Statemate Course

Kai Baukus

Statemate/SDL

W.-P. de Roever | D. Hogrefe
K. Baukus | H. Neukirchen
CAU Kiel | MU Lübeck

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".

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

Last Session

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 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.

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.

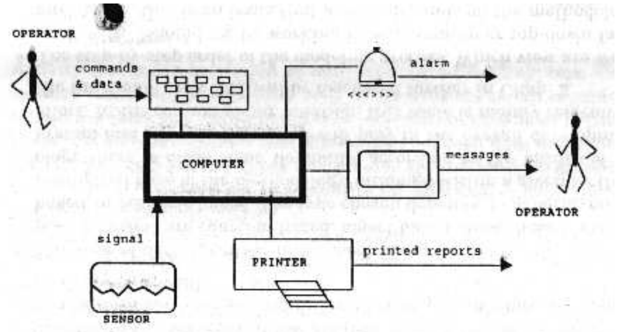


Figure 1.1 The early warning system (EWS).

STATEMATE

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 wher ever 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**

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.

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

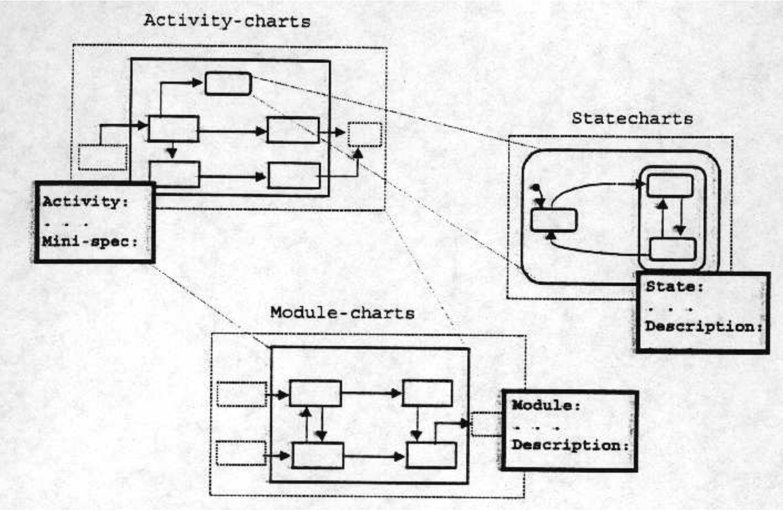


Figure 1.3 The modeling languages.

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.

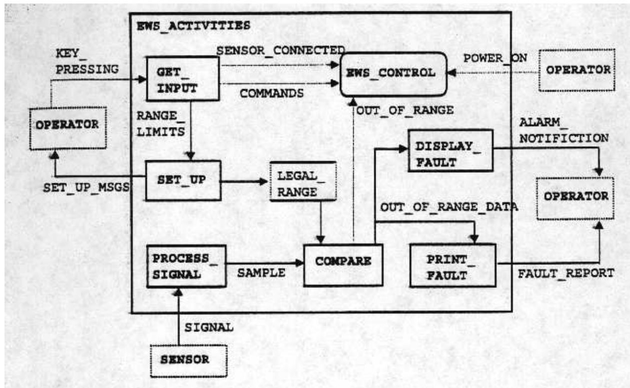


Figure 1.4 An activity-chart.

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.

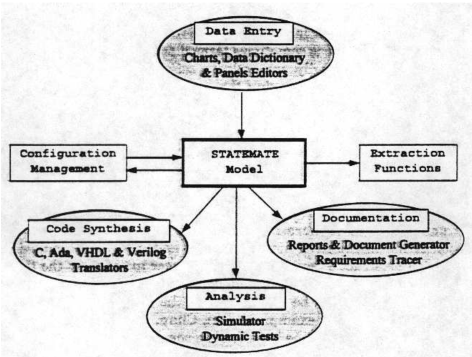


Figure 1.11 The STATEMATE toolset.

