Statemate Course

Kai Baukus

Statemate/SDL

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

Communication between Activities and Module-C

We discuss the mechanismens used to communicate bewteen actions.
As last language in STATEMATE we introduce Module-Charts. Module-charts describe the structural view – sometimes called the architectural view – of the system under development.
Module-charts are typically used in the high-level design stage of the project.

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

Communication between Activities

Specifying the communication between activities consists of the what and the when, just like for other parts of the specification.
The what is described by the flow-lines in the activity-charts and relevant parts of the Data Dictionary. The when is to be specified by the behavioral parts of the model, i.e., the statecharts and mini-specs.

Communication and Synchronization Issues

Functional components in systems communicate between themselves in order to pass along information and to help synchronize their processing. A number of attributes characterize the various communication mechanisms.

Attributes

Communication can be

- **instantaneous**, meaning that it is lost when not consumed immediately, or

  persistent, meaning that it stays around until it gets consumed.

- **synchronous**, i.e., the sender waits for an acknowledgment, or

  asynchronous, i.e., there is no waiting on the part of the sender

- **directly addressed**, i.e., the target is specified, or sent by

  broadcasting

Controlling the Flow of Information

In the following figure X is specified to flow between activities A and B:

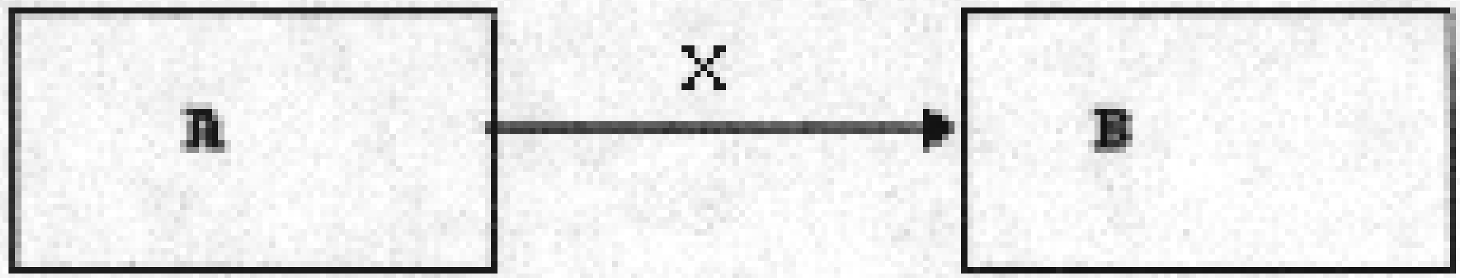


Figure 8.1. An information element flowing between activities

Flow of Information

If X is an event we may have the following situation:

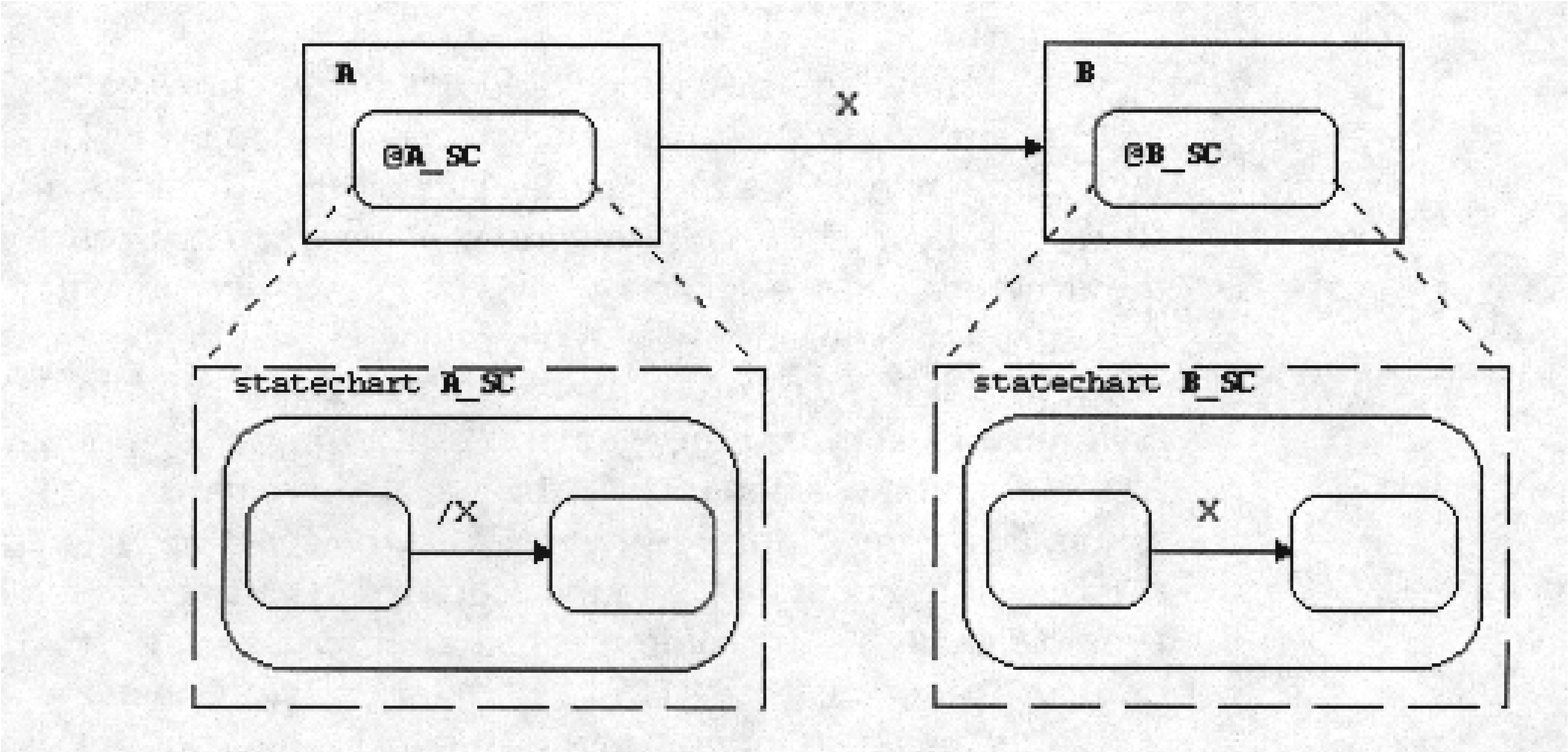


Figure 8.2. Producing and consuming an event

If X is a condition or data-item modified by A, B could sense the value or the change of the value (X, TR(X), WR(X)).

