SysML v1

Ansgar Radermacher / Asma Smaoui
ansgar.radermacher@cea.fr

- **Acknowledgments** – contains material from our CEA colleagues Shuai Li, Jérémie Tatibouët, François Terrier, Sébastien Gérard

Agenda – SysML v1

Motivation, history

SysML requirements

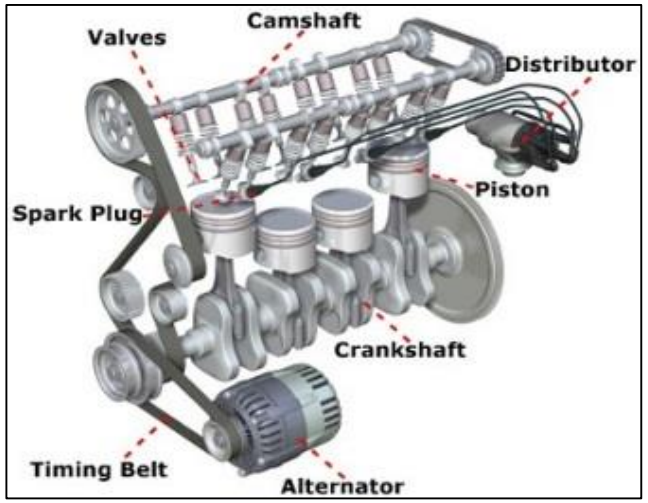
SysML composition structures

1. Block definition diagram (BDD)
2. Block internal structure diagram (IBD)

Exercise

Motivation

Need a modeling language to model **system engineering** aspects that can not be expressed with UML



Points of interest: piston diameter, cylinder volume, weight of components, etc.

