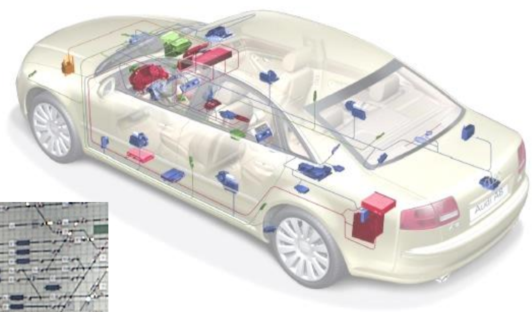
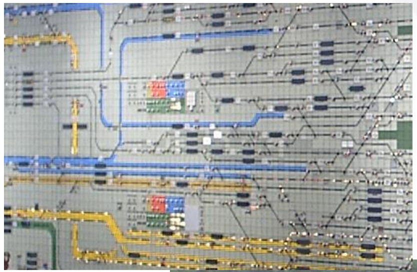
Modeling Requirements in SysML

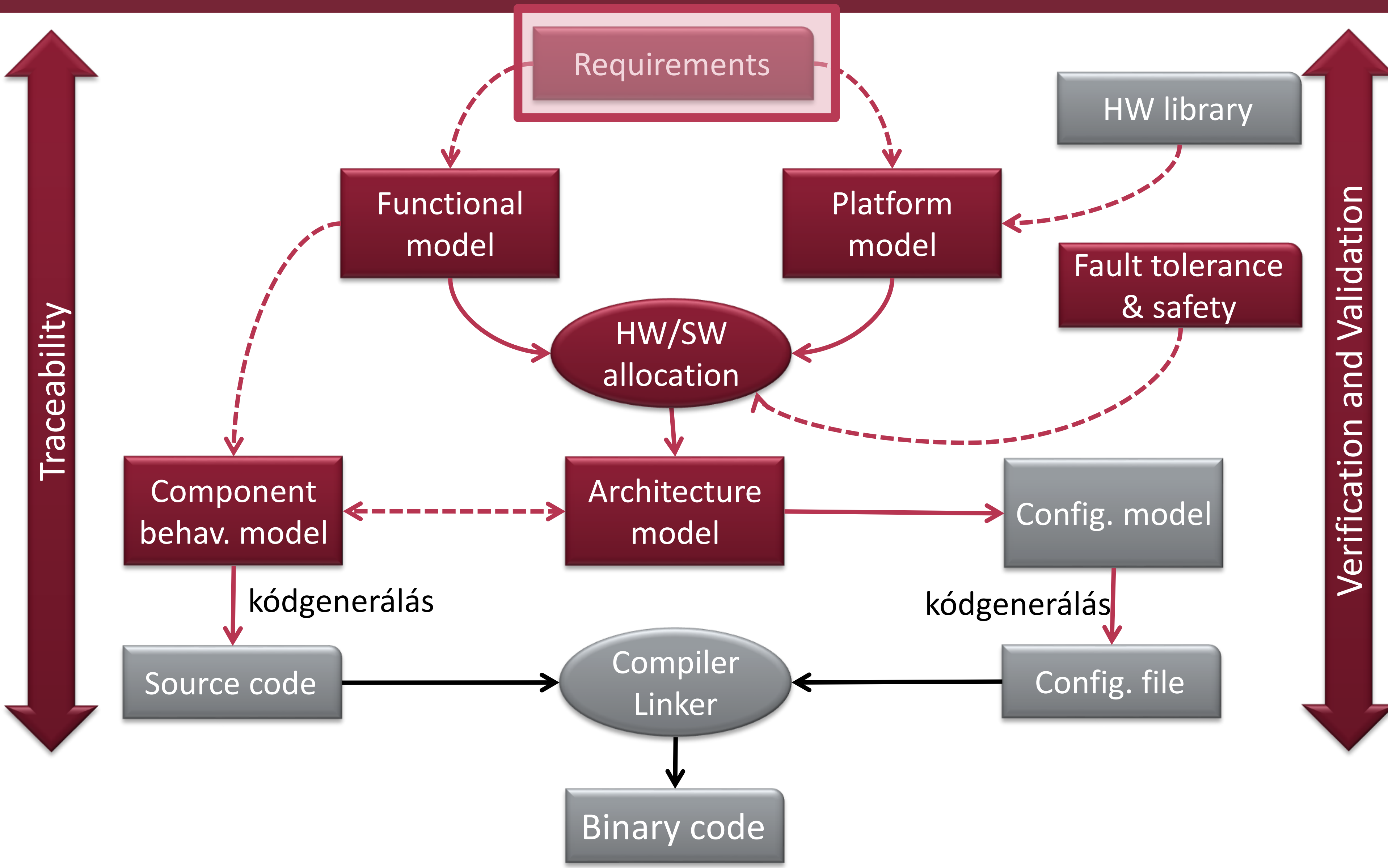
Systems Engineering

- Systems Engineering is a multidisciplinary approach to develop balanced system solutions in response to diverse stakeholder needs
- ~ Integration Engineering
  - Software engineering
  - Hardware engineering
  - Mechanical engineering
  - Safety engineering
  - Security engineering
  - ...
- ~ Process Engineering
- System
  - Military, airplane, car, aviation, railway interlocking, notebook, etc.





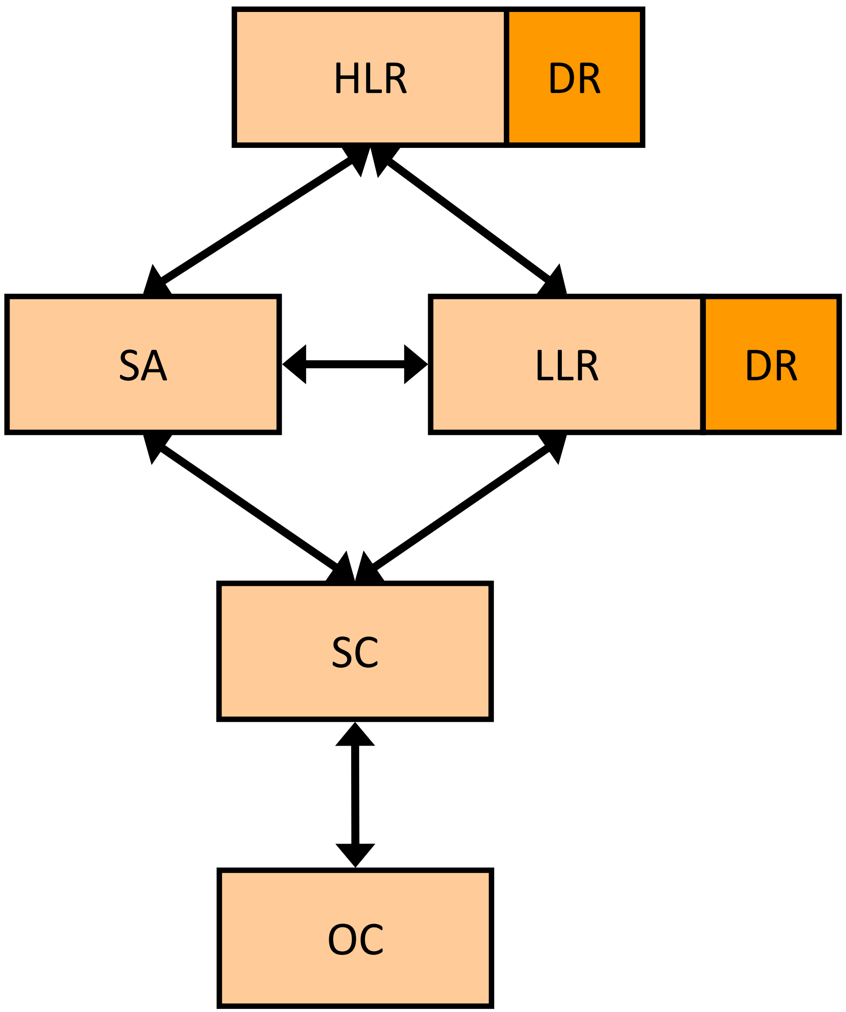
Platform-based systems design



Definition of a Requirement

- Definitions
  - A condition or capability a system must conform to (IBM Rational)
  - A statement of the functions required of the system (Mentor Graphics)
- Each requirements needs to be
  - **Identifiable + Unique**: unique IDs
  - **Consistent**: no contradiction
  - **Unambiguous**: one interpretation
  - **Verifiable**: e.g. testable to decide if met
- Captured with special statements and vocabulary

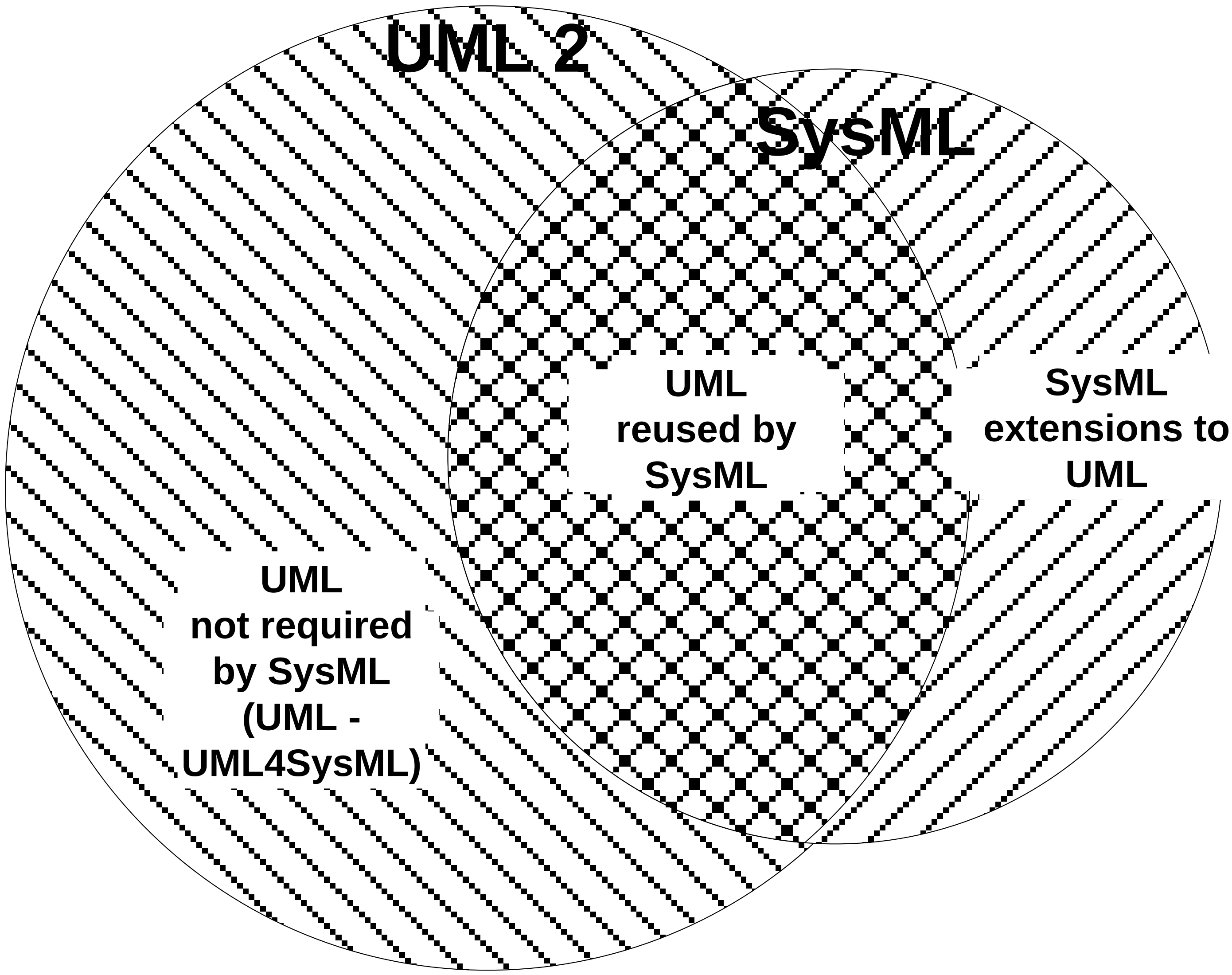
The Certification Perspective: High-level vs Low-Level



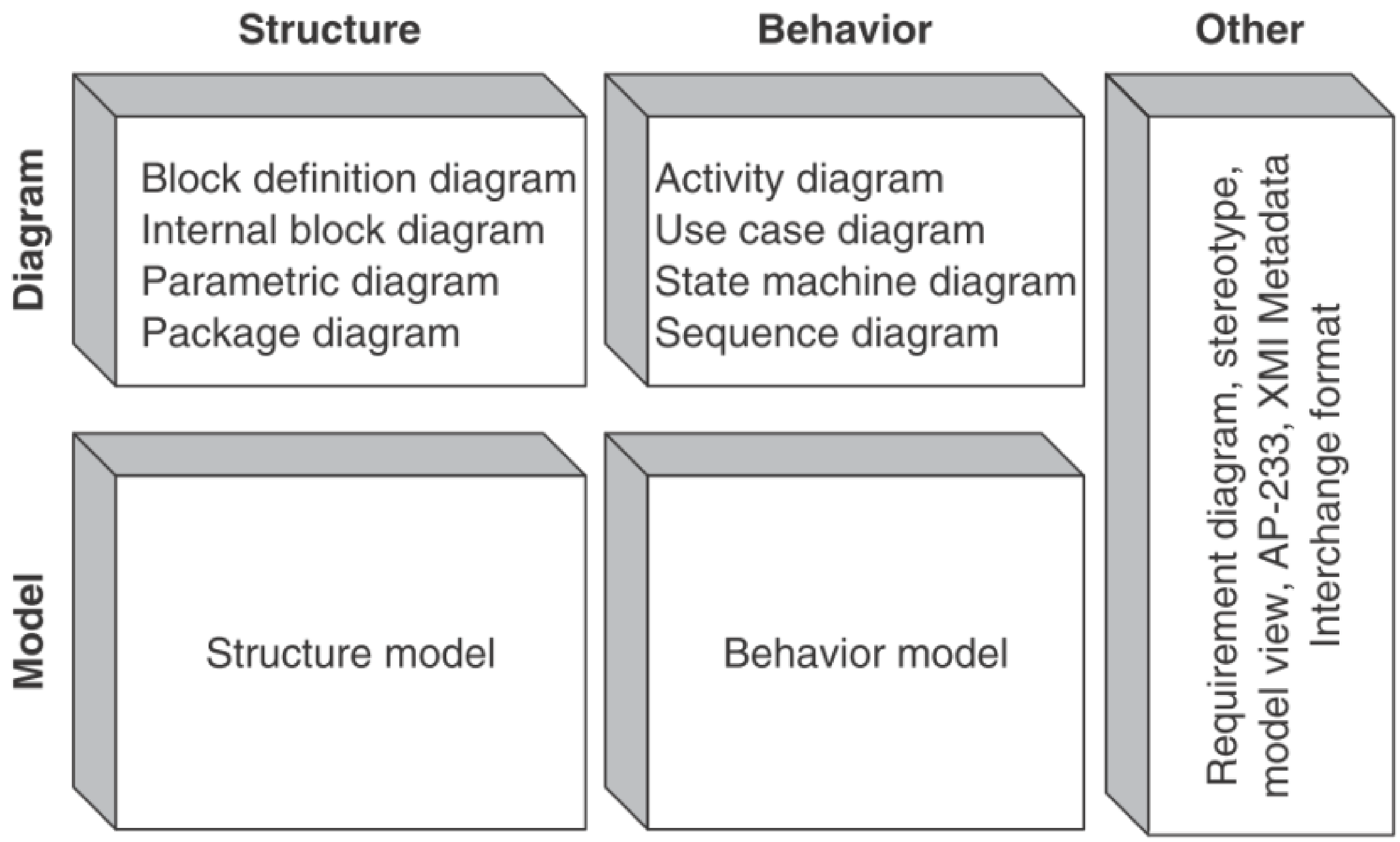
Concepts from DO-178C standard

- High Level Requirements (HLR):
  - customer-oriented
  - black-box view of the software,
  - captured in a natural language (e.g. using shall statements)
- Derived Requirements (DR)
  - Capture design decisions
- Low Level Requirements (LLR):
  - SC can be implemented without further information
- Software Architecture (SA)
  - Interfaces, information flow of SW components
- Source Code (SC)
- Executable Object Code (EOC)

