Systeme hoher Sicherheit und Qualität
Universität Bremen, WS 2017/2018

# Lecture 05:

# High-Level Design with SysML

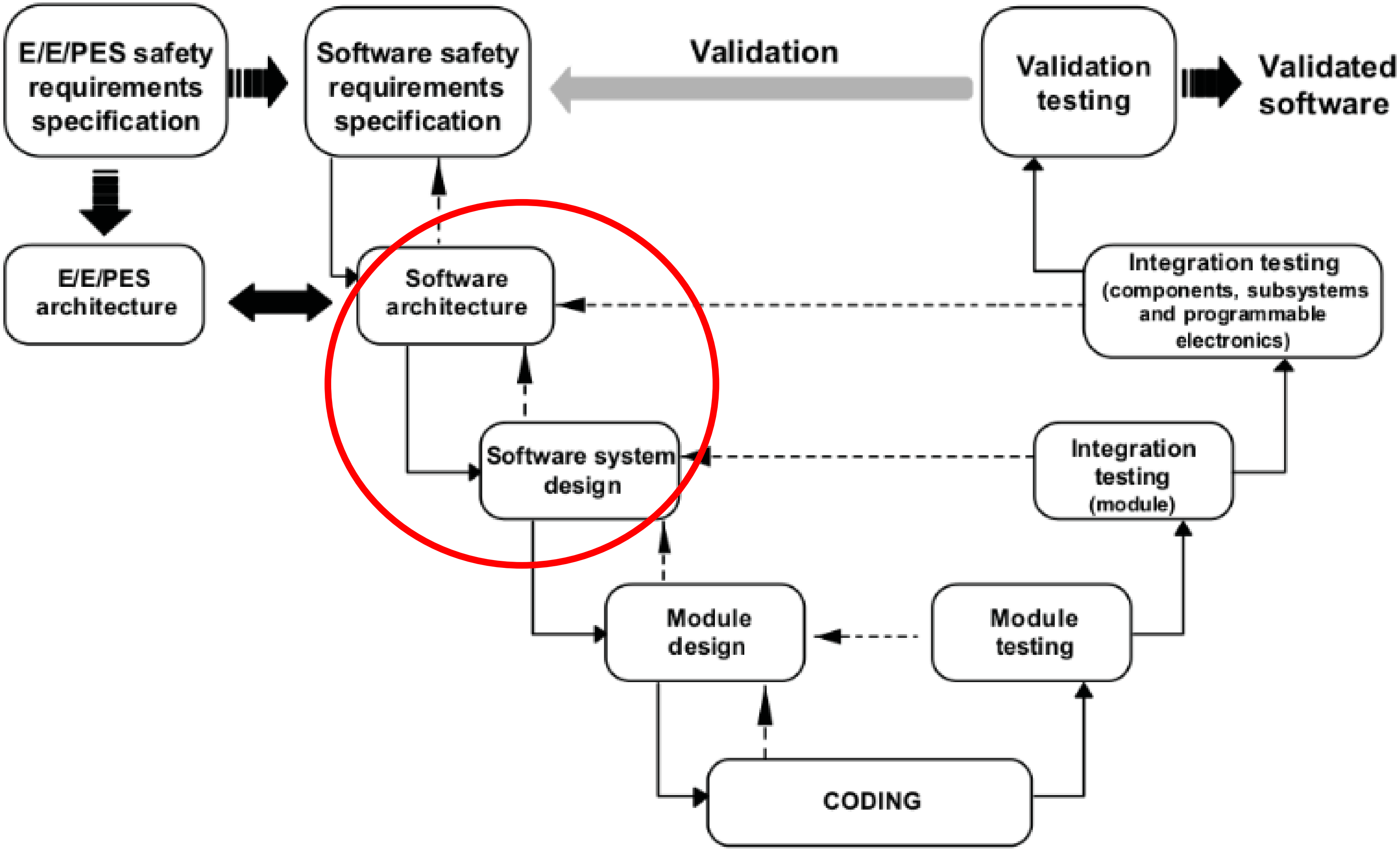
Christoph Lüth, Dieter Hutter, Jan Peleska

Universität Bremen

# Where are we?

- 01: Concepts of Quality
- 02: Legal Requirements: Norms and Standards
- 03: The Software Development Process
- 04: Hazard Analysis
- 05: High-Level Design with SysML
- 06: Formal Modelling with OCL
- 07: Testing
- 08: Static Program Analysis
- 09-10: Software Verification
- 11-12: Model Checking
- 13: Conclusions

# High-Level Design in the Development Cycle



# What is a model?

A model is a representation in a certain medium of something in the same or another medium.
The model captures the important aspects of the thing being modelled from a certain point of view and simplifies or omits the rest.

Rumbaugh, Jacobson, Booch: UML Reference Manual.

- Different notions of models in physics, philosophy or computer science
- Here: an abstraction of a system / a software / a development
- Purposes of models:
  - Understanding, communicating and capturing the design
  - Organizing decisions / information about a system
  - Analyzing design decisions early in the development process
  - Analyzing requirements

# An Introduction to SysML

# The Unified Modeling Language (UML)

- Grew out of a wealth of modelling languages in the 1990s (James Rumbaugh, Grady Booch and Ivar Jacobson at Rational)
- Adopted by the Object Management Group (OMG) in 1997, and approved as ISO standard in 2005.
- UML 2 consists of
  - the superstructure to define diagrams,
  - a core meta-model,
  - the object constraint language (OCL),
  - an interchange format
- UML 2 is not a fixed language, it can be extended and customized using profiles.
- SysML is a ***modeling language*** for systems engineering
- Standardized in 2007 by the OMG (May 2017 at Ver 1.5)
- Standard available at: http://www.omg.org/spec/SysML/About-SysML/

# What for SysML?

- Serving as a standardized notation allowing all stakeholders to understand and communicate the salient aspects of the system under development
  - the requirements,
  - the structure (static aspects), and
  - the behavior (dynamic aspects)

- Certain aspects (diagrams) of the SysML are **formal**, others are **informal**
  - Important distinction when developing critical systems

- All diagrams are **views** of one underlying model

# Different Views in SysML

- Structure:
  - How is the system constructed?
    How does it decompose?
- Behaviour:
  - What can we observe?   Does it have a state?
- Requirements:
  - What are the requirements?   Are they met?
- Parametrization:
  - What are the constraints (physical/design)?
- … and possibly more.

# Example: A Cleaning Robot (HooverBot)

- Structure:
  - Has an engine, wheels (or tracks?), a vacuum cleaner, a control computer, a battery...
- Behaviour:
  - General: starts, then cleans until battery runs out, returns to charging station
  - Cleaning: moves in irregular pattern, avoids obstacle
- Requirements:
  - Must cover floor when possible, battery must last at least six hours, should never run out of battery, ...
- Constraints:
  - Can only clean up to 5 g, can not drive faster than 1m/s, laws concerning movement and trajectory, ...

# SysML Diagrams

*Requirement Diagram **

**Structural Diagrams**

- Package Diagram
- Block Definition Diagram
- Internal Block Diagram
- Parametric Diagram

**Behavioural Diagrams**

- *Use Case Diagram **
- Activity Diagram
- State Machine Diagram
- Sequence Diagram

* Not considered further.