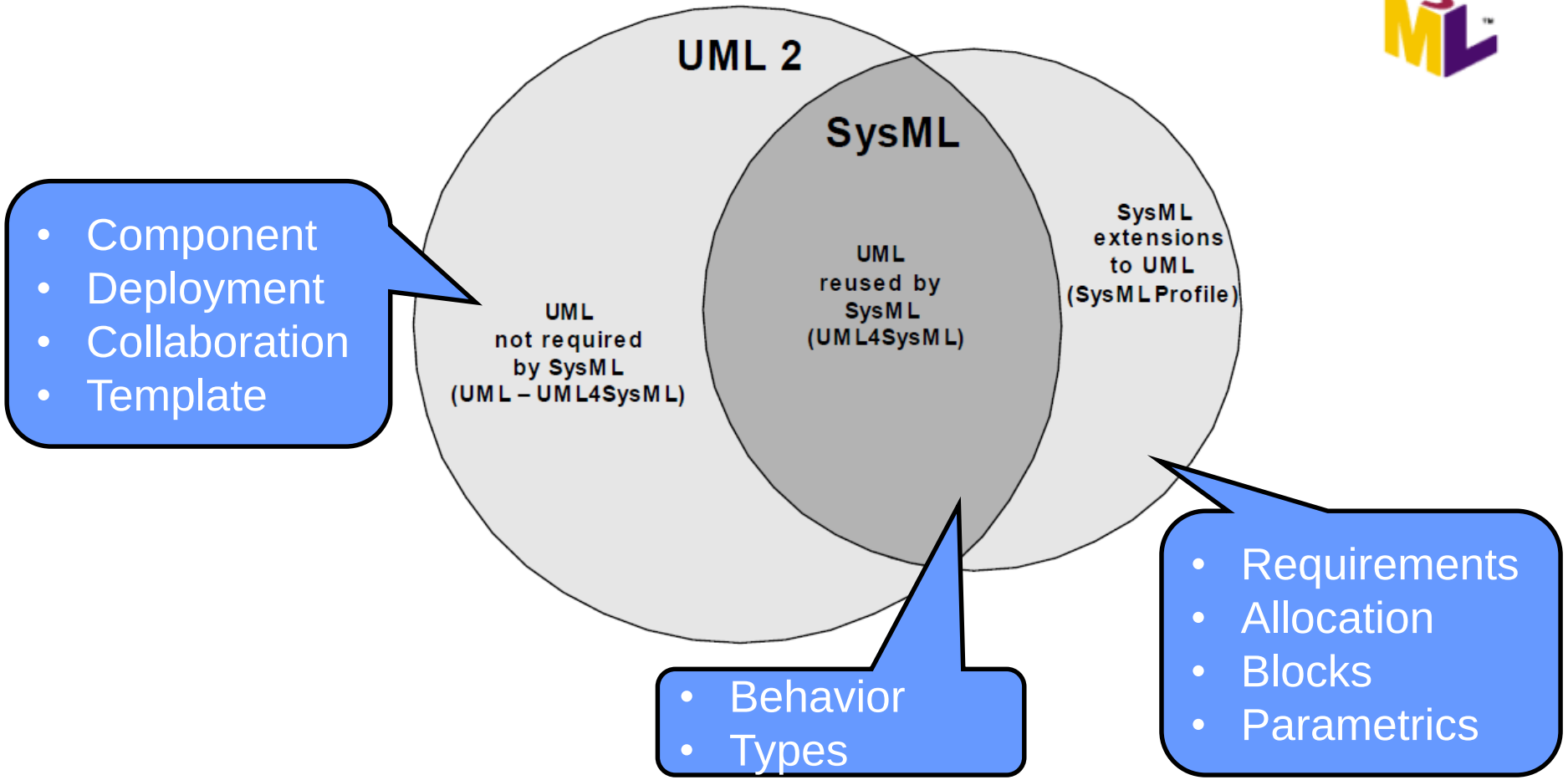
History

- 01/2001: International Council on Systems Engineering (INCOSE) start Model Driven Systems Design workgroup to customize UML for systems engineering applications.
- 07/2001: INCOSE and OMG charter Systems Engineering Domain Special Interest Group (SE DSIG)
- 03/2003: SE DSIG develops requirements for the modeling language => *UML for Systems Engineering Request for Proposal* (RFP)
- 2003: Joint response to UML for SE RFP, “Systems Modeling Language”, aka SysML, logo, language design team
- 01/2005: SysML v0.9 draft, *open source specification*