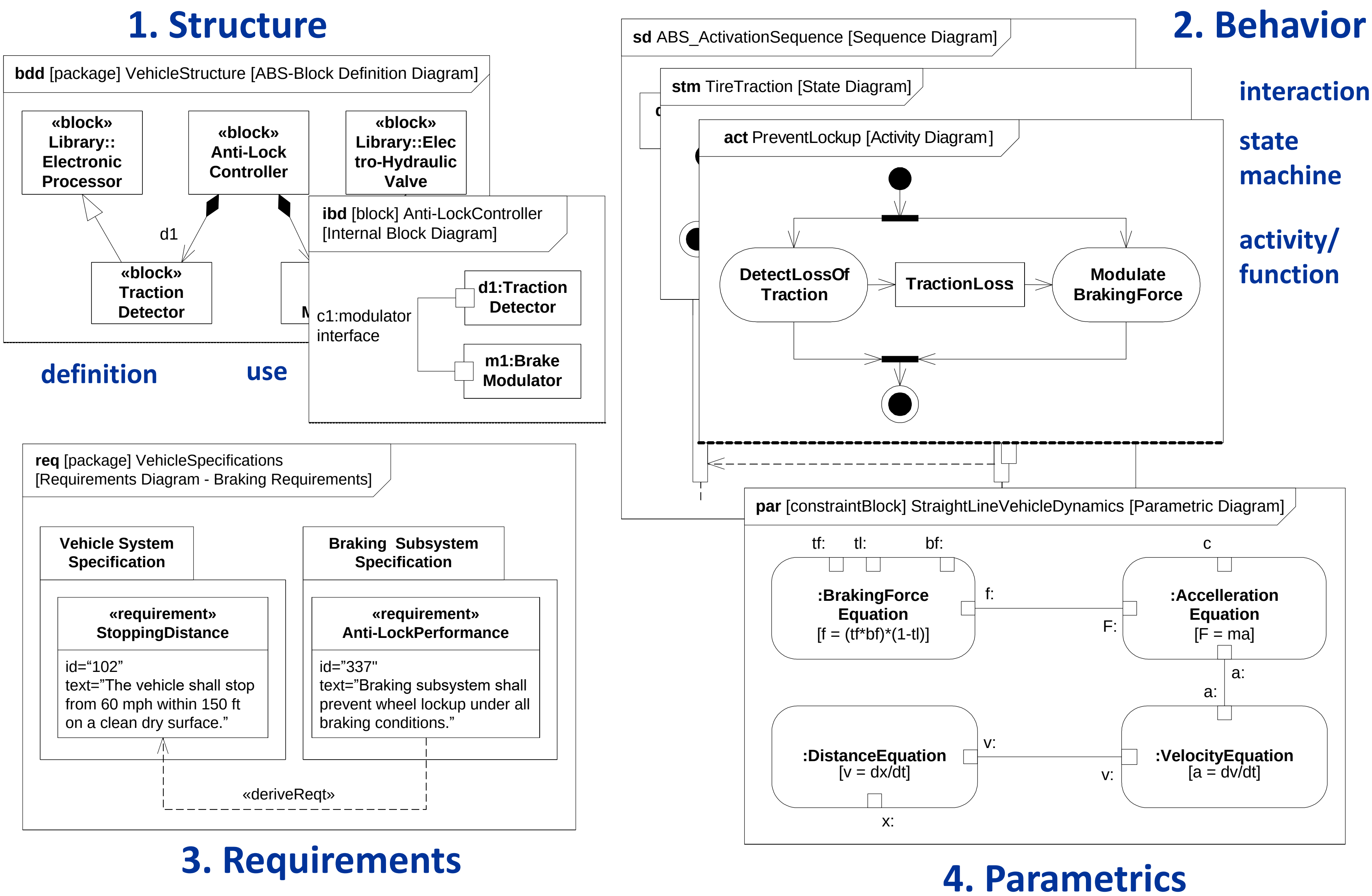
Relationship Between SysML and UML



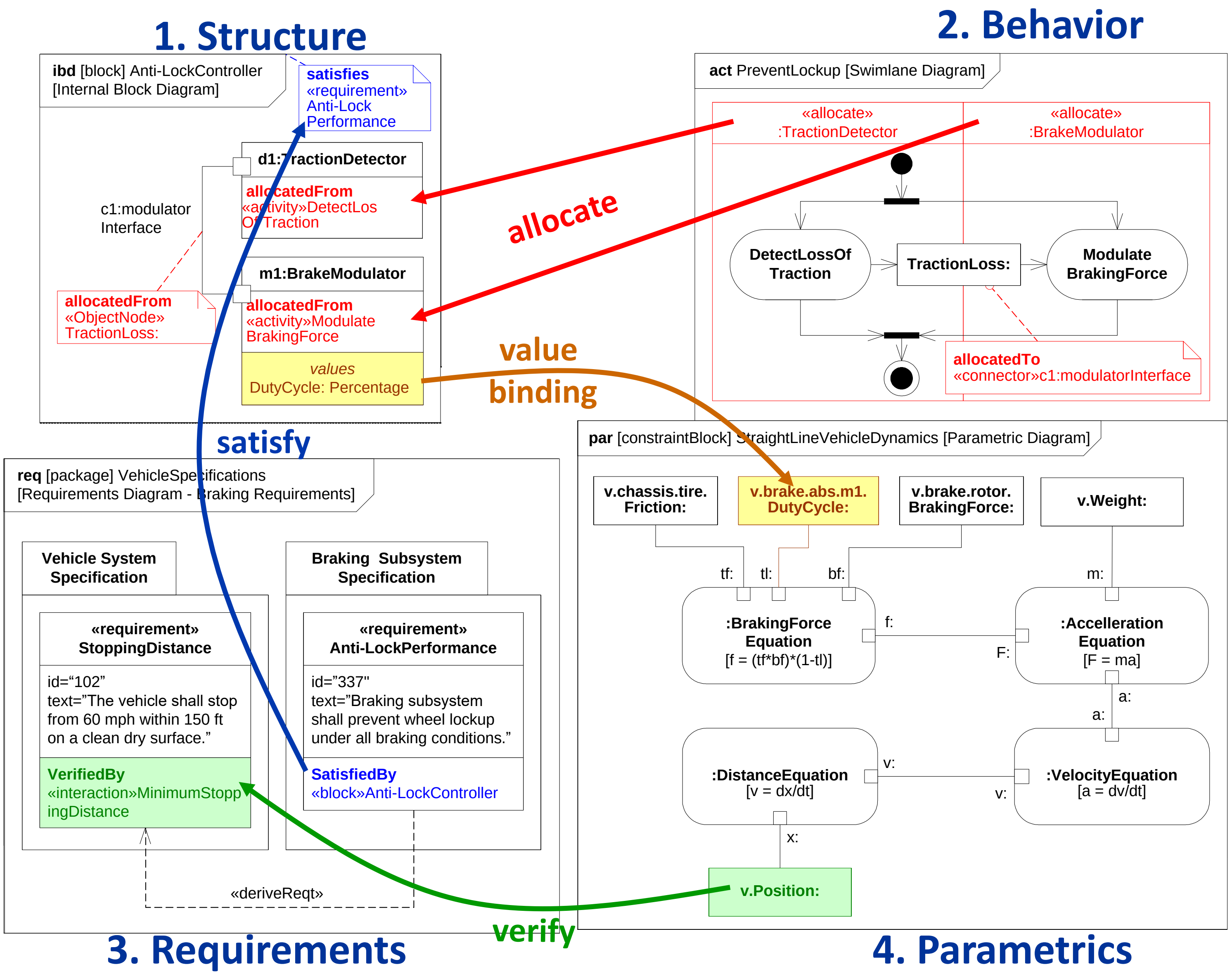
Aspects of SysML



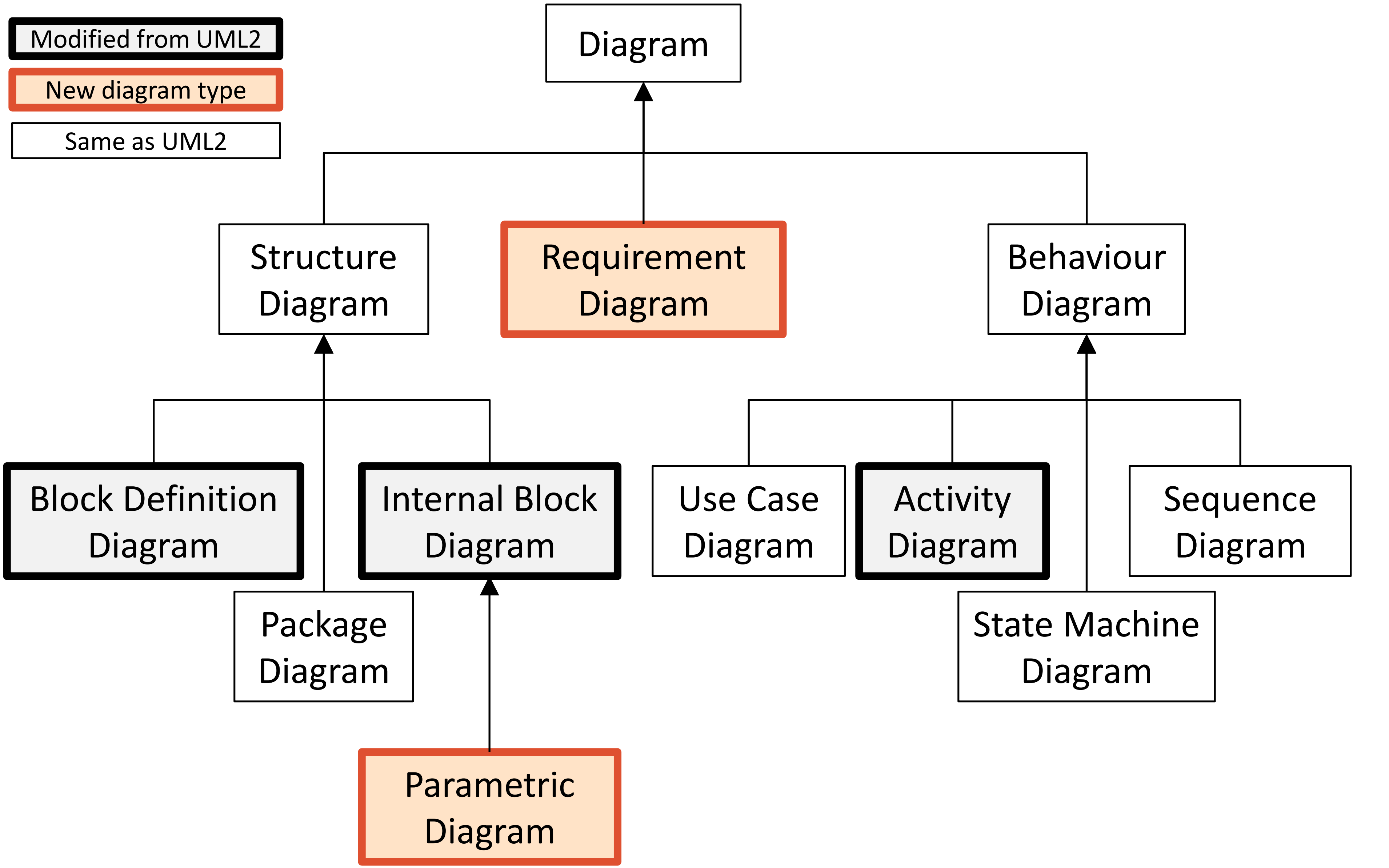
4 Pillars of SysML – ABS Example



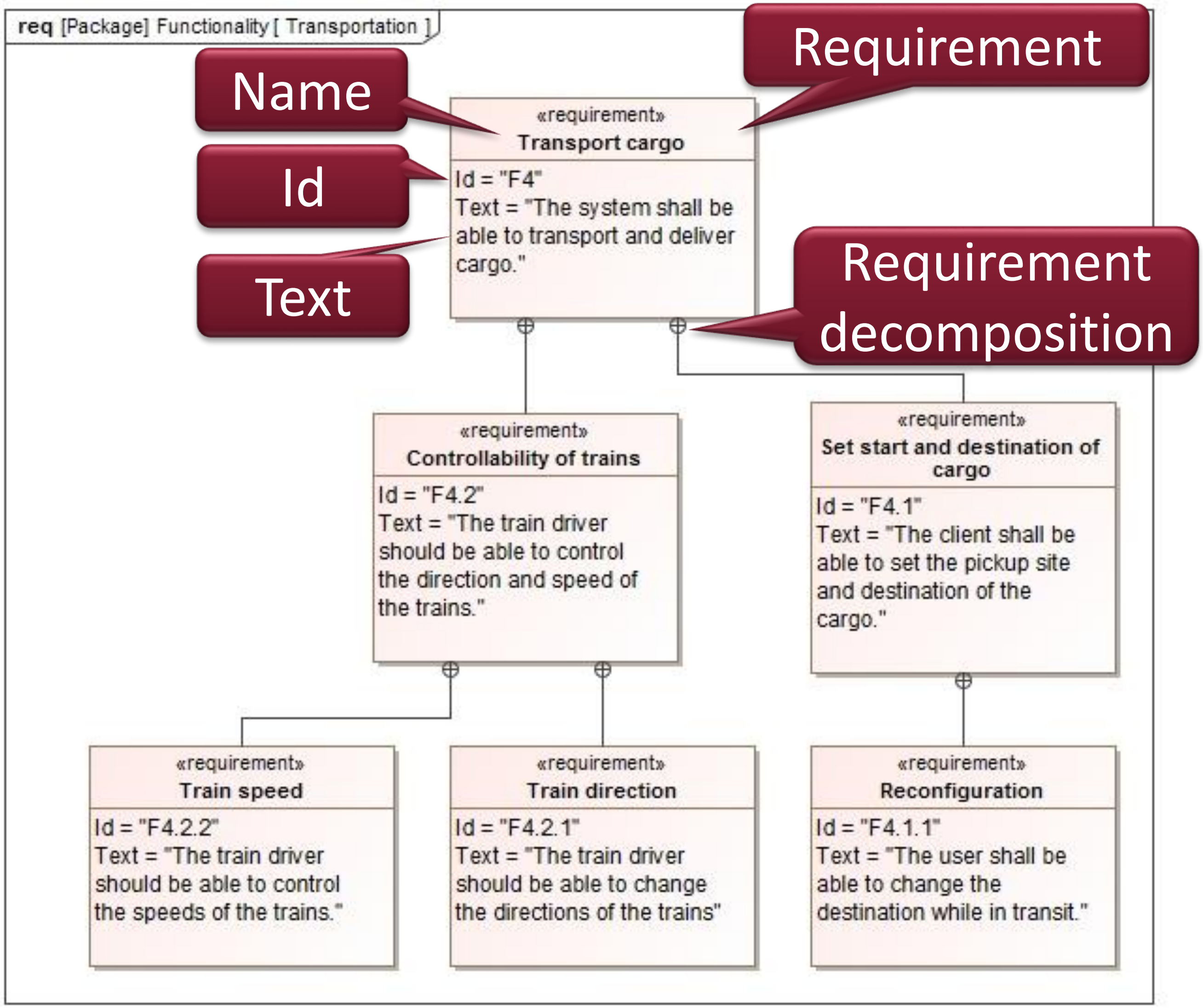
Cross Connecting Model Elements



SysML Diagram Taxonomy

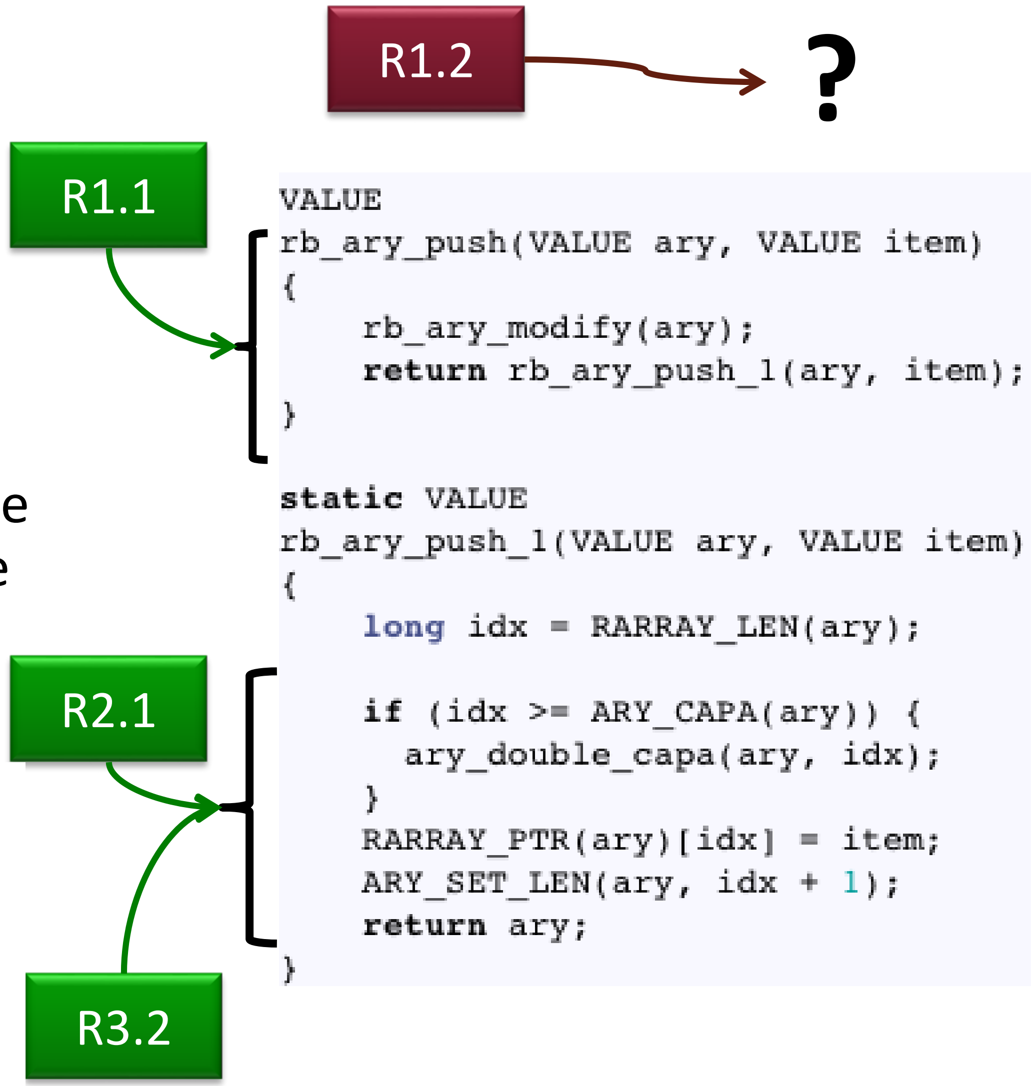


SysML Example – Requirements



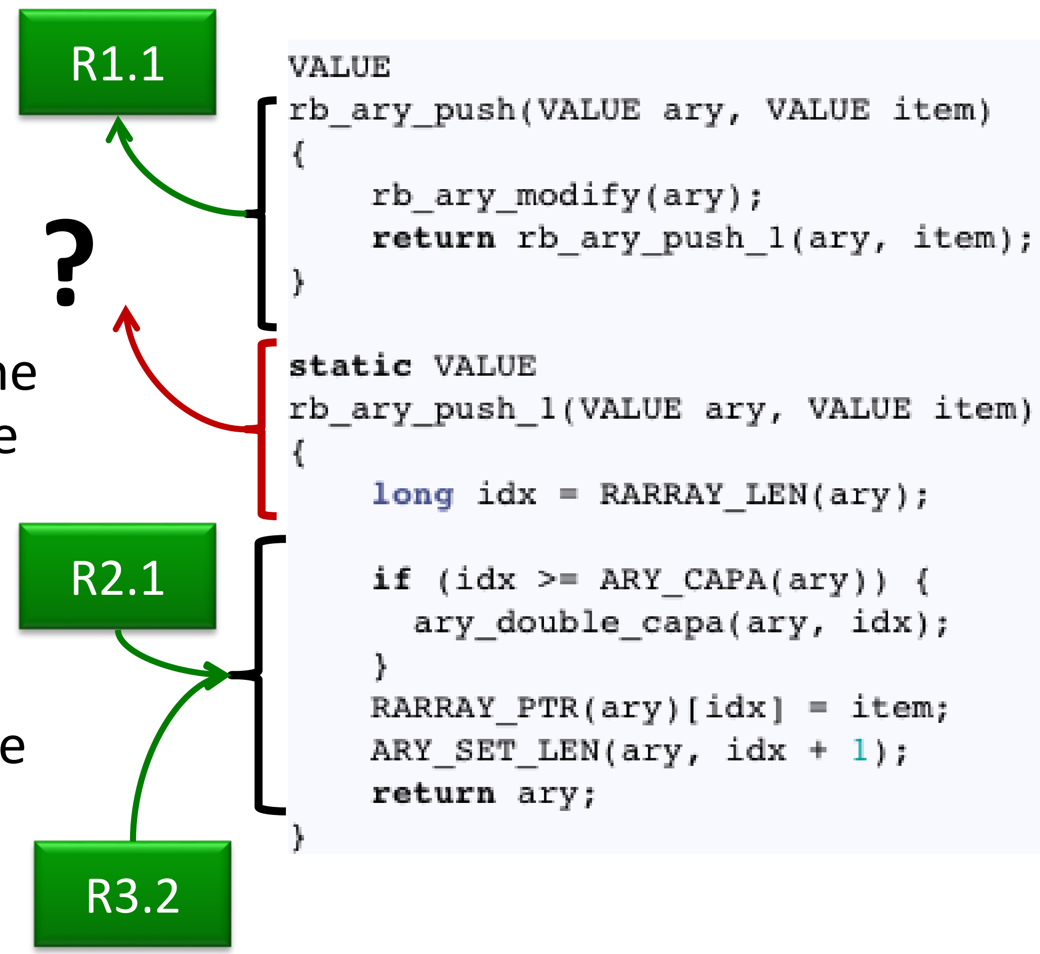
The Concept of Traceability

- Traceability is a core **certification concept**
  - For safety-critical systems
  - See safety standards (DO-178C, ISO 26262, EN 50126)
- **Forward traceability**:
  - From each requirement to the corresponding lines of source code (and object code)
  - Show responsibility