# Structural Diagrams in SysML

# Block Definition Diagram

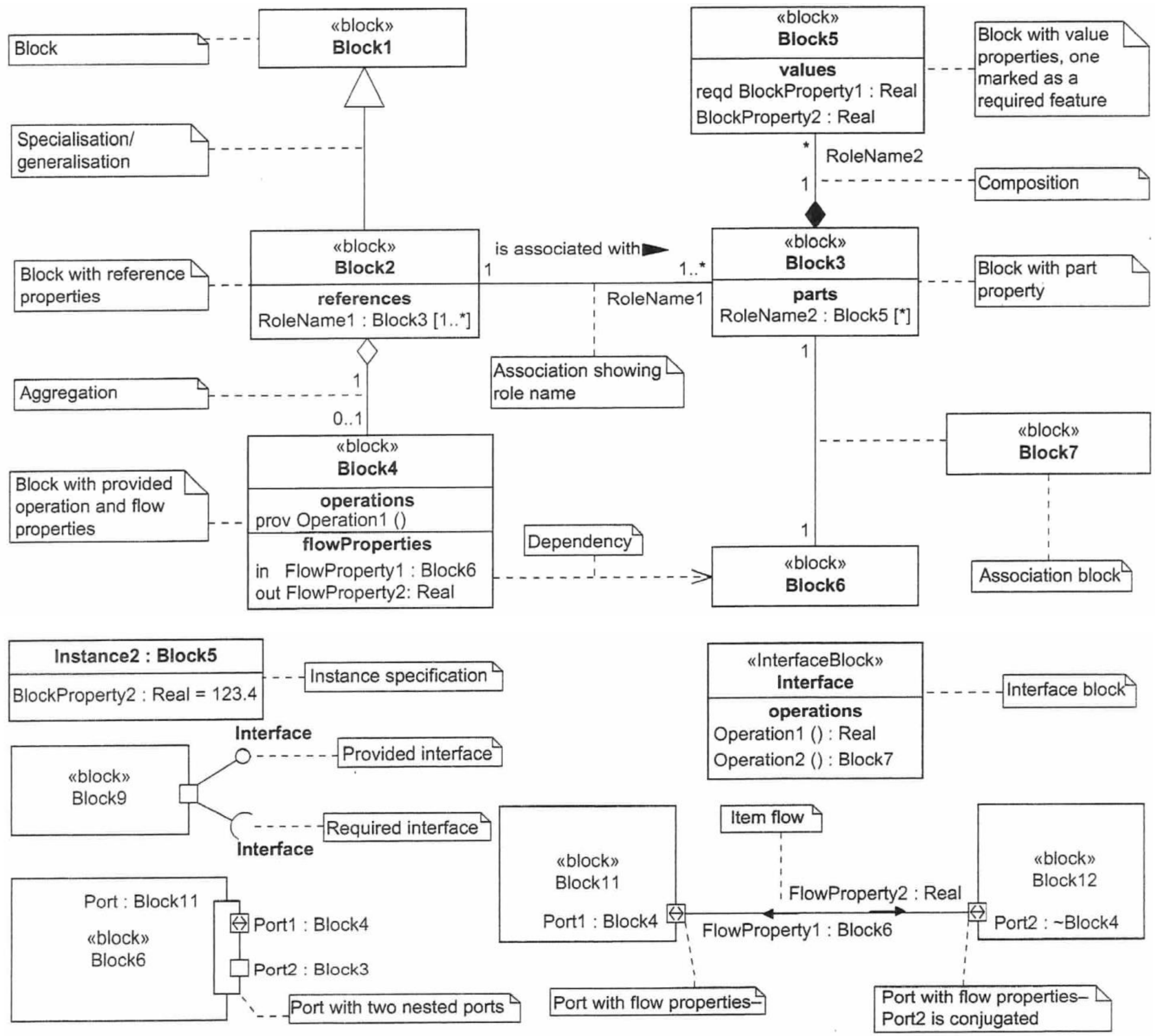
- Blocks are the **basic building elements** of a model
  - Models are *instances* of blocks

- Block definition diagrams model **blocks** and their **relations**:
  - Inheritance
  - Association

- Blocks can also model interface definitions.

- Corresponds to **class diagrams** in the UML.

# BDD – Summary of Notation

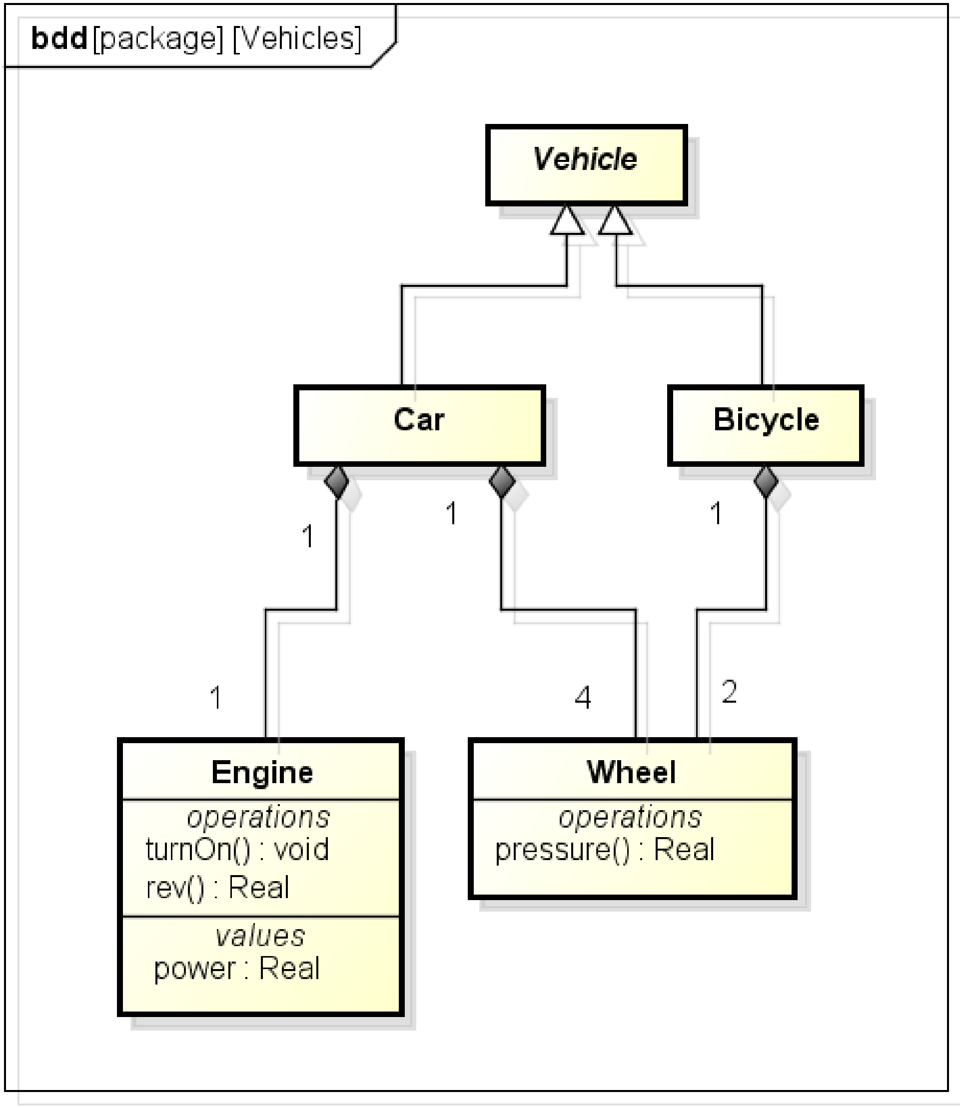



Quelle: Holt, Perry. SysML for Systems Engineering.