Configuration Management

Configuration managment

Basic idea of configuration management (CM) in the software development process is to guarantee the integrity (completeness and intactness) of the software product at any moment of the development

CM (cont'd)

Aims of CM:

- structure and discipline in the development process
- reusability of software

result of CM is

- better software quality and
- increased efficiency of the software development

CM is a management task that includes

- people responsible for it
- CM strategy/methods, plans
- support tools

Aspects of configuration management

- Identification: The imposed structure of the product provides access to parts of the product.
- Control: Product changes are authorized by a formal procedure that distinguishes different releases of the same product and its parts. Consistent releases constitute a baseline.
- Documentation: of status (releases, baselines).
- Verification: guarantees completeness and consistency
- Construction: of the product from its constituents
- process management: support the software life cycle.
- teamwork: several teams/developers of one product

CM informally

programming-in-the-many: $\Rightarrow$ potential for confusion. CM is meant to decrease the confusion. CM must

- identify,
- organize, and
- control

changes by different developers. The aim of CM is to increase quality by avoiding errors.

Architecture of a CM system

- repository: to provide consistency, releases and baselines;
- workareas: parallel development/test and parallel (sic!) changes of the same parts of the software
- Makefiles: construction and dependency checks

Tasks of a CM system

Typical CM activities from the viewpoint of a developer (= user)

- check out current product parts
- build product from checked out parts (if available)
- check in modifications
- compare own version to one in the repository
- update to actual status
- change the structure of the product: add or remove parts.

Activity-Charts

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

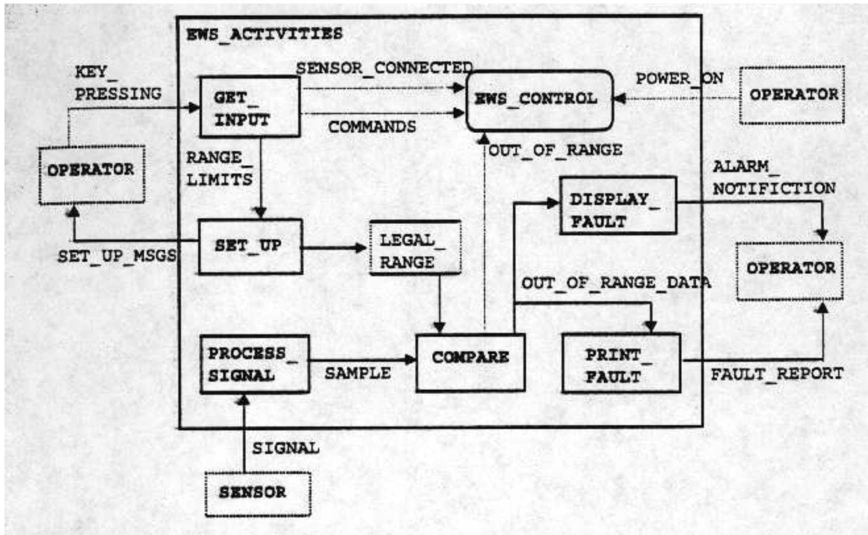


Figure 1.4 An activity-chart.

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:

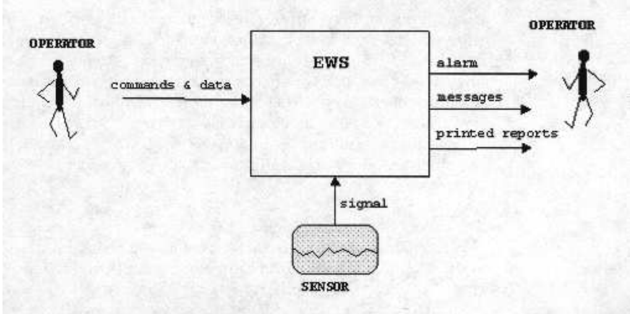


Figure 2.1. The context of the EWS

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**:

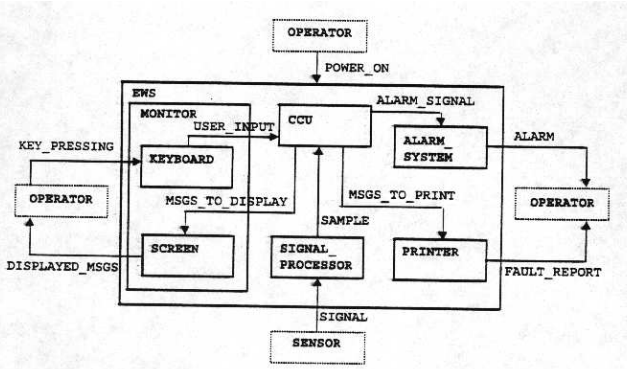


Figure 1.7 A module-chart.

