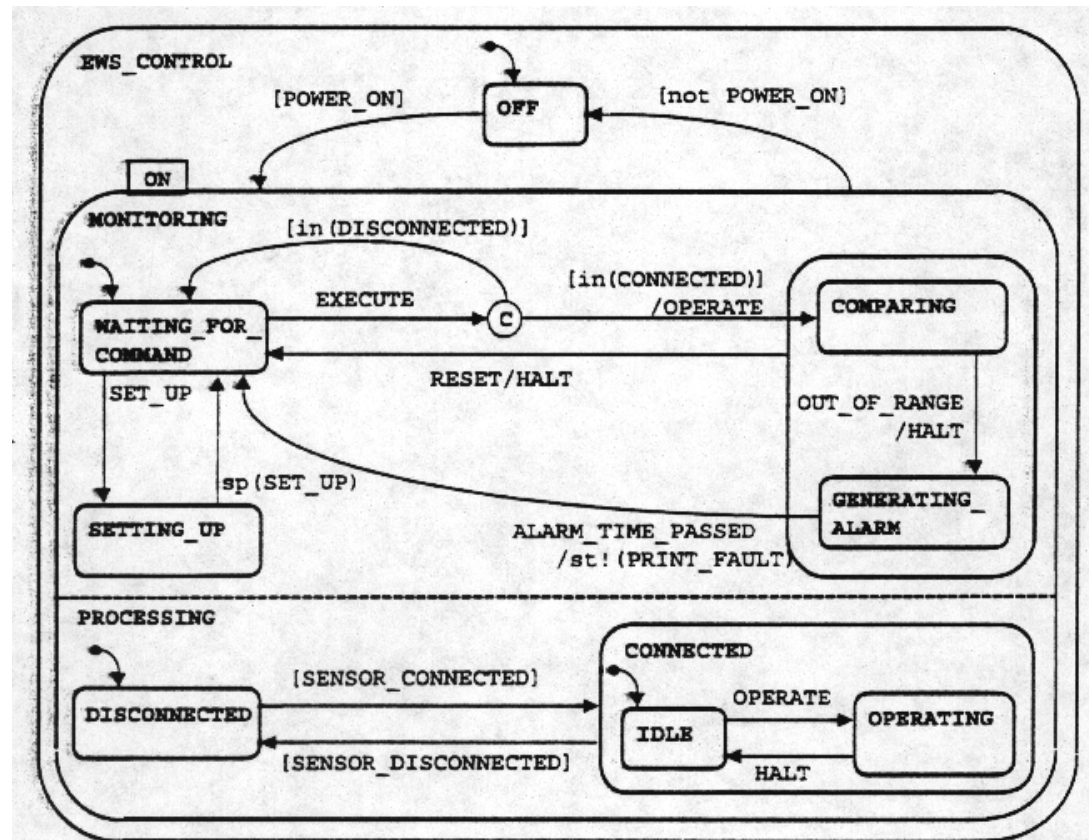


Figure 1.6 A statechart.

Example

The **functional view** tells whether a medical diagnosis system can monitor a patient's functions, and, if so, where it gets its input data and which functions have access to the output data.

The **behavioral view** tells under which conditions monitoring is started, whether it can be carried out parallel to temperature monitoring, and how the flow of control of the process of monitoring develops.

The **structural view** deals with the sensors, processors, monitors, software modules and hardware necessary to implement the monitoring system

The three views

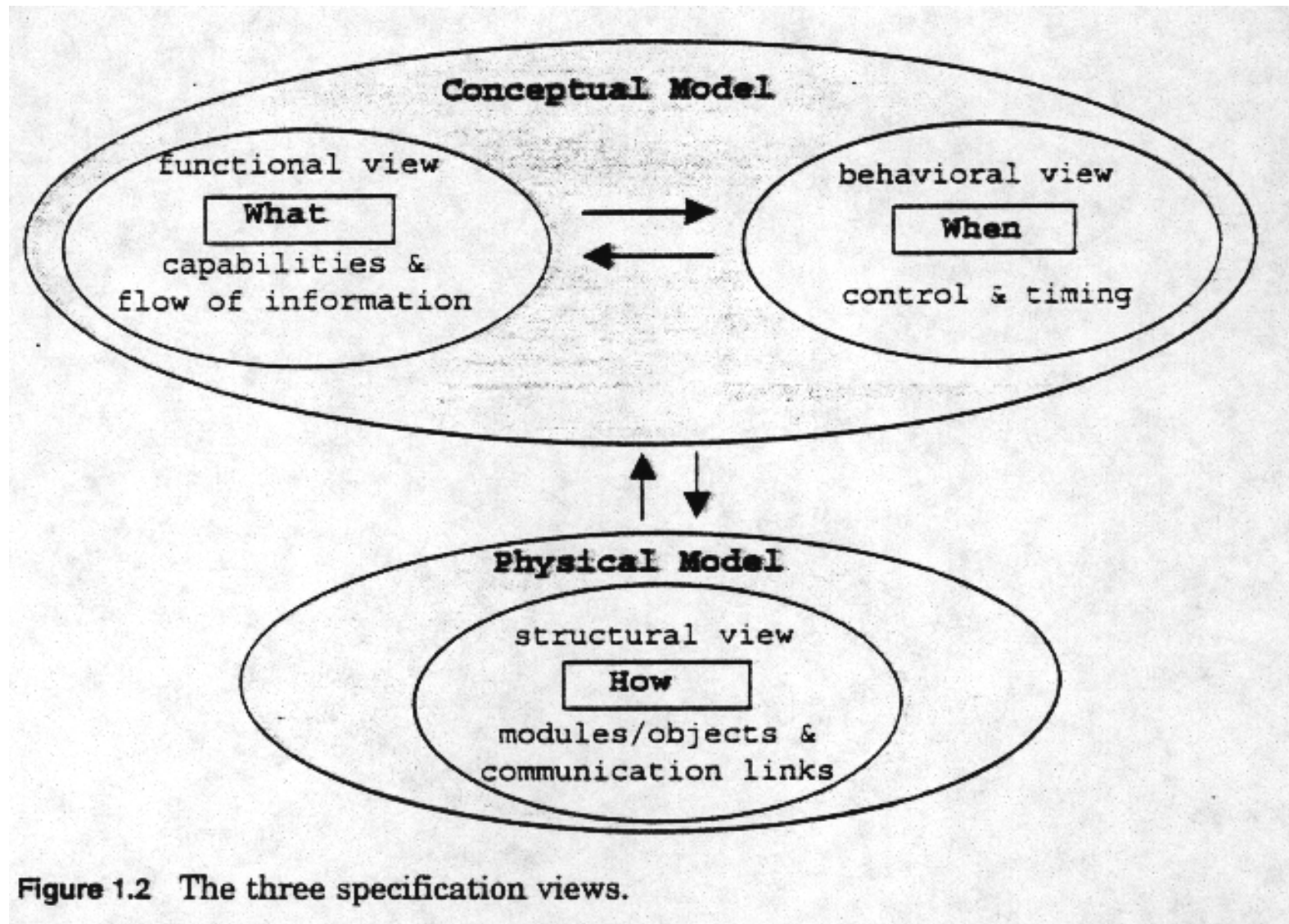


Figure 1.2 The three specification views.