# Example 1: Vehicles

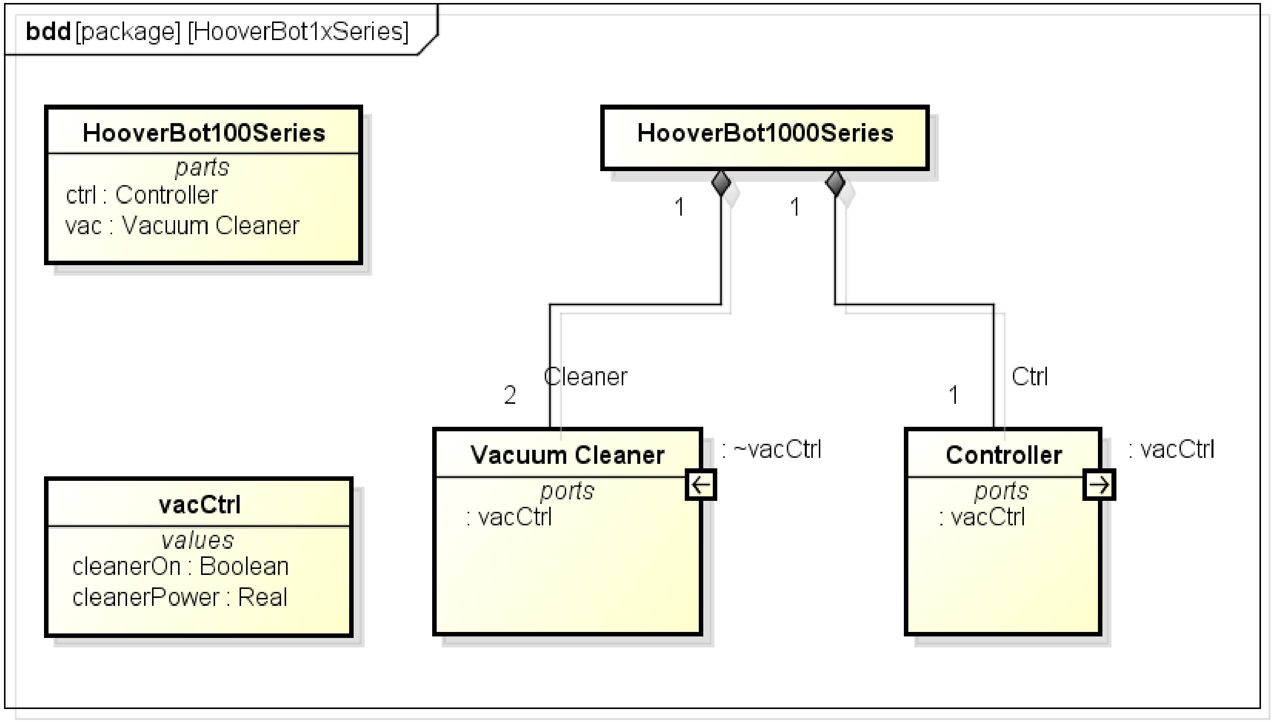
- A vehicle can be a car, or a bicycle.
- A car has an engine
- A car has 4 wheels,
  a bicycle has 2 wheels
- Engines and wheels have operations and values

- In SysML, engine and wheel are **parts** of car and bicycle.



# Example 2: HooverBots

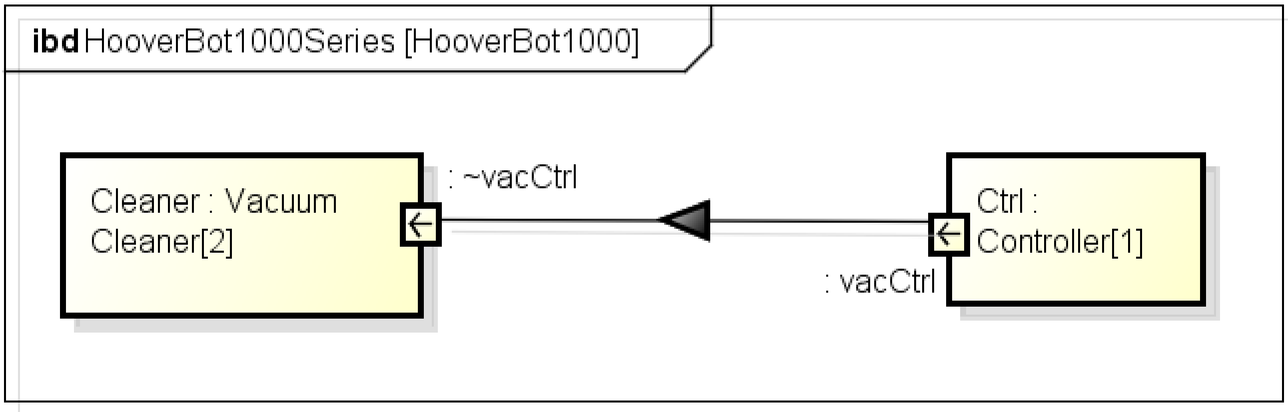
- The hoover bots have a control computer, and a vacuum cleaner (v/c).
  - HooverBot 100 has one v/c, Hoover 1000 has two.
  - Two ways to model this (i.e. two views):

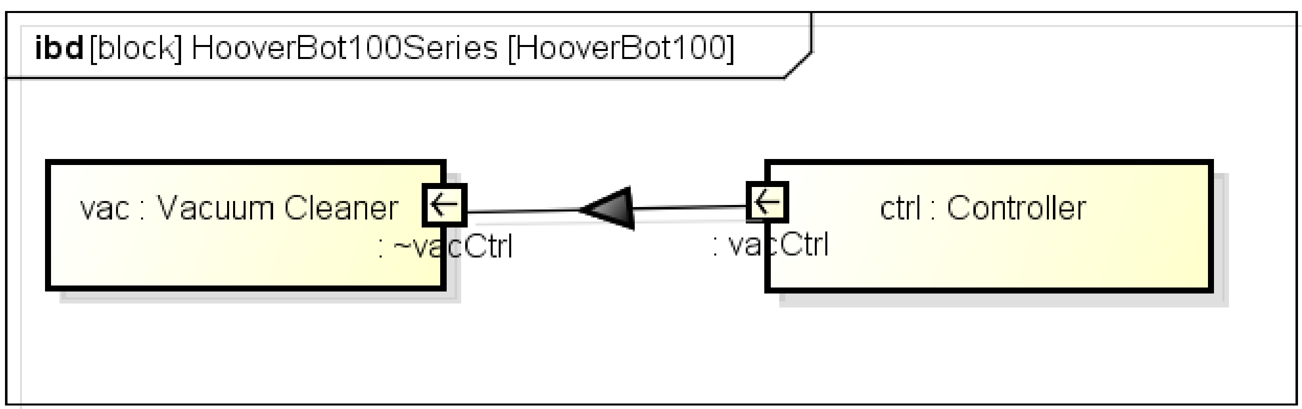


# Internal Block Diagrams

- Internal block diagrams decribe instances of blocks
- Here, instances for HooverBots
- On this level, we can describe connections between ports (flow specifications)
  - Flow specifications have directions.

