

Outline

Case Study

Method

Requirements

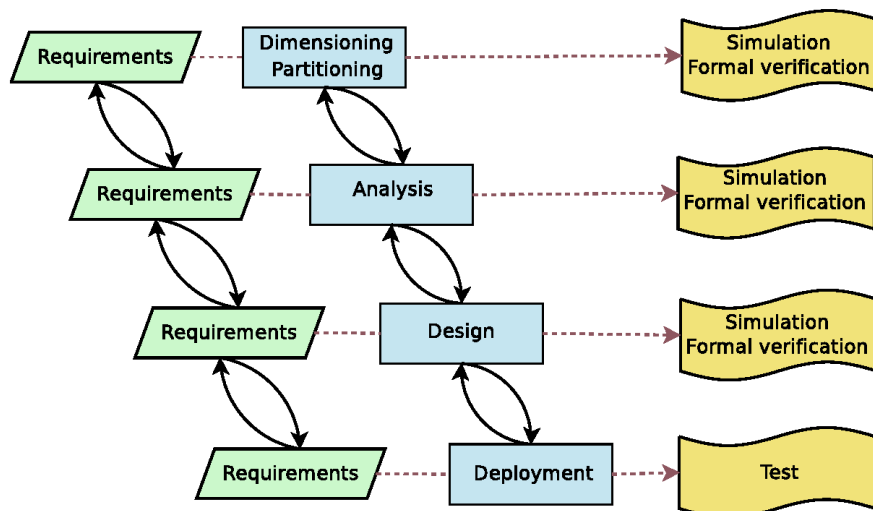
(Partitioning)

Analysis

Design



Overview of the V Cycle



Outline

Case Study

Method

Requirements

(Partitioning)

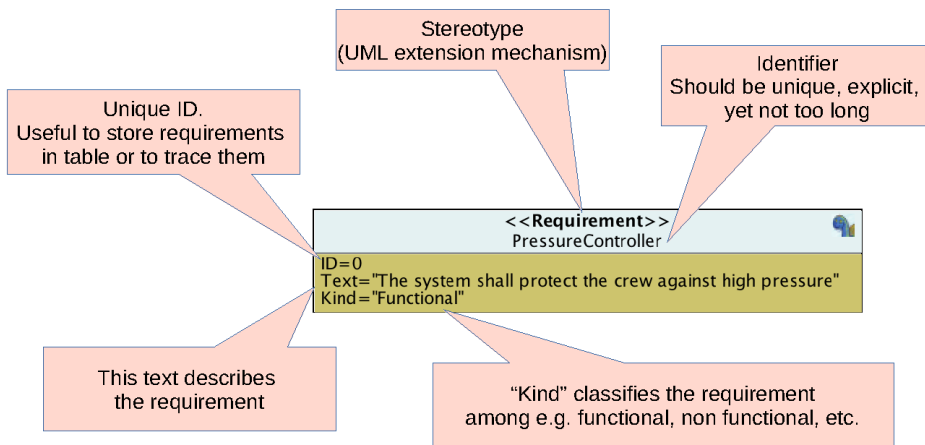
Analysis

Design



Requirement Node

- **A requirement node identifies a requirement by:**
 - A unique identifier (so as to achieve tracability)
 - A description in plain text
 - A type (functional, non functional, performance, security, ...).