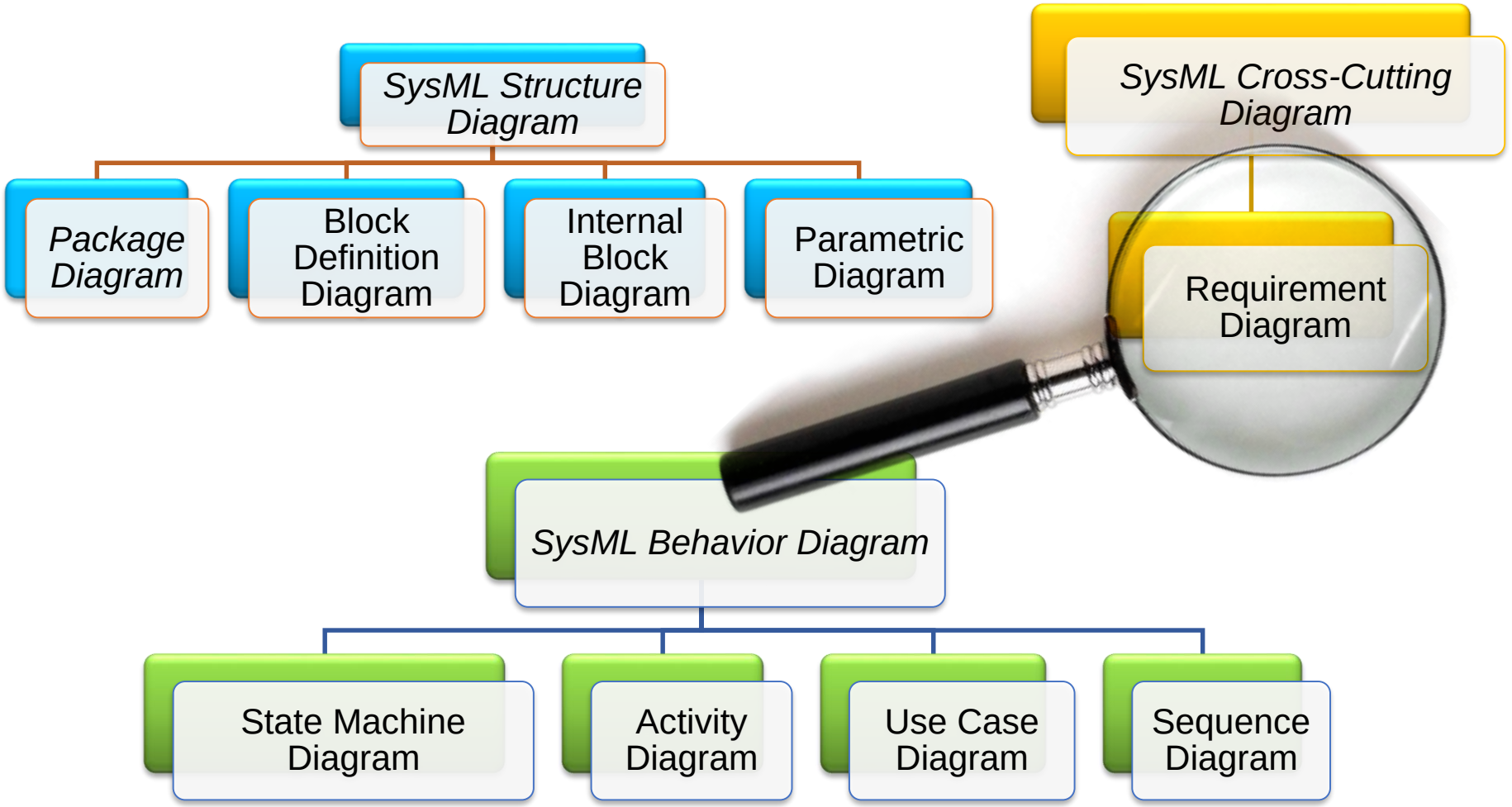
History

- 11/2005: SysML v1.0 alpha, but competing SysML specification proposals
- 07/2006: Merge Team was proposed and voted
- 01/2007: OMG SysML v1.0, as SysML is derived from open source SysML, it also includes an open source license for distribution and use.
- … several revisions
- 12/2019: OMG SysML v1.6
- 2017: Published by ISO as international standard, ISO/IEC 19514:2017 (Information technology – OMG systems modeling language)
- > 2018: OMG works on next generation and issues an RFP for **SysML v2**

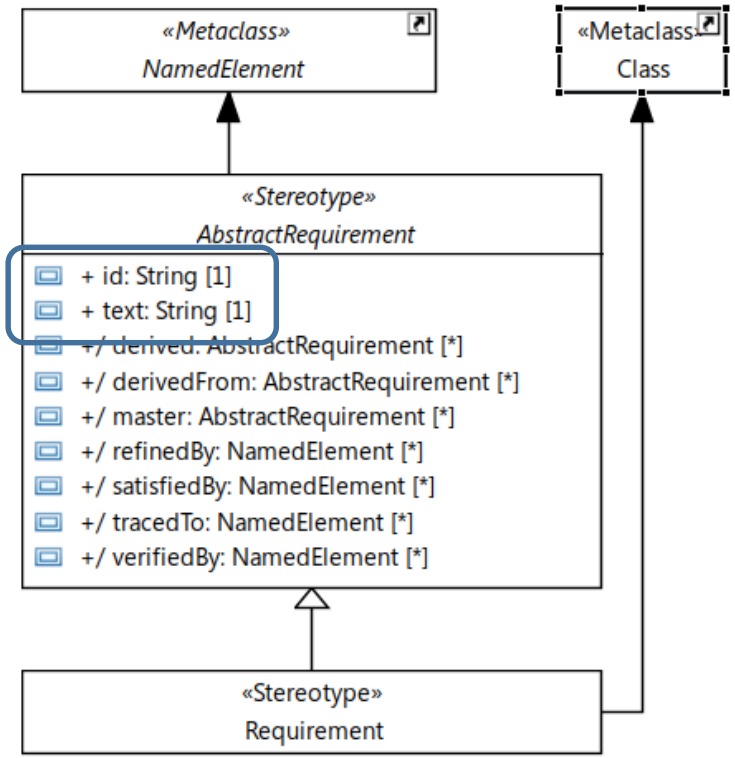
SysML: a UML specialization for system modeling