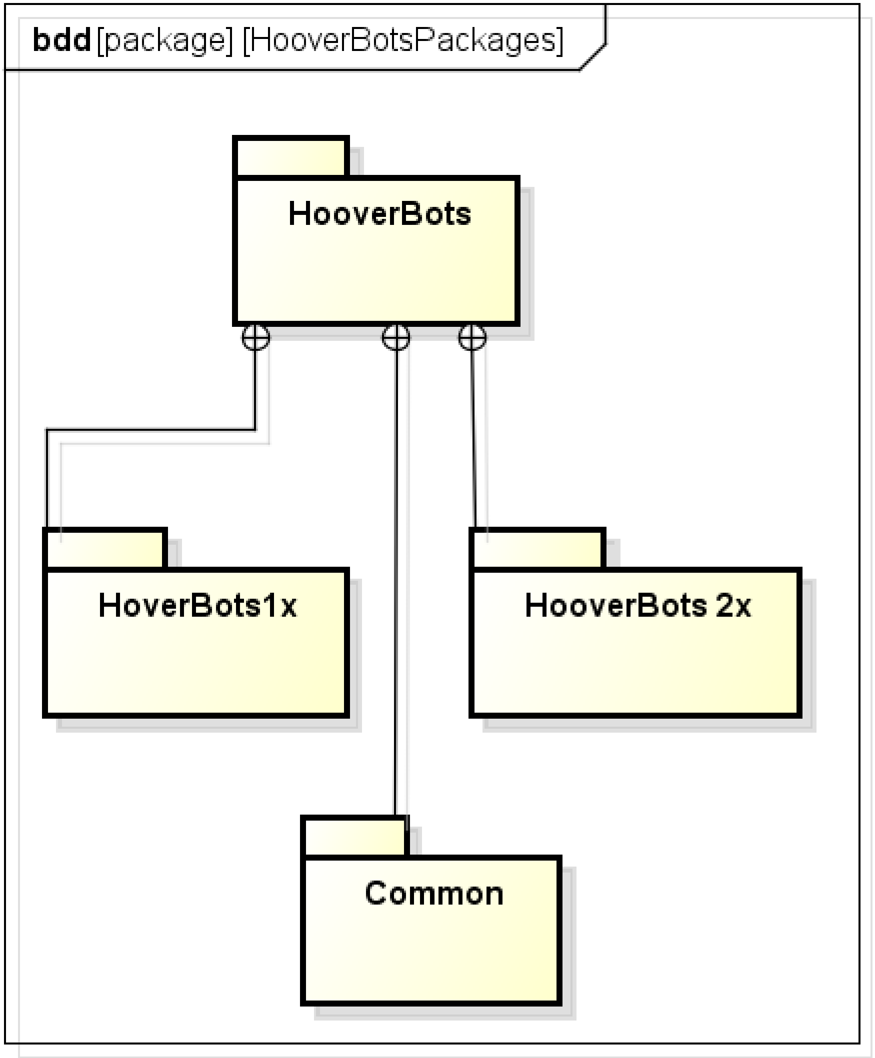
# Example: HooverBot 100 and 1000



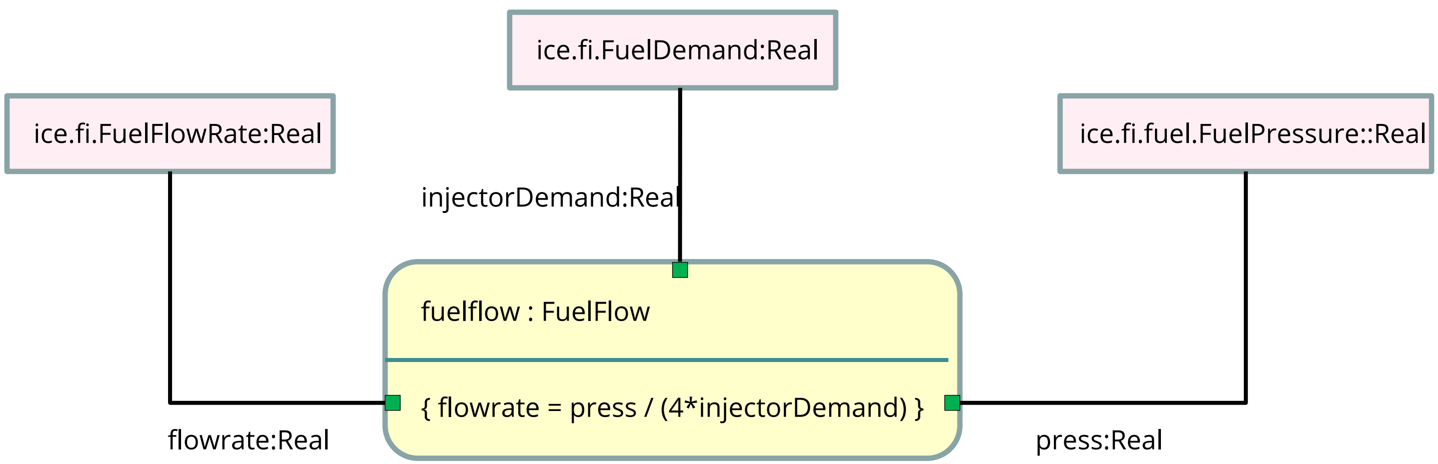
# Package Diagrams

- Packages are used to group diagrams, much like directories in the file system.
- Not considered much in the following