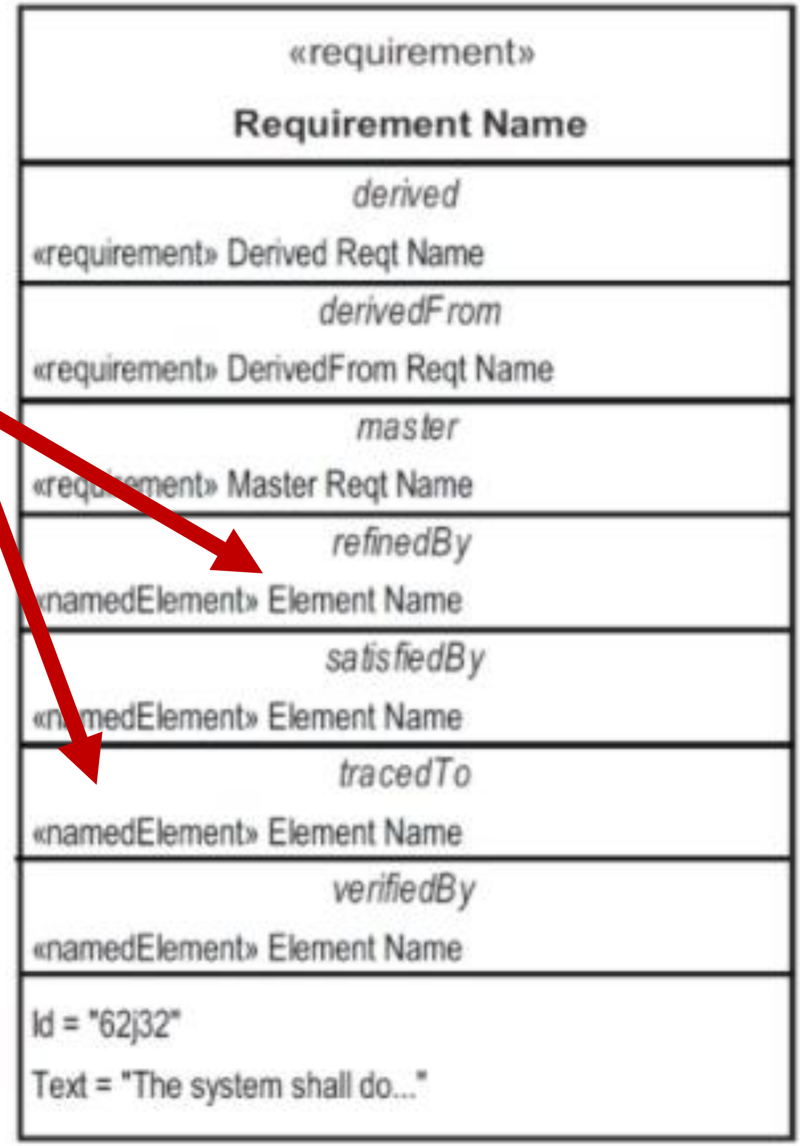
The Concept of Traceability

- Traceability is a core **certification concept**
  - For safety-critical systems
  - See safety standards (DO-178C, ISO 26262, EN 50126)
- **Forward traceability**:
  - From each requirement to the corresponding lines of source code (and object code)
  - Show responsibility
- **Backward traceability**:
  - From any lines of source code to one ore more corresponding requirements
  - No extra functionality

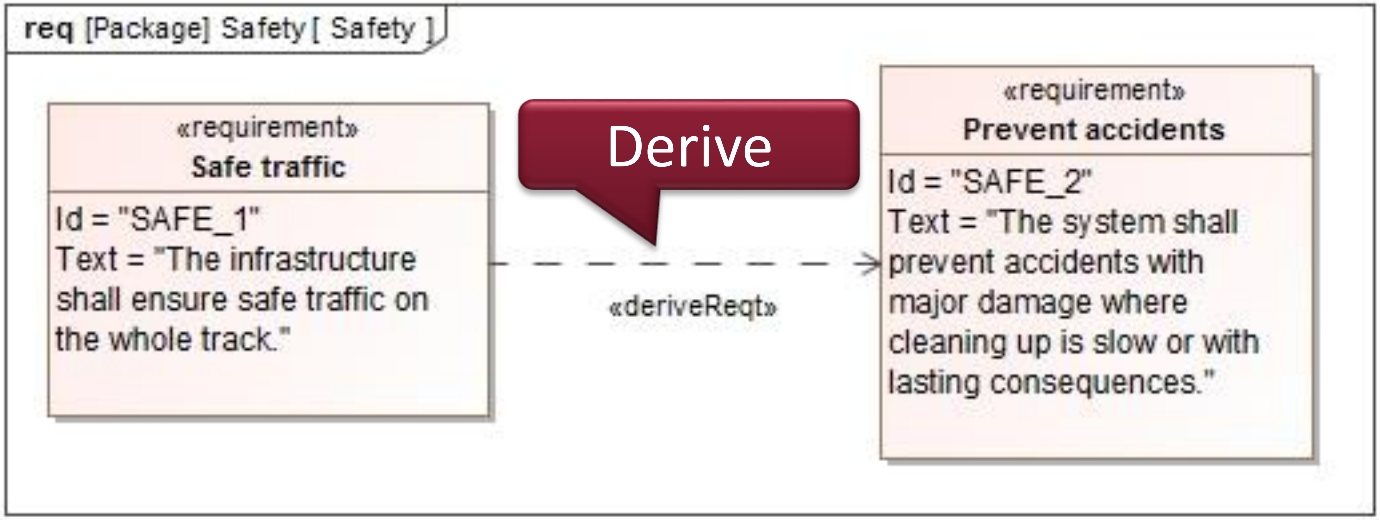


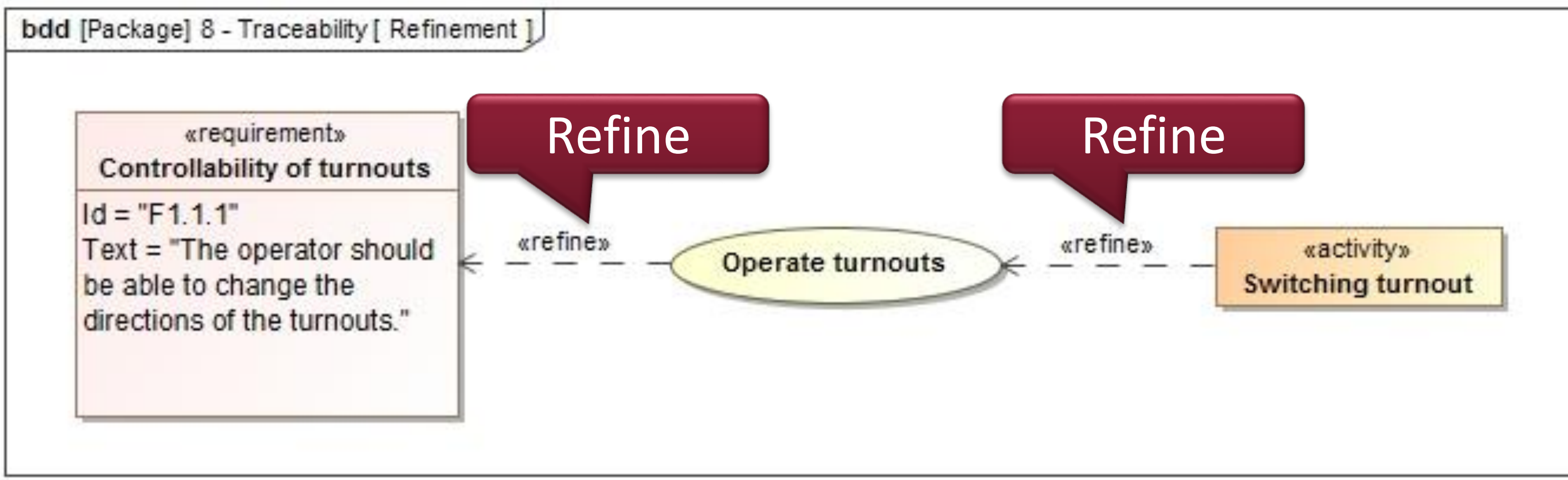
Relations between Requirements

- **Trace**
  - General trace relationship
  - Between requirement and any other model element
- **Refine**
  - Depicts a model element that clarifies a requirement
  - Typically a use case or a behavior
- **Derive**
  - A requirement is derived from another requirement by analysis or decision
  - Typically at the next level of the system hierarchy
- **Copy**
  - Supports reuse by copying requirements to other namespaces
  - Master-slave relation between requirements
- **Satisfy**
  - Depicts a design or implementation model element that satisfies the requirement
- **Verify**
  - Used to depict a test case that is used to verify a requirement

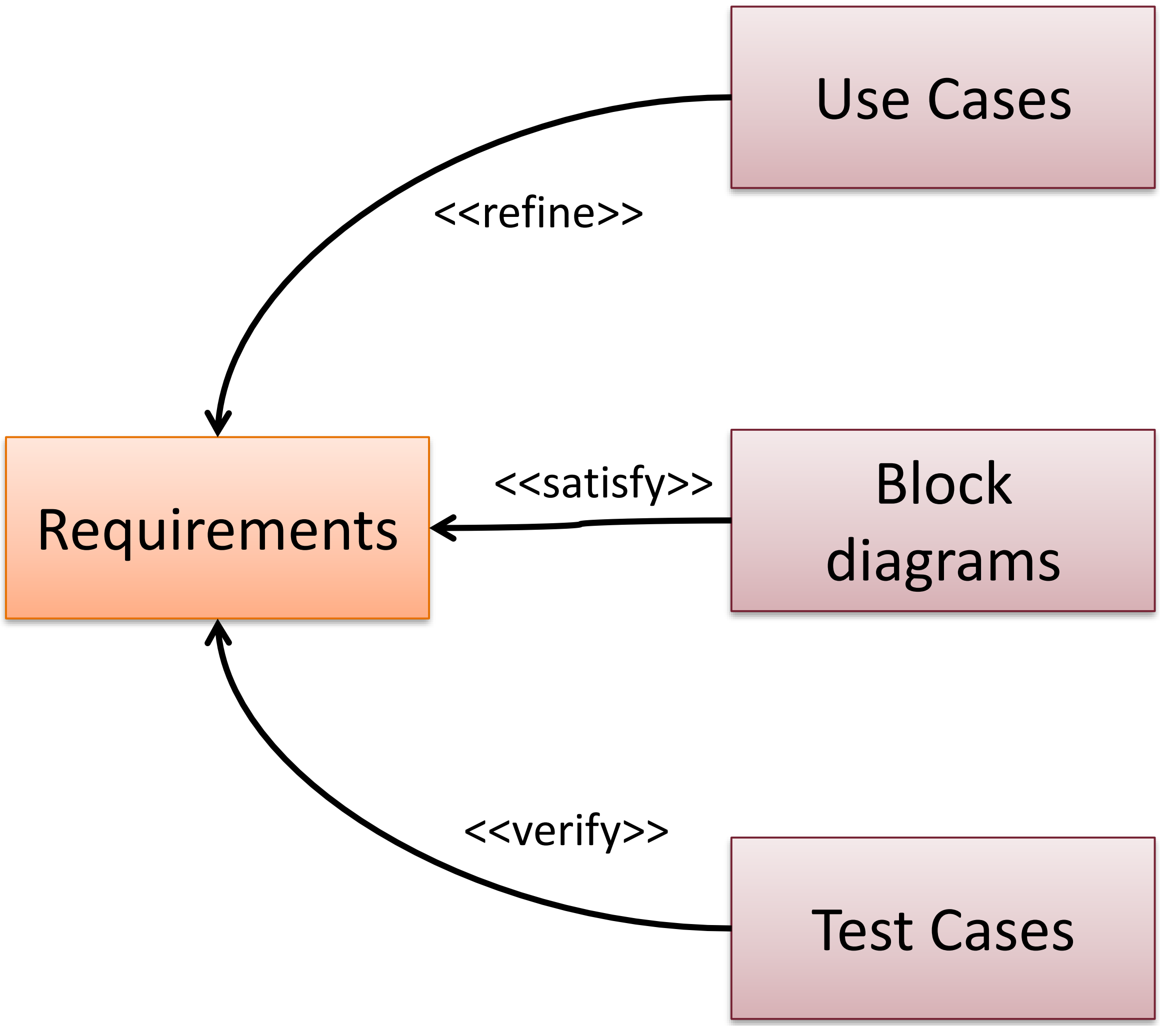


Examples of Relations between Requirements





Traceability of Requirements in SysML Models



SysML 1.5 requirement modeling changes

May 2017

Requirements Relations in Table

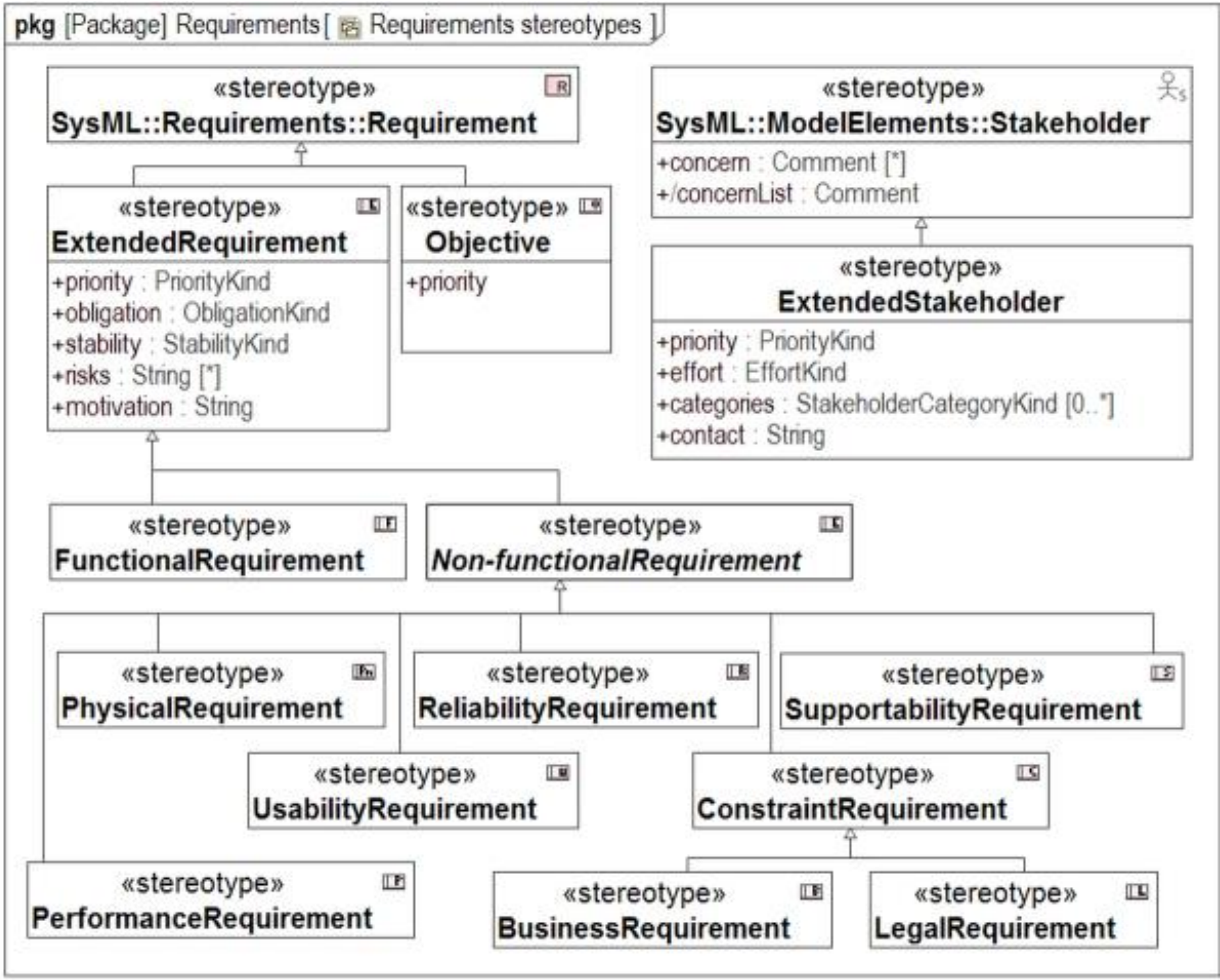
#	Id	Name	Text	Traced To
24	P1	Cost efficiency	The system shall choose one of the cheapest ways of delivering the cargo to the destination in a safe way.	SAFE_1 Safe traffic
25	P2	Swift delivery	The delivery of the cargo shall be as fast as the safe operation of the railway allows and the route is economical.	P1 Cost efficiency R2 High availability
26	R2.1	Low downtime	Allowed downtime of the system is one hour per year.	
27	R2.2	Fast recovery	The system should continue normal operation withi[...] hours after a failure. (MTTR = 8h)	
28	R2	High availability	The transportation system shall provide its services [...]	
29	S1.1	[illegible]	The system shall provide remote access to the staff [...]bers.	
30	S1.2.1	[illegible]	[...]nnel only with extra authority may access the system.	
31	S1.2	Secure access	[...]tenance staff should access the system securely.	
32	S1	Maintainability	There shall be access points for the system for maintenance and update.	
33	SAFE_1.	Safety within a zone	The infrastructure shall ensure safe traffic within a zone.	