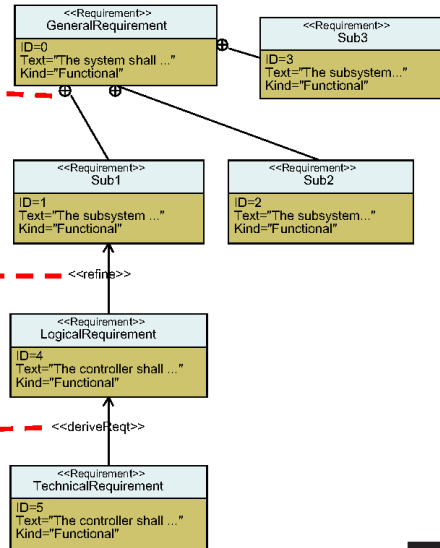


Relations Between Requirement Nodes

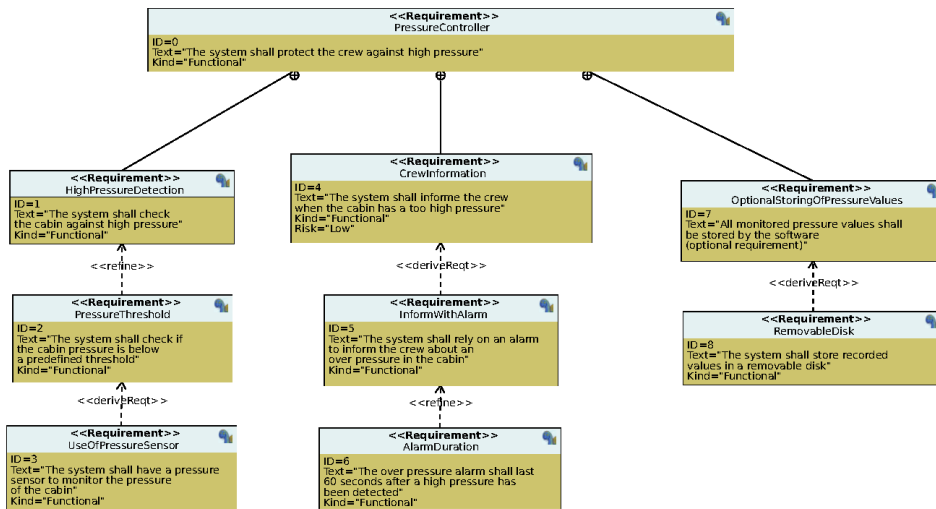
Containment relation
 Splits up a compounded requirement into elementary ones

Refinement
 Relates two requirements of different abstraction levels

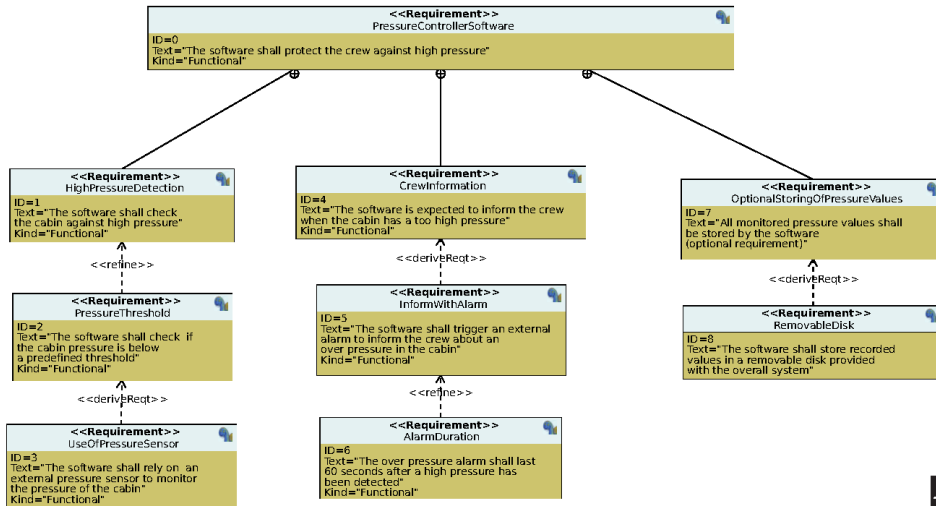
Derivation
 Builds a new requirement from the reuse of other requirements



Requirement Diagram - Pressure Controller - System View



Requirement Diagram - Pressure Controller - Software View



Outline

Case Study

Method

Requirements

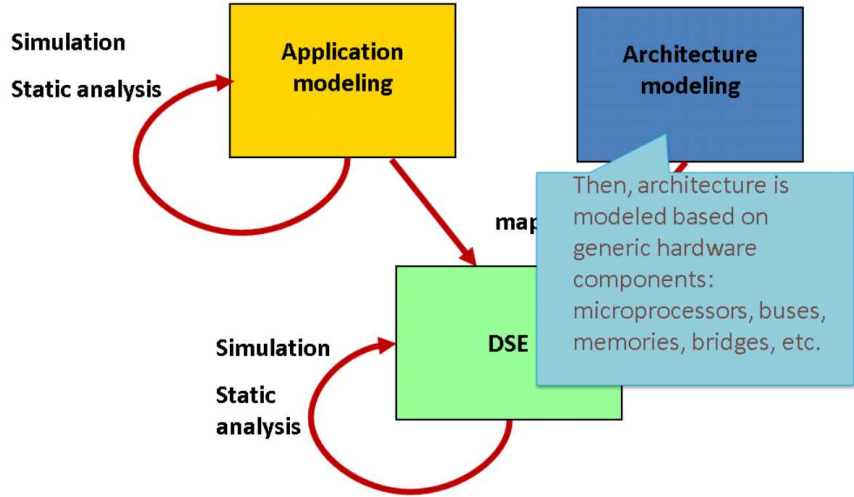
(Partitioning)