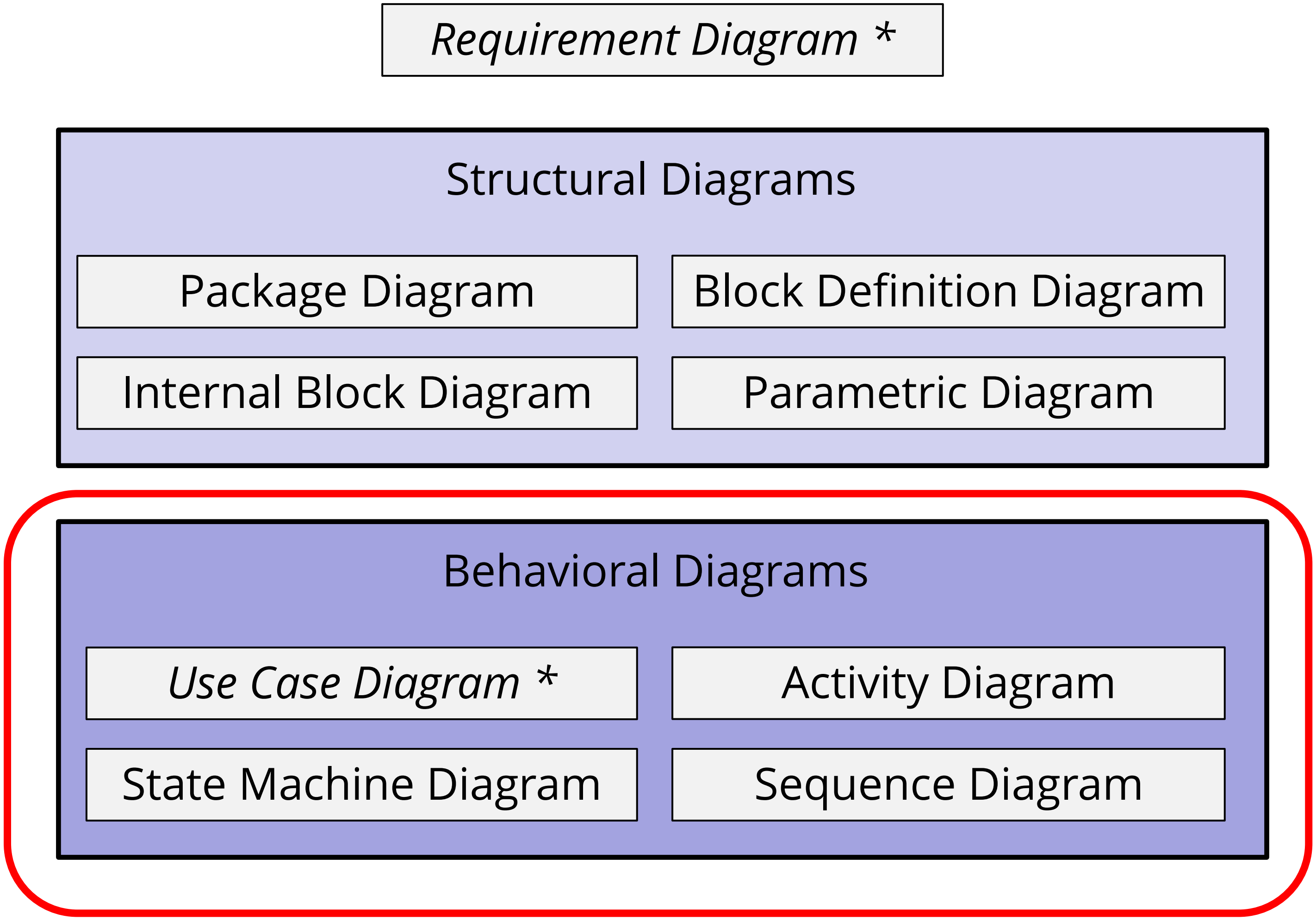
# Parametric Diagrams

- Parametric diagrams describe constraints between properties and their parameters.
- It can be seen as a restricted form of an internal block diagram, or as equational modeling as in Simulink.



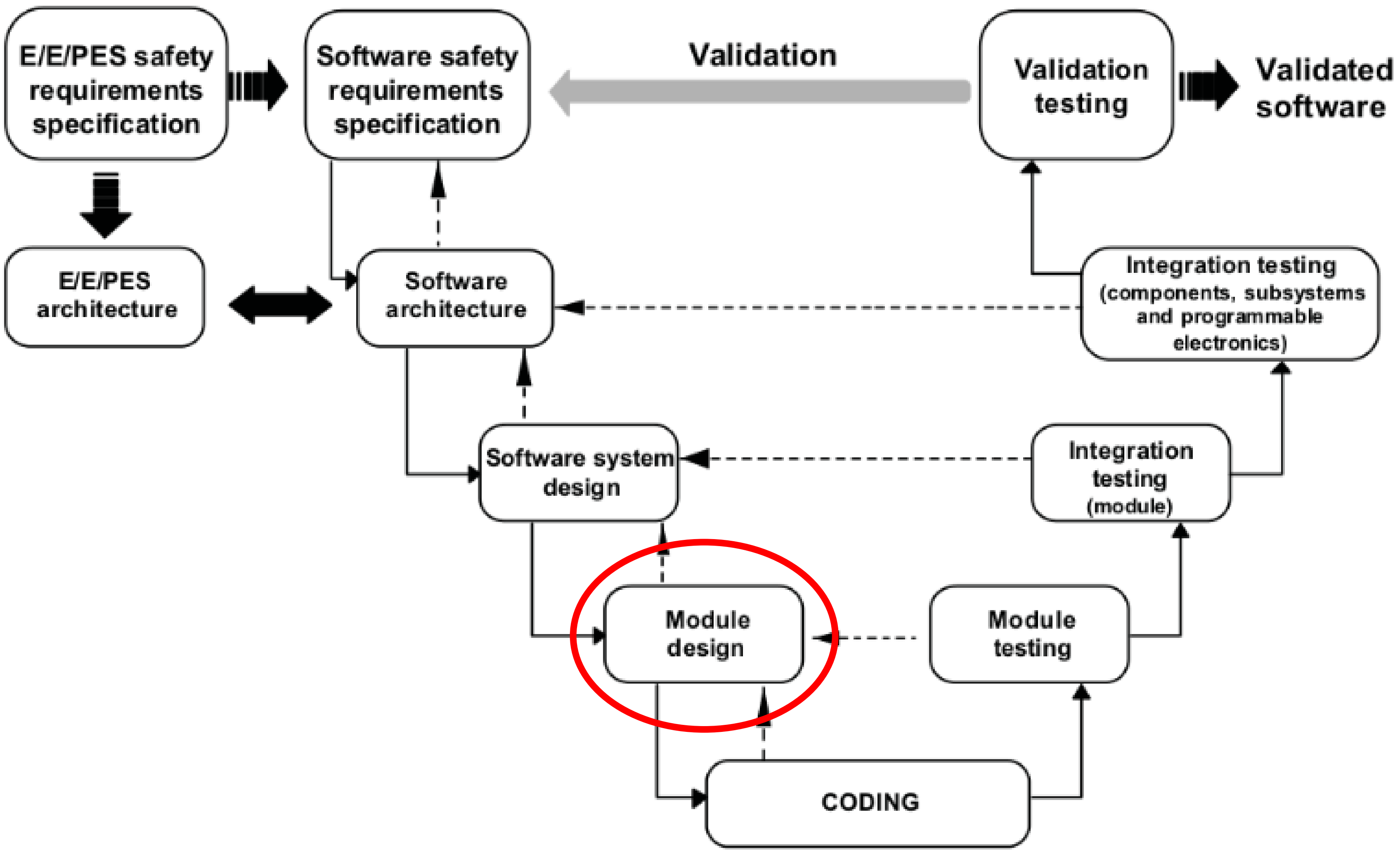

Relation of fuel flowrate to FuelDemand and FuelPressure value properties (Source: OMG SysML v1.2)

# SysML Diagrams Overview




* Not considered further.

# Detailed Specification in the Development Cycle



# Why detailed Specification?

- **Detailed specification** is the specification of single modules making up our system.
- This is the „last“ level both in abstraction and detail before we get down to the code – in fact, some specifications at this level can be automatically translated into code.
- Why **not** write code straight away?
  - We want to stay platform-independent.
  - We may not want to get distracted by details of our target platform.
  - At this level, we have a better chance of finding errors or proving safety properties.