Traceability links

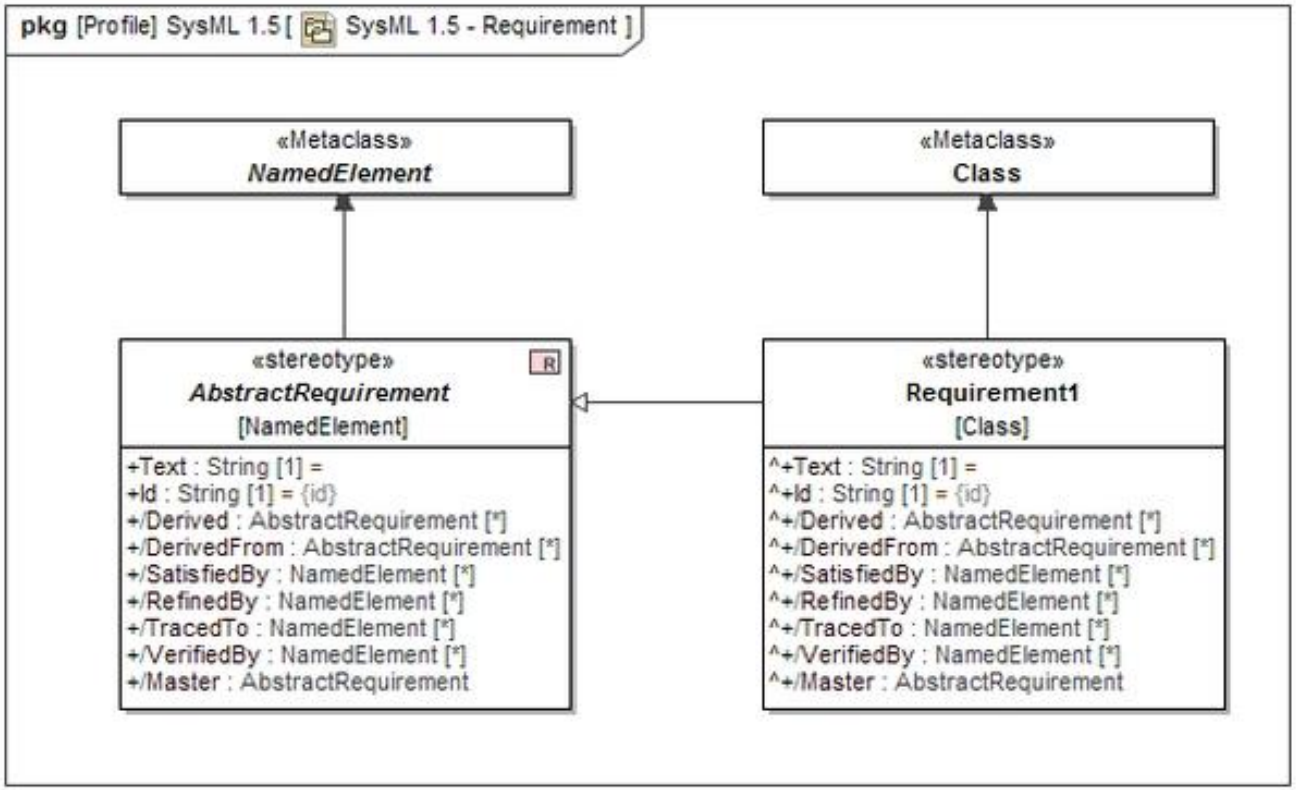
Hierarchical numbering

Subclassing the SysML

Additional requirement properties taxonomies



SysML 1.5 Requirement Definition

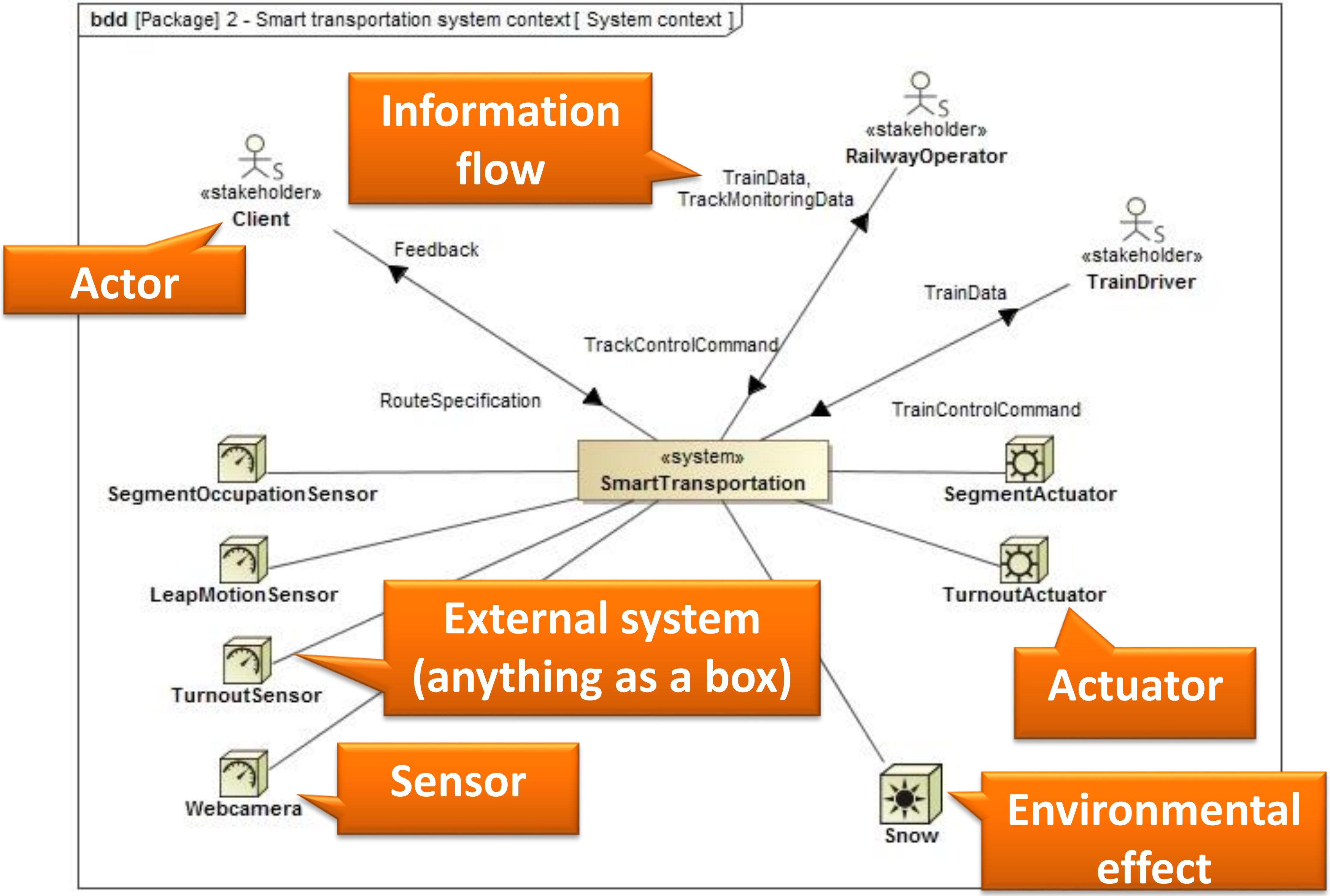


Any named model element could represent a requirement.

- a constraint, a
- block with value properties,
- behavior element
  - state machine
  - activity,

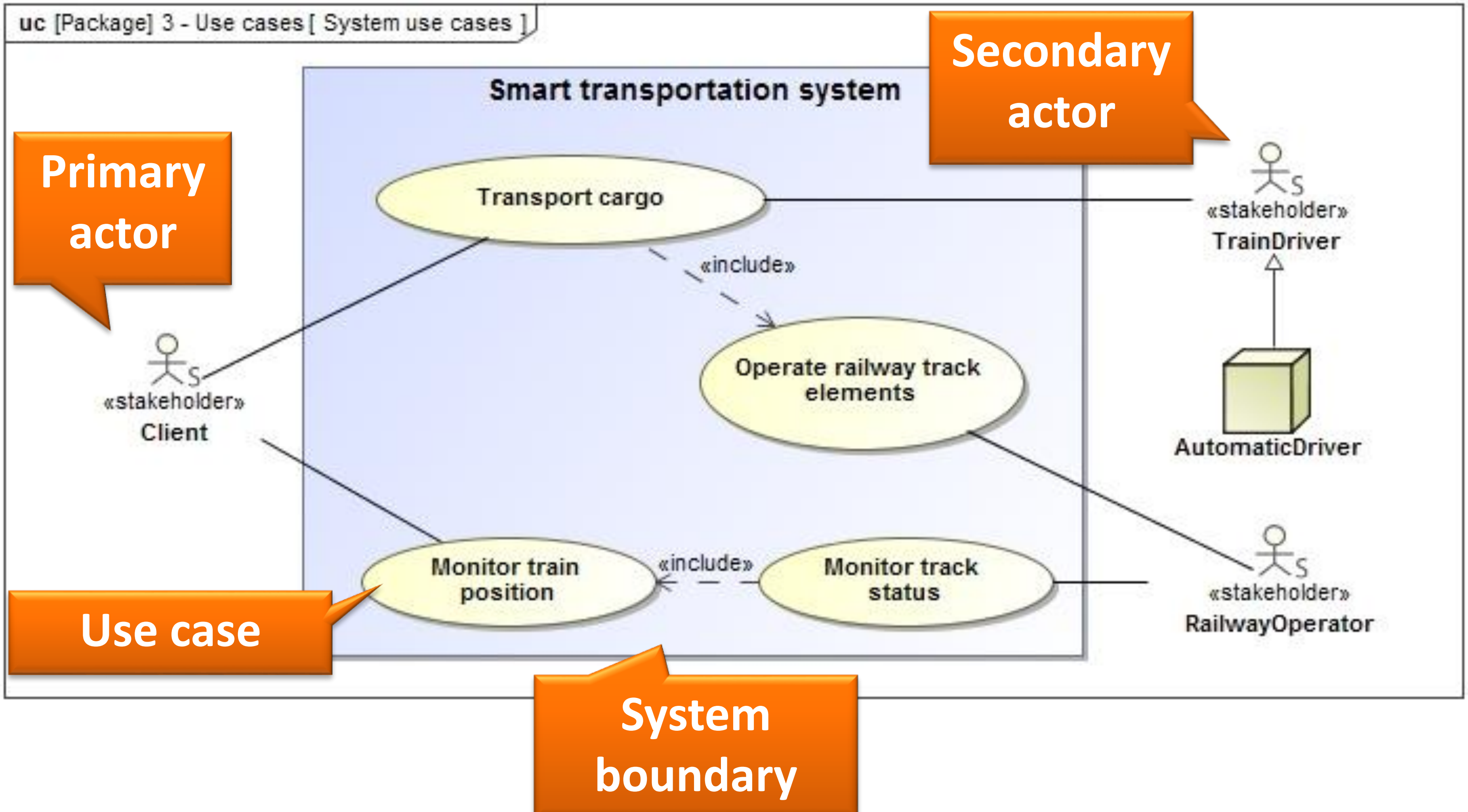
Modeling System Functions with Use Cases

SysML notation: Actors and External systems



Use cases

*Who will use the system **and for what**?*



Use Case Descriptions

- Additional textual description to detail use cases
  - **Preconditions**: must hold for the use case to begin
  - **Postconditions**: must hold once the use case has completed
  - **Primary flow**: the most frequent scenario(s) of the use case (aka. main success scenario)
  - **Alternate flow**: less frequent (or not successful)
  - **Exception flow**: not in support of the goals of the primary flow
- Elaborated behavior in SysML (discussed later)
  - Activity diagrams: scenarios with complex control logic
  - Interaction diagrams: for message based scenarios

Overview of UC Relations

Association

- Actor – use case (rarely: actor – actor)
- an actor initiates (or participates in) the use of the system

Generalization

- actor – actor OR use case – use case
- a UC (or actor) is more general than another UC or actor

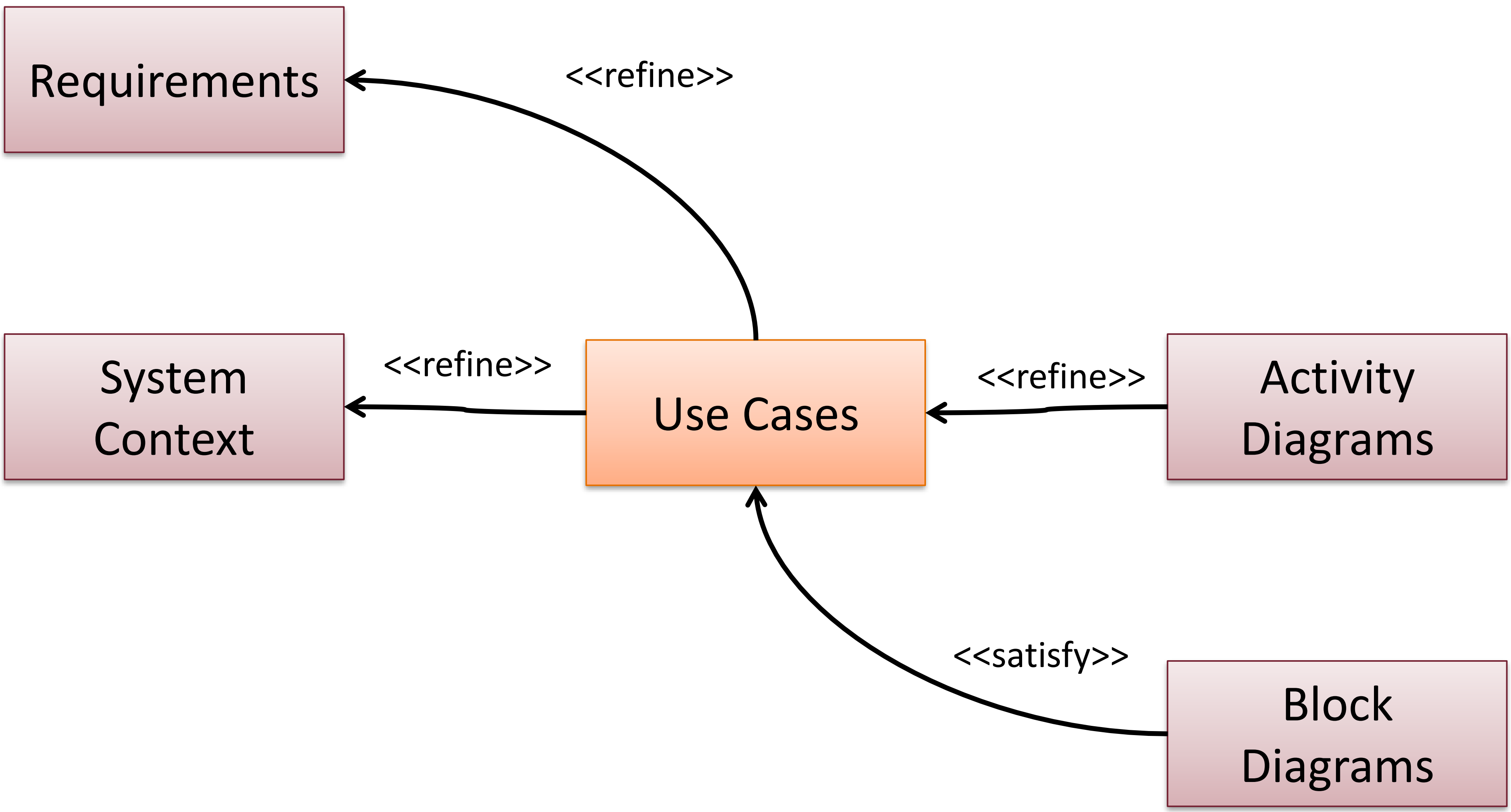
Includes

- use case – use case
- a complex step is divided into elementary steps
- a functionality is used in multiple UCs

Extend

- use case – use case
- a UC may be extended by another UC
- typically solutions for exceptional situations

Traceability of Use Cases in SysML Models



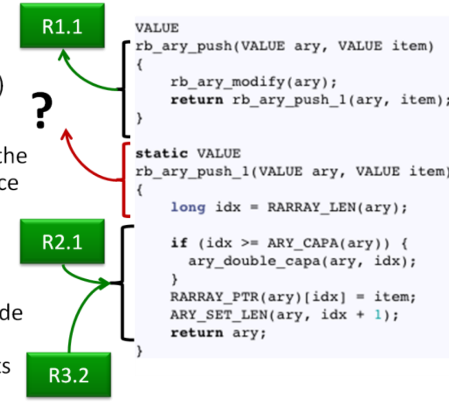
Summary

Definition of a Requirement

- Definitions
  - A condition or capability a system must conform to (IBM Rational)
  - A statement of the functions required of the system (Mentor Graphics)
- Each requirements needs to be
  - **Identifiable + Unique**: unique IDs
  - **Consistent**: no contradiction
  - **Unambiguous**: one interpretation
  - **Verifiable**: e.g. testable to decide if met
- Captured with special statements and vocabulary

The Concept of Traceability

- Traceability is a core **certification concept**
  - For safety-critical systems
  - See safety standards (DO-178C, ISO 26262, EN 50126)
- **Forward traceability**:
  - From each requirement to the corresponding lines of source code (and object code)
  - Show responsibility
- **Backward traceability**:
  - From any lines of source code to one ore more corresponding requirements
  - No extra functionality



Definition of Use Cases

- **Use case (használati eset)** captures a main functionality of the system corresponding to a functional requirements
- UCs describe
  - the typical interactions
  - between the *users* of a *system* and
  - **the system itself,**
  - by providing a narrative of how a system is used
- A set of scenarios tied together by a common user goal
- Language template: **Verb + Noun (Unique)!**
  - Example: Drive train, Switch turnout