Analysis

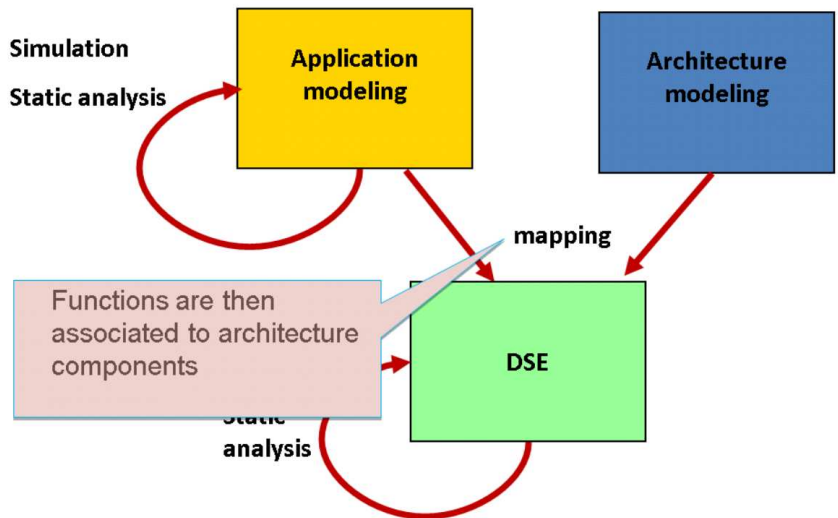
Design



Architecture Modeling



Mapping



Outline

Case Study

Method

Requirements

(Partitioning)

Analysis

Design



System Analysis

Analysis = Understanding what a client wants

- So, it does not mean "creating a system", but rather "understanding the main functionalities" of the system to be designed
- Can be performed before or after the partitioning stage

Analysis method

1. System boundary and main functions → *Use Case Diagram*
2. Relations between main functions → *Activity Diagram*
3. Communications between main system entities and actors → *Sequence Diagram*



Use Case Diagram: Method

■ Shows what the system does and who uses it

1. Define the boundary of the system

- Inside of the rectangle → What you promise to design
- Outside of the rectangle → System environment (= Actors)
 - This is not part of what you will have to design