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:

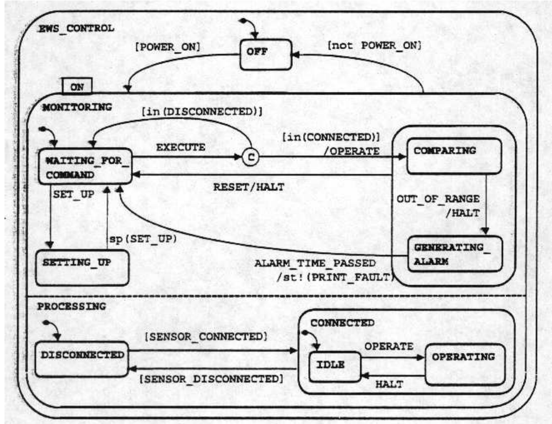


Figure 1.6 A statechart.

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

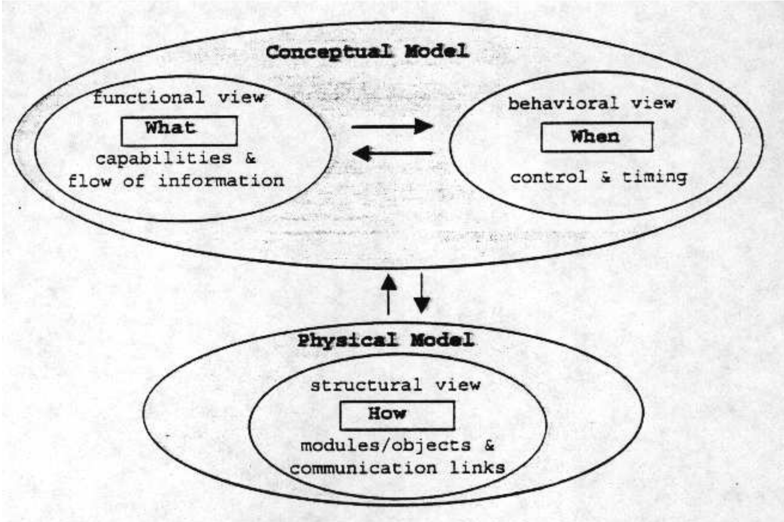


Figure 1.2 The three specification views.

Illustration

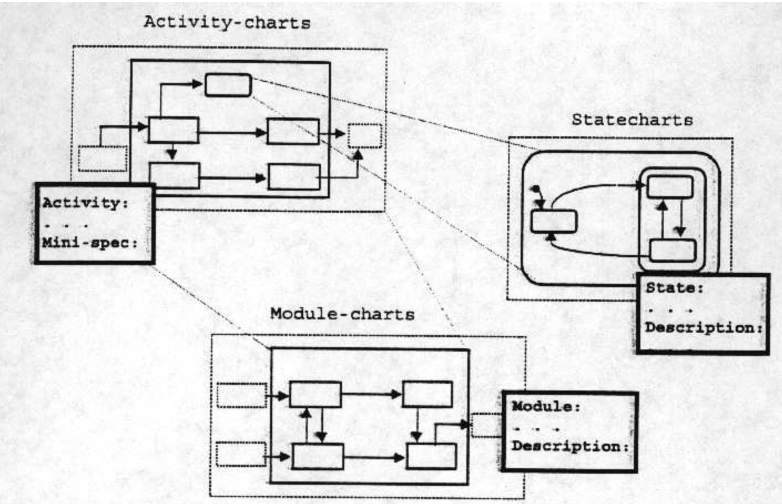


Figure 1.3 The modeling languages.

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:

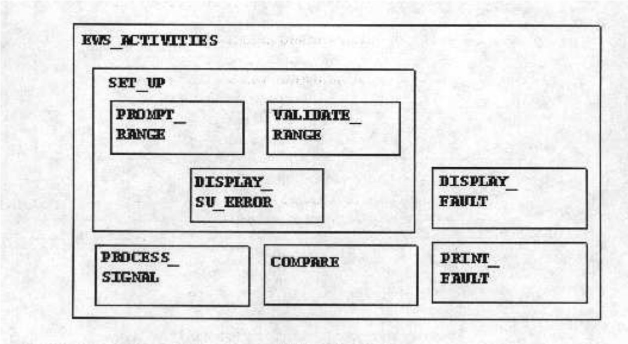


Figure 2.3. Multi-level decomposition of an activity

Functional Decomposition (cont'd)