Block Definition Diagram



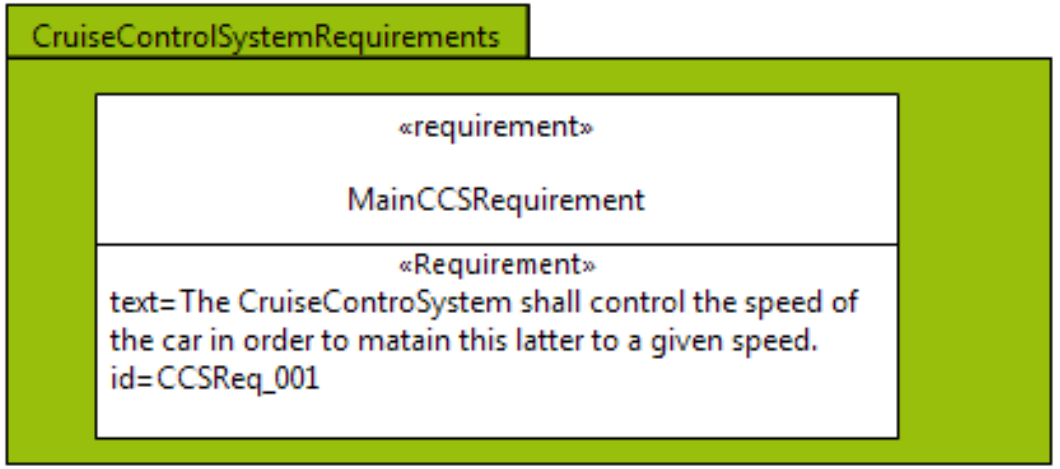
Details of the «Requirement» Concept



- Extends meta-class Class
- Inherits attributes from «AbstractRequirement»
- Properties
  - id = unique identifier (defined as a string)
  - text = requirement description