M. Fowler: UML Distilled. 3rd Edition. Addison-Wesley

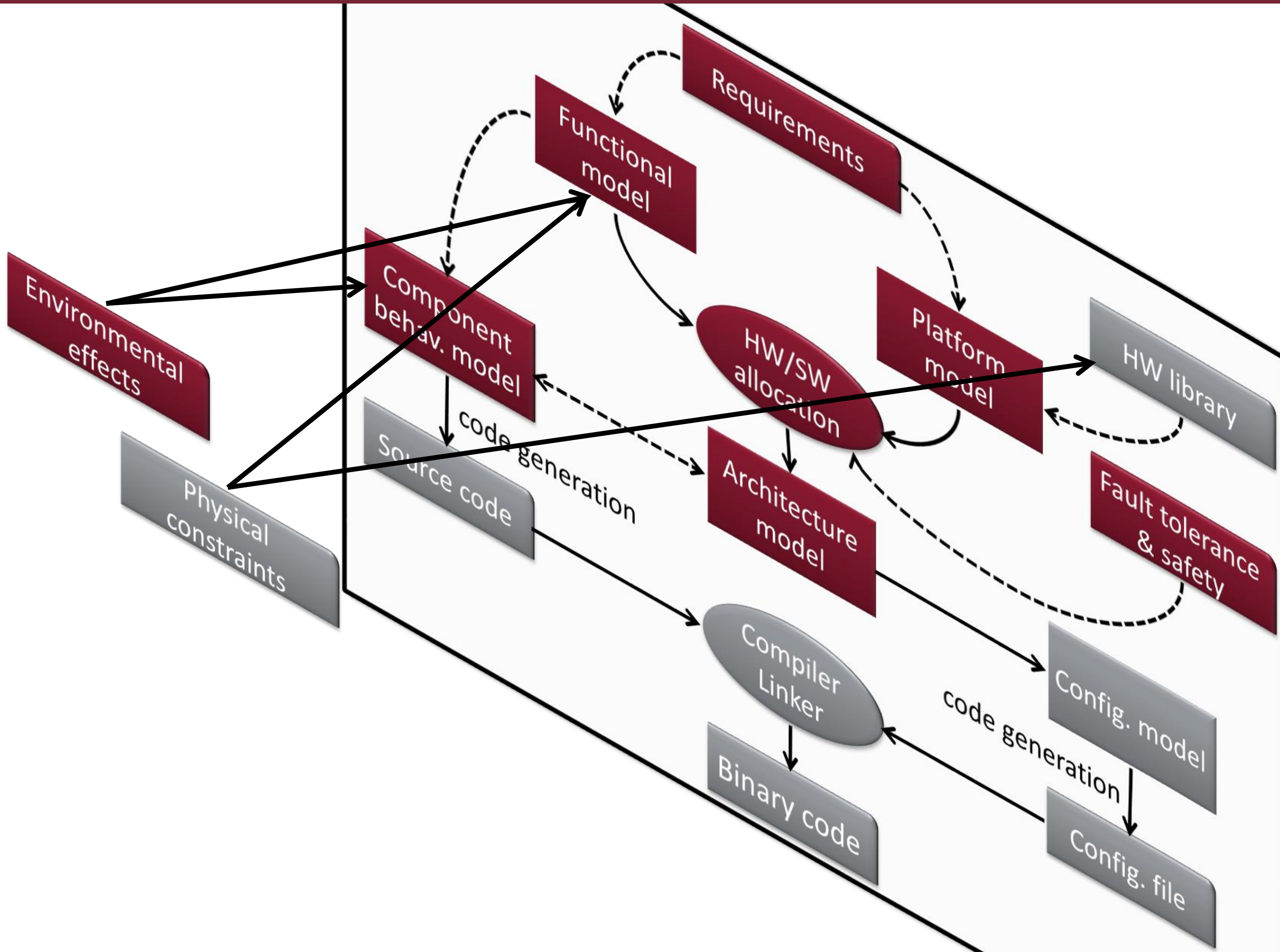
Definition of Actors

- **Actor (aktor, szereplő)** is a **role** that a user plays with respect to the system.
  - *Primary actor*: calls the system to deliver a service
  - *Secondary actor*: the system communicates with them while carrying out the service
- An actor is outside the boundary of the system
- *Characteristics:*
  - One person may act as more than one actor
    - Example: The farmer may also act as a laborer who performs the spraying
  - Can be an existing subsystem (and not a person)

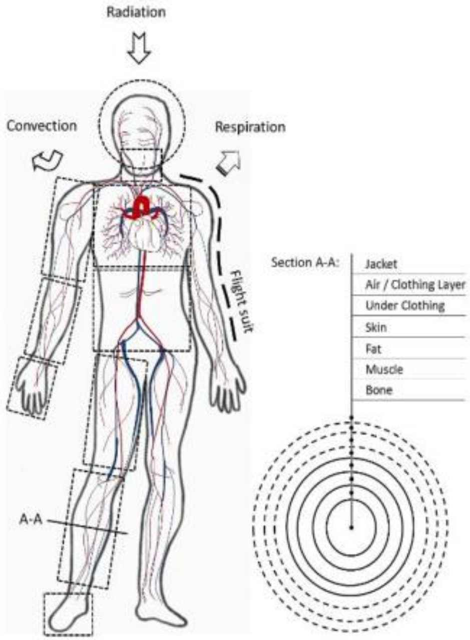
Modeling physical properties

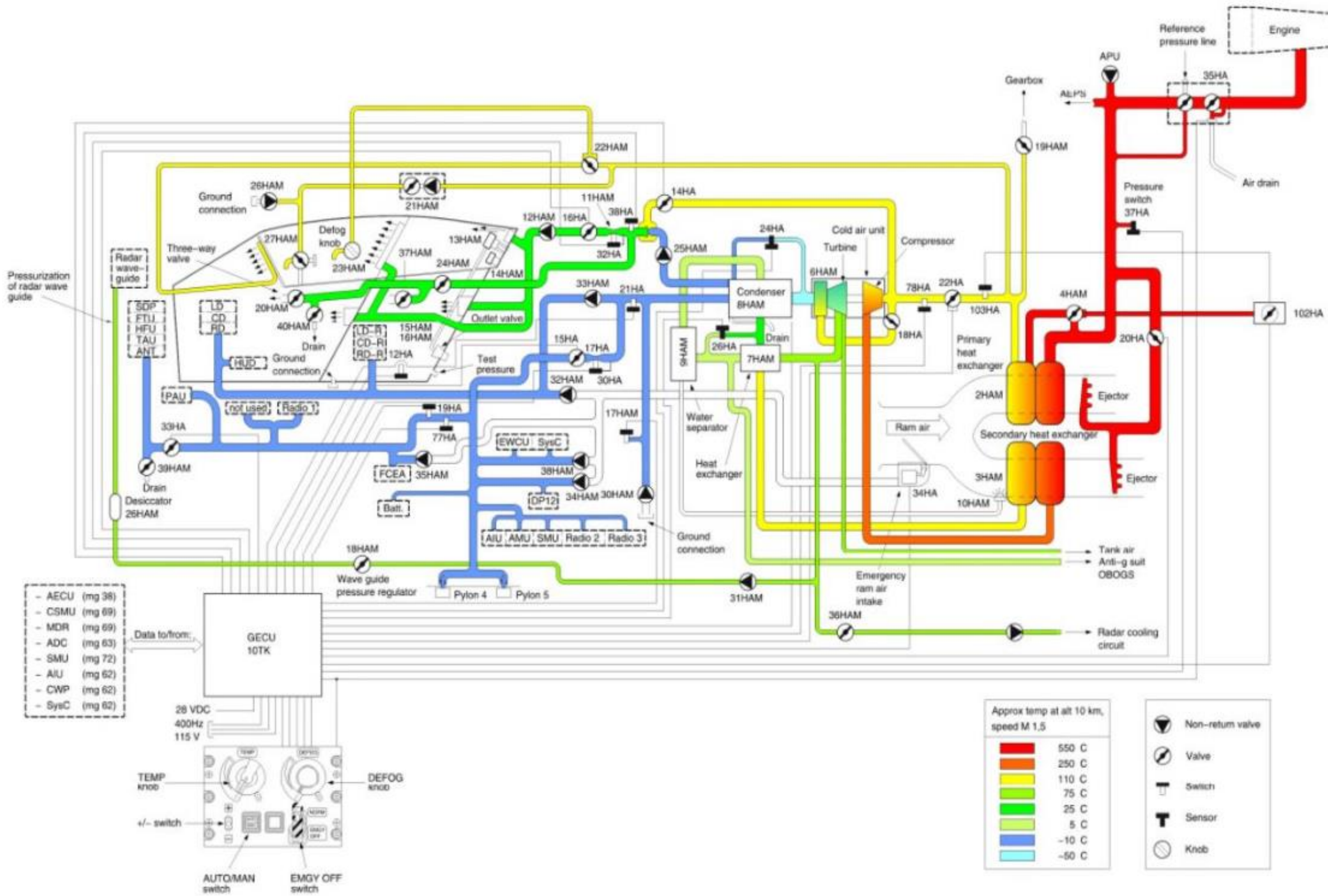
Controller, plant and environment model

Platform-based systems design



Thermal model of an aircraft

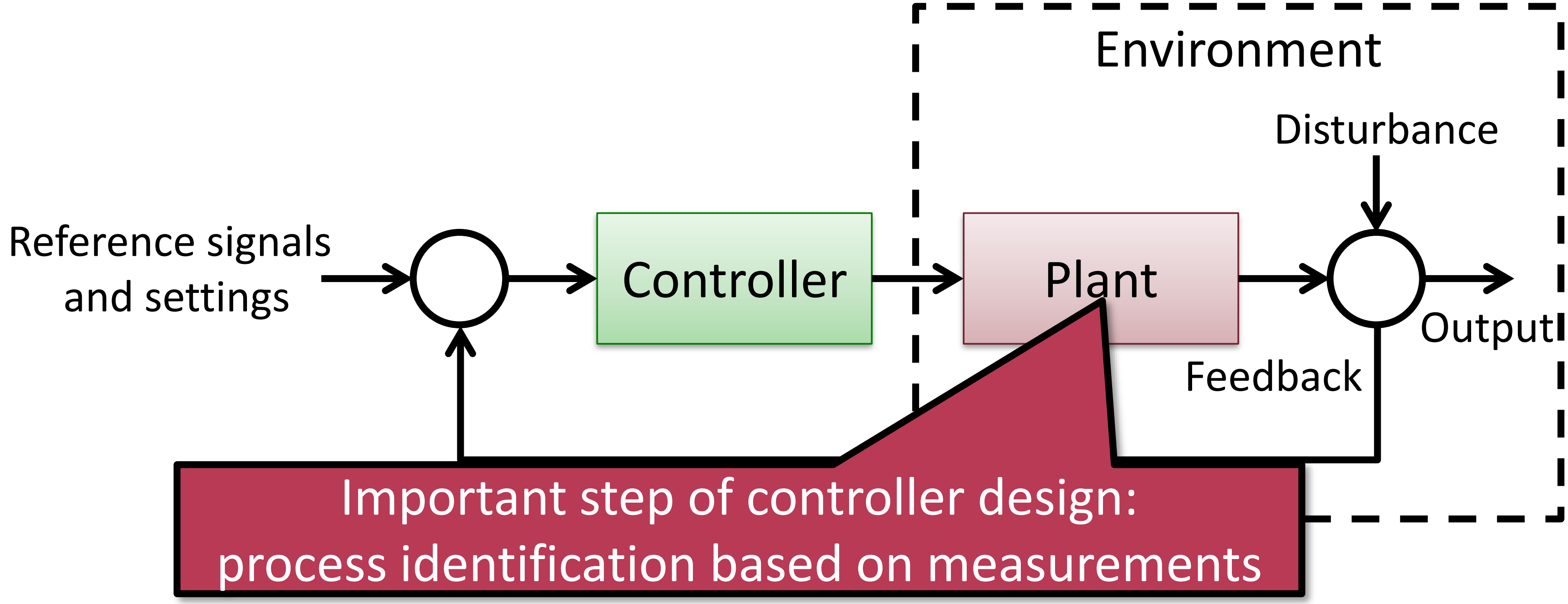




Copyright: SAAB

Controller, Plant, and Environment

- Typical system control loop



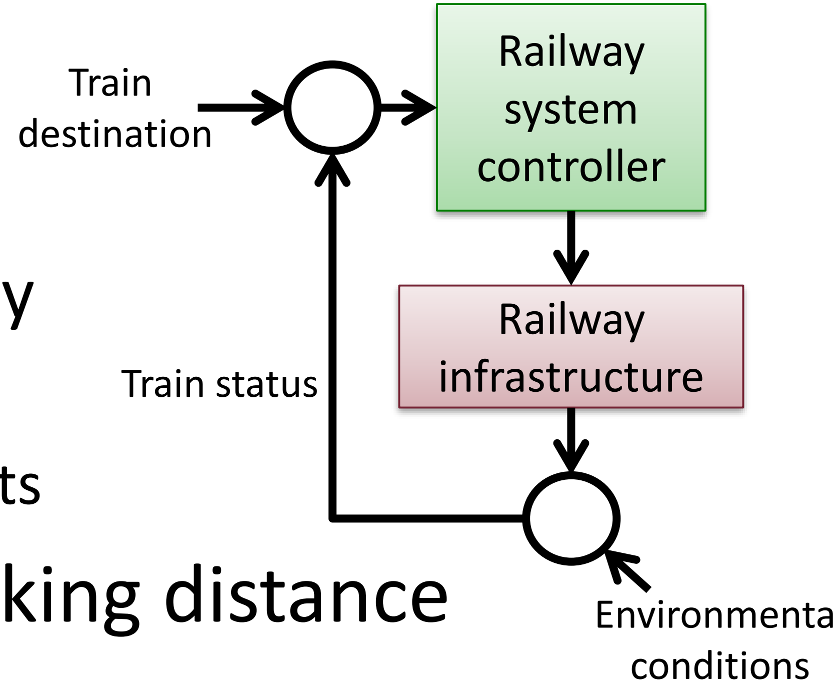
- Co-designing controller and the plant would be the ideal setting

Controller design

- Controller functional design using blocks
  - BDD: defines element hierarchy and containment
  - IBD: template for component internal structure
- Challenge: validate the design of the controller
  - On-site testing and calibration can be
    - Expensive (time + cost)
    - Dangerous
  - Instead:
    - create plant model and environment model with physical properties and
    - run simulations

Example railway system controller

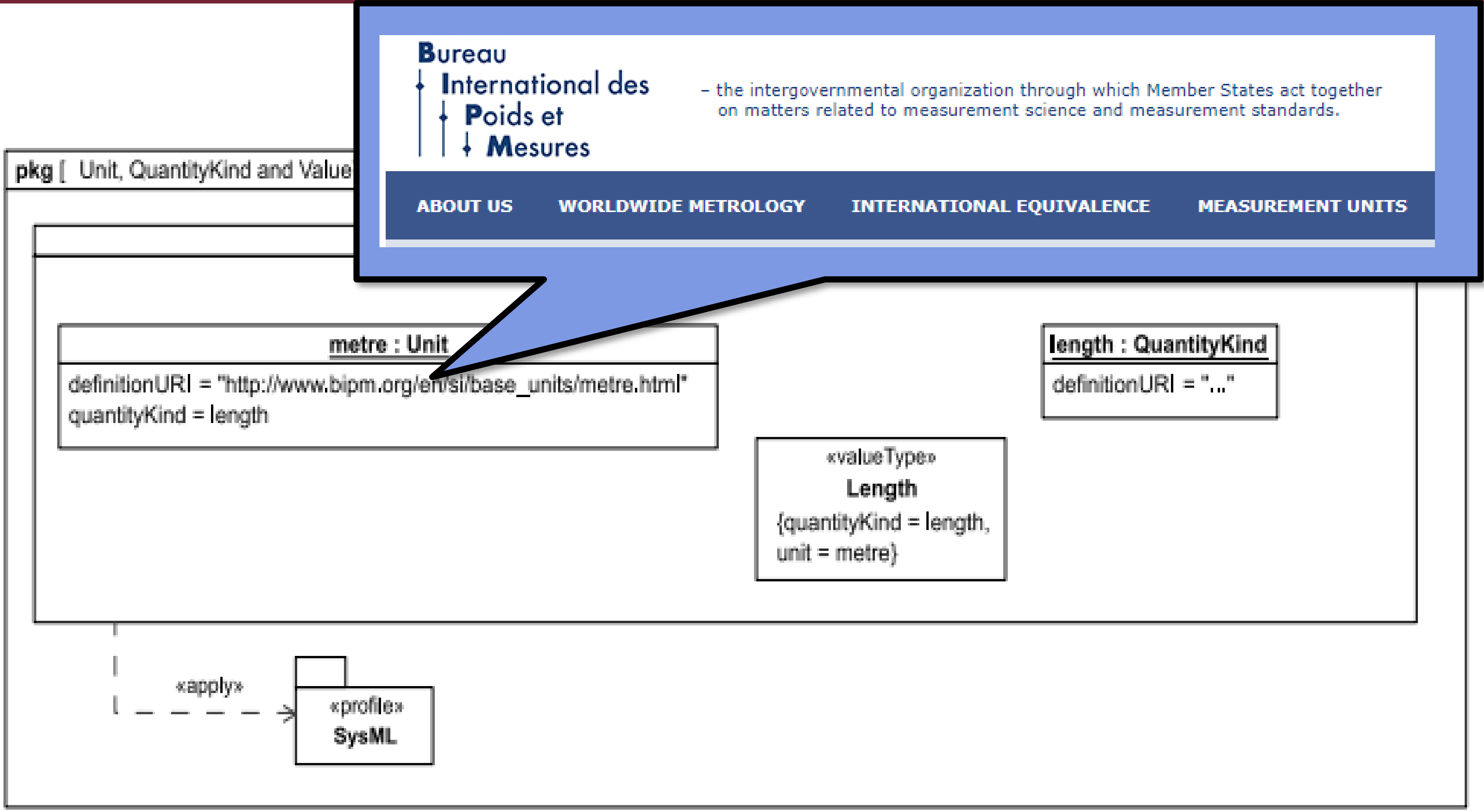
- Controller aims to
  - monitor the trains
  - apply brakes when necessary
    - too close to each other
    - prevent derailment at turnouts
- Parameters influencing braking distance
  - Weather conditions
  - Speed
  - Landscape
  - … (anything else?)