Examples of Communication Control

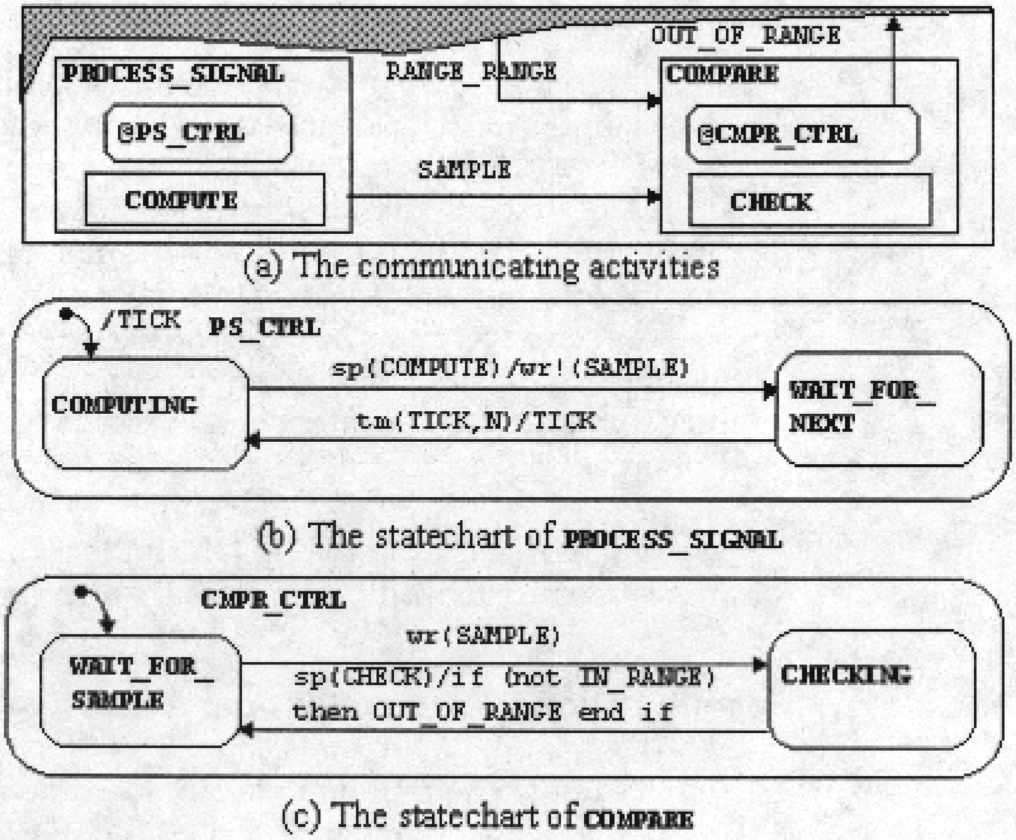


Figure 8.4. Communication between periodic activities

Message Passing

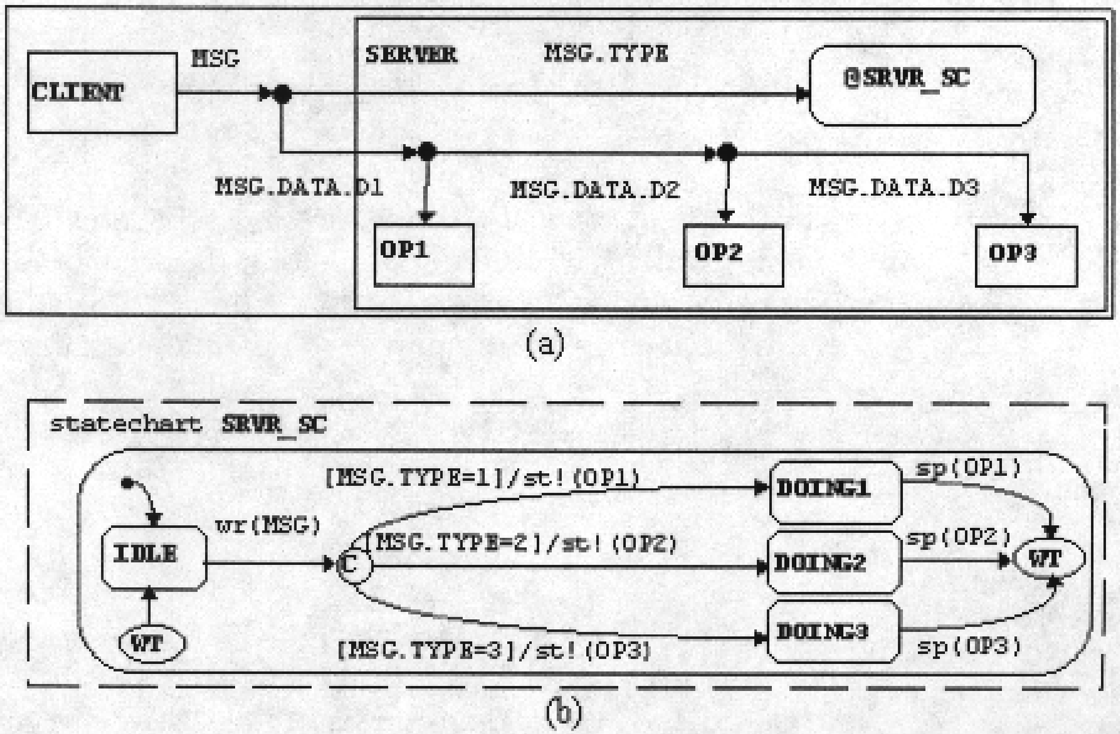


Figure 8.5. Server responding to three service requests

Activities Communicating Through Queues

Queuing facilities for messages are virtually indispensable in modeling multi-processing environments, and especially multiple client-server systems.
We want to have:

- ability to sent unlimited number of messages to the same address, while the receiver is not always in a position to accept them,
- no message is consumed before one that was sent earlier,
- possibility for concurrently active components to write messages to the same address at the same moment
- possibility for concurrently active components to read different messages to the same address at the same moment

Queues and their operations

A queue is an ordered, unlimited collection of data-items, all of the same data type. The queue is usually shared among several activities, which can employ special actions to add elements to the queue and read and remove elements from it.

- **q_put(Q,D)** add the value of expression D to the queue
- **q_urgent_put(Q,D)** add the value of expression D to the head of the queue
- **q_get(Q,D,S)** extract the element at the head of Q and place it in D
- **q_peek(Q,D,S)** same as above without removing the element from Q
- **q_flush(Q)** clears Q totally

Combination with Data Stores

The following figure illustrates the order in which operations on a queue are performed during a step:

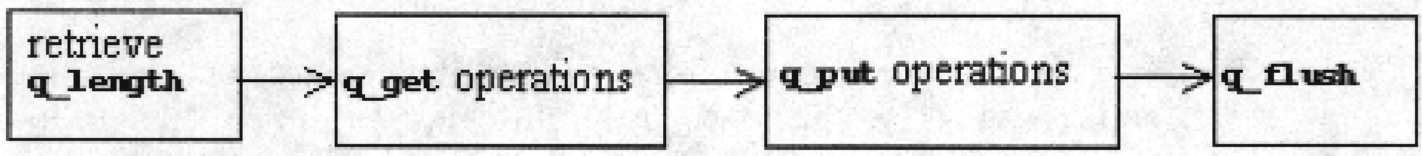


Figure 8.6. Operations on a queue during a step

Queues can be associated with data stores just like data-items of other types can.

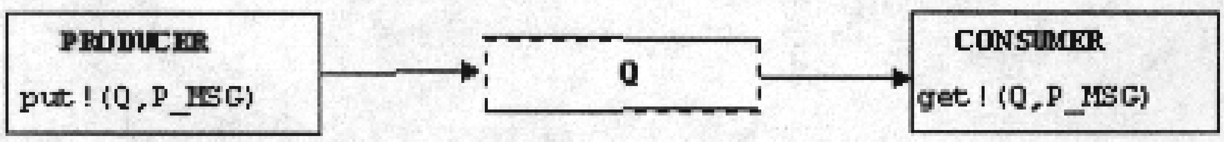


Figure 8.7. A queue associated with a data-store

Example

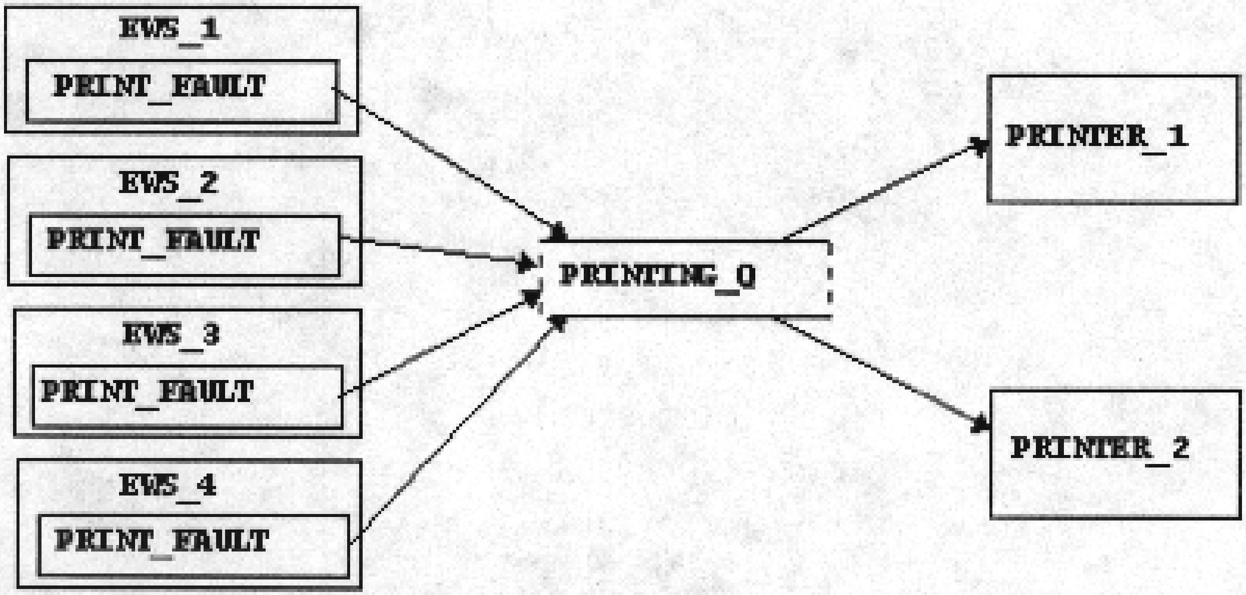
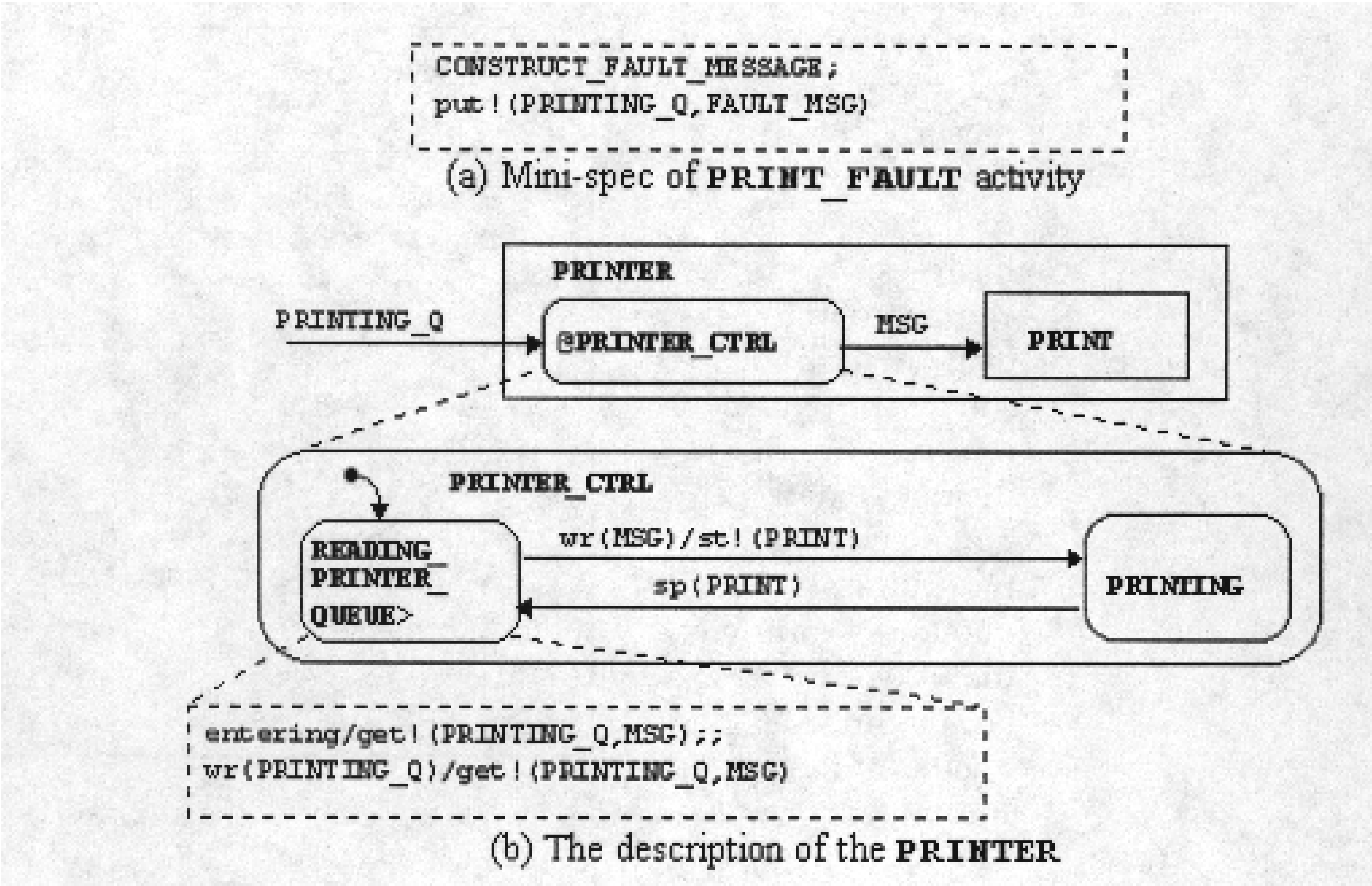