# Levels of Detailed Specification

We can specify the basic modules

- By their (external) **behavior**
  - Operations defined by their pre/post-conditions and effects (e.g. in OCL)
  - Modeling the system‘s internal states by a state machine (i.e. states and guarded transitions)
- By their (internal) **structure**
  - Modeling the control flow by flow charts (aka. activity charts)
  - By action languages (platform-independent programming languages) for UML
    (but these are not standard for SysML)

# State Diagrams: Basics

- State diagrams are a particular form of (hierarchical) FSMs:

**Definition: Finite State Machine (FSM)**

A FSM is given by $\mathcal{M} = \langle \Sigma, I, \rightarrow \rangle$ where

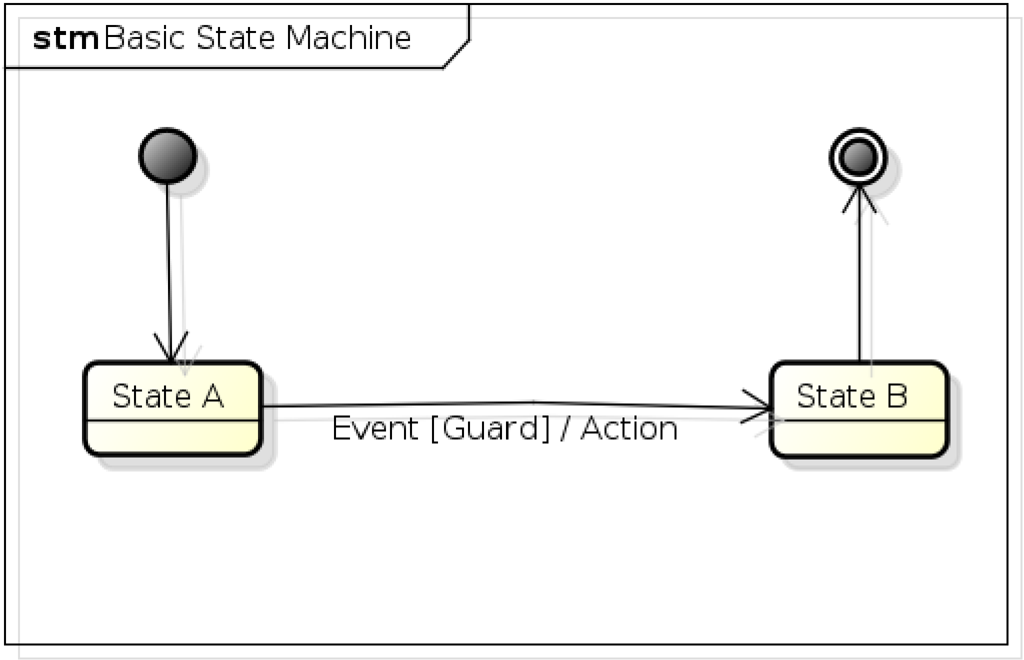
- $\Sigma$ is a finite set of **states**,
- $I \subseteq \Sigma$ is a set of **initial** states, and
- $\rightarrow \subseteq \Sigma \times \Sigma$ is a **transition relation**, s.t. $\rightarrow$ is left-total:

$$\forall s \in \Sigma. \exists s' \in \Sigma. s \rightarrow s'$$

- Example: a simple coffee machine.
- We will explore FSMs in detail later.
- In hierarchical state machines, a state may contain another FSM (with initial/final states).
- State Diagrams in SysML are taken unchanged from UML.

# Basic Elements of State Diagrams

- States
  - Initial/Final
- Transitions
- Events (Triggers)
- Guards
- Actions (Effects)

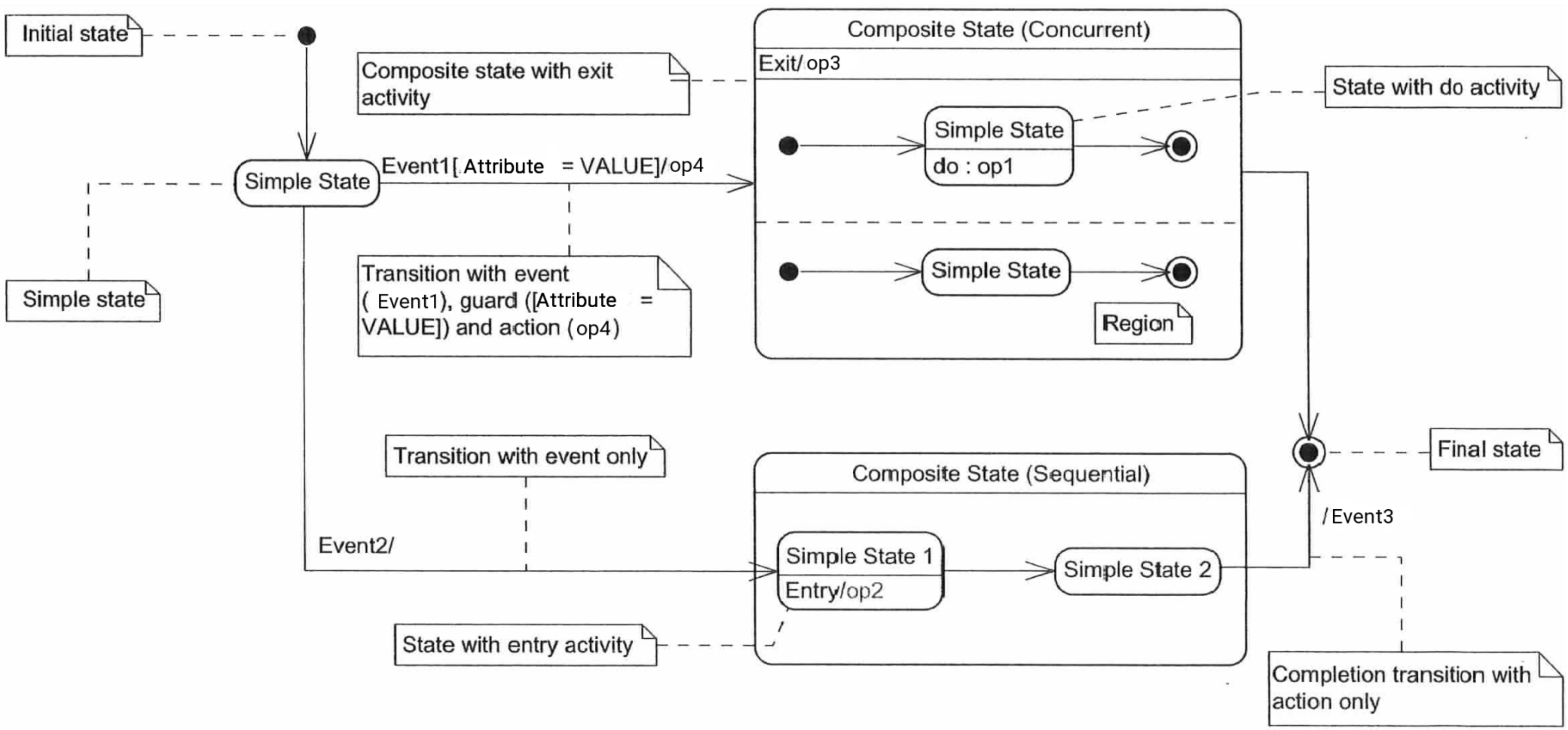


# What is an Event?

- *„The specification of a noteworthy occurence which has a location in time and space."* (UML Reference Manual)