Illustration

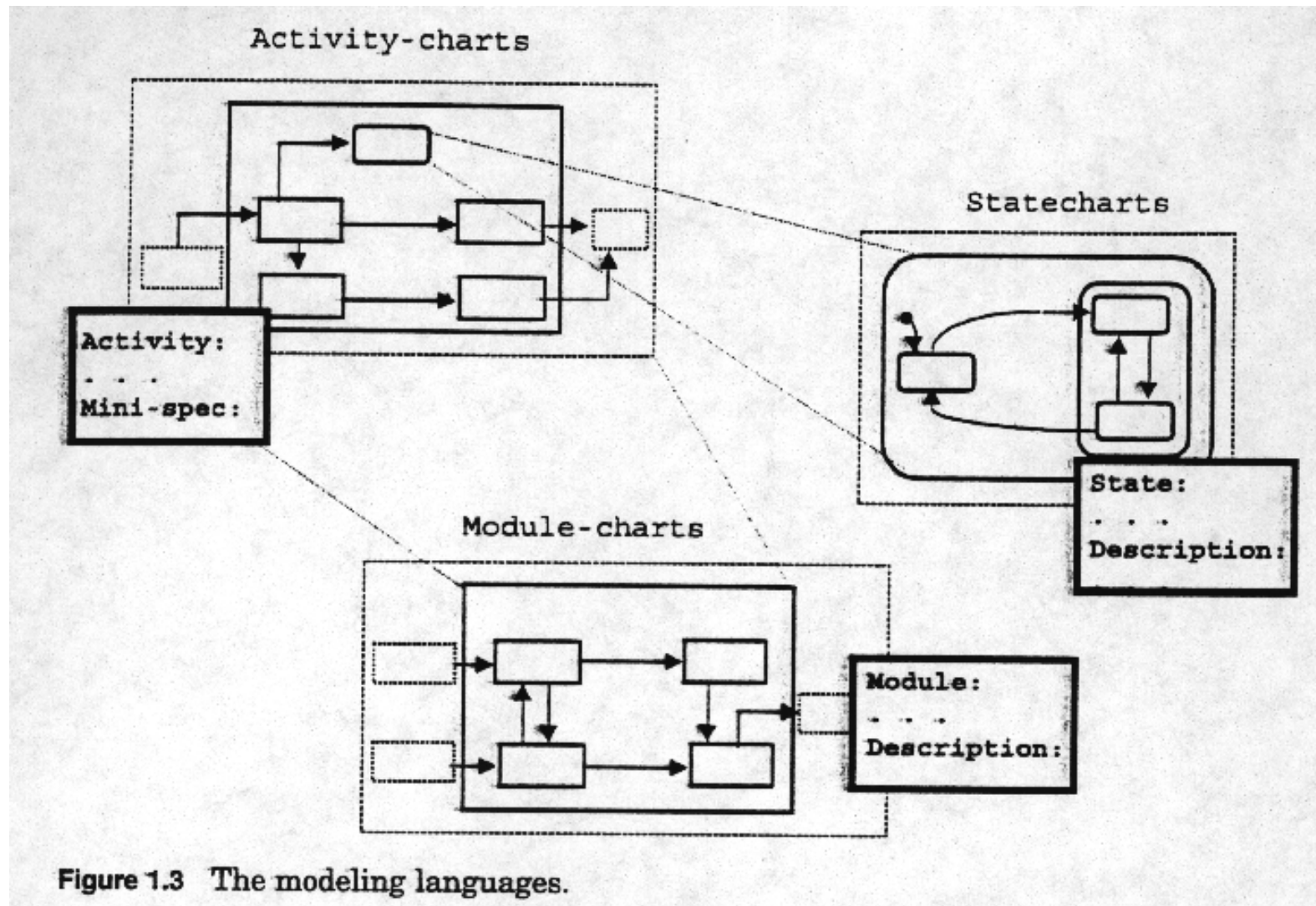


Figure 1.3 The modeling languages.

Functional Decomposition

In the Statechart approach, the functionality of a system is described by **functional decomposition**, by which a system is viewed as a collection of interconnected functional components, called **activities**, organized into a hierarchy.

EWS_ACTIVITIES

E.g., in the activity-chart EWS_ACTIVITIES, the SET_UP components can be decomposed leading to a multi-level decomposition of EWS_ACTIVITIES:

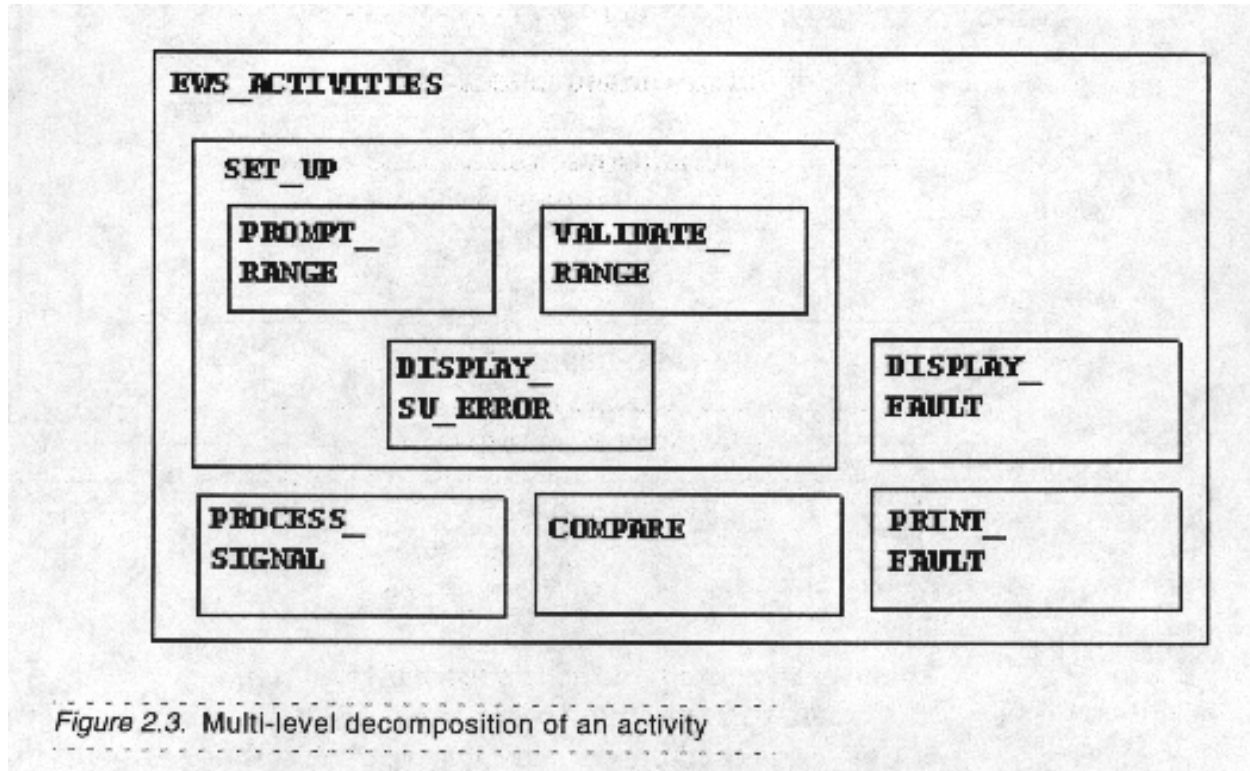


Figure 2.3. Multi-level decomposition of an activity

Functional Decomposition (cont'd)

- Each of the activities may be decomposed into subactivities repeatedly until the system is specified in terms of **basic activities**.
They are specified using textual description (formal or informal), or code in a programming language, inside the Data Dictionary.
- The intended meaning of the functional decomposition is that the **capabilities of the parent activity are distributed between its subactivities**.

Functional Decomposition (cont'd)

- The order in which these subactivities are performed, and the conditions that cause their activation or deactivation are not represented in the functional view and are specified in the behavioral view, i.e., in the (one) statechart associated with the parent activity-chart.
- Activities can represent objects, processes, functions, logical machines, or any other kind of functionally distinct entity.
- In the following sections we'll confine ourselves to function-based decomposition of an activity-chart. We shall not discuss object-based decomposition (see Section 2.1.3 of Harel & Politi)

Function-based Decomposition

In function-based decomposition, the activities are (possibly reactive) functions.

The EWS receives a signal from an external sensor. When the sensor is connected, the EWS processes the signal and checks if the resulting value is within a specified range. If the value of the processed signal is out of range, the system issues a warning message on the operator display and posts an alarm. If the operator does not respond to this warning within a given time interval, the system prints a fault message on a printing facility and stops monitoring the signal. The range limits are set by the operator. The system becomes ready to start monitoring the signal only after the range limits are set. The limits can be re-defined after an out-of-range situation has been detected, or after the operator has deliberately stopped the monitoring.

Function-based Decomposition (cont'd)

Next we decompose this narrative to describe its functionality:

- 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.

Function-based Decomposition (cont'd)

Thirdly, we identify the various functions that are described by there 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.

Function-based Decomposition (cont'd)

Notice that this description also contains info about handled data. An activity may transform its input into output to be consumed by other functions, which are internal or external to the system:

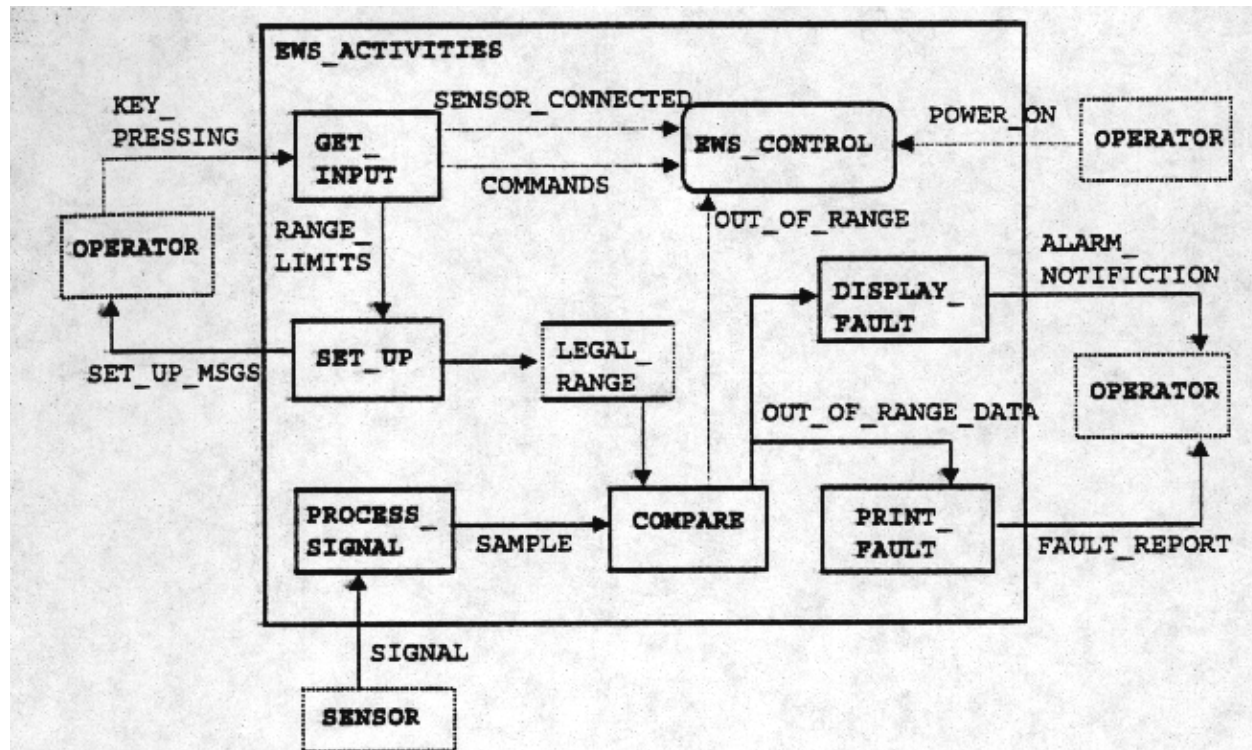
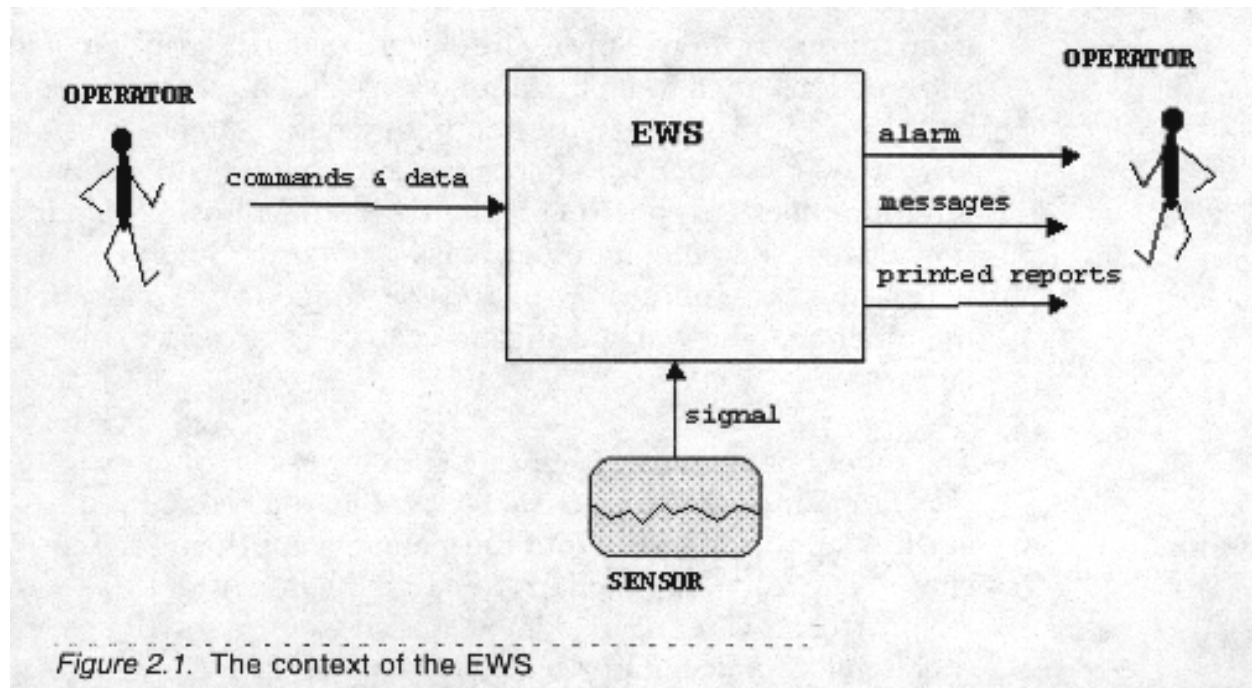


Figure 1.4 An activity-chart.

System Context

One of the first decisions to be made when developing a system involves its **boundaries**, or, **context**. I.e., one must determine which entities are part of the environment of a system, and how they communicate with the system. The latter are called **external activities** of the system.

EWS-Example



Notice that for the EWS one might have chosen for the printer to be external, leading to printer as external activity. Different occurrences of the same entity (here: operator) denote the same entity; these are multiplied of ease of drawing.

Decomposition process

The functional view is specified by **Activity-charts**, together with a **Data Dictionary** that contains additional information about the elements appearing in the charts, e.g., about their basic activities.

Activities and their representation

We continue the functional decomposition of the EWS, started with:

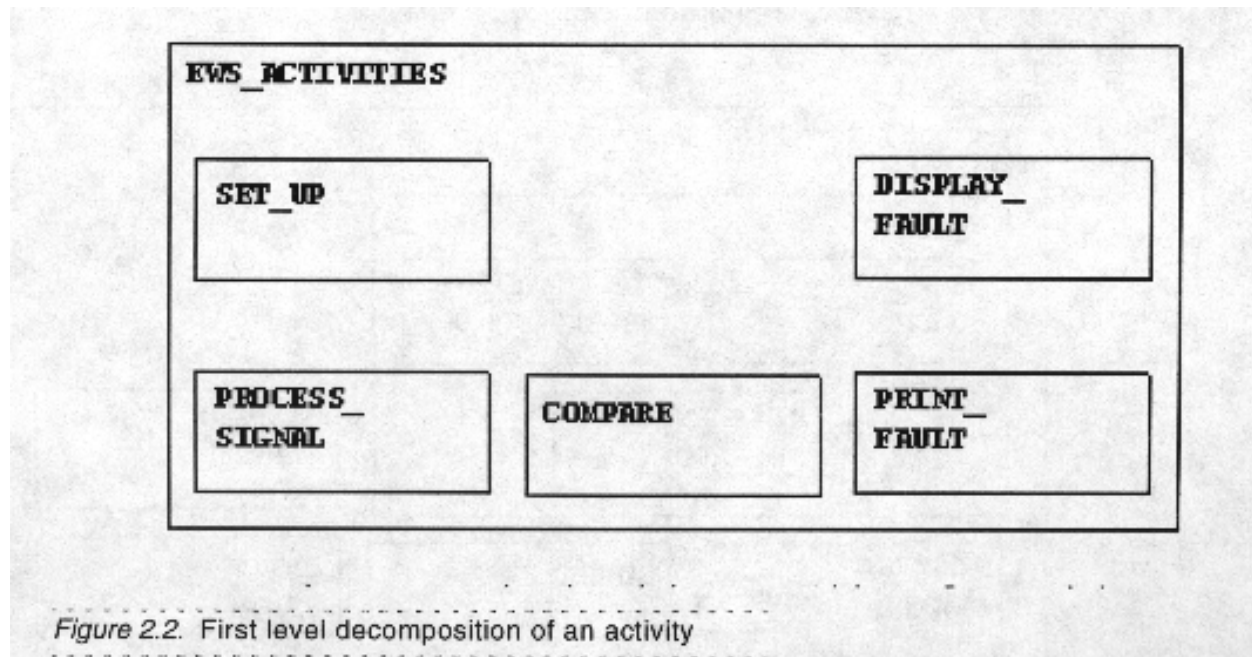
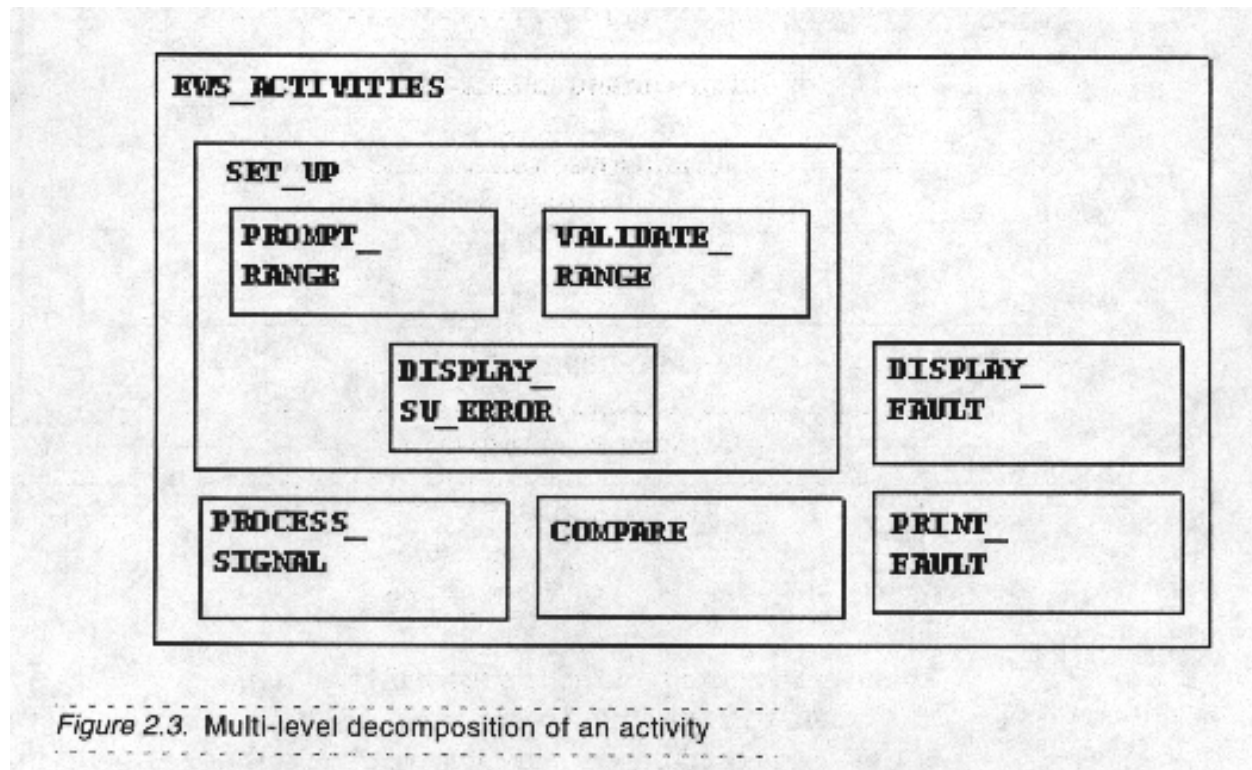