Figure 8.8. Multiple clients served by multiple servers via a queue

Example



(a) Mini-spec of **PRINT_FAULT** activity

(b) The description of the **PRINTER**

Figure 8.9. Writing and reading messages from a queue

Conditions and Events Related to States

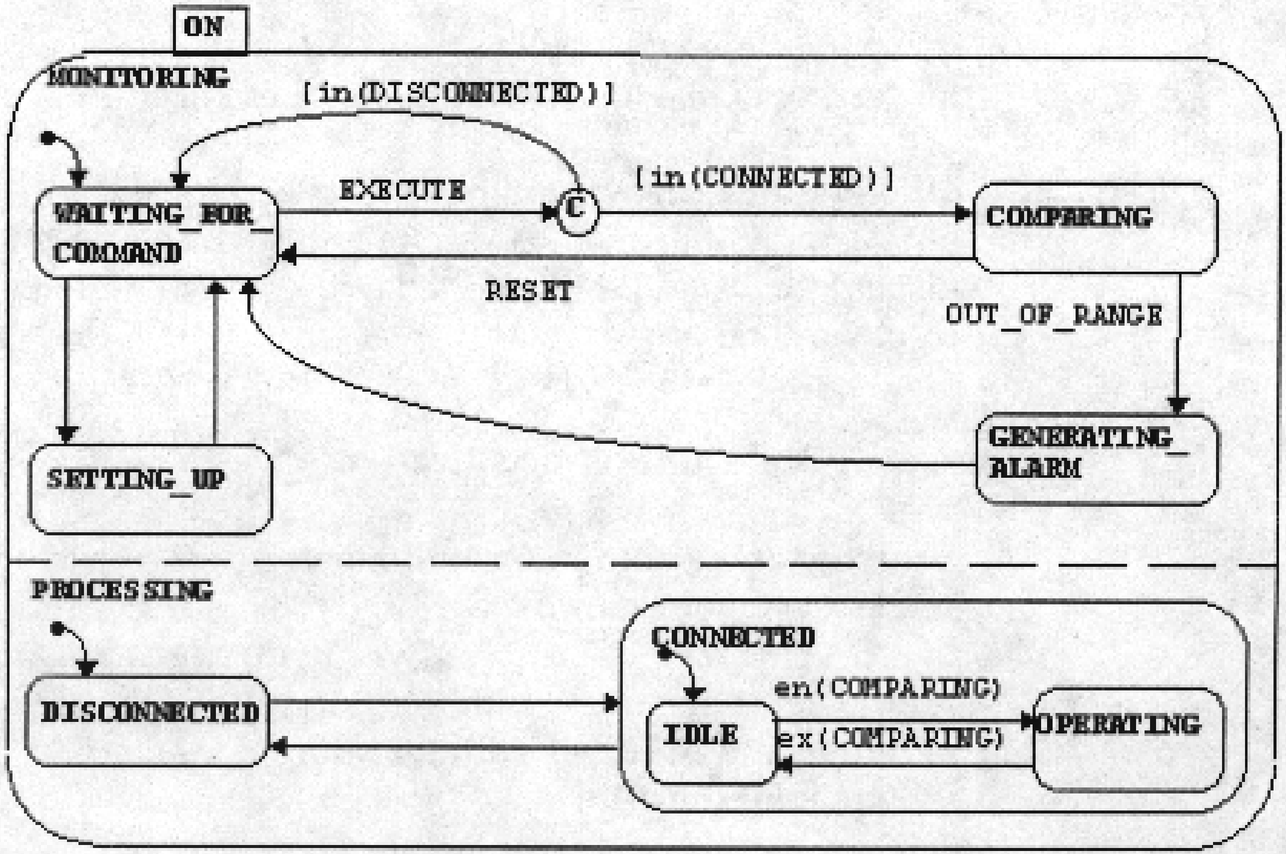


Figure 4.15. Conditions and events related to states

Condition Connector

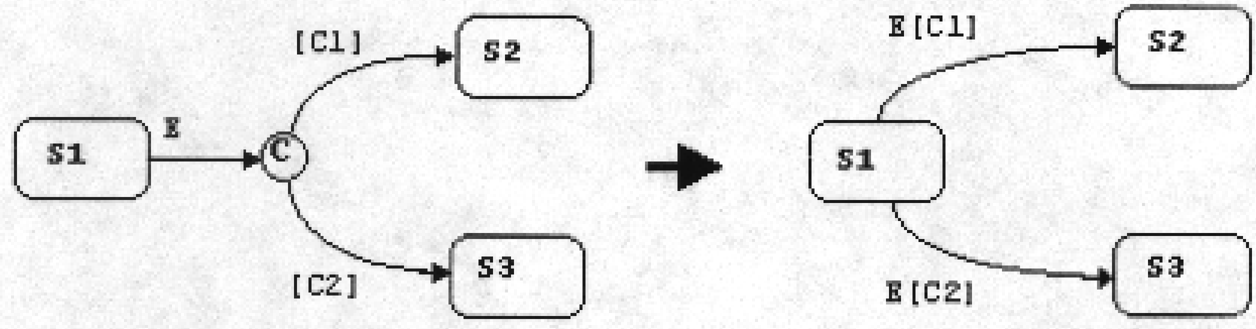


Figure 4.17. A condition connector and compound transitions

Switch Connector

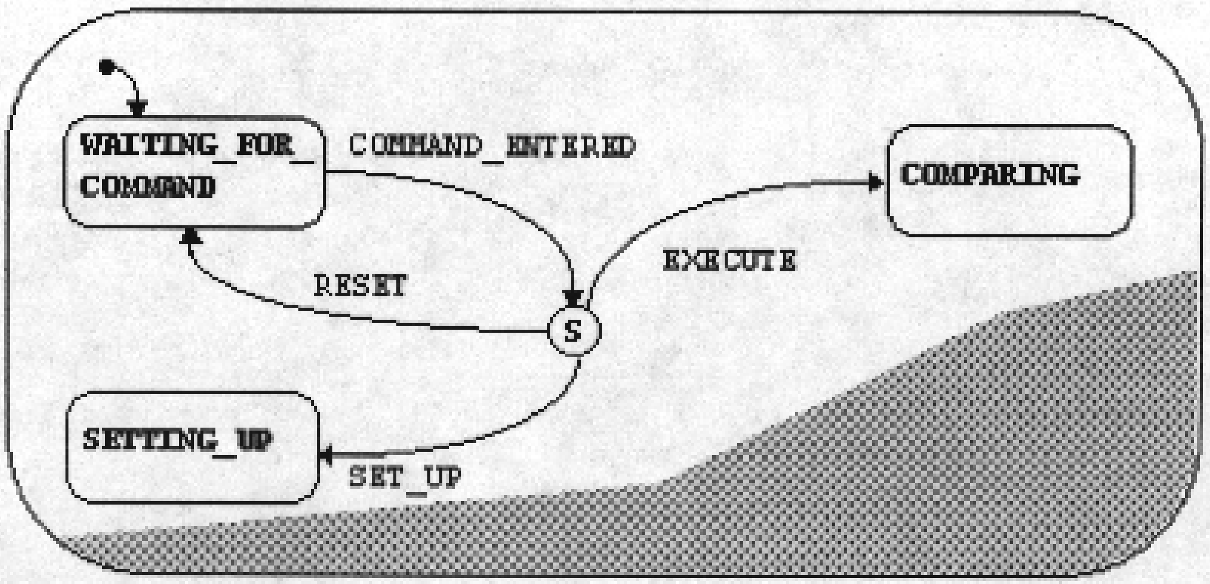
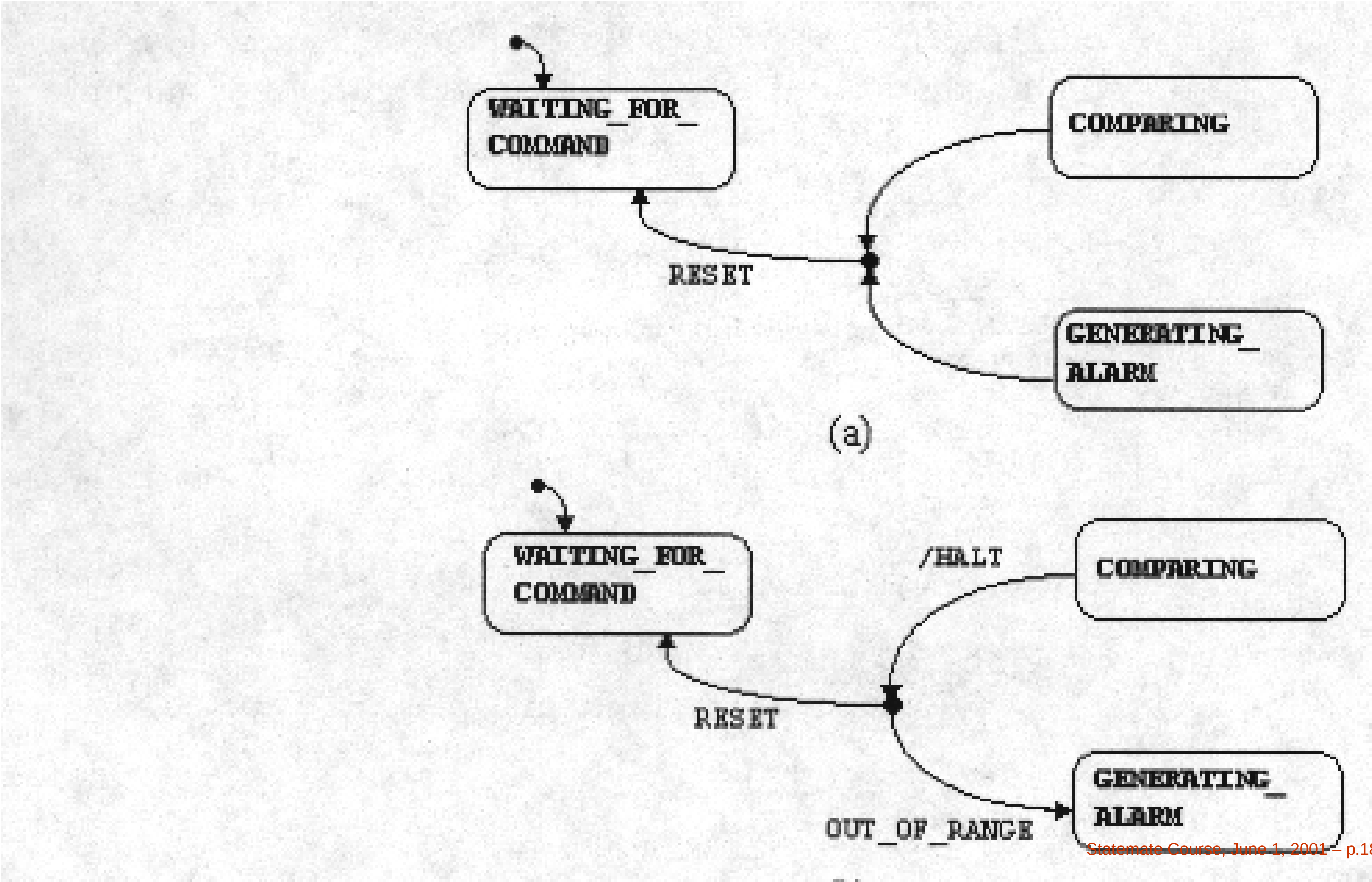