- Each of the activities may be decomposed into subactivities repeatedly until the system is specified in terms of **basic activities.**
  There 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:

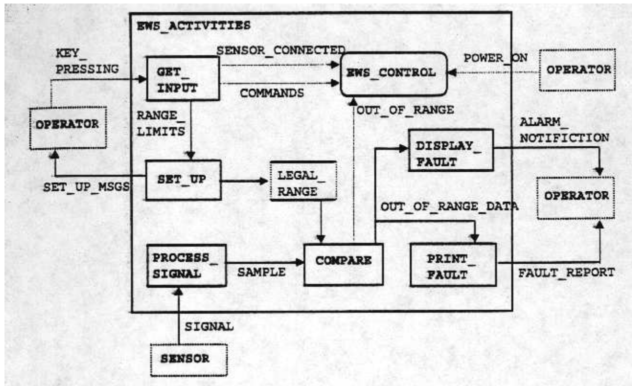


Figure 1.4 An activity-chart.

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

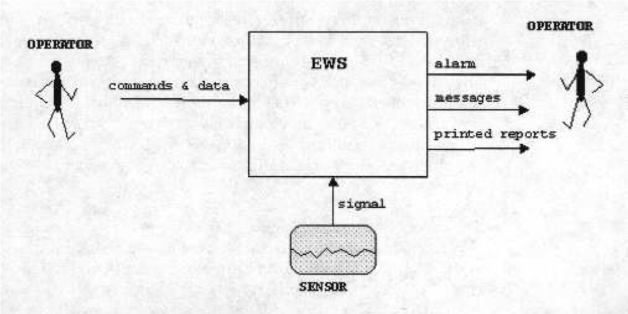


Figure 2.1. The context of the EWS

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 multiplicated 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:

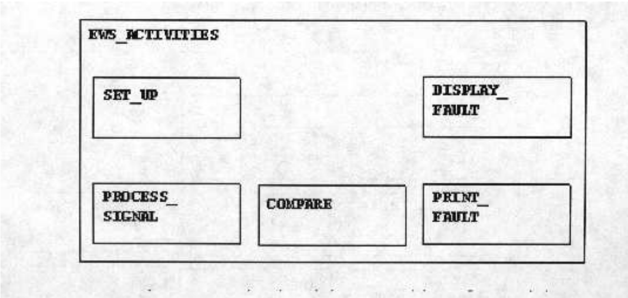


Figure 2.2. First level decomposition of an activity

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:

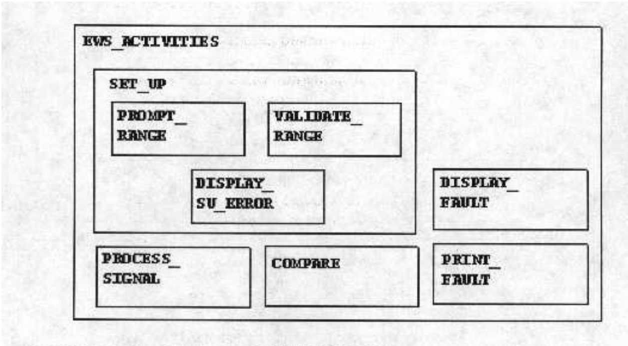


Figure 2.3. Multi-level decomposition of an activity

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

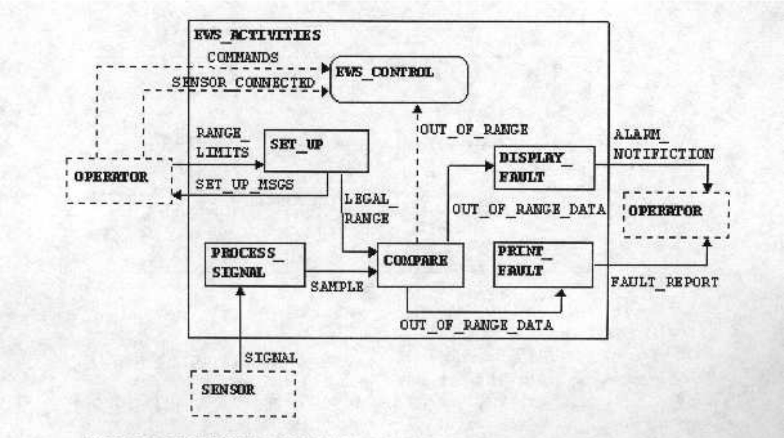


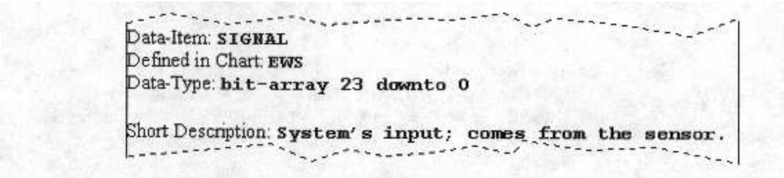
Figure 2.5. **EWS_ACTIVITIES**, its environment and flow of information

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.:


Data-Item: `SIGNAL`
Defined in Chart: `EWS`
Data-Type: `bit-array 23 downto 0`

Short Description: `System's input; comes from the sensor.`

Figure 3.2. A bit- array data-item in the Data Dictionary

Records

```
User-Defined Type: RANGE
Defined in Chart: EWS
Data-Type: record
        Field Name: LOW_LIMIT      Field Type: real
        Field Name: HIGH_LIMIT     Field Type: real

Data-Item: LEGAL_RANGE
Defined in Chart: EWS
Data-Type: RANGE

Data-Item: FAULT_REPORT
Defined in Chart: EWS
Data-Type: record
        Field Name: FAULT_TIME     Field Type: TIME
        Field Name: FAULT_VALUE    Field Type: real
        Field Name: FAULT_RANGE    Field Type: RANGE
```

Figure 3.5. User-defined type **RANGE** in the Data Dictionary

Such groups are called **information-flow**.

Representation

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

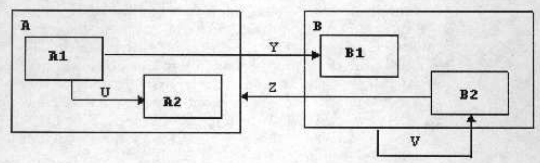


Figure 2.4. Flow of information among levels

- 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**.

```
Information-Flow: COMMANDS
Defined in Chart: EARLY_WARNING_SYSTEM
Consists of:      SET_UP
                  EXECUTE
                  RESET
```

Figure 2.7. Information-flow **COMMANDS** in Data Dictionary

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:

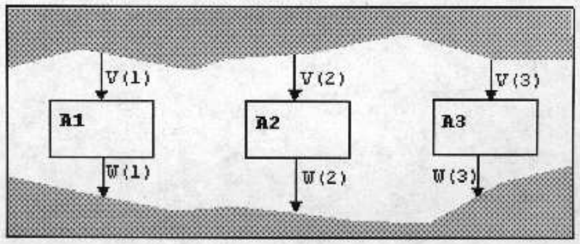


Figure 2.6. Array components labelling flow-lines

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:

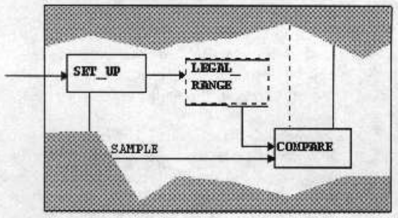


Figure 2.8. Data-store containing **LEGAL_RANGE** data

- 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:

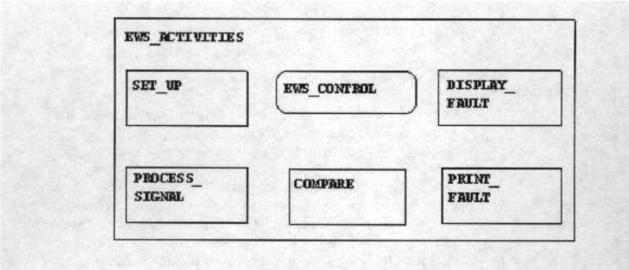
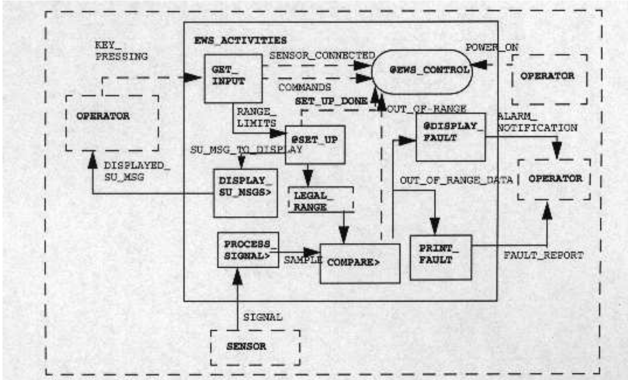