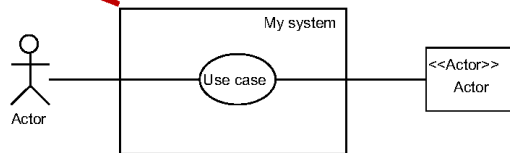
2. Name the system

3. Identify the services to be offered by the system

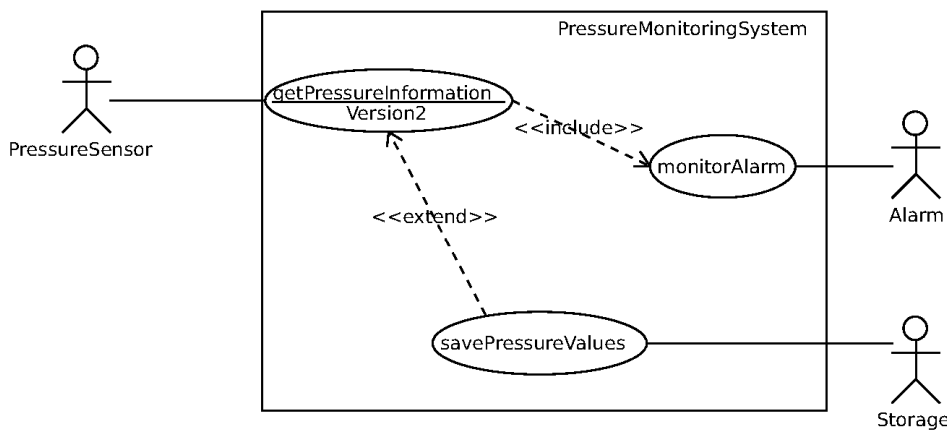
- Only services interacting with actors

4. Draw interactions between functions and actors

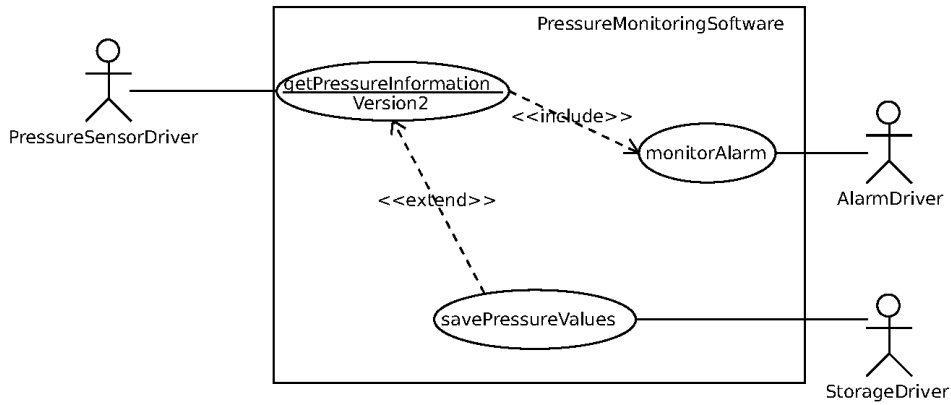
Boundary



Use Case Diagram - Pressure Controller - System View

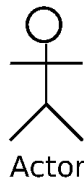


Use Case Diagram - Pressure Controller - Software View

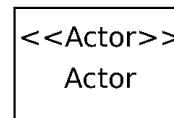


Actors

■ **Syntax 1:** Stickman



■ **Syntax 2:** <<Actor>>



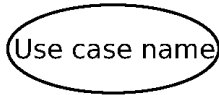
Method

- An actor identifier is a substantive
- An actor must interact with the system



Use Case

- **Syntax:** ellipse with exactly one use case



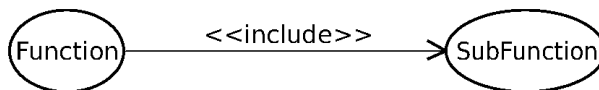
Method

- A use case is described by a verb
 - The verb should describe **the point of view of the system**, not the point of view of the actors
- A use case diagram must **NOT** describe a step-by-step algorithm
 - A use case describes a high-level service/function, not an elementary action of the system

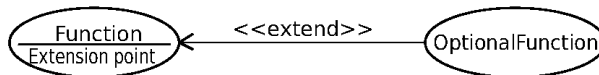


Use Case to Use Case Relations

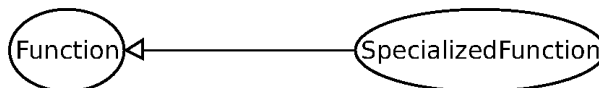
- **Inclusion**
 - A function mandatorily includes another function



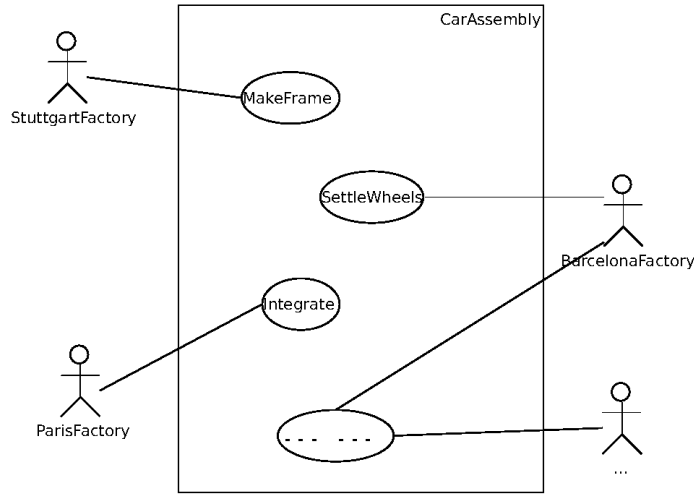
- **Extension**
 - A function optionally includes another function