Figure 4.18. A switch connector

Junction Connector



Example

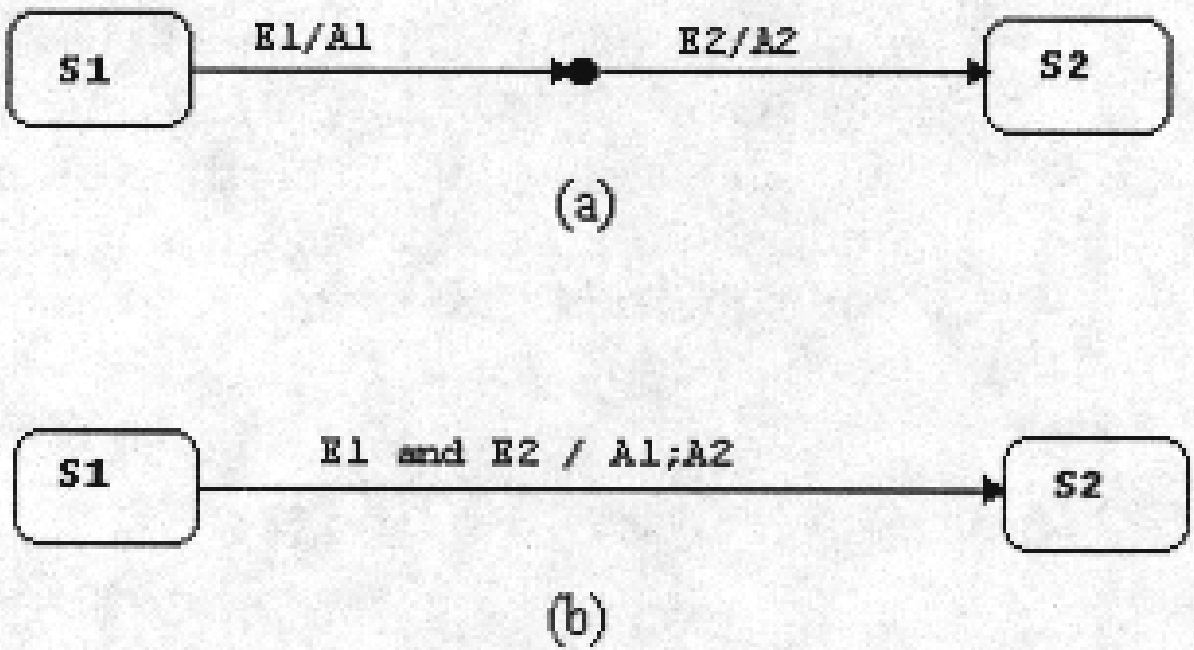


Figure 4.20. Two equivalent transition constructs

Diagram Connector

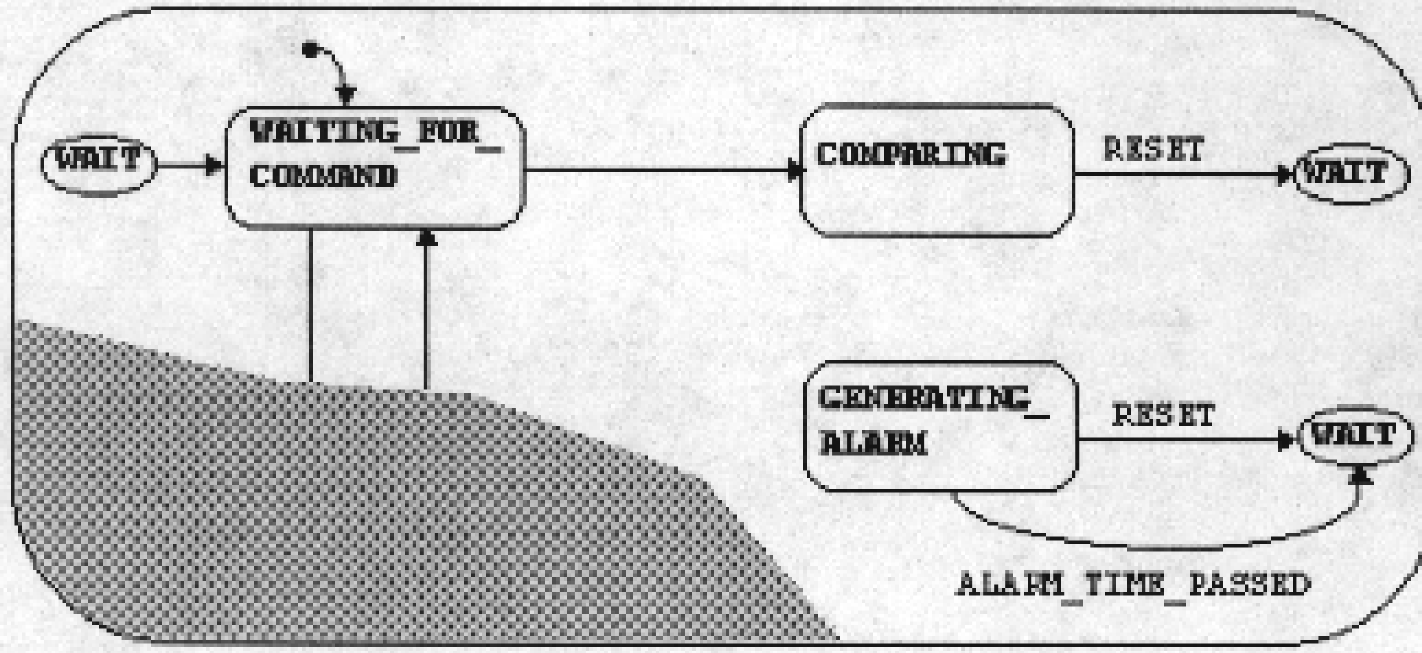


Figure 4.21. Diagram connectors

Transitions to and from And-States

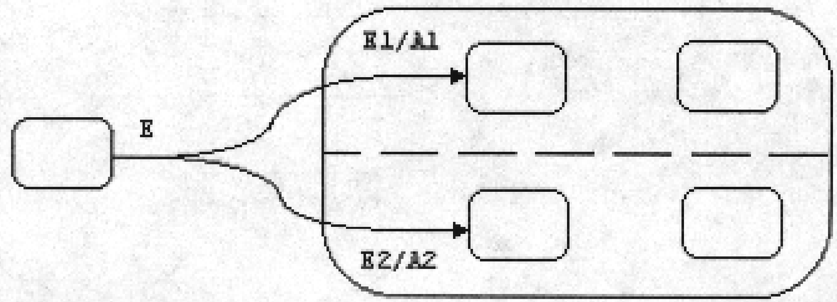


Figure 4.23. Triggers and actions on a fork construct

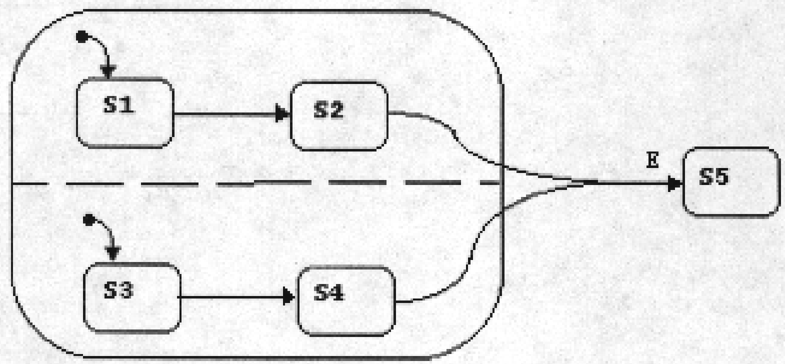


Figure 4.24. A joint connector in a merge construct

Asymmetric Cases

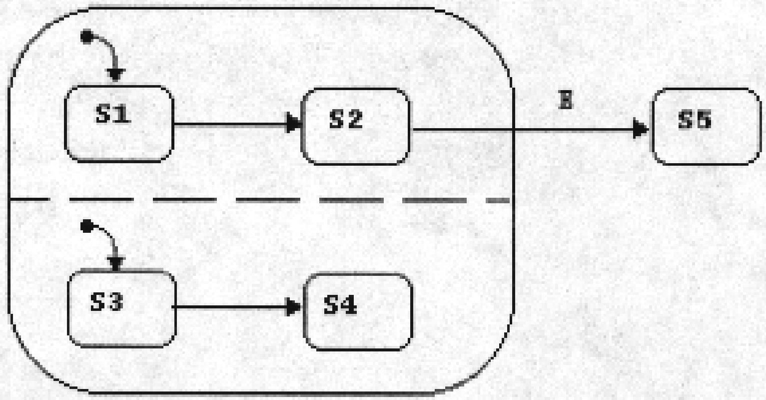
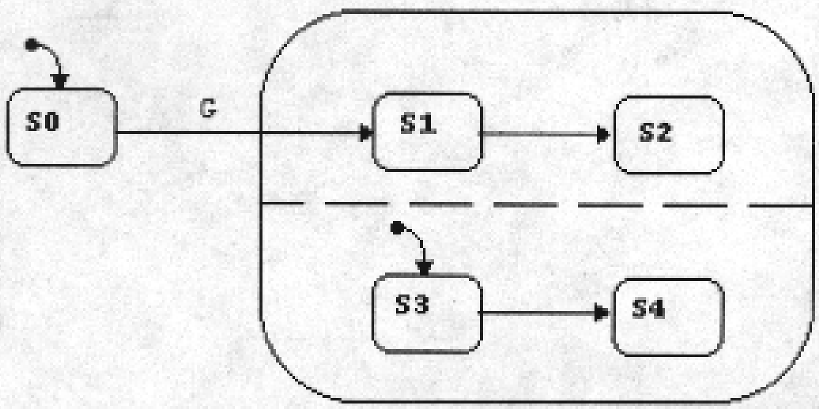


Figure 4.25. A transition from an and-state



Module-Charts

Structural Description: High-Level Design

Module-charts describe the structural view – sometimes called the architectural view – of the system under development. Module-charts are typically used in the high-level design stage of the project.