Figure 2.2. First level decomposition of an activity

This activity chart contains one top-level box, representing the top-level activity of the chart.

Top-Down Development

On their turn, the activities appearing above can be decomposed themselves, as SET_UP:



Some terminology

- EWS_ACTIVITIES is called **top-level activity**
- EWS_ACTIVITIES is also called **parent activity** of SET_UP, COMPARE, etc., which are called **descendants** of EWS_ACTIVITIES, as are the subactivities PROMPT_RANGE etc. of SET_UP, who have SET_UP and EWS_ACTIVITIES as **ancestor**.

Each activity has a corresponding item in the Data Dictionary, which may contain additional information.

Flow of Information between Activities

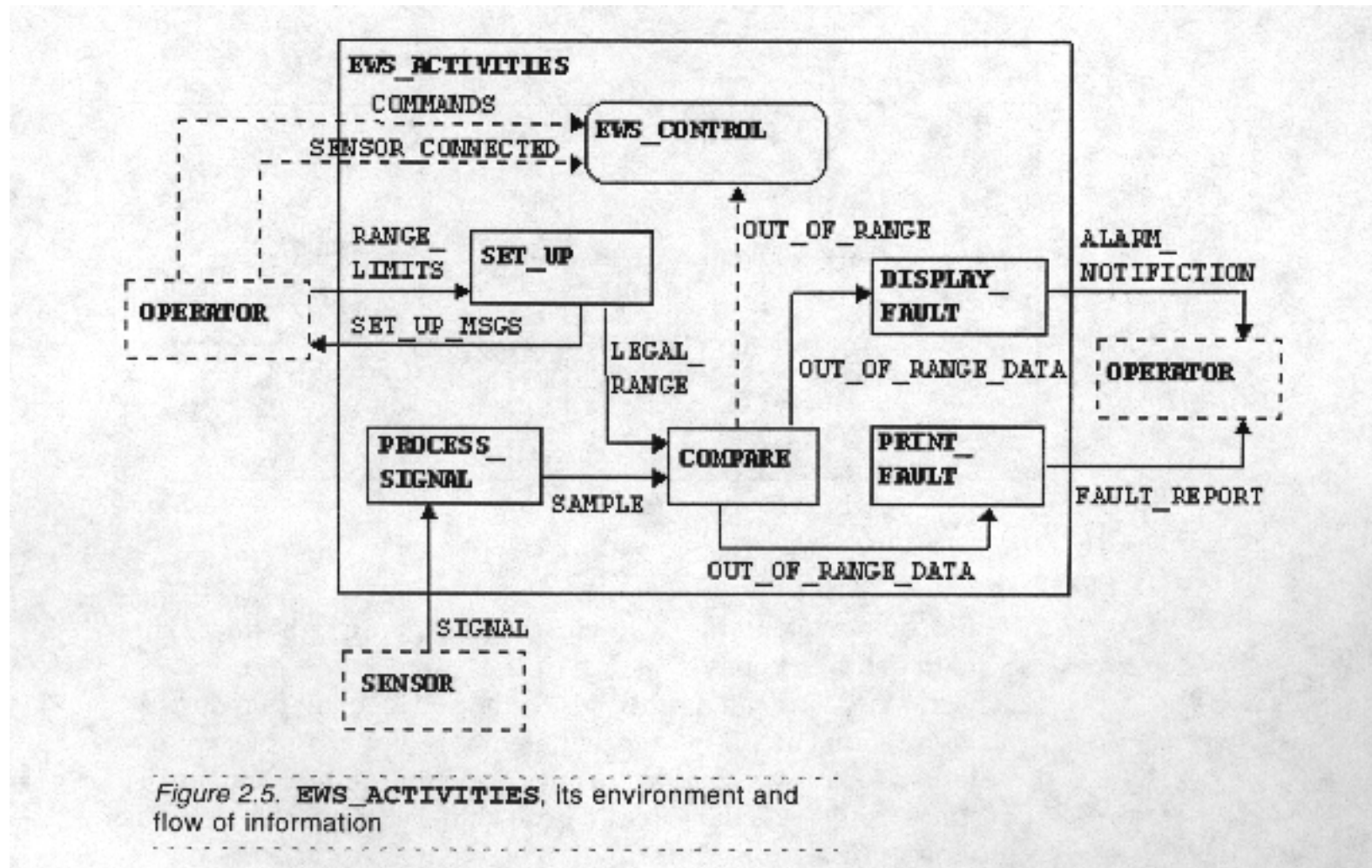


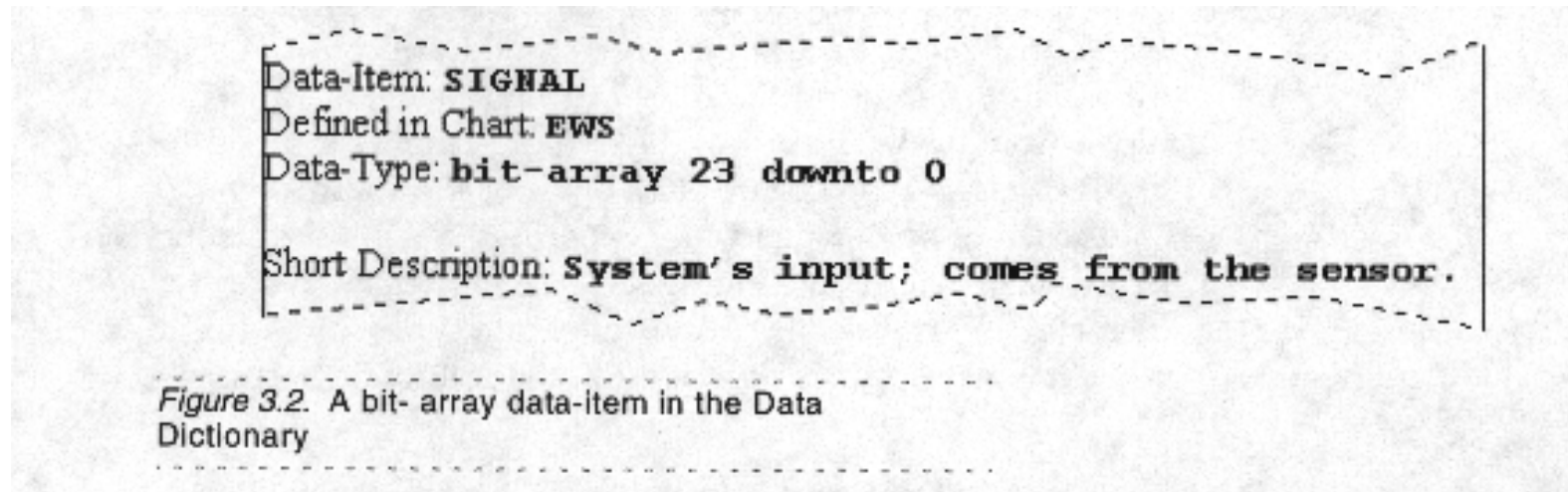
Figure 2.5. **EWS_ACTIVITIES**, its environment and flow of information