Profile Level

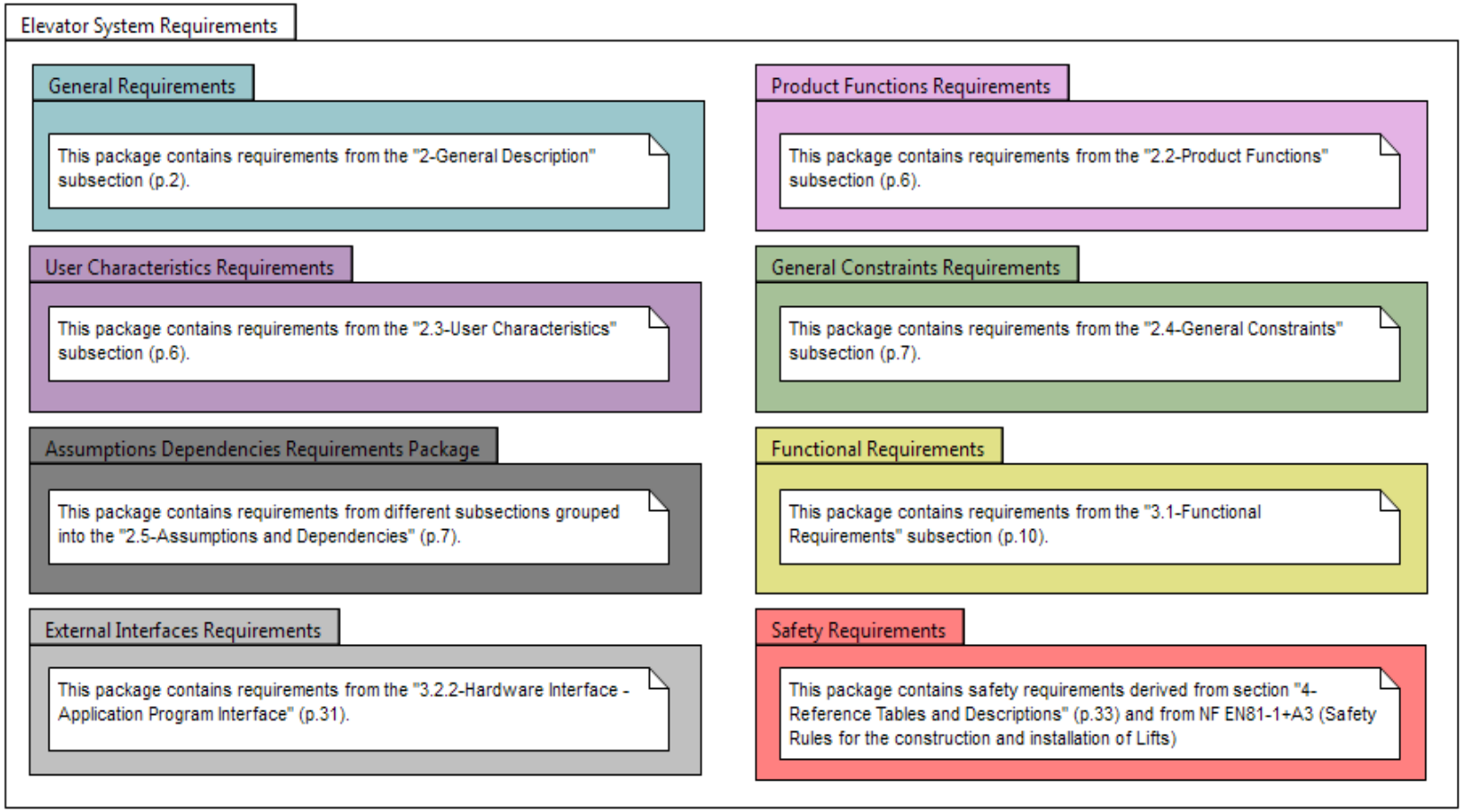
User Level



Organization/Structuration of Requirements

- Organize requirements in packages (hierarchical structuration).
  - Can import other packages or model elements in other packages

- Example



Concepts for Traceability

- **Traceability** includes several complementary notions [From Gotel, 1994]
  - Relations among requirements (behaviors refining requirements):

    Answer the question: "*How can a requirement evolve?"*

    Decomposition, **refinement**, **copy, derive**
  - Relations between the requirements and the system

    Answer the question: "*How can a system be defined to ensure requirements?"*

    Relations such as: **satisfy**
  - Relations between requirements and V&V:

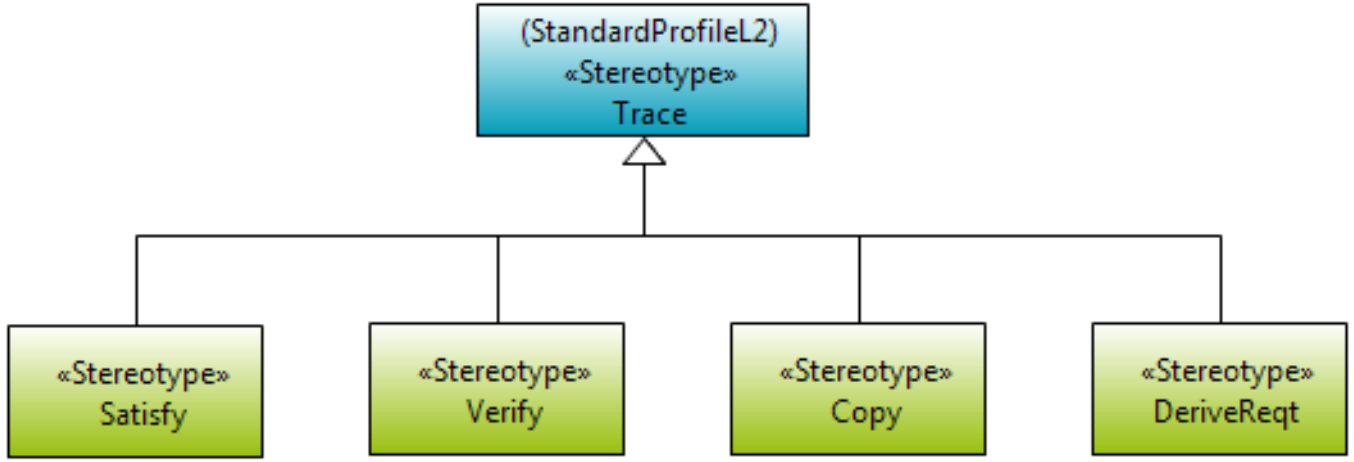
    Answer the question: "*How does a system meet its requirements?"*