- SysML knows:
  - Signal events **`event name/`**
  - Call events **`operation name/`**
  - Time events **`after(t)/`**
  - Change event **`when(e)/`**
  - Entry events **`Entry/`**
  - Exit events **`Exit/`**

# SMDs – Summary of Notation



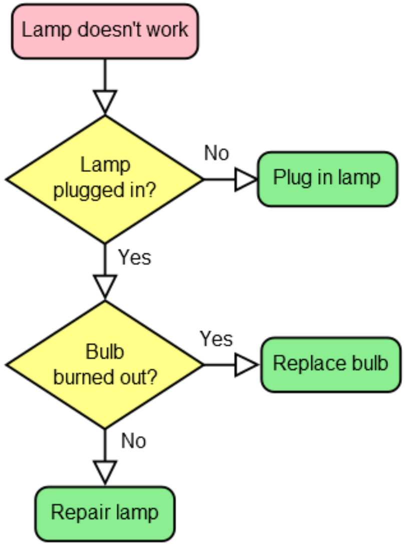

Quelle: Holt, Perry. SysML for Systems Engineering.

# State Diagram Elements (SysML Ref. §13.2)

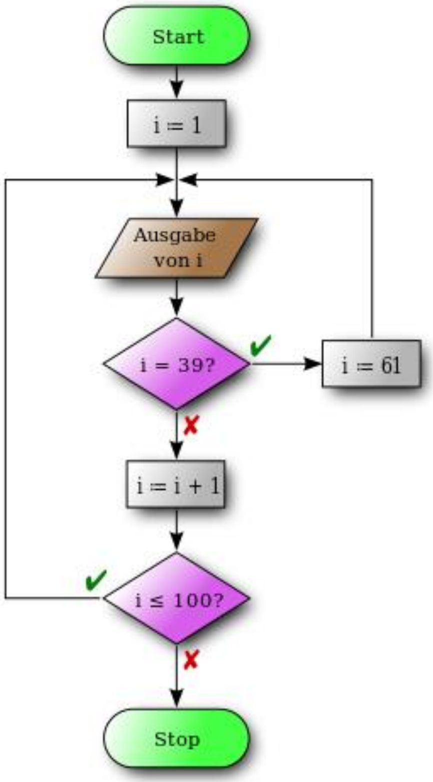
- Choice pseudo state
- Composite state
- Entry point
- Exit point
- Final state
- *History pseudo states*
- Initial pseudo state
- Junction pseudo state
- Receive signal action
- Send signal action
- Action
- Region
- Simple state
- State list
- State machine
- Terminate node
- Submachine state

# Activity Charts: Foundations

- The activity charts of SysML (UML) are a variation of good old-fashioned **flow charts.**
  - Those were standardized as DIN 66001 (ISO 5807).
- Flow charts can describe programs (right example) or non-computational activities (left example)
- SysML activity charts are extensions of UML activity charts.




Quelle: Wikipedia




Quelle: Erik Streb, via Wikipedia

# Basics of Activity Diagrams

- Activities model the work flow of low-level behaviours:
  "*An activity is  the specification of parameterized behaviour as the coordinated sequencing of subordinate unites whose individual elements are actions.*" (UML Ref. §12.3.4)

- Diagram comprises of actions, decisions, joining and forking activities, start/end of work flow.

- Control flow allows to disable and enable (sub-) activities.

- An activity execution results in the execution of a set of actions in some specific order.

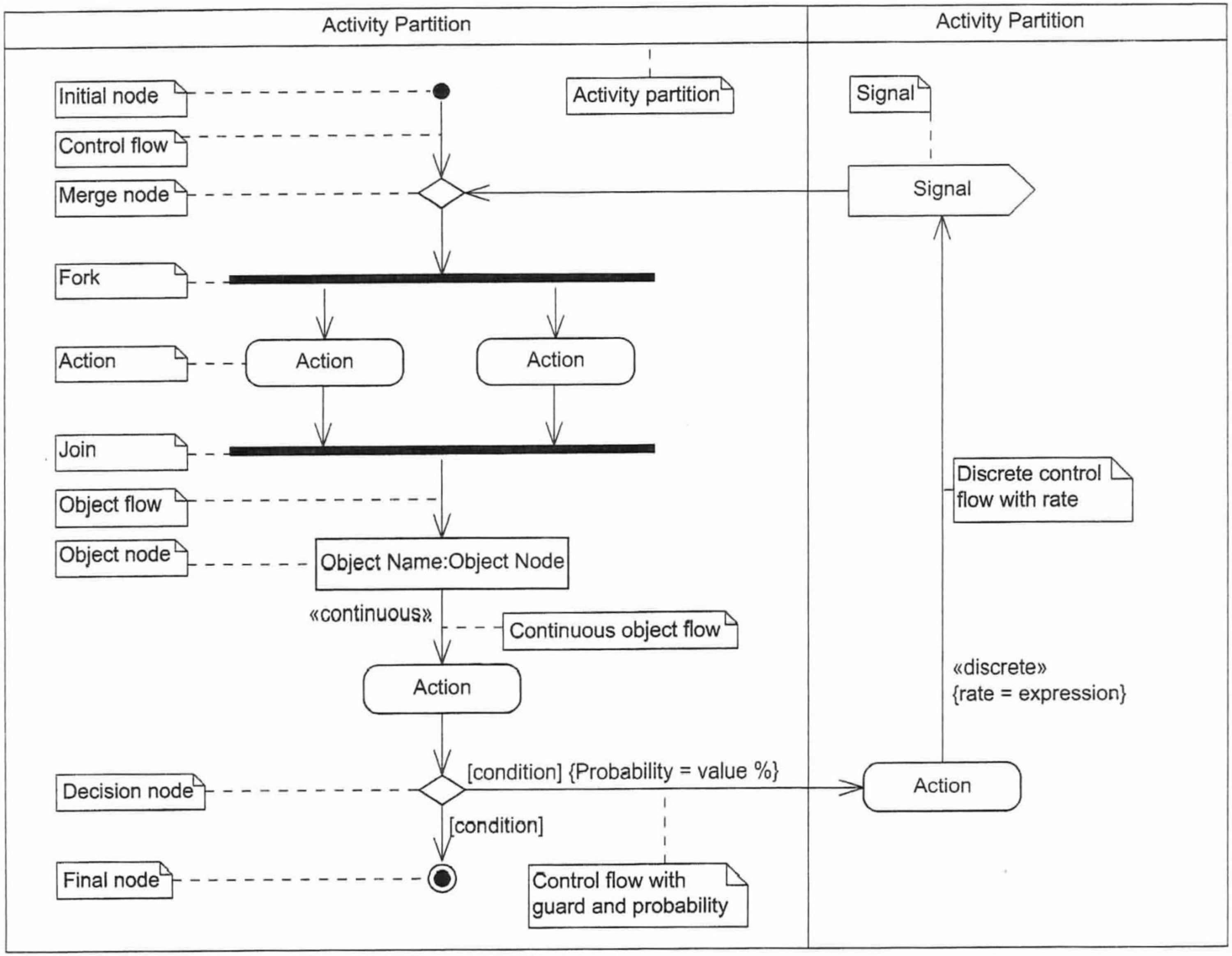
# What is an Action?