Structural View

The structural view captures the system's high-level design. A structural description of the system specifies the components that implement the capabilities described by the functional and behavioral views.

These components may be:

- hardware,
- software,
- or even humans.

EWS Example

CCU (control and computation unit): The central CPU, within which the main control of the system and the basic computations take place.

SIGNAL_PROCESSOR: The subsystem that processes the signal produced by the sensor and computes the value to be checked. It consists of an analog-to-digital unit, and a high speed processor that works at the required checking rate.

MONITOR: The subsystem that communicates with the operator. It consists of a **KEYBOARD** for commands and data entry, and a **SCREEN** for displaying messages.

ALARM_SYSTEM: The subsystem that produces the alarm, in visual and/or audible fashion.

PRINTER: The subsystem that receives the messages (text and formatting instructions) and prints them.

Connections to functional view

Sometimes There is a clear correspondence between the top-level activities in the functional view and the top-level subsystems in the structural view, e.g., SIGNAL_PROCESSOR implements the activity PROCESS_SIGNAL.

In other cases the structural decomposition is quite different from the functional decomposition. E.g., the CCU subsystem carries out both the EWS_CONTROL and COMPARE activities, whereas the DISPLAY_FAULT activity is divided into subactivities that are distributed among the ALARM_SYSTEM and MONITOR subsystems.

Internal and External Modules

The structural view is represented by the language of Module-charts.

- There exist two types of **internal** modules:
  - **execution modules**
  - **storage modules**
- And there exist **external modules**

Modules

- Execution modules may be submodules of other external modules only.
- Storage modules may be submodules of other storage modules or of execution modules.
- External modules are always external to an execution module or storage module, and there is no hierarchy of external modules.

EWS-Example

The next figure shows the structural decomposition of the EWS, including a storage module DISK, that stores the fault messages:

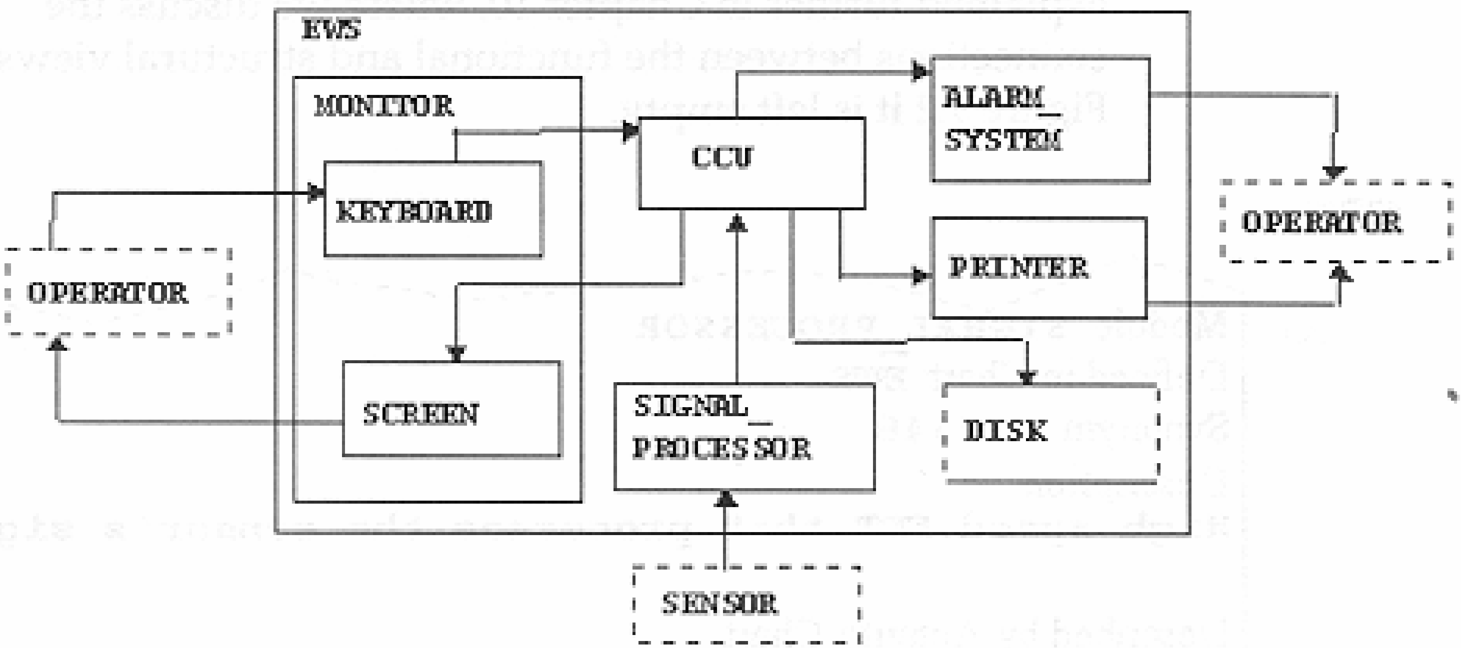


Figure 9.1. Structural decomposition of the EWS

Data Dictionary Entry

The Data Dictionary contains a special field, DESCRIBED BY ACTIVITY-CHART, which is used to connect modules with their functional description:

Module: **SIGNAL_PROCESSOR**
Defined in Chart: **EWS**
Synonym: **FFT548**
Description:
High speed FFT that processes the sensor's signal.

Described by Activity-Chart:

Attributes:

Name	Value
IMPLEMENTATION	**HARDWARE**

Long Description:
This subsystem processes the analog signal coming from the sensor. It is a standard FFT, that also contains an A/D unit.

Communication Between Modules

As in Activity-charts we use labeled arrows between modules to denote communication between them. They are called **flow-lines** or **m-flow-lines** to emphasize that they connect modules.

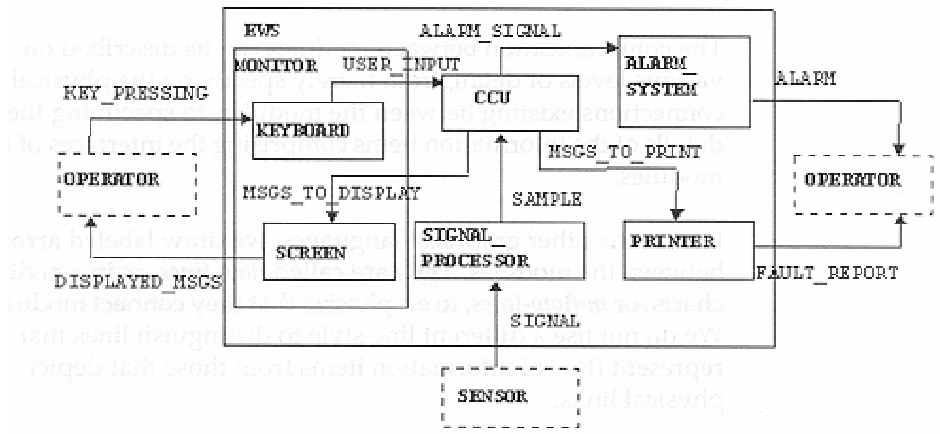


Figure 9.3. Flow of information among modules

Here, USER_INPUT contains the information-flow COMMANDS, the data-item RANGE_LIMITS and the condition SENSOR_CONNECTED.

Physical Links Between Modules

Arrows in a module-chart may also denote physical communication links, or channels, between modules:

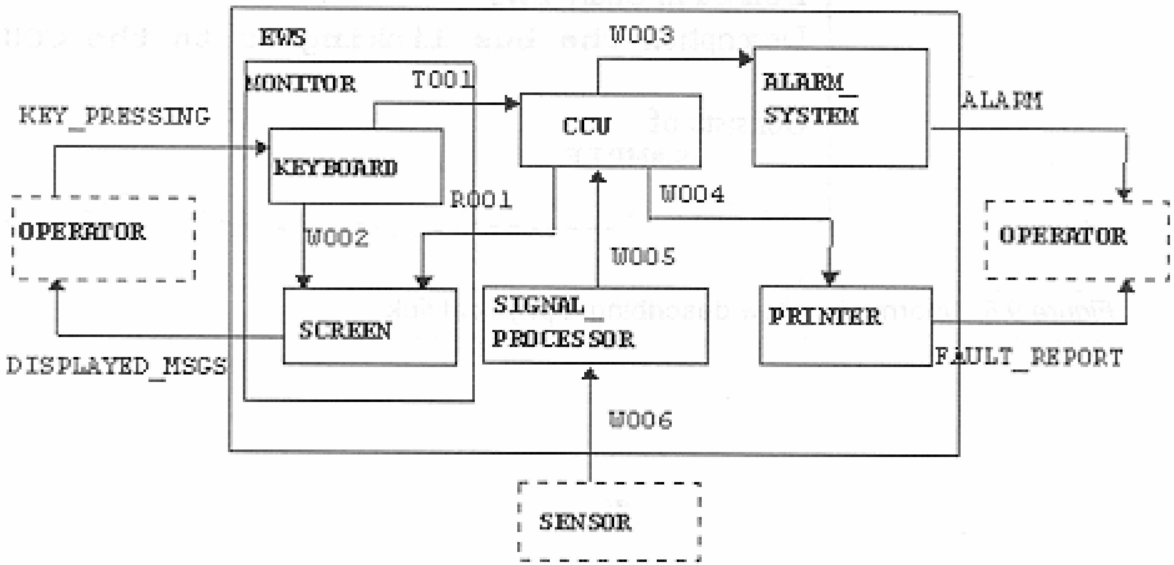
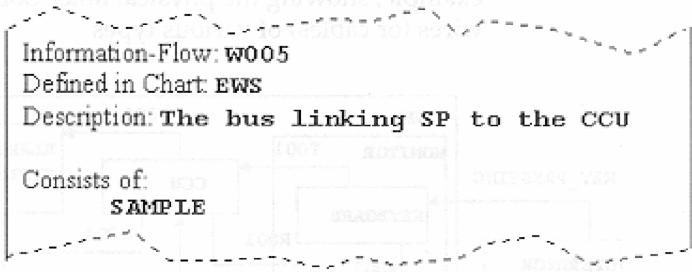