    A system can be proved with respect to requirements: test cases, code review, …

    Relations such as **verify**

Requirements Traceability: «Trace» concept

Refine and decomposition already present in UML. New SysML stereotypes for satisfy, verify, copy and derive.



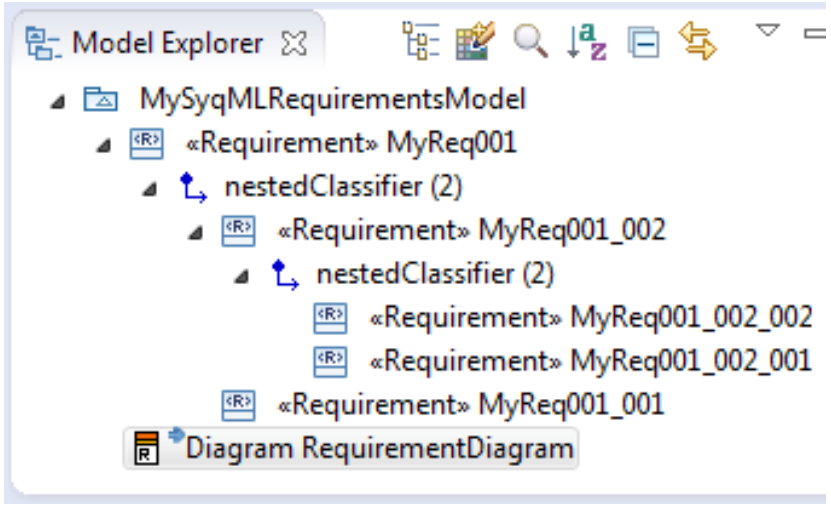
The generic «UML::trace» relationship has *very generic (i.e., weak) semantics!*

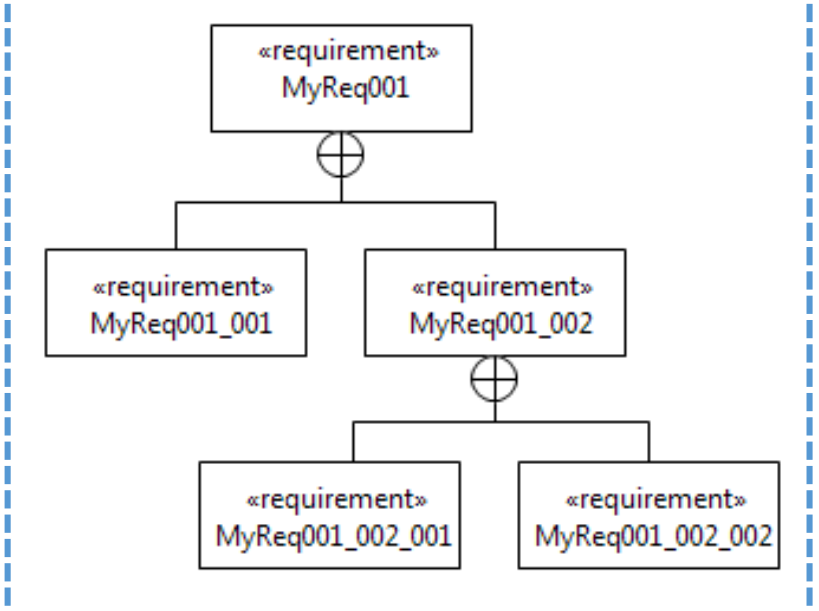
➔ Not recommended to use in conjunction with other SysML requirements relationships.