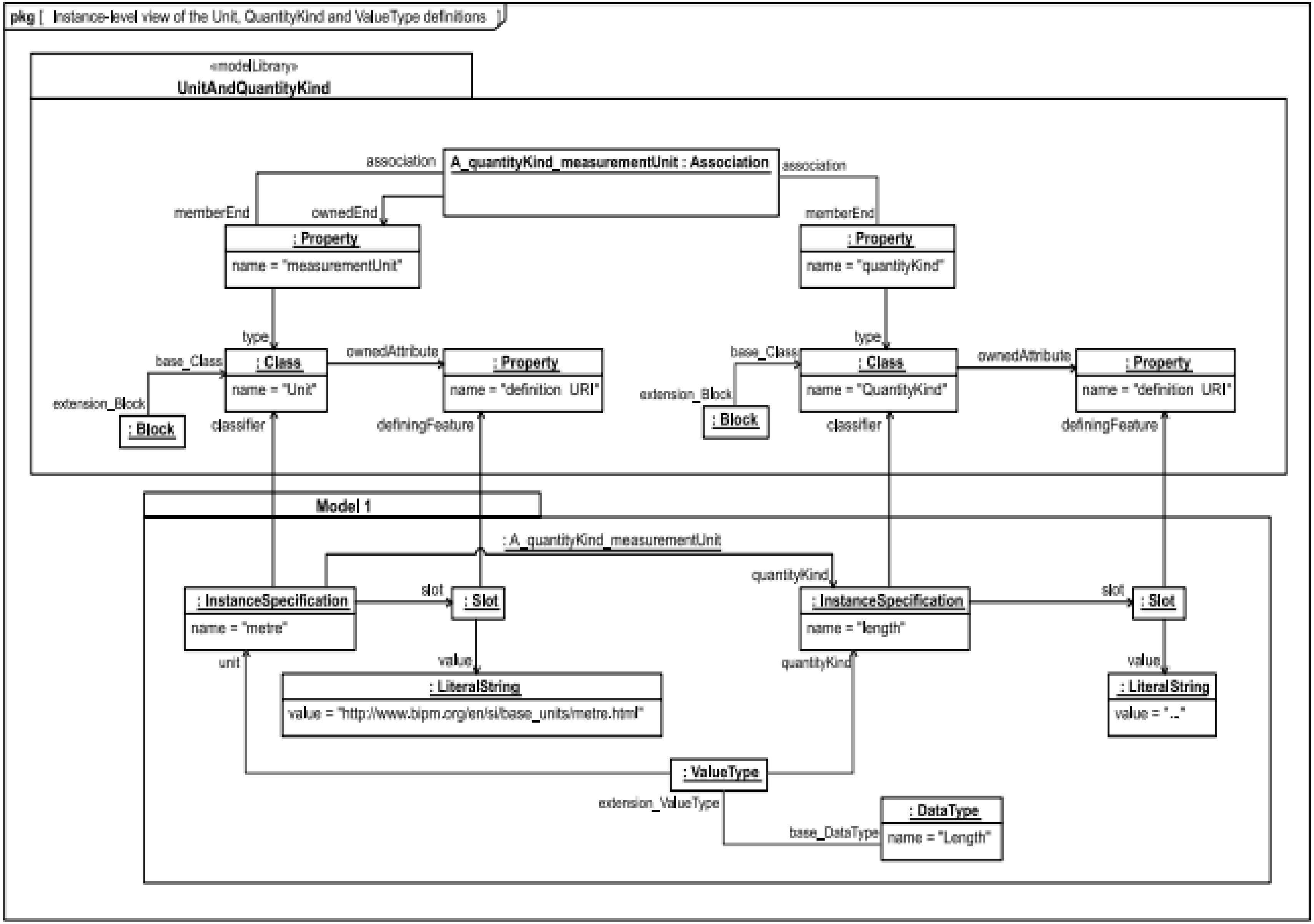
Constraints and physical parameters in SysML

Constraint blocks

Units and Quantity Kinds

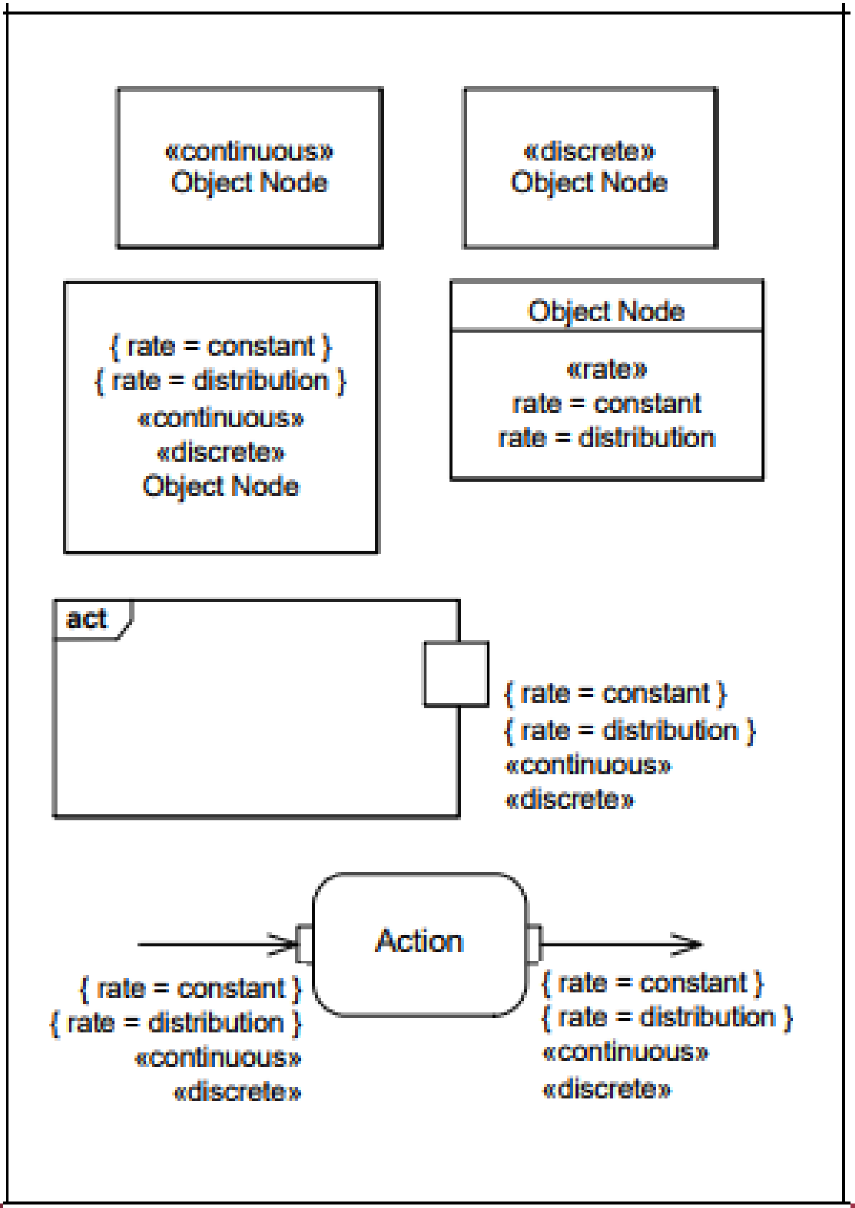


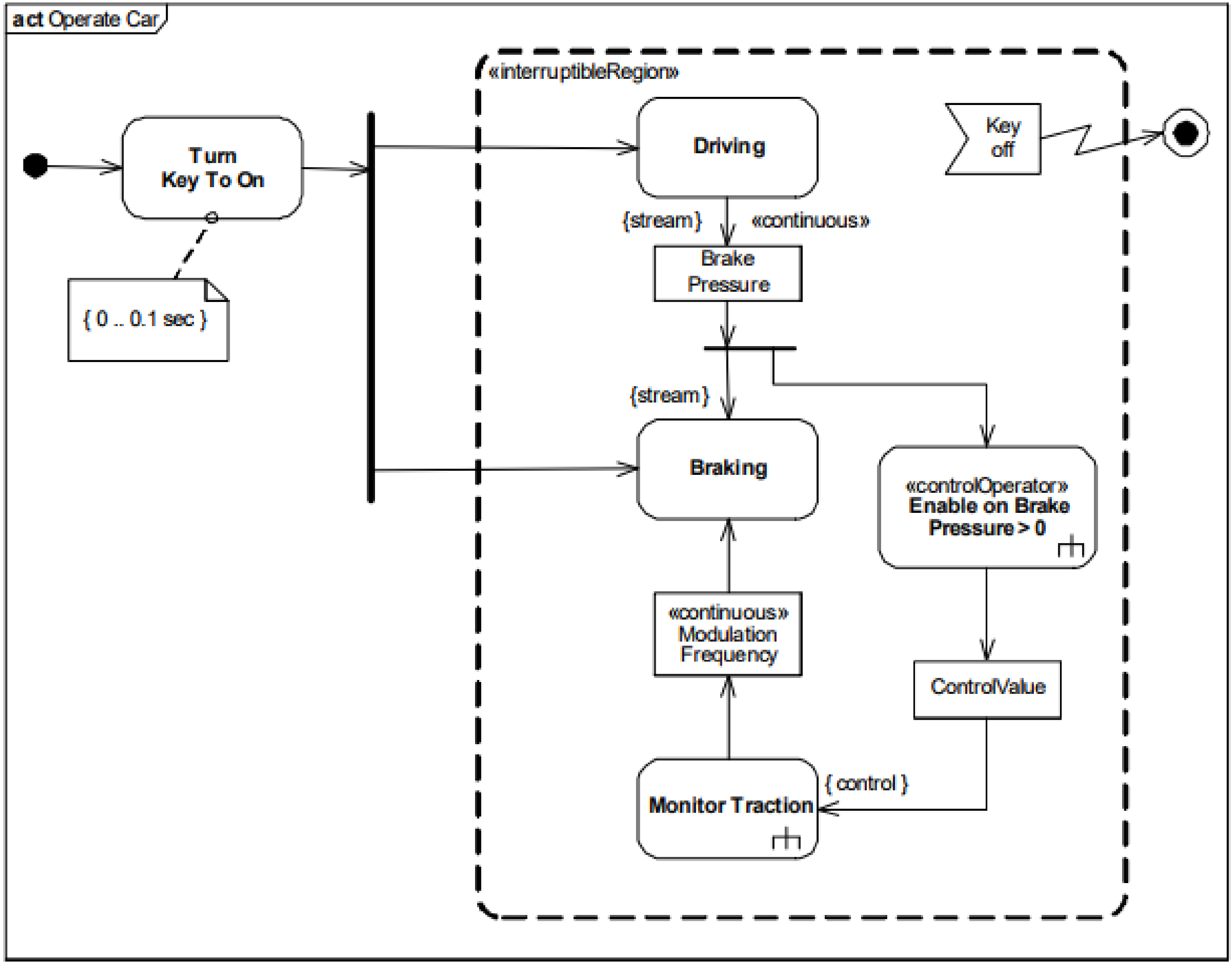
Unit, QuantityKind and ValueType definitions

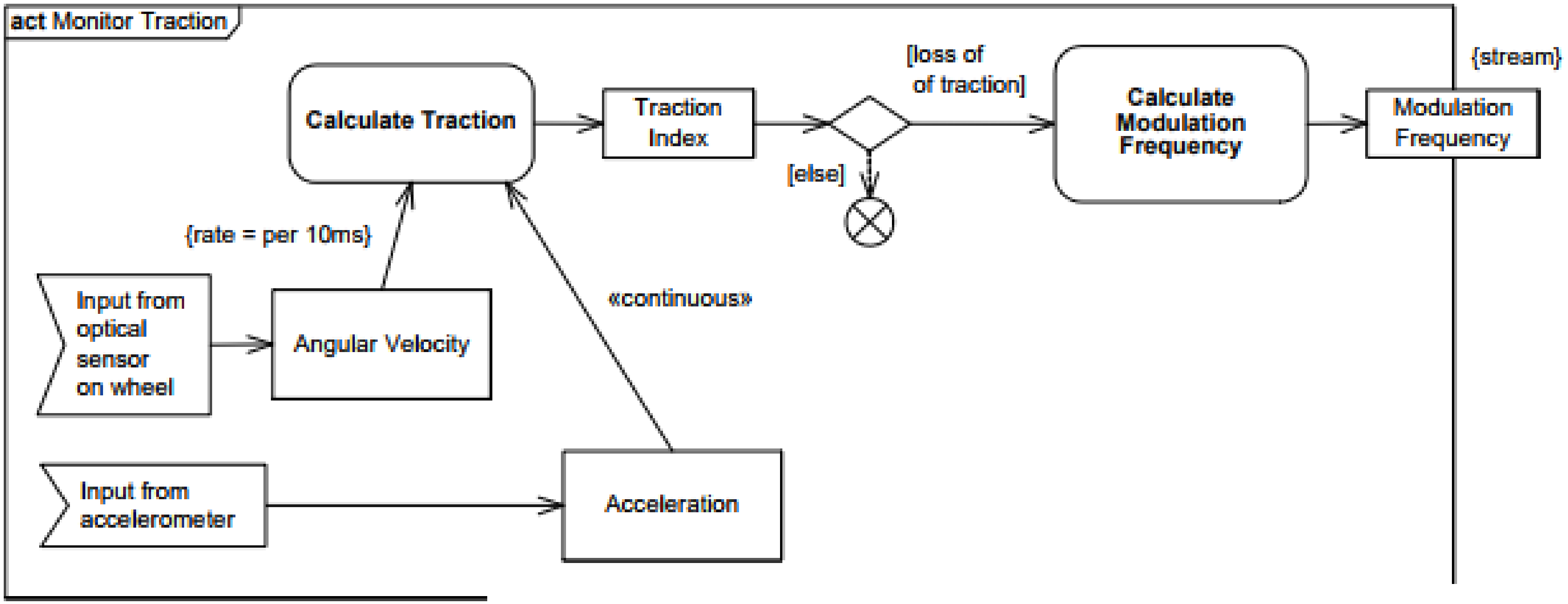


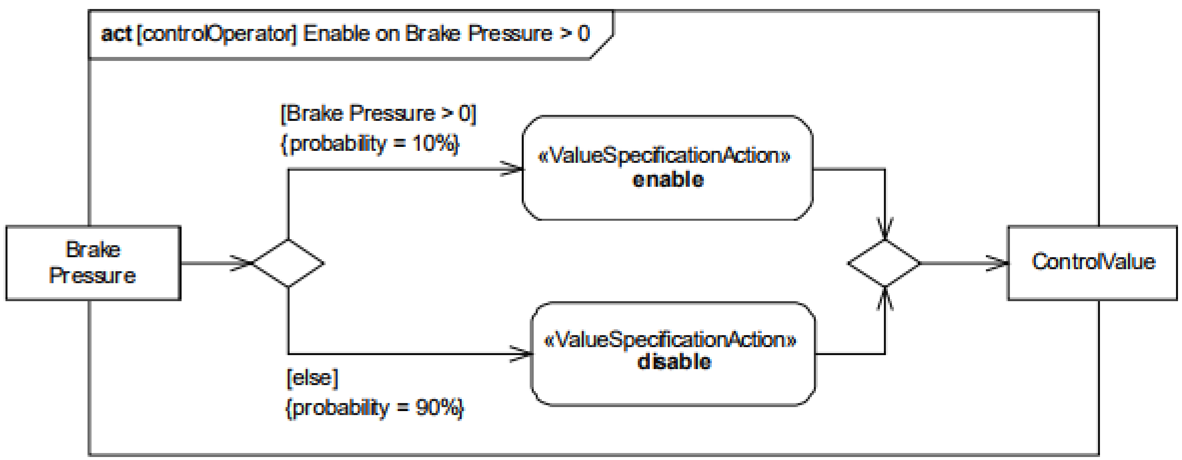
Continuous elements :Rate

SysML::Activities::Rate,
SysML::Activities::Continuous,
SysML::Activities::Discrete



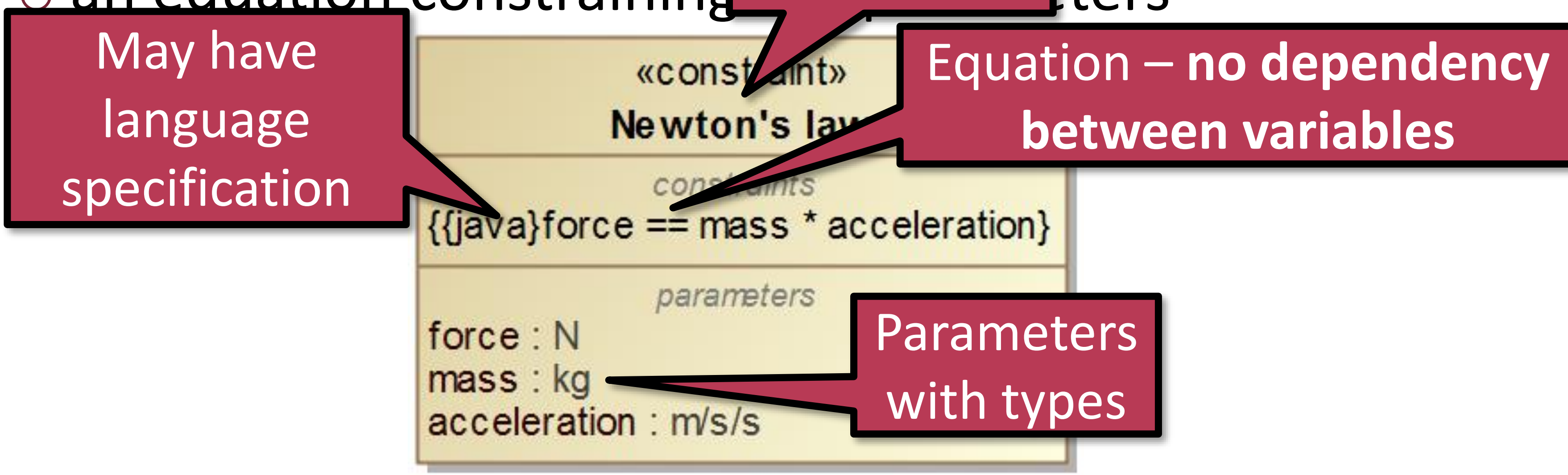

act Operate Car
«interruptibleRegion»
Turn
Key To On
{ 0 .. 0.1 sec }
Driving
{stream}
«continuous»
Brake
Pressure
{stream}
Braking
«controlOperator»
Enable on Brake
Pressure > 0
«continuous»
Modulation
Frequency
ControlValue
{ control }
Monitor Traction
Key
off


act Monitor Traction
Calculate Traction
Traction Index
[loss of of traction]
Calculate Modulation Frequency
Modulation Frequency
{stream}
[else]
{rate = per 10ms}
«continuous»
Input from optical sensor on wheel
Angular Velocity
Input from accelerometer
Acceleration


act [controlOperator] Enable on Brake Pressure > 0
[Brake Pressure > 0]
{probability = 10%}
«ValueSpecificationAction»
enable
Brake Pressure
ControlValue
«ValueSpecificationAction»
disable
[else]
{probability = 90%}