Information Flow in EWS

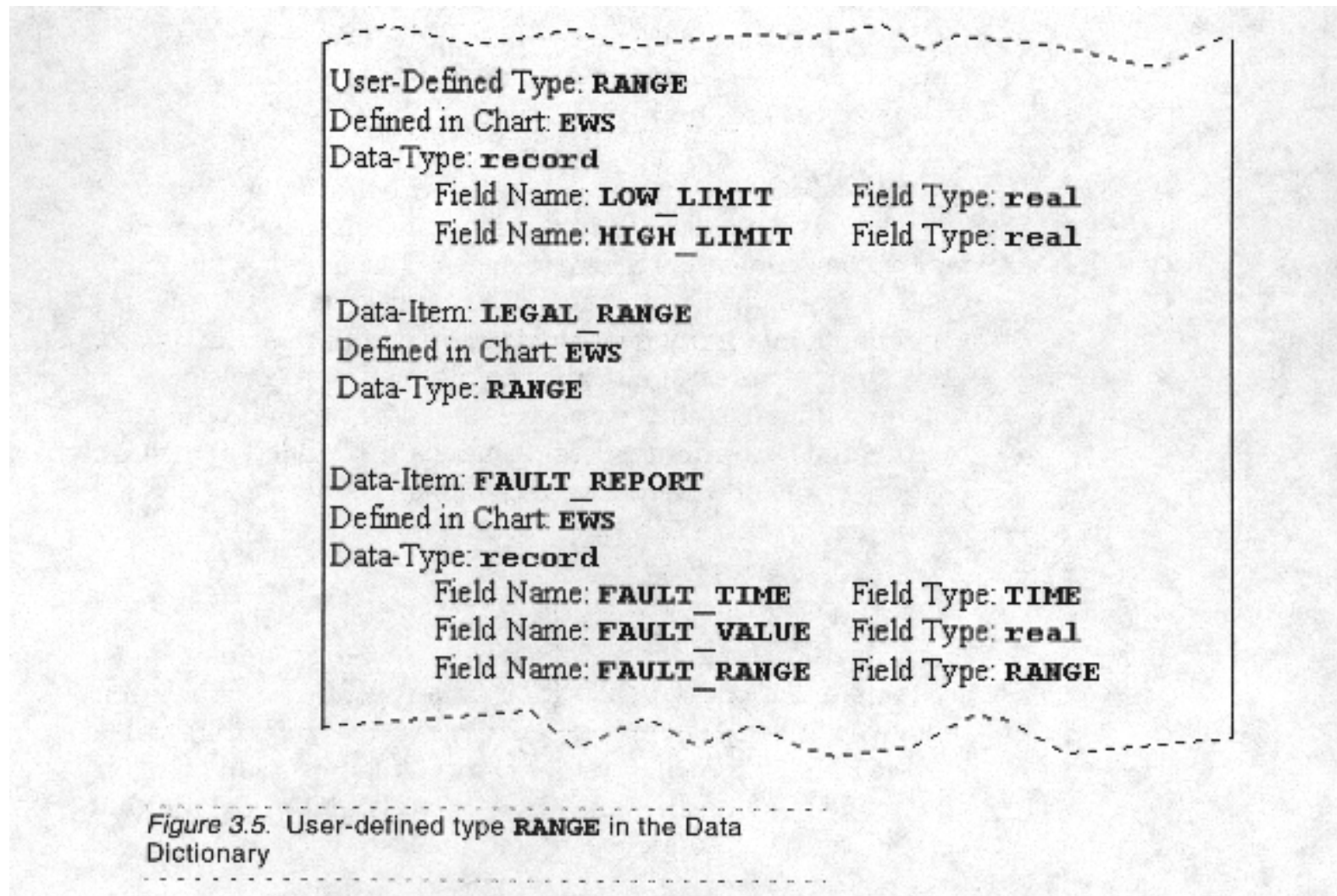
- OPERATOR and SENSOR are external activities, drawn using dotted lines.
- Different occurrences of OPERATOR refer to the same entity.
- Solid arrows denote **data-flow-lines** between activities.
- Control of EWS_ACTIVITIES is handled in its **control activity chart** EWS_CONTROL, a statechart (drawn using rounded corners).
- Dotted arrows denote **control-flow-lines**, carrying info or signals used in making **control decisions**.

Flow lines

A label on a flow line denotes either a single information element that flows along the line, i.e., a **data-item**, **condition**, or **event** or a group of such elements, as in, e.g.:



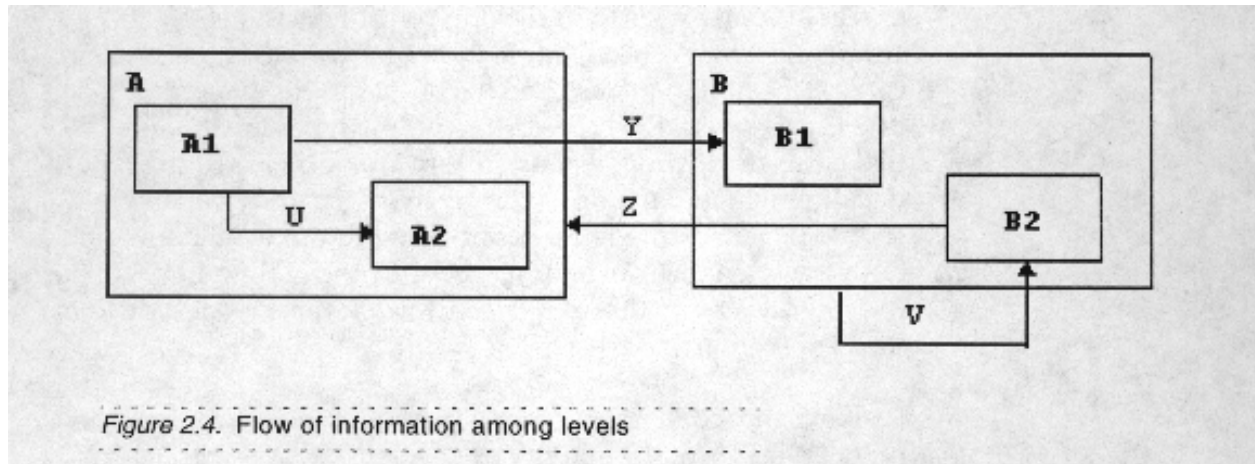
Records



Such groups are called information-flow.

Representation

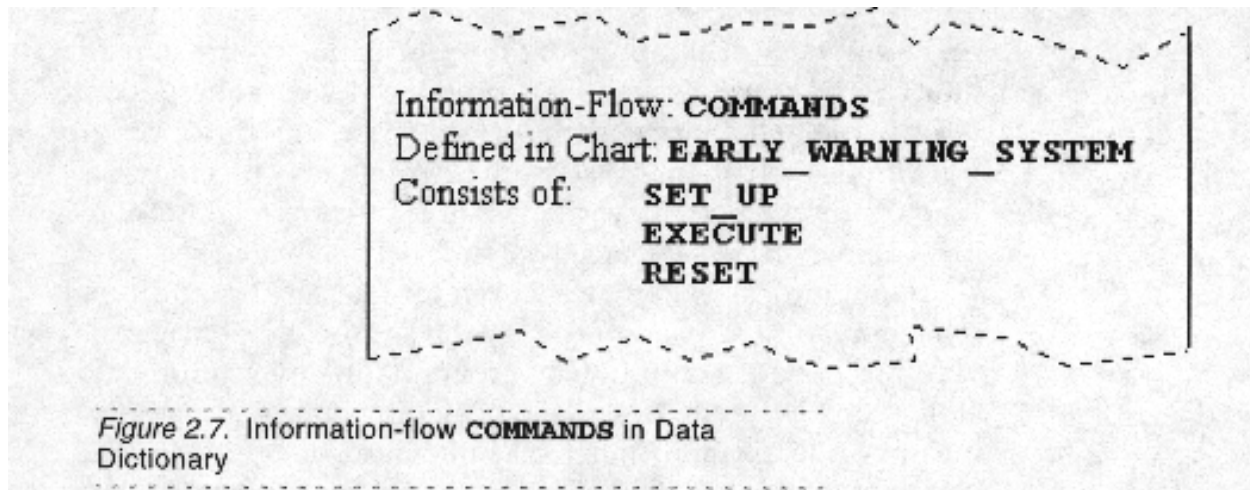
- A flow-line originates from its **source activity**, and leads to its **target activity**:



- An arrow can be connected to a non-basic box, meaning it relates to **all the subboxes** within the box in question, see above the data flow lines labeled *v* and *z*.
- Information flow `SIGNAL` in Figure 2.5 is declared in the Data Dictionary as in Figure 3.2 and is used in data processing.