- **Inheritance**
 - A "child" function specializes a "parent" function

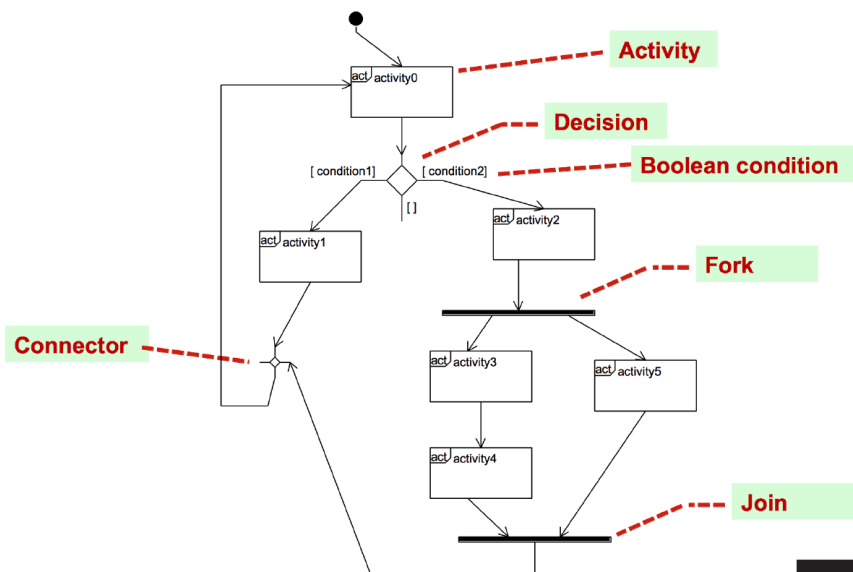


Location-Driven Use Case Diagram

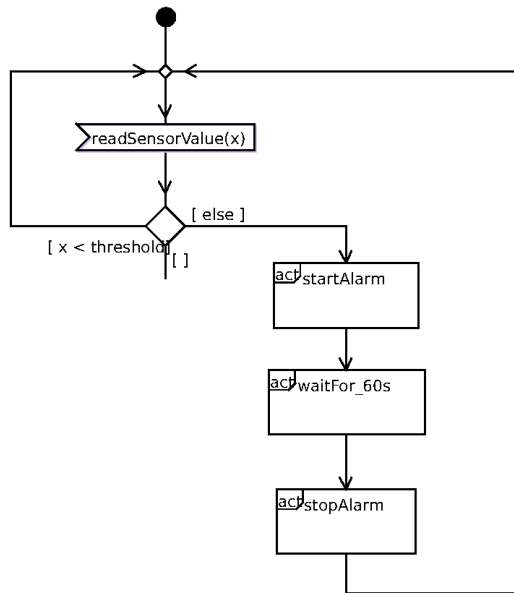


Activity Diagram - Syntax

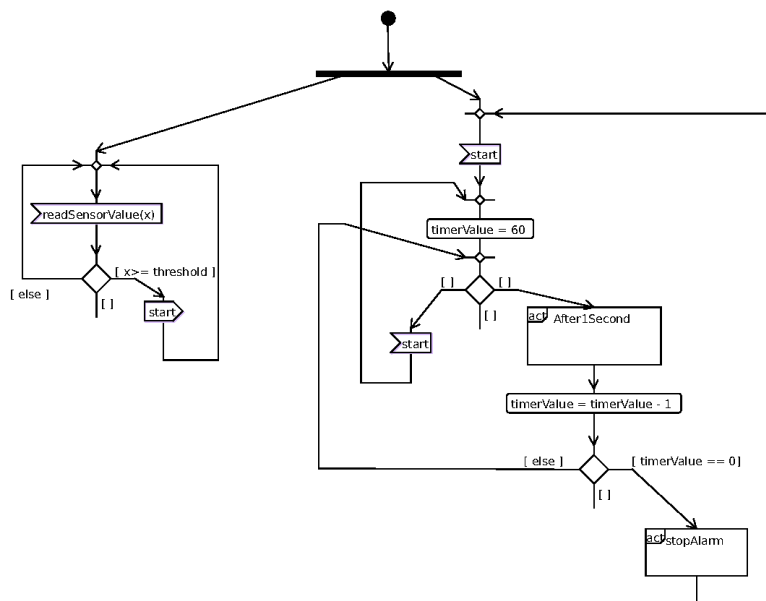
Shows functional flows in the form of succession of actions



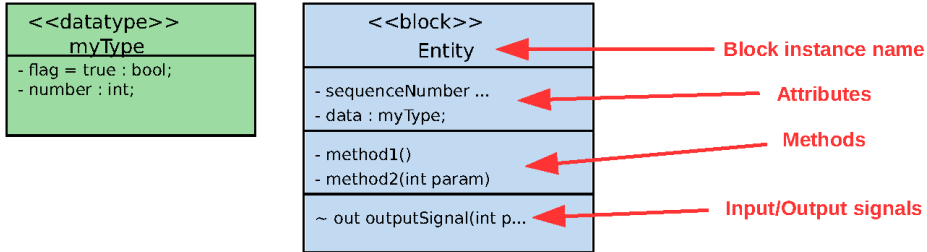
Activity Diagram - Pressure Controller



Activity Diagram - Pressure Controller



(Instance) Block Diagram: Syntax of Blocks

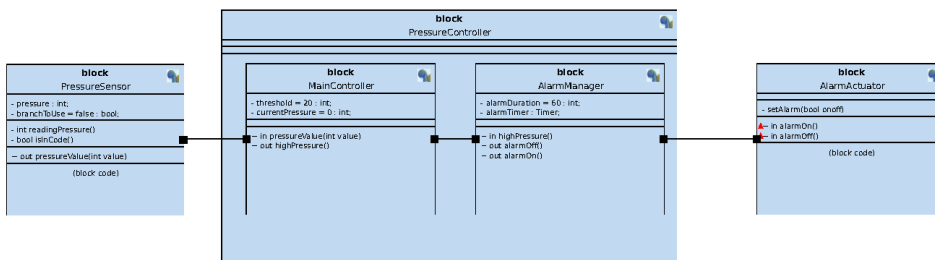


Note

- Upper-case and lower case characters
- Attributes are *private* elements
- Signals have the *package* access right
 - Package = a block + its sub-blocks