Constraint blocks

- **Constraint**: equations with parameters bound to the properties of the system
- **Constraint block**: supports the definition and the reuse of constraints. It holds
  - a set of parameters and
  - an equation constraining … eters



Assignments and equations

- An assignment in a typical programming language is a **causal** connection, where the left hand side is the dependent variable:

```
y := x + 3
```

- An **acausal** connection is like a mathematical equation; there is no notion of inputs/outputs. So

```
y = x + 3
```

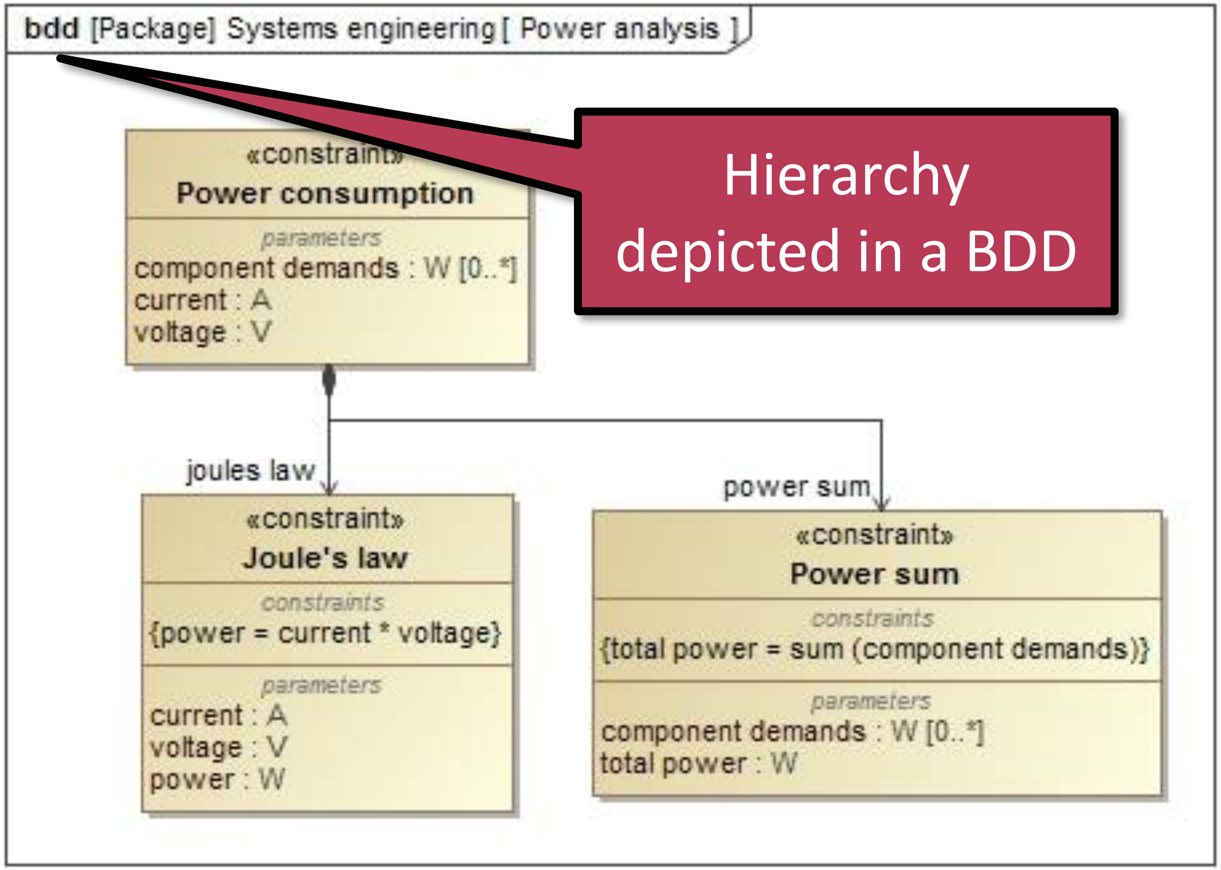
  and

```
y - 3 – x = 0
```

  have the same meaning.
  - If any of the variables has a new value, it enforces that the other variables change accordingly.

Constraint definition

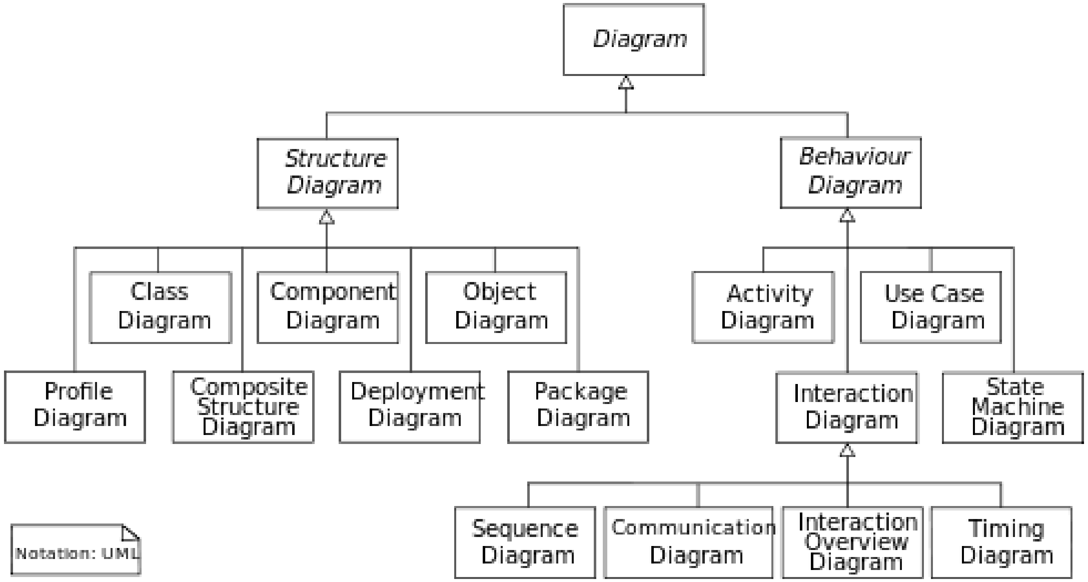
- **Composition** is used to define complex constraints from simple equations



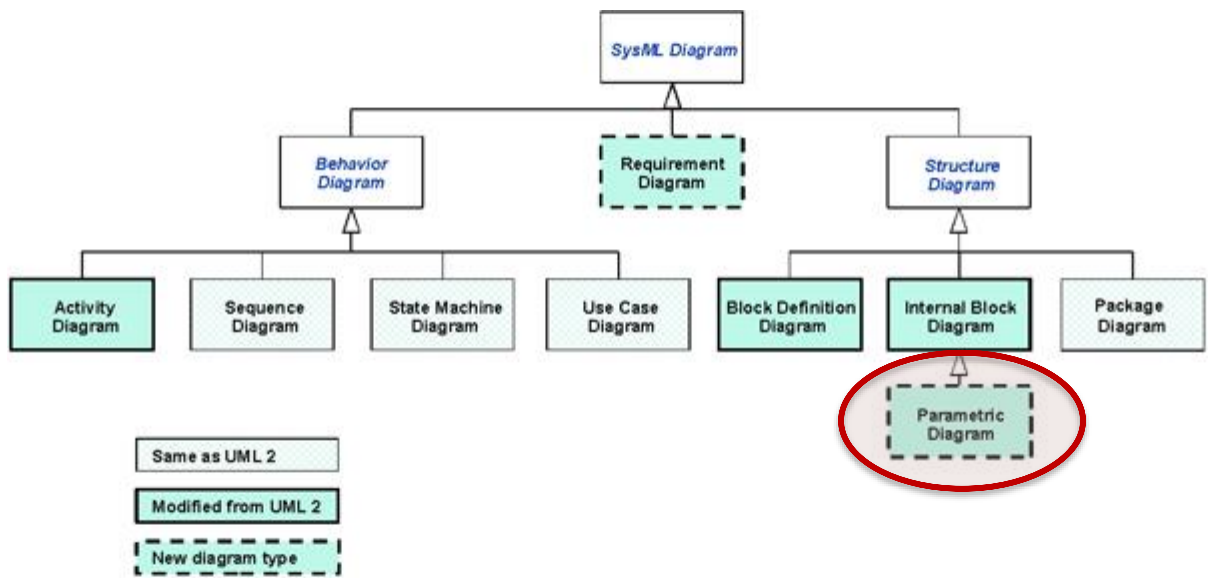
Parametric diagram

Specification of bindings between system parameters

Parametric Diagram (PAR)



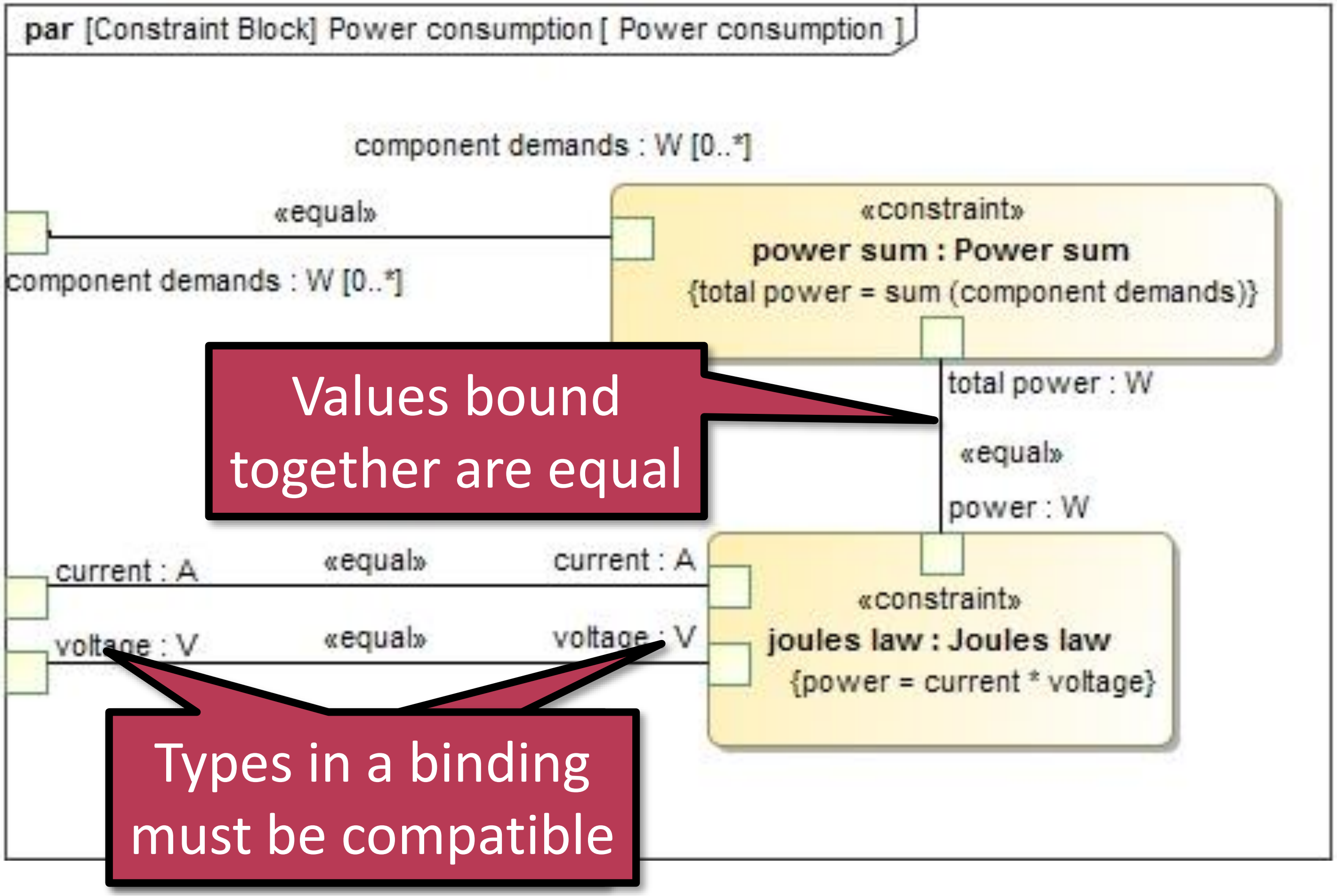
UNIFIED MODELING LANGUAGE



OMG SYSTEMS MODELING LANGUAGE

Parameter bindings

- Goal: describe the application of constraints in a particular context



Applications of parametrics

- Parametric specification
  - Define parametric relationships in the system structure
- Parametric analysis
  - Evaluating constraints on the system parameters to calculate values and margins for a given context
  - Checking design alternatives
  - Tool support: ParaMagic plug-in for MagicDraw
- There are modeling standards with better support for this modeling aspect…
  - …such as Modelica

Modelica

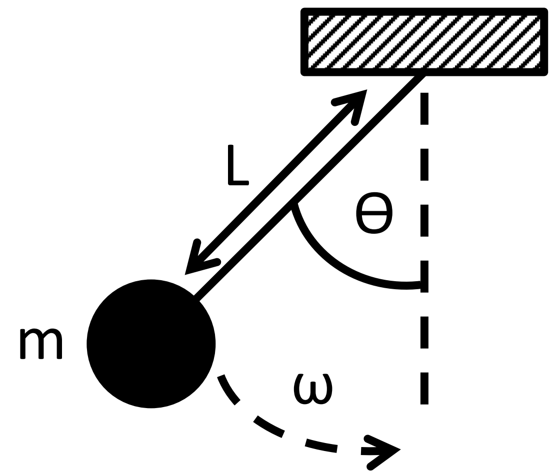
A language for modeling and simulating
complex physical systems

Overview of Modelica

- **Modelica** is an object-oriented, equation based language designed to model complex physical systems containing process-oriented subcomponents of different nature
  - Describing both continuous-time and discrete-time behaviour
- The **Modelica Standard Library** provides more than 1000 ready-to-use components from several domains
  - Full high-school + 1st year university physics (and much more)
- Implementations
  - Commercial e.g. by Dymola, Maplesoft, Wolfram MathCore
  - Open-source: JModelica
- Modeling and simulation IDE: OpenModelica

Example: modeling a simple pendulum

- Simple pendulum



- Behavior of the pendulum as a function of time:

$$\begin{pmatrix} \dot{\theta}(t) \\ \dot{\omega}(t) \end{pmatrix} = \begin{pmatrix} \omega(t) \\ -\frac{g}{L}\theta(t) \end{pmatrix}$$

Modelica code for simple pendulum

```
model SimplePendulum
    parameter Real L=2.0;
    constant Real g=9.81;
    Real theta (each start = 1.0);
    Real omega;
equation
    der(thetha) = omega;
    der(omega) = -(g/L)*thetha;
end SimplePendulum;
```

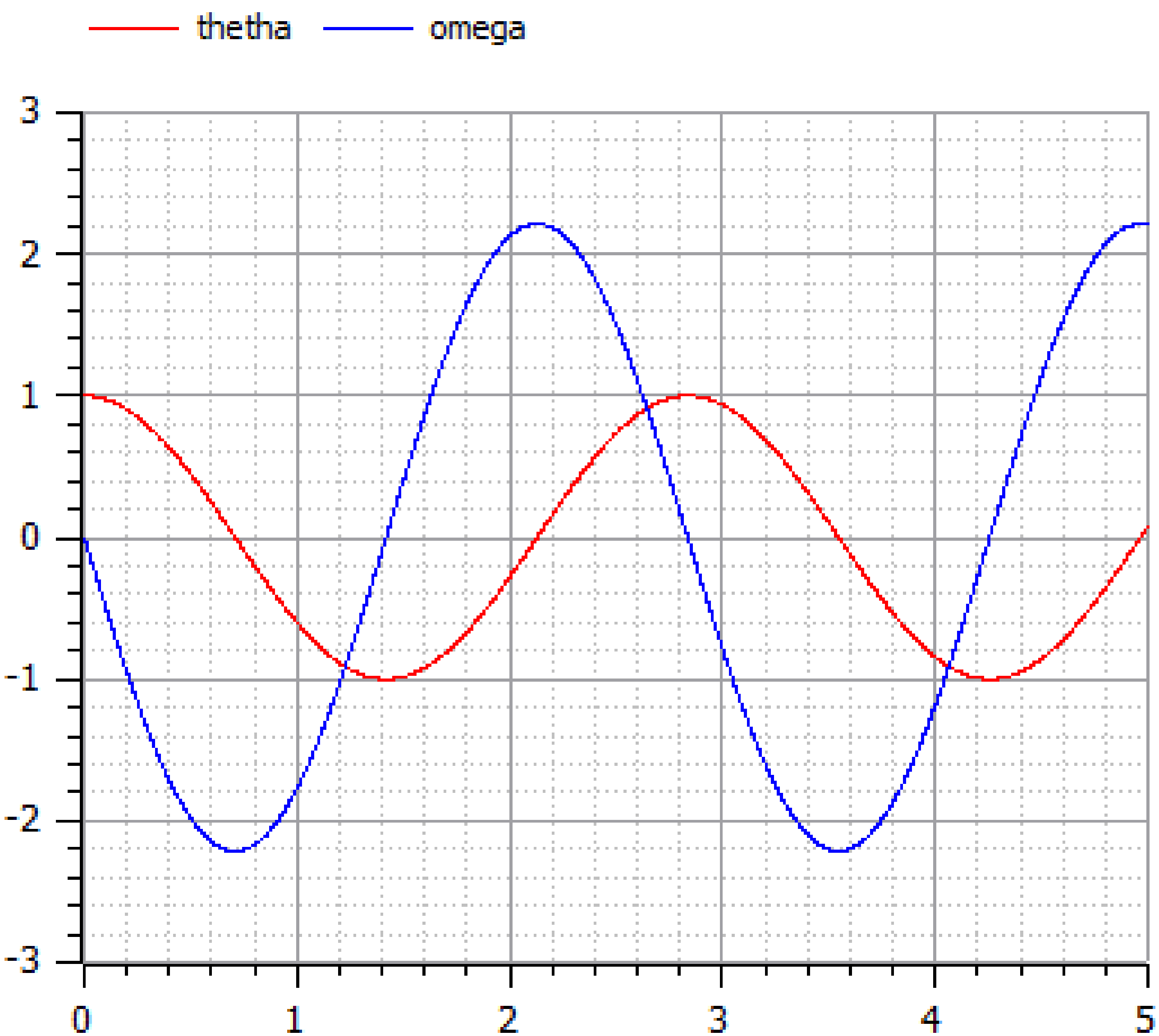
Model name

Continuous time variables, constants

Initial value

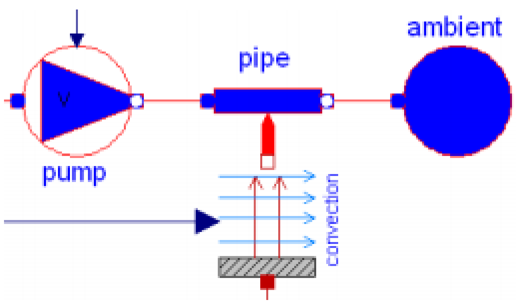
(Differential) equations

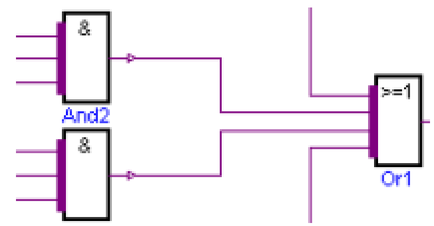
Pendulum simulation results

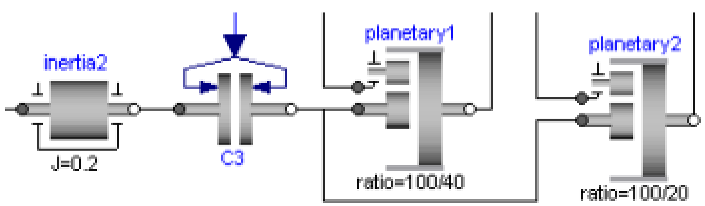


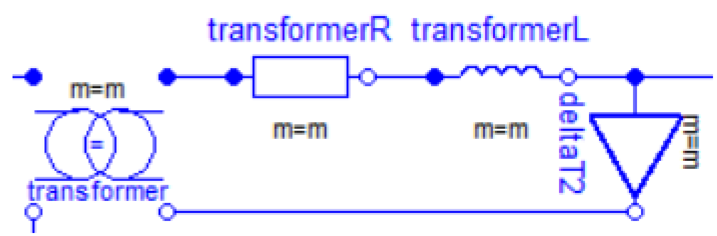
Modelica Standard Library

- Provides reusable building blocks (called classes) for Modelica models
- Version 3.2.1. has more than 1340 classes and models
- Various domains