Information-flow in Data Dictionary

Information flow **COMMANDS** in the Data Dictionary declared as below, is used to denote control issues.



The number of lines in an activity chart can be reduced by grouping information elements into an **information-flow**, used to label a common flow line, e.g. **COMMANDS** consists of **SET_UP**, **EXECUTE**, **RESET**.

Flow Lines

- Flow lines may represent, e.g.,
 - parameter passing to procedures
 - passing of values of global variables
 - messages transferred in distributed systems
 - queues between tasks in real-time applications
 - signals flowing along physical links in hardware systems
- Flows can be continuous or discrete in time.

Flowing Elements

- Three types of information elements flow between activities: **events, conditions, data-items.**
- Their differences are in their **domain of values and timing characteristics:**

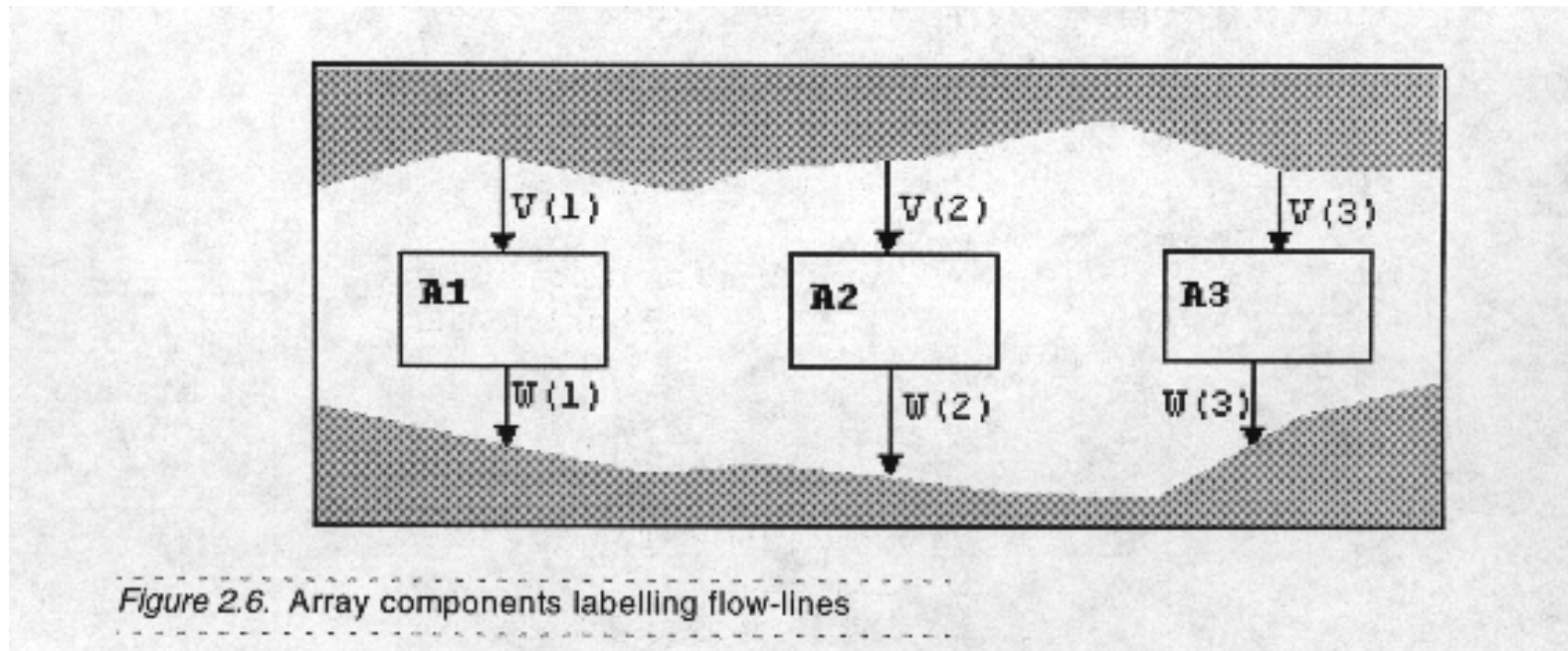
Events are instantaneous signals used for synchronization purposes, e.g., `OUT_OF_RANGE` in Figure 2.5.

Conditions are persistent signals that are either true or false, e.g., `SENSOR_CONNECTED` in Figure 2.5.

Data-item are persistent and may hold values of various types and structures, e.g., `SIGNAL`, a **bit-array**, or `LEGAL_RANGE`, a **record** with two fields of type `real`, `HIGH_LIMIT` and `LOW_LIMIT`.

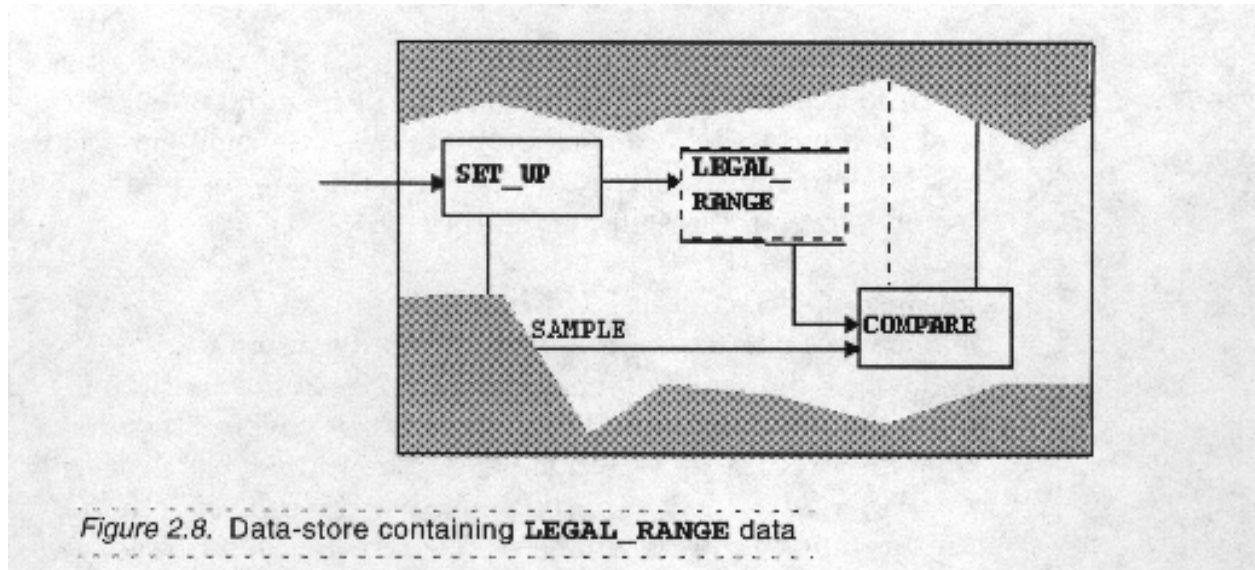
Complex Information Structures

All three types of information elements can be arranged in array and record structures:



Data Stores

- There are no restrictions on the time that data reside on a flow line. Nevertheless it is often more natural to incorporate an explicit data store in the chart:



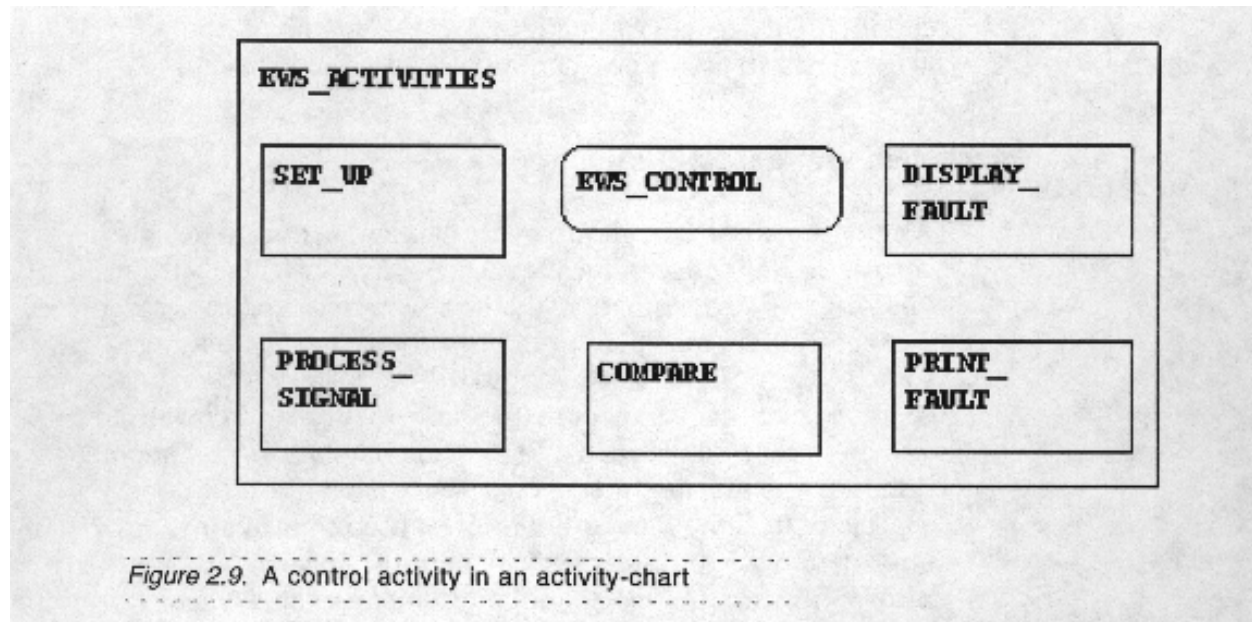
- A data item is defined in the Data Dictionary with the same name as the data store. Any structure given to a data item is inherited by the data store.