(Instance) Block Diagram: Connecting Blocks



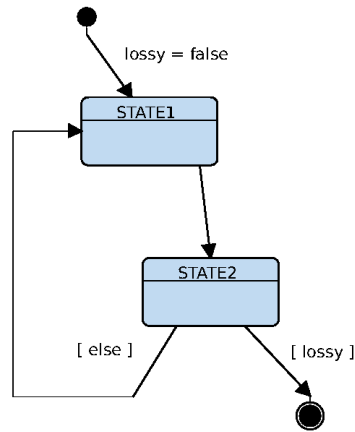
- Ports are connected to allow the state machines of blocks to exchange signals
- A block instance may nest one or several block instances



State Machine - States and Transitions

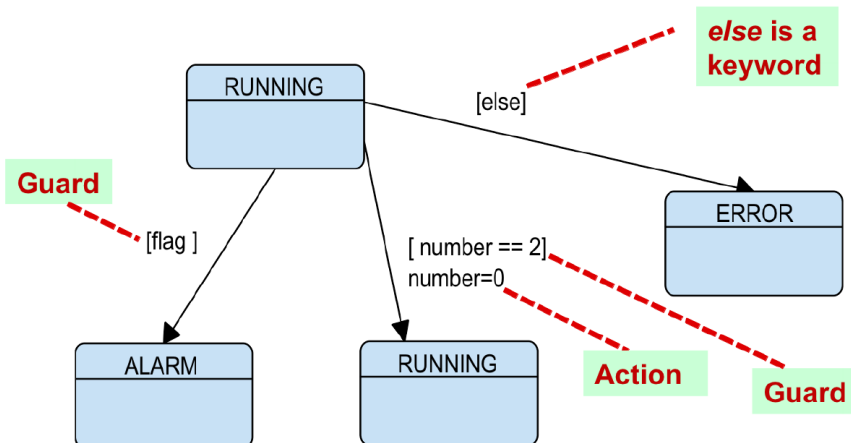
Note

- No parallelism
- States can have several output transitions



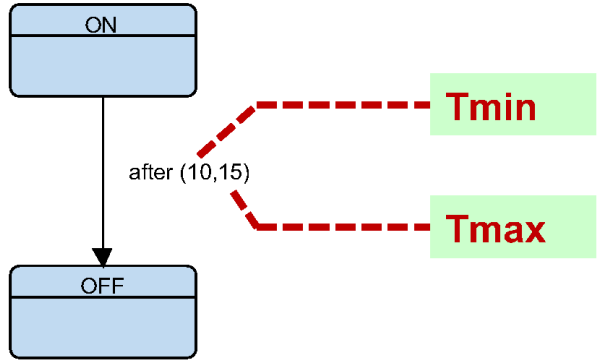
State Machines - Guards

- A transition guard contains a *boolean expression* built upon boolean operators and attributes



State Machines - Time Intervals

- *after* clause with a [Tmin, Tmax] interval

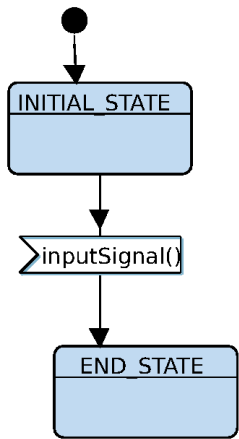


- A transition with no *after* clause has de facto an **after(0, 0)** clause, which means the transition may be fired "immediately"



State Machine - Inputs (1/3)

- A signal reception is a **transition trigger**

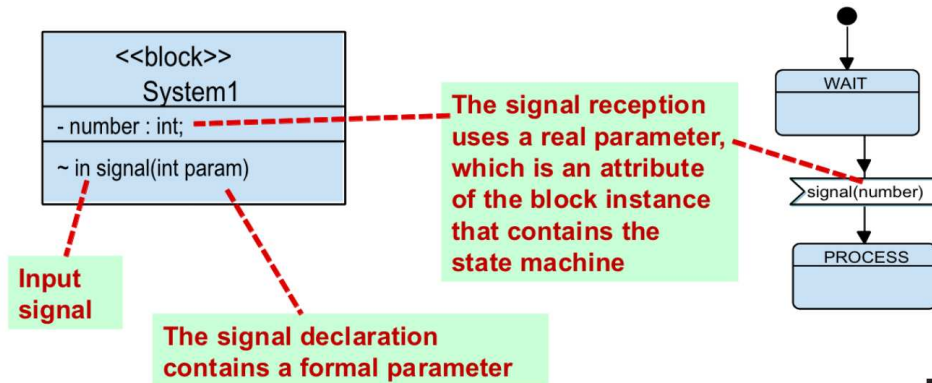


- The transition between INITIAL_STATE and END_STATE is triggered by a signal reception
- **Asynchronous communication**
 - FIFO-based
 - The transition is fired if $size(FIFO, inputSignal) > 0$
- **Synchronous communication**
 - The transition is fired whenever a rendezvous is possible
- Signals can convey parameters