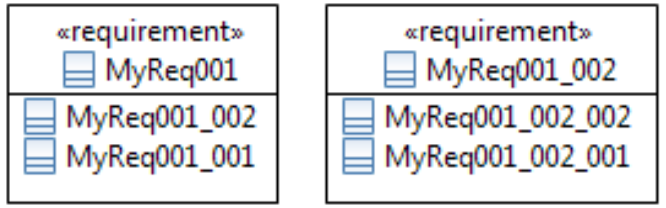
Decomposition

Composite requirements (i.e., hierarchical description)

- Use UML namespace containment mechanism.







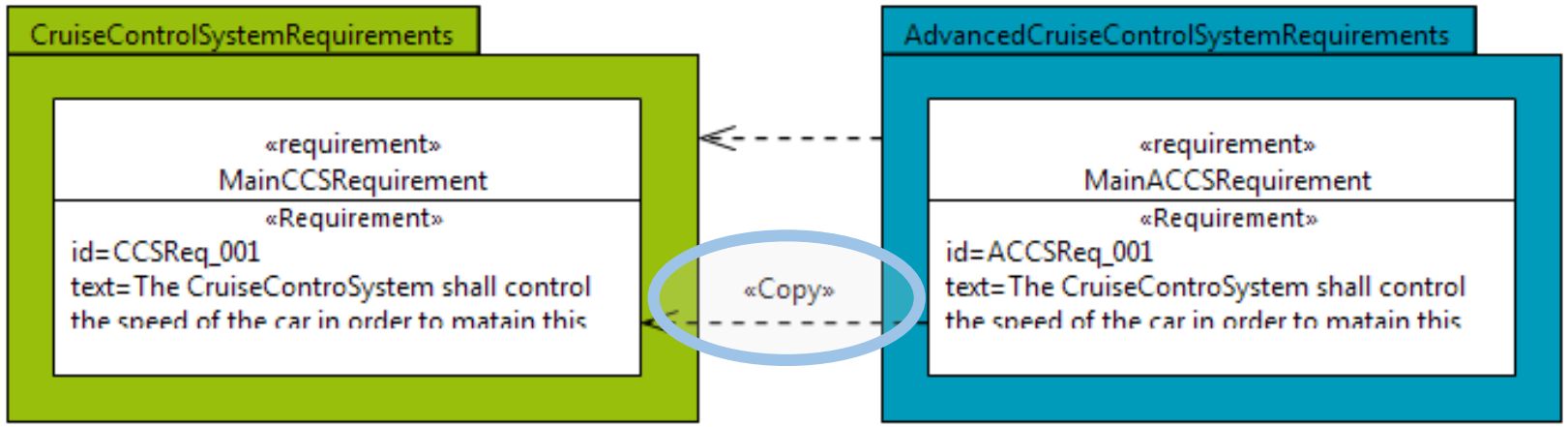
Requirements reuse: the «Copy» concept

The use of namespace containment to specify requirements hierarchies precludes reusing requirements in different contexts since a given model element can only exist in one namespace!

Slave requirements for enabling reuse

- A **slave** requirement is a requirement whose text property is a read-only copy of the text property of a master requirement.
- The master/slave relationship is denoted via a «Copy» dependency.