Figure 9.4. Physical links among modules


Information-Flow: W005
Defined in Chart: EWS
Description: The bus linking SP to the CCU

Consists of:
SAMPLE

Connectors and Compound Flow-Lines

Connectors and compound flow-lines are allowed in module-charts exactly as in activity-charts:

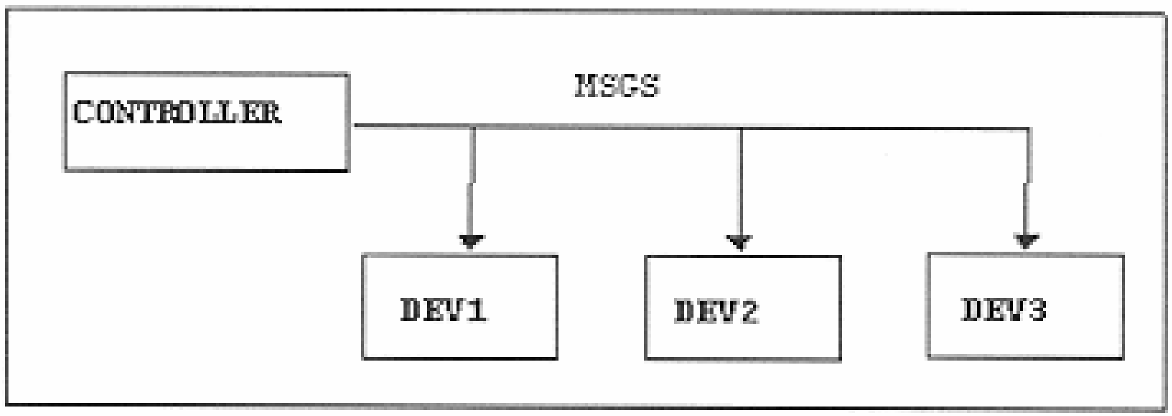


Figure 9.6. Communication link to several devices

Connections Between the Functional and Structu

- The functional view provides a decomposition of the system under development into its functional components, i.e., its capabilities and processes.
- The structural view provides a decomposition of the system into the actual subsystems that will be part of the final system, and which implement its functionality.

Types of connections

There are three types of connections between the functional and structural views:

1. describe the functionality of a module by an activity-chart: **Activity-chart Describing a Module**
2. allocate specific activities in an activity-chart to be implemented in a module: **Activities Implemented by Modules**
3. map activities in the functional description of one module to activities in that of some other module: **Activities Associated with a Module's Activ ities**

Conclusion

In conclusion, we may wish to attach functional descriptions, i.e., activity-charts, to modules on different levels of the structural decomposition:

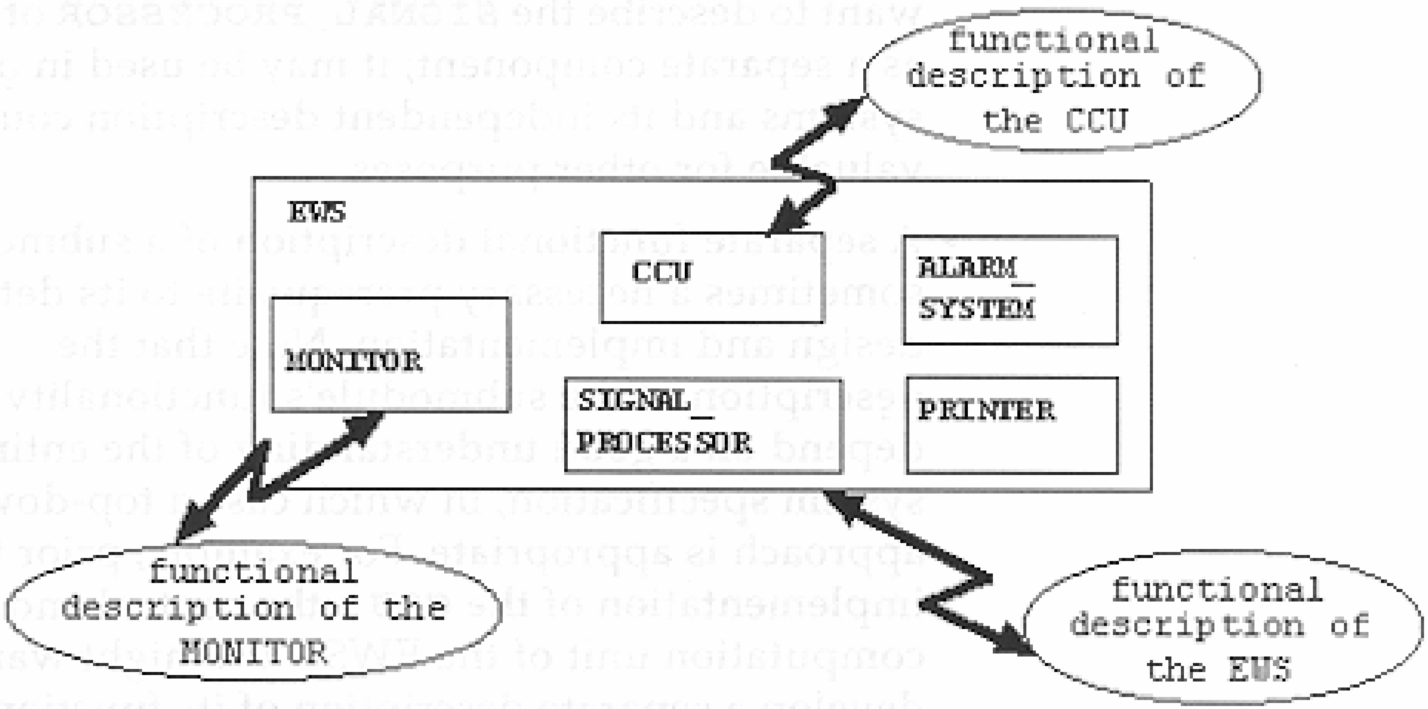


Figure 10.1. Functional descriptions attached to different modules

Activity-chart Describing a Module

The activity-chart EWS_ACTIVITIES

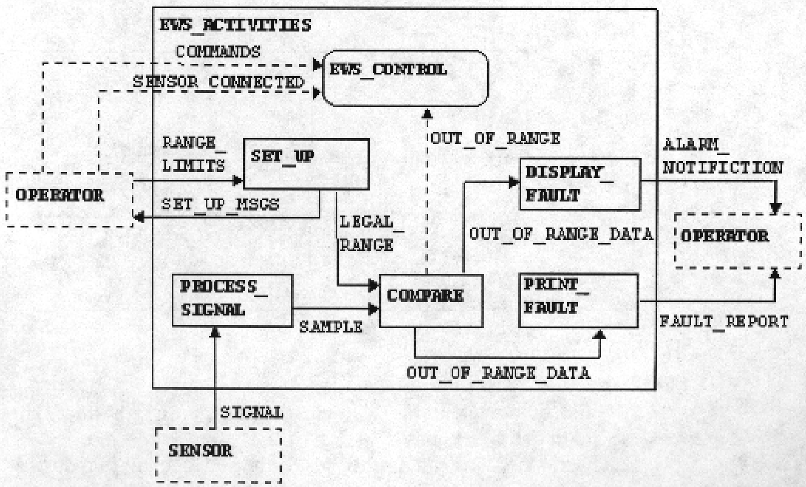
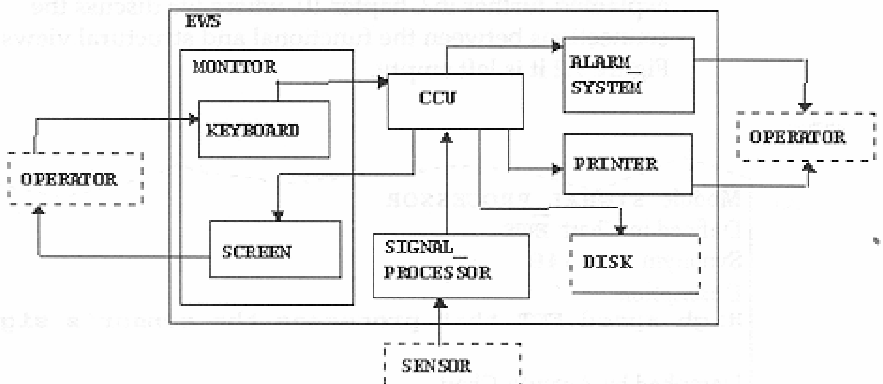


Figure 2.5. **EWS_ACTIVITIES**, its environment and flow of information

describes the functionality of the module EWS



This connection is specified in the Data Dictionary:

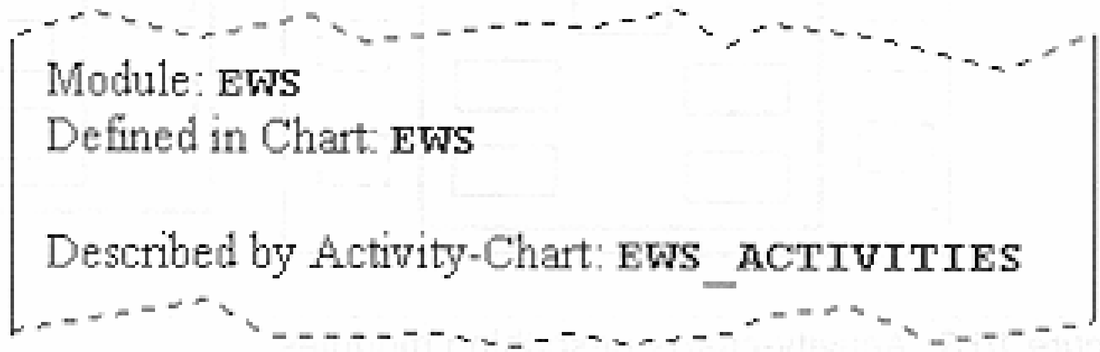


Figure 10.2. A module described by activity-chart

Notice that the connection is between an activity-chart and a module!