- A terminating basic behaviour, such as
  - Changing variable values [UML Ref. §11.3.6]
  - Calling operations [UML Ref. §11.3.10]
  - Calling activities [UML Ref. §12.3.4]
  - Creating and destroying objects, links, associations
  - Sending or receiving signals
  - Raising exceptions .
- Actions are part of a (potentially larger, more complex) behaviour.
- Inputs to actions are provided by ordered sets of pins:
  - A pin is a typed element, associated with a multiplicity
  - Input pins transport typed elements to an action
  - Actions deliver outputs consisting of typed elements on output pins

# Elements of Activity Diagrams

- Nodes:
  - Action nodes
  - Activities
  - Decision nodes
  - Final nodes
  - Fork nodes
  - Initial nodes
  - Local pre/post-conditions
  - Merge nodes
  - Object nodes
  - *Probabilities and rates*
- Paths (arrows):
  - Control flow
  - Object flow
  - Probability and rates
- Activities in BDDs
- Partitions
- Interruptible Regions
- Structured activities

# Activity Diagrams – Summary of Notation



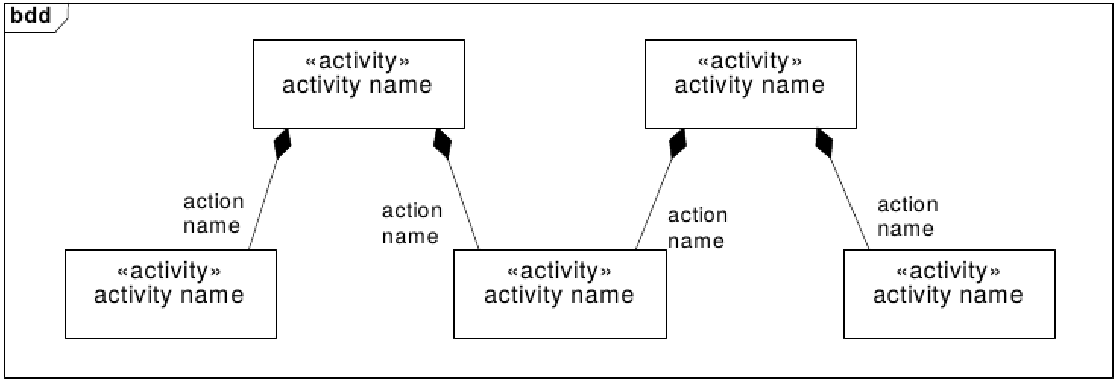

Quelle: Holt, Perry. SysML for Systems Engineering.

# Behavioural Semantics

- Semantics is based on **token flow** – similar to Petri Nets, see [UML Ref. pp. 326]
  - A token can be an input signal, timing condition, interrupt, object node (representing data), control command (call, enable) communicated via input pin, …
  - An executable node (action or sub-activity) in the activity diagram begins its execution, when the required tokens are available on their input edges.
  - On termination, each executable node places tokens on certain output edges, and this may activate the next executable nodes linked to these edges.

# Activity Diagrams – Links With BDDs

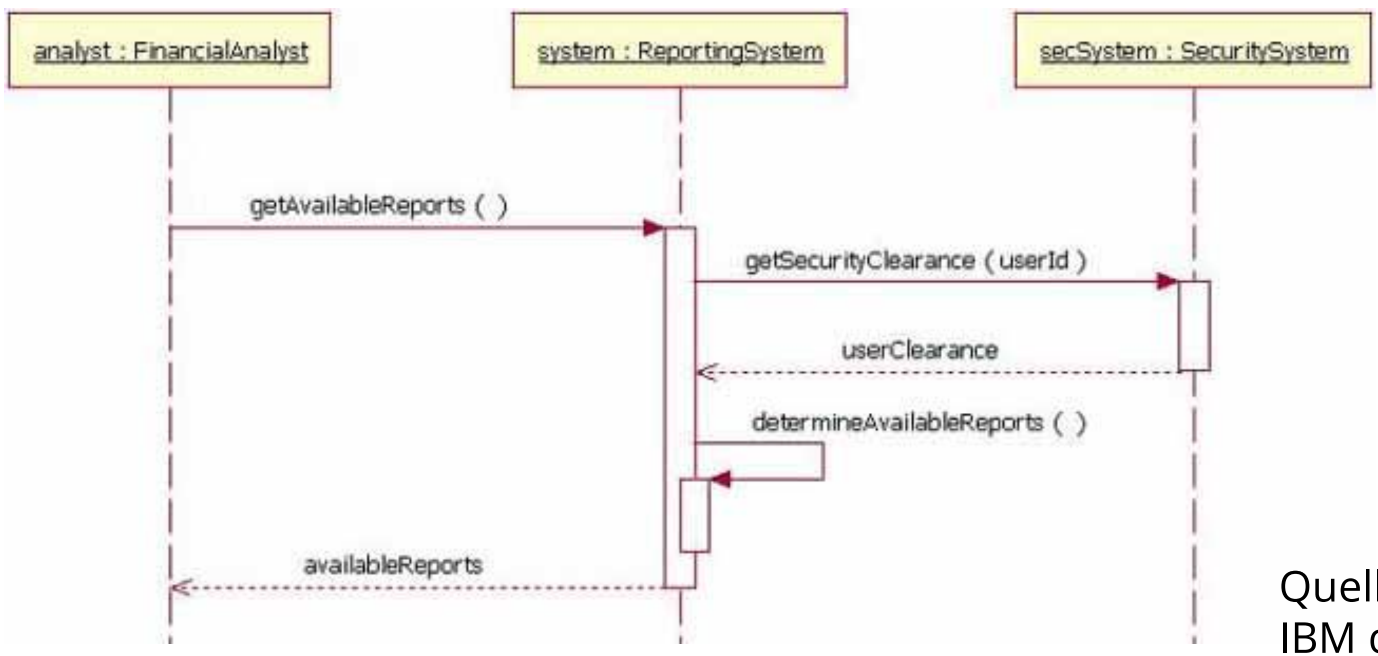
- Block definition diagrams may show
  - Blocks representing activities




  - One activity may be composed of other activities – composition indicates parallel execution threads of the activities at the “part end”.
  - One activity may contain several blocks representing **object nodes** (which represent data flowing through the activity diagram).

# Sequence Diagrams

- Sequence Diagrams describe the flow of messages between actors.
- Extremely useful, but also extremely limited.




Quelle:
IBM developerWorks

- We may consider concurrency further later on.

# Summary

- High-level modeling describes the structure of the system at an abstract level
- SysML is a standardized modeling language for systems engineering, based on the UML
  - We disregard certain aspects of SysML in this lecture
- SysML  structural diagrams describe this structure.
  - Block definition diagrams
  - Internal block definition diagrams
  - Package diagrams
- We may also need to describe formal constraints, or invariants.

# Summary (cont.)

- Detailed specification means we specify the internal structure of the modules in our systems.
- Detailed specification in SysML:
  - State diagrams are hierarchical finite state machines which specify states and transitions.
  - Activity charts model the control flow of the program.
- More behavioral diagrams in SysML:
  - Sequence charts model the exchange of messages between actors.
  - Use case diagrams describe particular uses of the system.