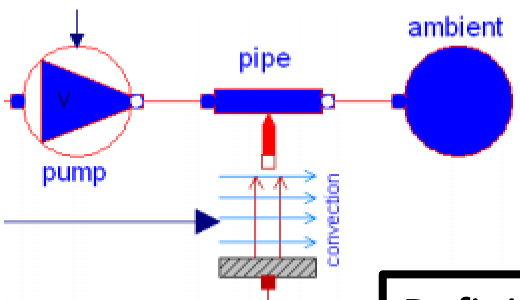


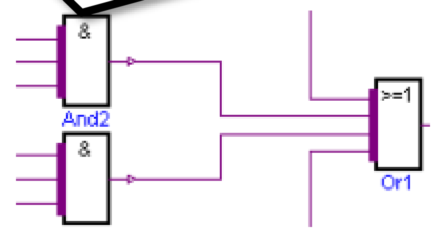
Modelica Standard Library

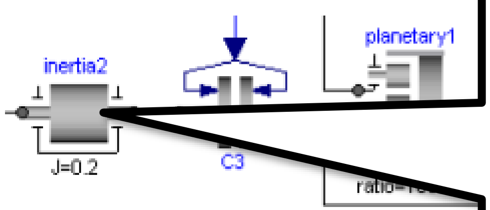
- P
  M
- V
- V

Definition in Modelica:

```
equation
  auxiliary[1] = x[1];
  for i in 1:n - 1 loop
    auxiliary[i + 1] = D.Tables.AndTable[auxiliary[i], x[i + 1]];
       end for;
  y = pre(auxiliary[n]);
```







Definition in Modelica:

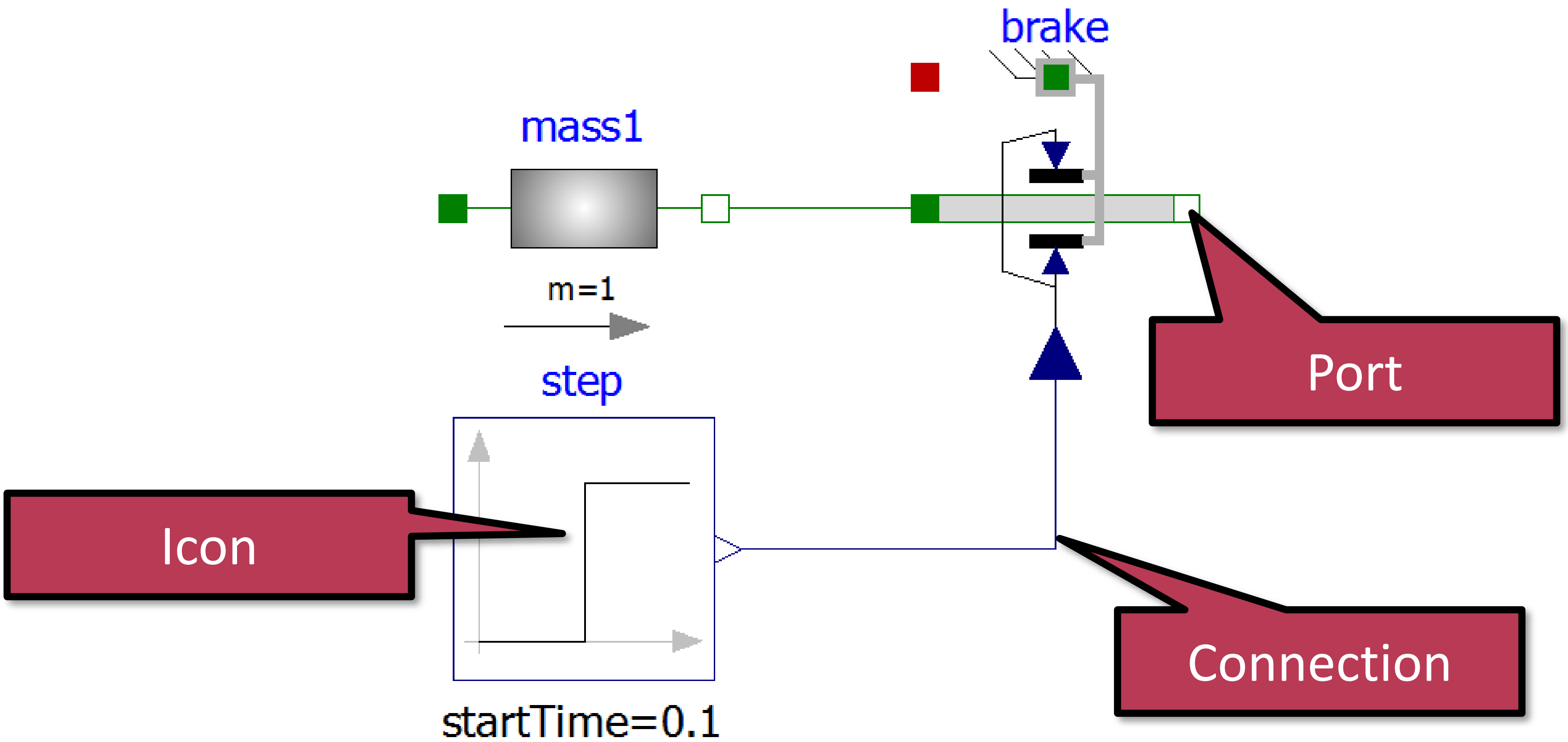
```
equation
        phi = flange_a.phi;
        phi = flange_b.phi;
        w = der(phi);
        a = der(w);
        J*a = flange_a.tau + flange_b.tau;
```

Modelica and Simulation

- Simulating a model means to calculate the values of its variables at certain time instants
- Advantages
  - Observing dangerous/expensive bevaviour at low cost with no risks
  - Resolves scaling issues (size, duration)
- Different algorithms and strategies for simulation
  - The task is to solve Ordinary Differential Equations (ODEs) generated from the model
  - Numerical techniques

Example plant model – train brakes

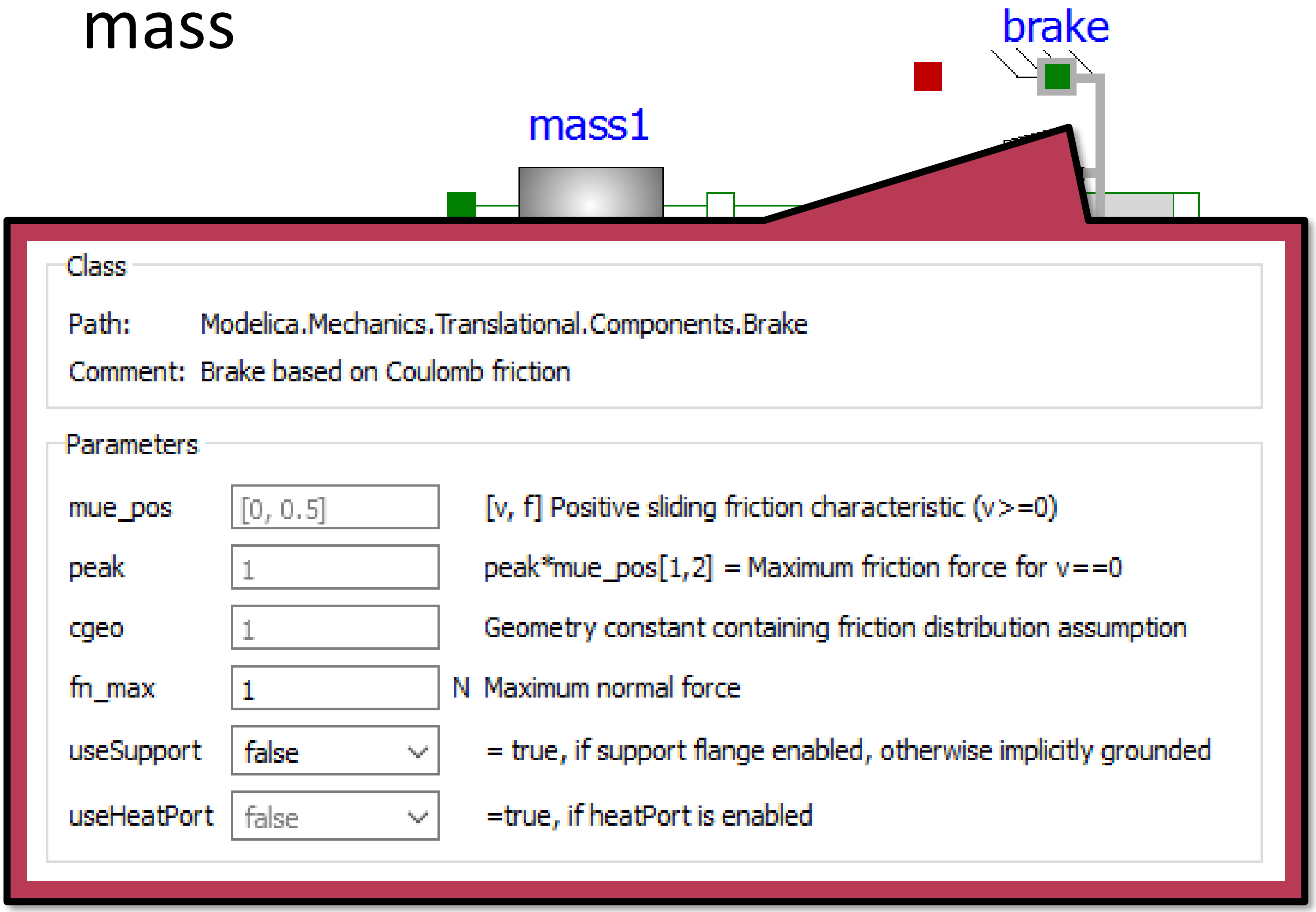
- Physical model for braking system carrying a mass



- Graphical notation in OpenModelicaEditor

Example plant model – train brakes

- Physical model for braking system carrying a given mass



Class

Path: Modelica.Mechanics.Translational.Components.Brake

Comment: Brake based on Coulomb friction

Parameters

mue_pos	[0, 0.5]		[v, f] Positive sliding friction characteristic (v>=0)
peak	1		peak*mue_pos[1,2] = Maximum friction force for v==0
cgeo	1		Geometry constant containing friction distribution assumption
fn_max	1	N	Maximum normal force
useSupport	false		= true, if support flange enabled, otherwise implicitly grounded
useHeatPort	false		=true, if heatPort is enabled

Example plant model – train brakes

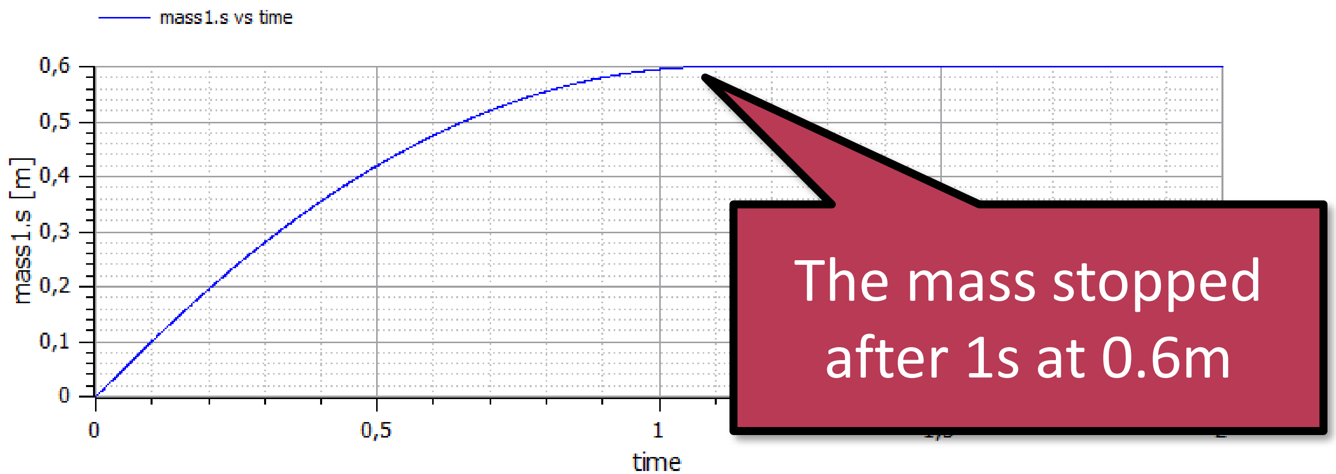
```
model BrakeExample
      Brake brake(
            fn_max=1
            useSupport=false);
      Mass mass1(
            m=1,
            s(fixed=true),
            v(start=
      Step step(
            startTime
            height=2)
equation
      connect(mass1.flange_b, brake.flange_a);
      connect(step.y, brake.f_normalized);
end BrakeExample;
```

Brake, Mass, and Step are inbuilt classes to Modelica Library

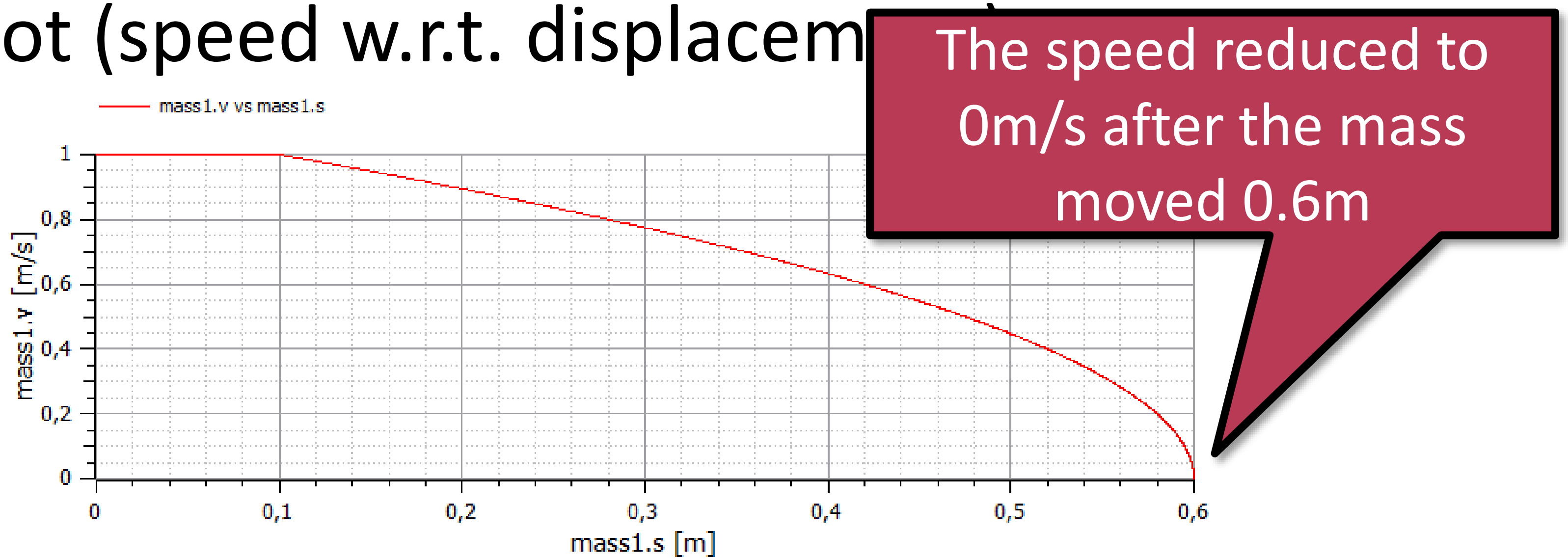
Can describe both causal and acausal connections between ports

Brake times and distance

- Plot values w.r.t. time (displacement)



- X-Y plot (speed w.r.t. displacement)



Summary

- Complex system design requires modeling of physical parameters
  - SysML constraint block, parametric diagram
- Modeling both discrete-time and continuous-time behaviour of cyber-physical systems
  - Modeling language for this purpose: Modelica
- Connecting models to study joint behavior
  - Simulation of models is especially useful when implementing and testing the system is expensive