The Behavioral Functionality of Activities

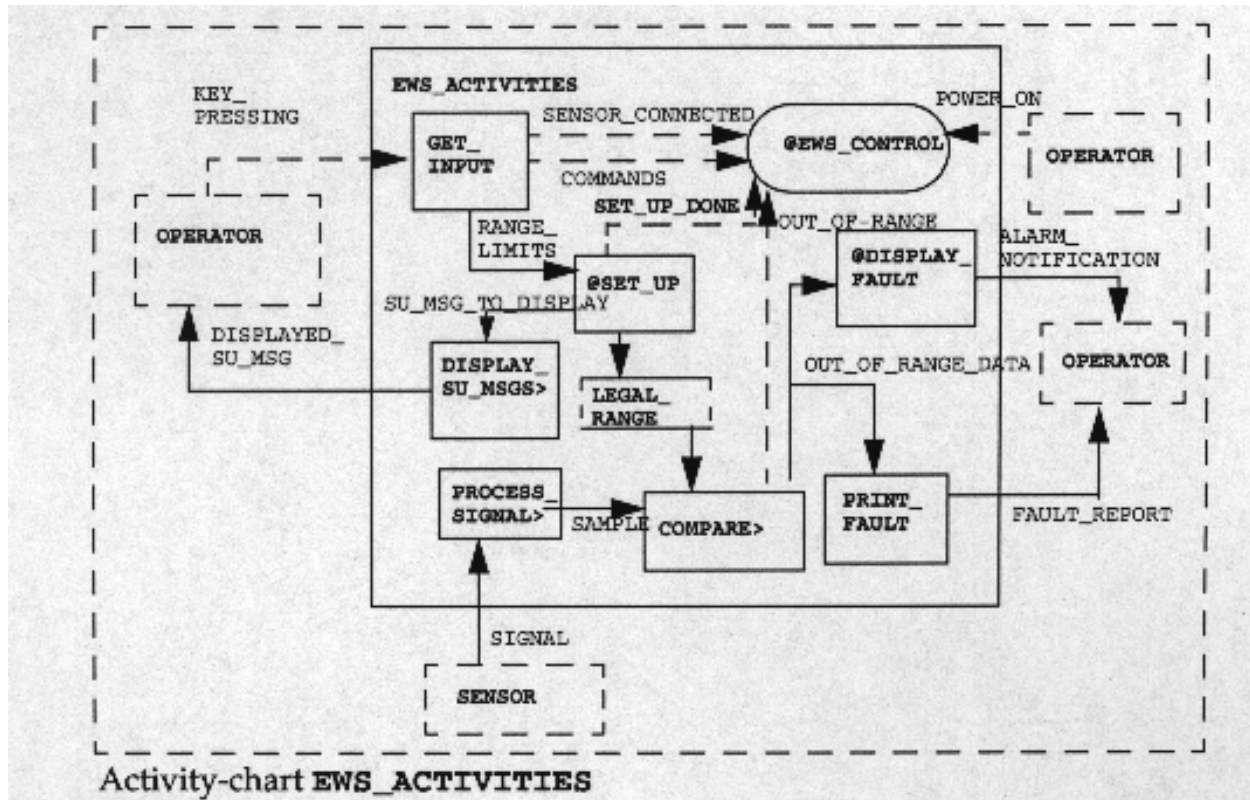
- The behavior of subactivities of an activity chart is described by its **control activity**, whose function is to control their **sibling** activities (i.e., the other subactivities in the chart).
- Each activity may have **at most one** control activity.
- The control activity, depicted as a rectangle with rounded corners, cannot have subactivities.

Control Activity

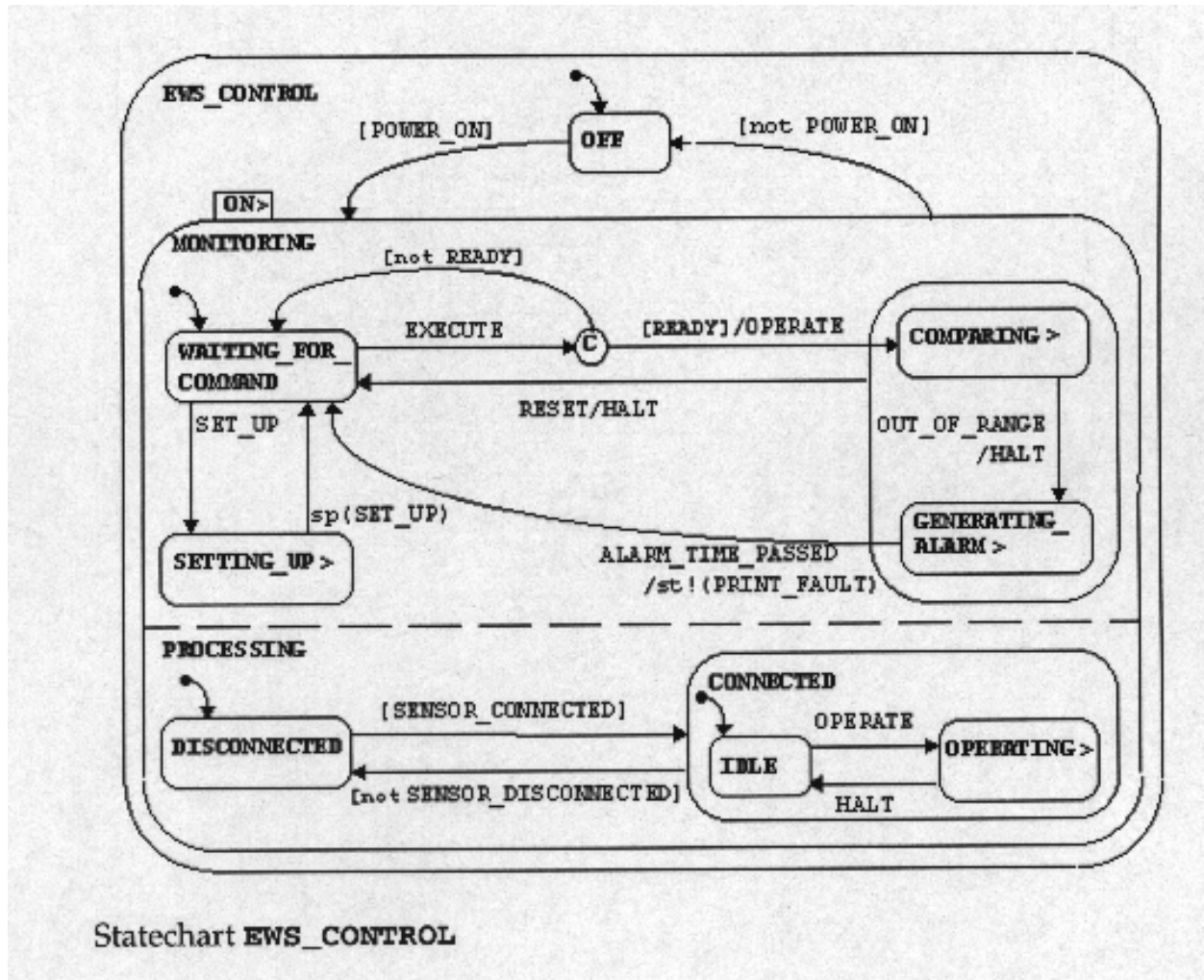
A control activity may explicitly start and stop its sibling activities, i.e., EWS_CONTROL controls SET_UP, PROCESS_SIGNAL, and COMPARE:



EWS-Activities



EWS-Control



Statechart EWS_CONTROL

Activities in the Data Dictionary

- Every activity can be described more extensively in the Data Dictionary using **textual information**.
- **Basic activities** are described in the Data Dictionary by **executable textual descriptions**, specifying patterns of behavior.