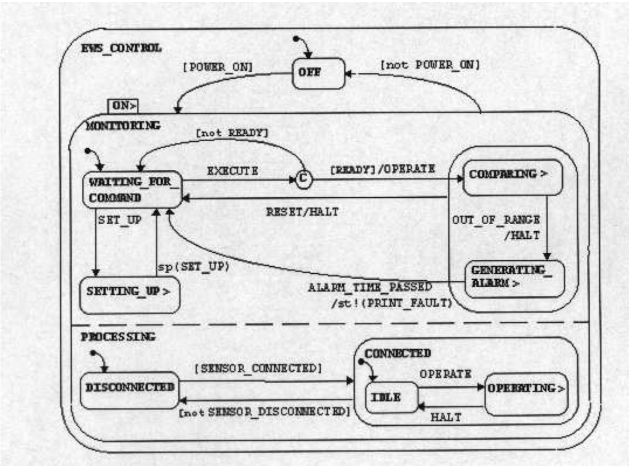
Figure 2.9. A control activity in an activity-chart

EWS-Activities



Activity-chart 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

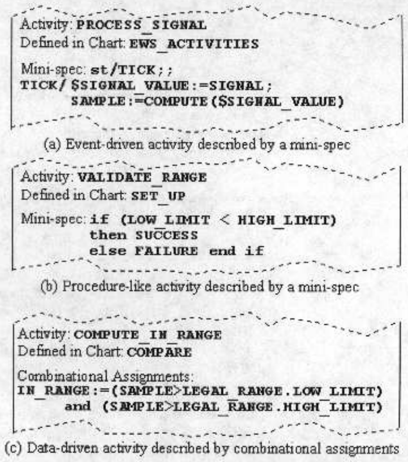


Figure 2.10. Data Dictionary entries describing activities

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:

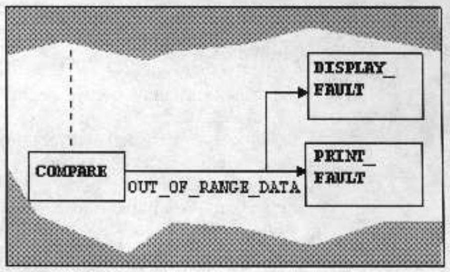


Figure 11. A joint connector (a fork construct)

Junction Connectors

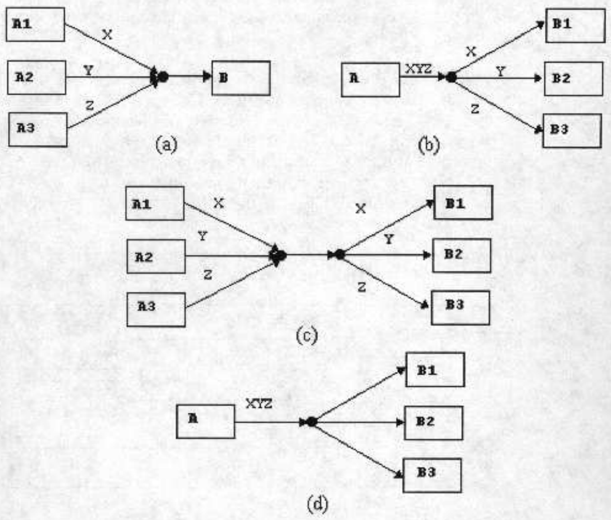


Figure 2.12. Junction connectors

Diagram Connectors

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

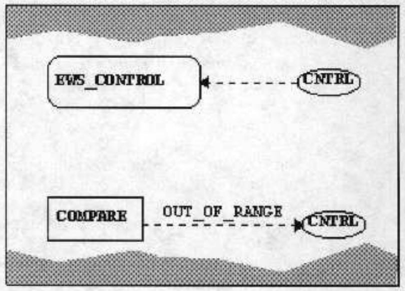


Figure 2.14. A diagram connector