Top-Down Approach

One may now want to specify an activity-chart CCU_AC for the module CCU:

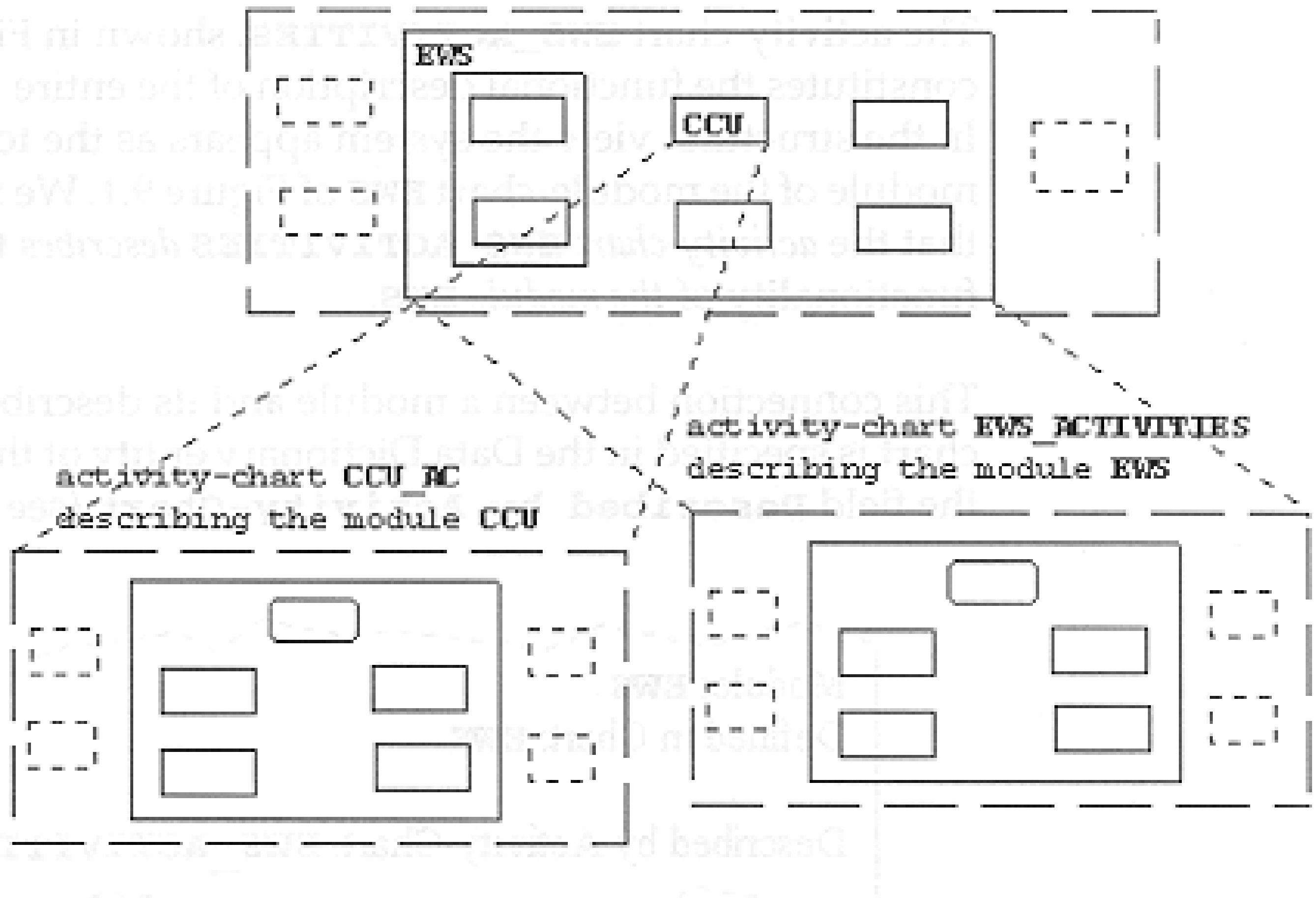
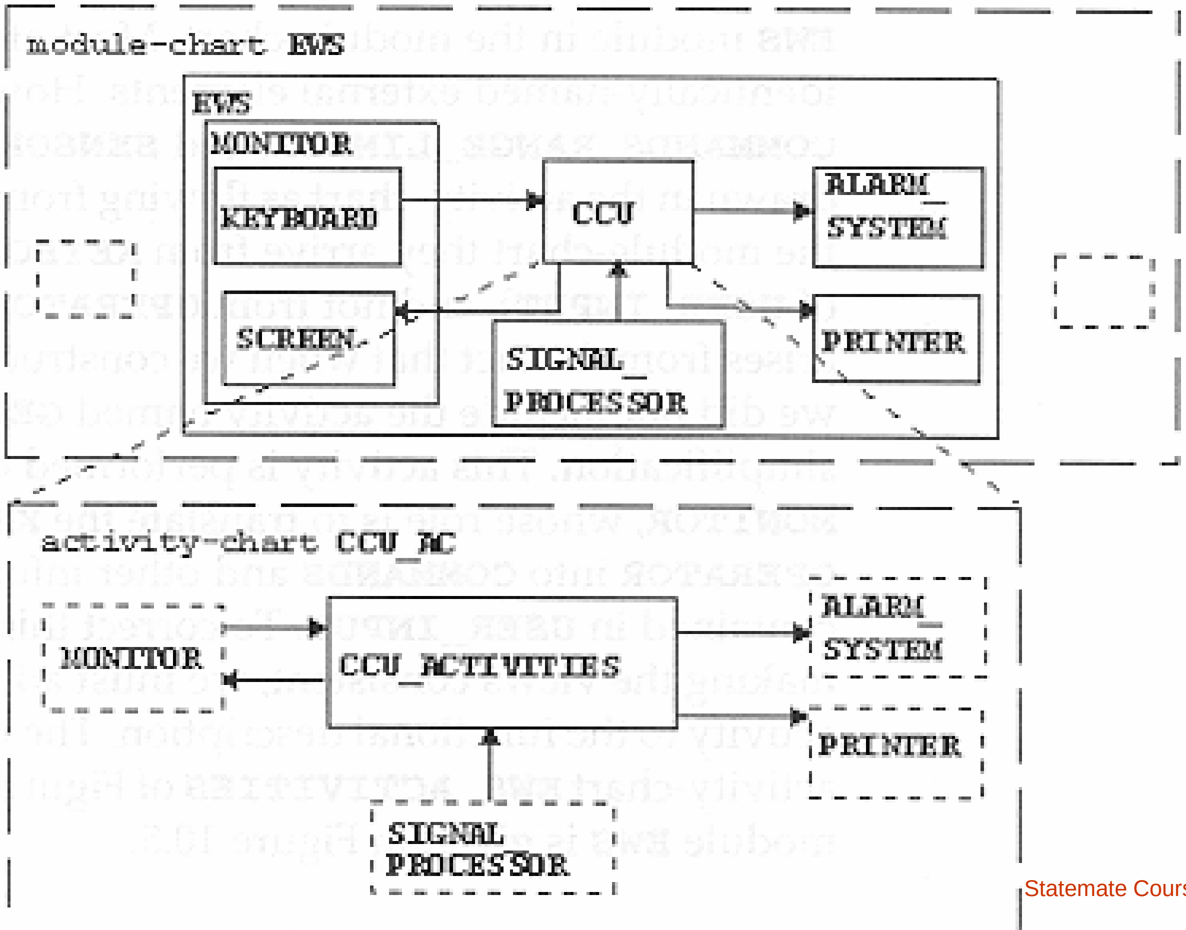


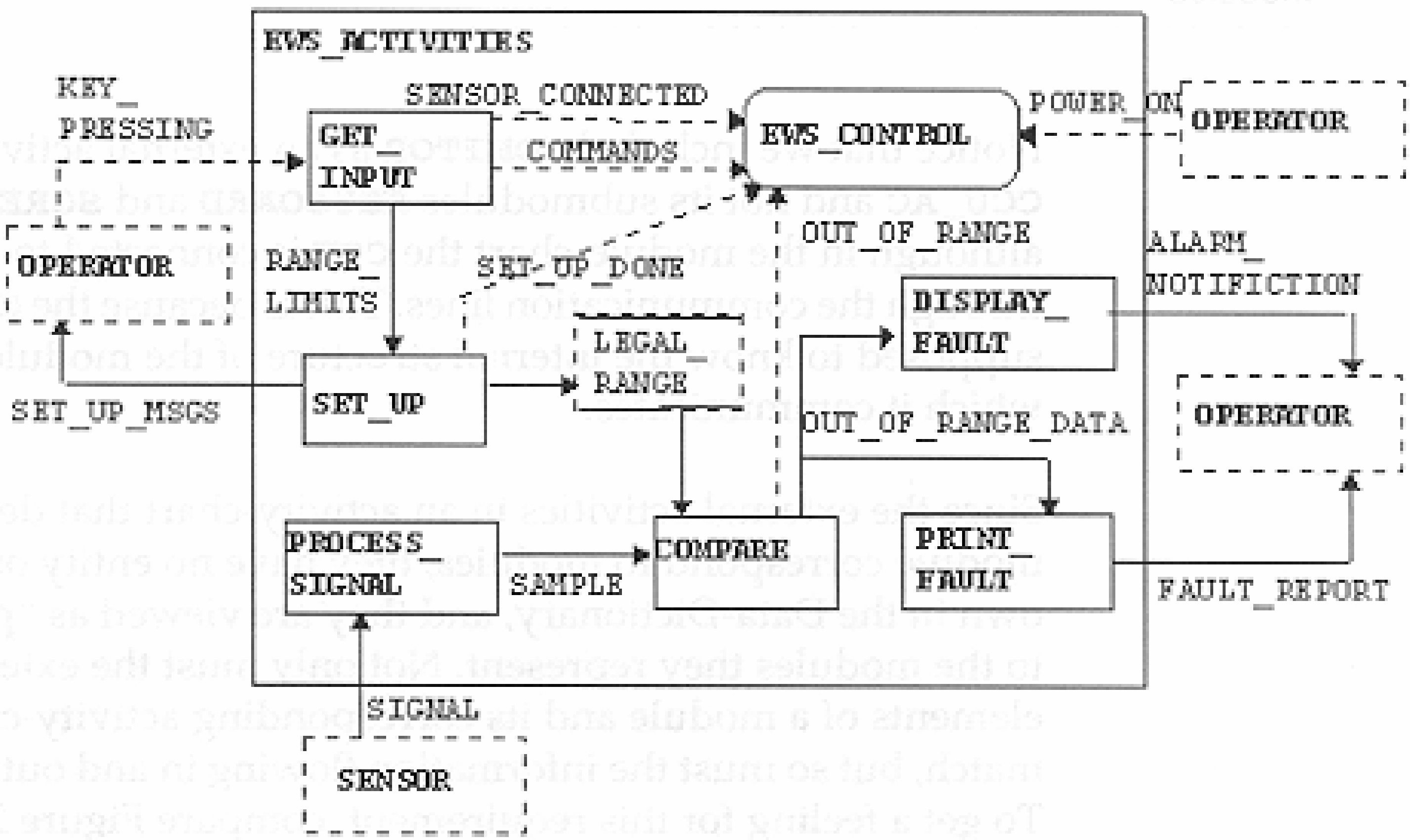
Figure 10.3. Activity-charts describing modules