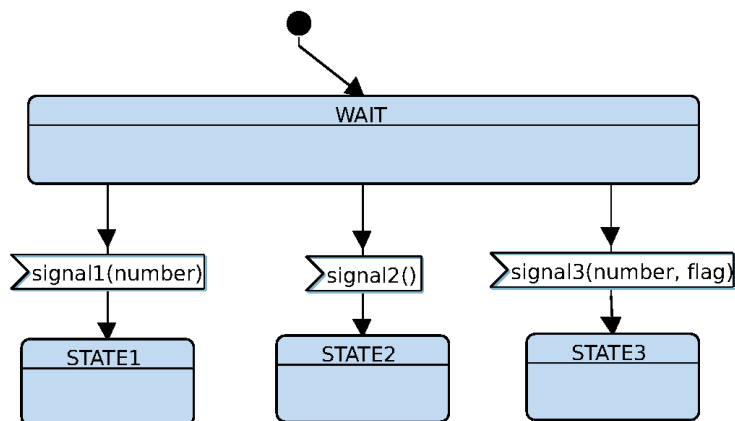
State Machine - Inputs (2/3)

- Signal parameters, if any, are stored in attributes of the block instance that receives the signal



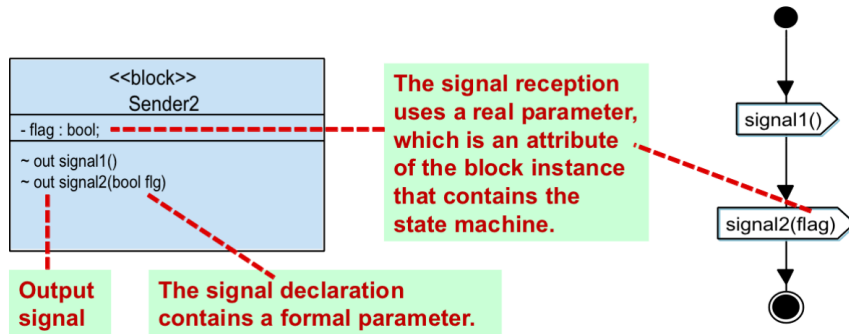
State Machine - Inputs (3/3)

- From the same state it is possible to wait for several signals
 - Asynchronous communication: the first signal in the input queues triggers the transition
 - Synchronous communication: The first ready-to-execute rendezvous triggers the transition



State Machine - Ouputs

- A block instance can send signals with several parameters
 - Constant values may not be used as real parameters → use attributes instead

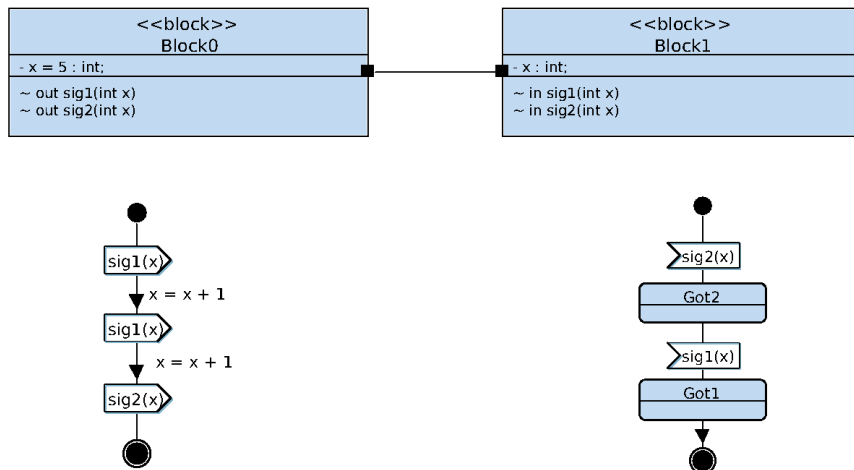


- A block instance cannot send two or several signals in parallel but it can send two or more signals in sequence



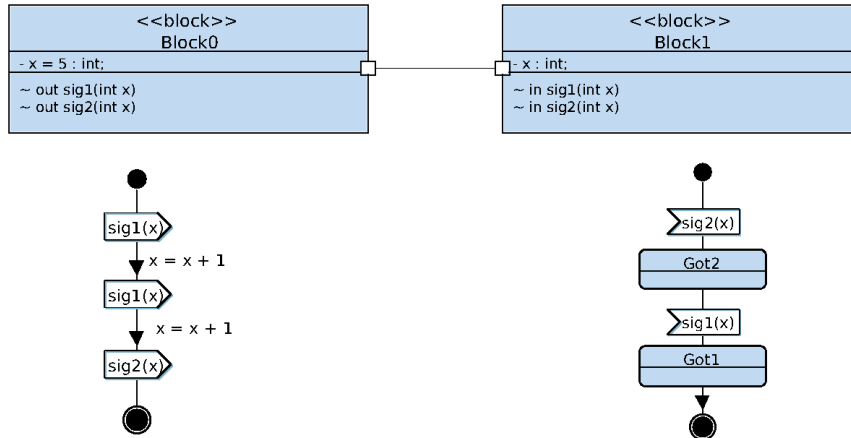
Synchronous Communications

- Sender and receiver synchronizes on the same signal
- Data exchange from the writer to the reader