EWS-Example

```
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

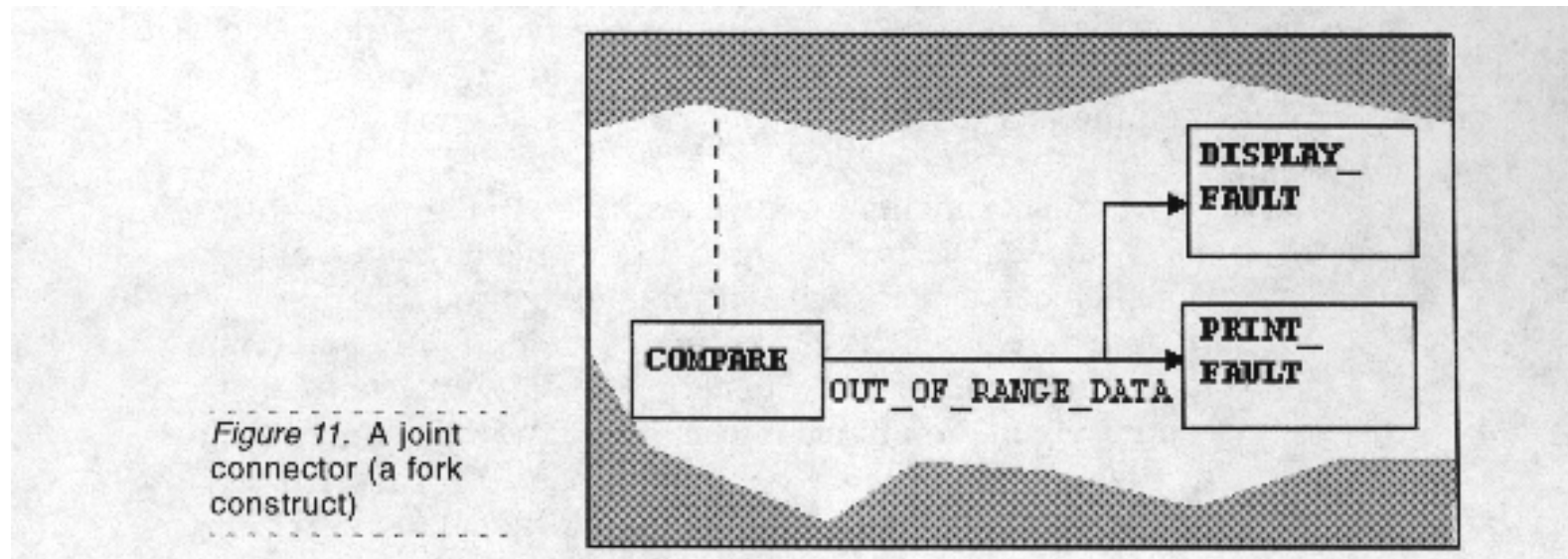
```
Activity: COMPUTE_IN_RANGE  
Defined in Chart: COMPARE  
  
Combinational Assignments:  
IN_RANGE := (SAMPLE > LEGAL_RANGE.LOW_LIMIT)  
and (SAMPLE > LEGAL_RANGE.HIGH_LIMIT)
```

(c) Data-driven activity described by combinational assignments

Figure 2.10. Data Dictionary entries describing activities

Connectors and Compound Flow-Lines

The data flow lines leaving activity COMPARE in Figure 2.5 can be drawn with a joint connector as below:



Junction Connectors

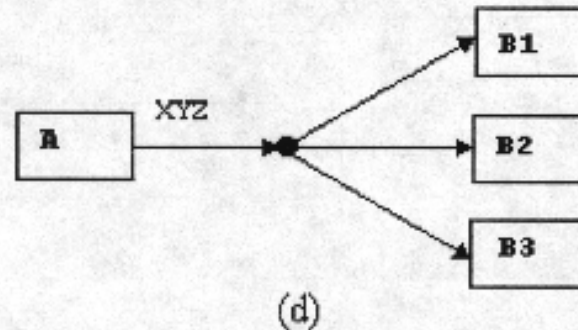
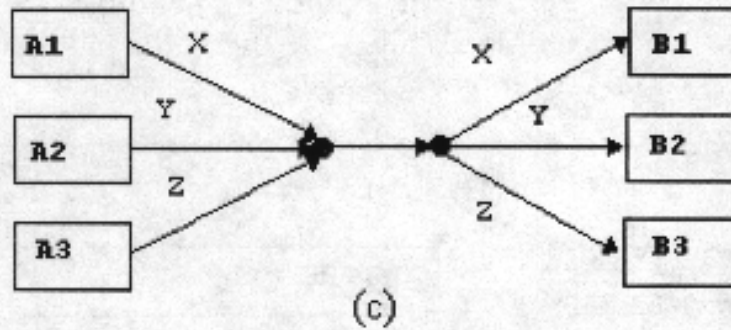
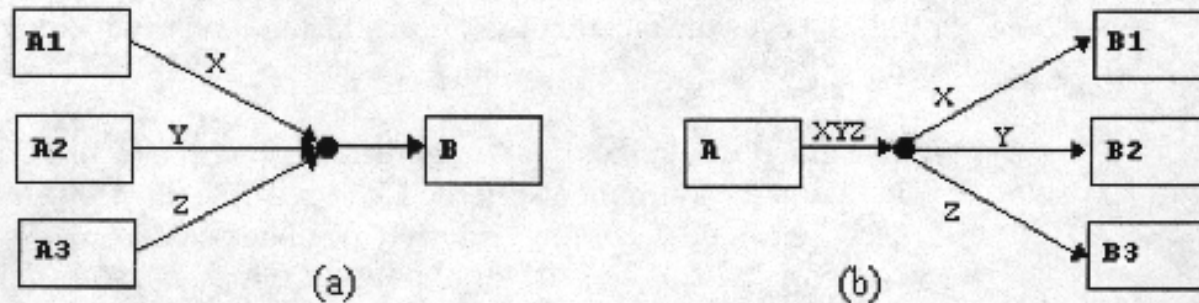


Figure 2.12. Junction connectors

Diagram Connectors

Diagram connectors are used when the source of a flow line is far from its target:

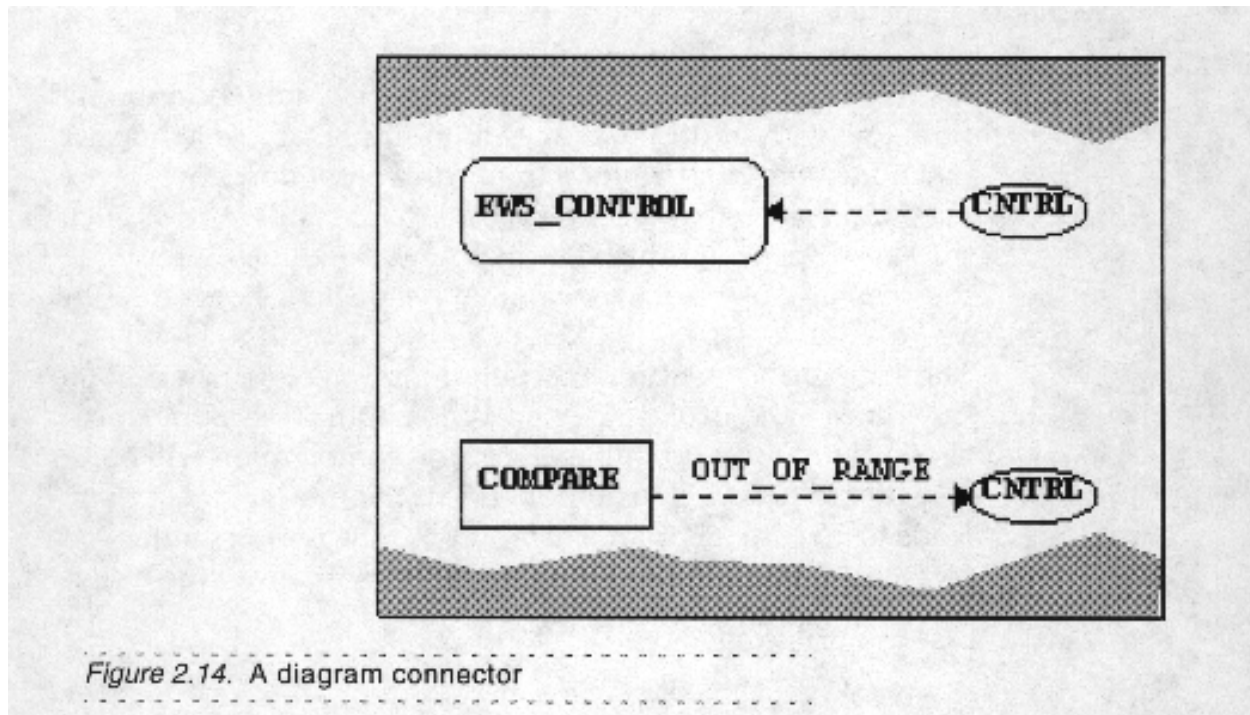


Figure 2.14. A diagram connector