Correspondence between views

There must be a correspondence between the functional and structural decompositions of a module in terms of the environment and the interface with it:



Since also the flow-lines have to be correct we have to introduce an activity GET_INPUT which will be implemented by the MONITOR module:



Activities implemented by Modules

When the module described by the activity-chart is eventually decomposed into submodules, we may be more concrete and allocate the relevant activities and data-stores to the submodules:

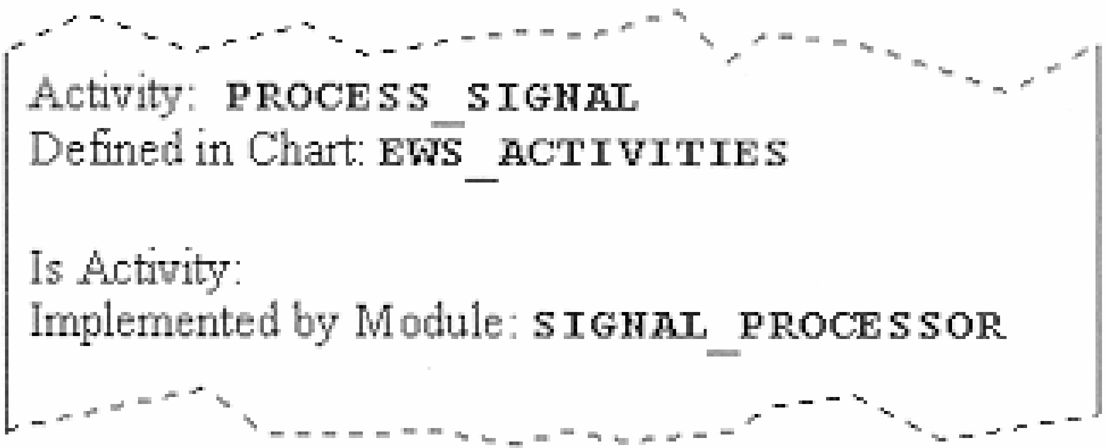
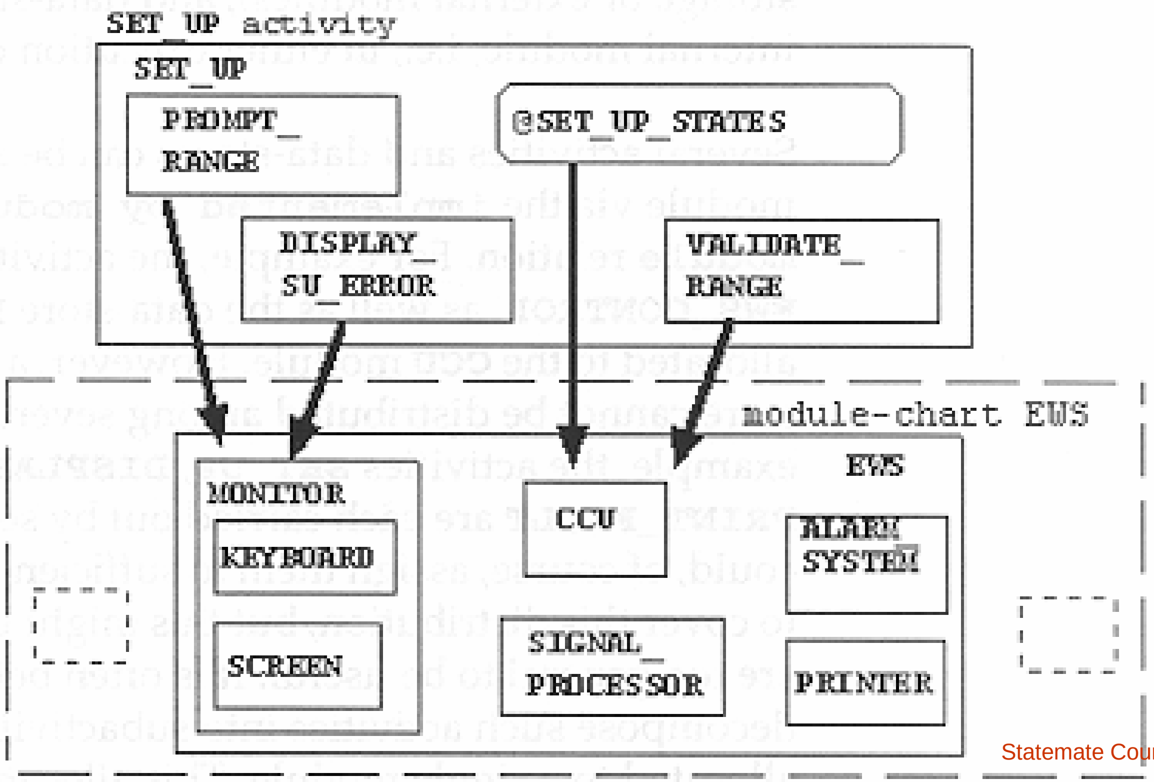


Figure 10.6. An activity implemented by a module

A single activity or data-store cannot be distributed among several modules.
Therefore, one has to decompose such activities (or data-stores) into subactivities that can each be allocated to a single module:



Activities Associated with a Module’s Activities

On the one hand, there is the EWS_ACTIVITIES describing the functionality of the whole system. On the other hand, also the submodules implement activities:

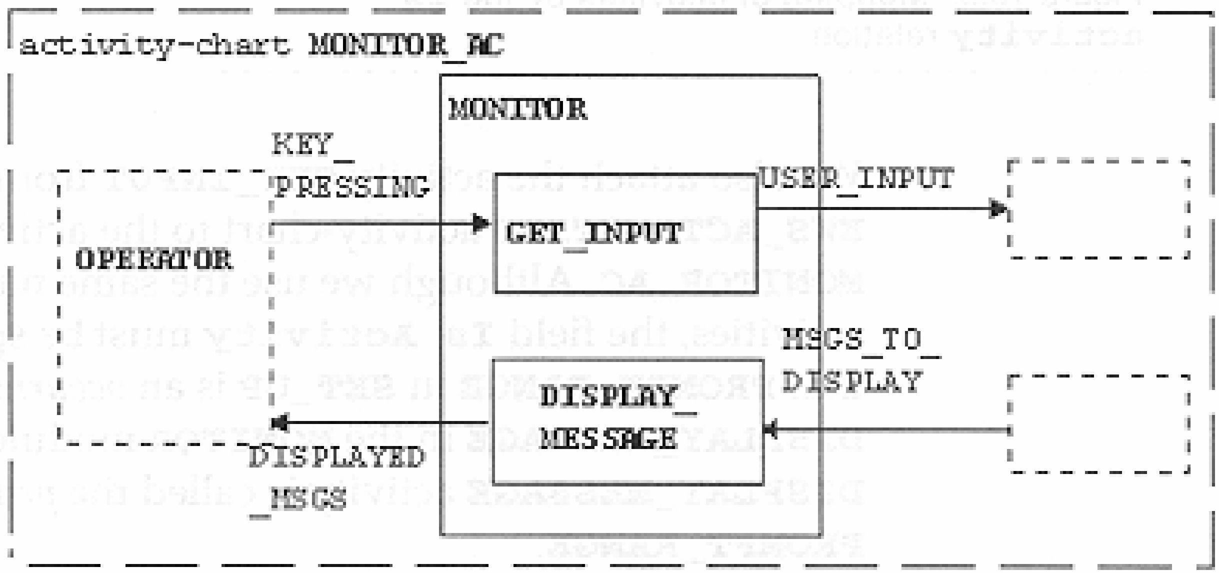


Figure 10.8. Activity-chart of **MONITOR**

Then, one wishes to associate subactivities of EWS_ACTIVITIES with those implemented by a submodule:

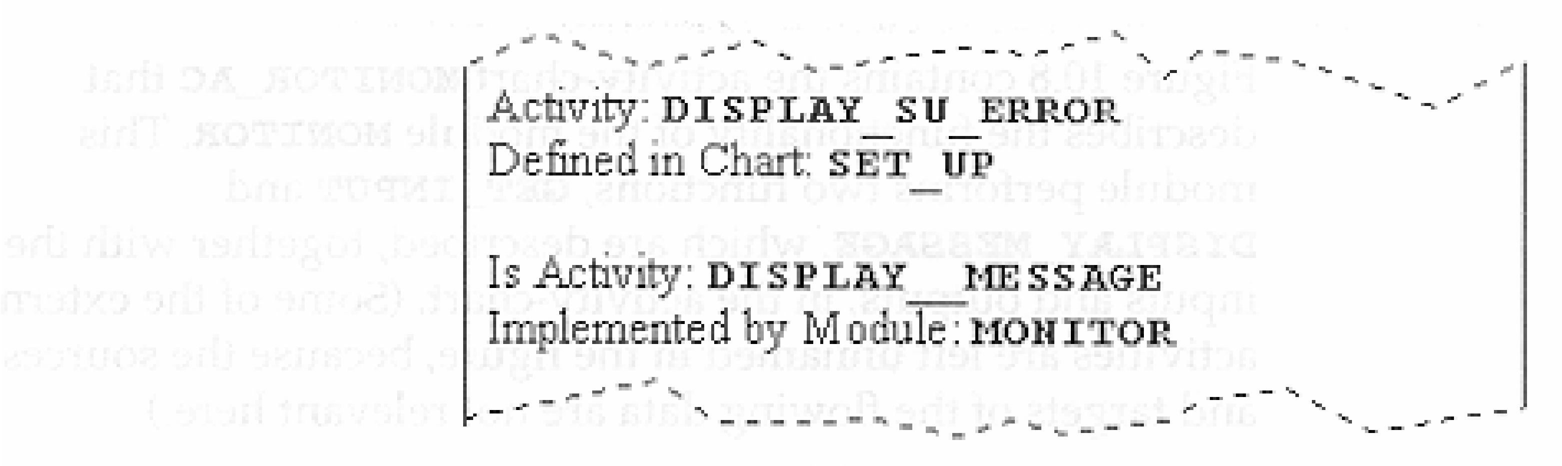


Figure 10.9. Mapping of activities by the **is activity** relation