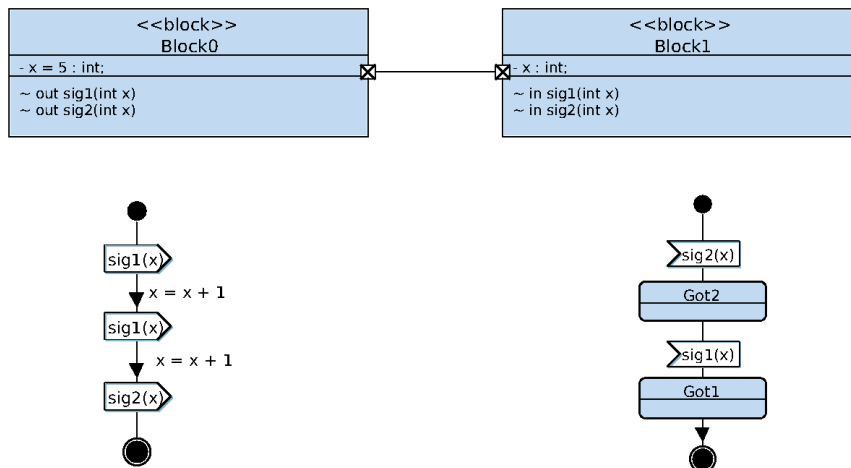
Non-Blocking Asynchronous Communications

- One FIFO per signal association
- Writing is **NOT** blocked when the FIFO is full
 - Bucket approach when FIFO is full: new messages are dropped
- Example: we assume a FIFO of size 1



Blocking Asynchronous Communications

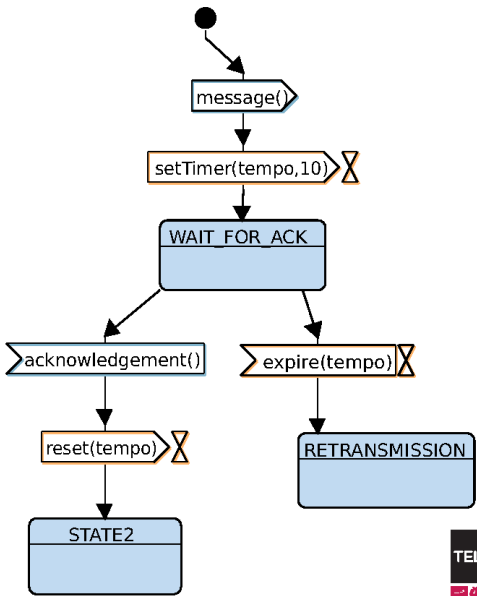
- One FIFO per signal association
- Writing is blocked when the FIFO is full
- Example: we assume a FIFO of size 1



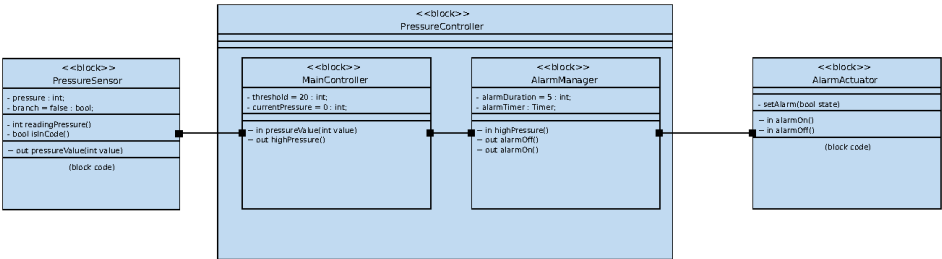
State Machines - Timers (3/3)

"Temporally limited acknowledgement" with timers
 A block instance may take decisions depending on the signal which arrives first: either a "normal" signal or a timer expiration

Question
 Could we use an *after* clause instead of the *tempo* timer?



Pressure Controller



How to Make "Good" Models?

Practice, Practice and Practice!!!

- Knowledge of various diagrams capabilities
- Accurate understanding of the system to model
- "Reading" your diagrams, reading diagrams of your friends, reading diagrams on Internet
- **Experience is a key factor**

→ **Make exercises!**