Example



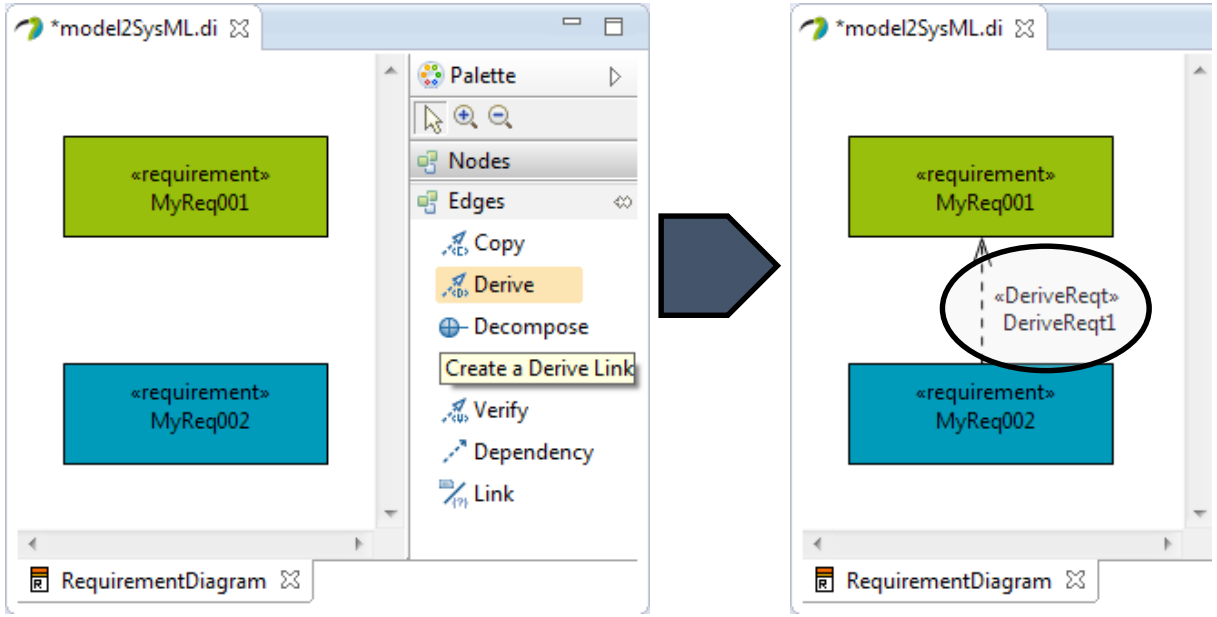
Requirements reuse: the «DeriveReq» concept

Usage

- Relates a derived requirement to its source requirement.

Example

- Vehicle **acceleration** requirement implies derived requirements for engine power, vehicle weight…



Alternative tabular notation:

	name	derived	derivedFrom
1	MyReq001	«Requirement» MyReq002	
2	MyReq002		«Requirement» MyReq001

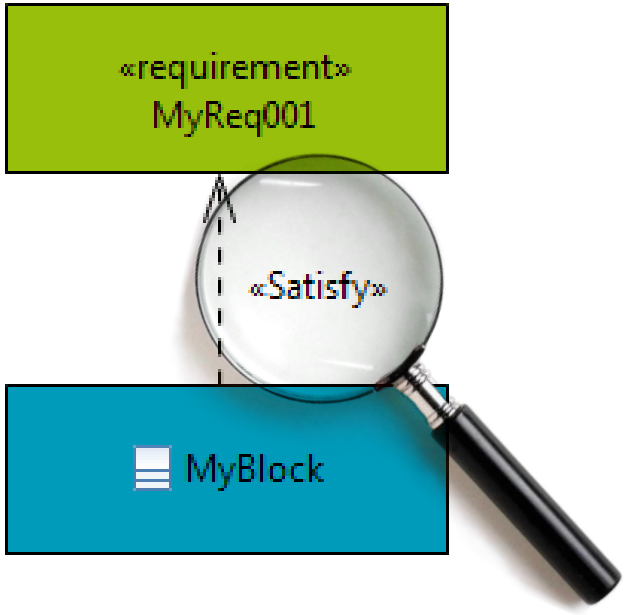
Satisfy and Refine

«Satisfy» Dependency

- Dependency from design or implementation model element to the requirement that it satisfies.

Example

- A block of the design model satisfies a requirement.

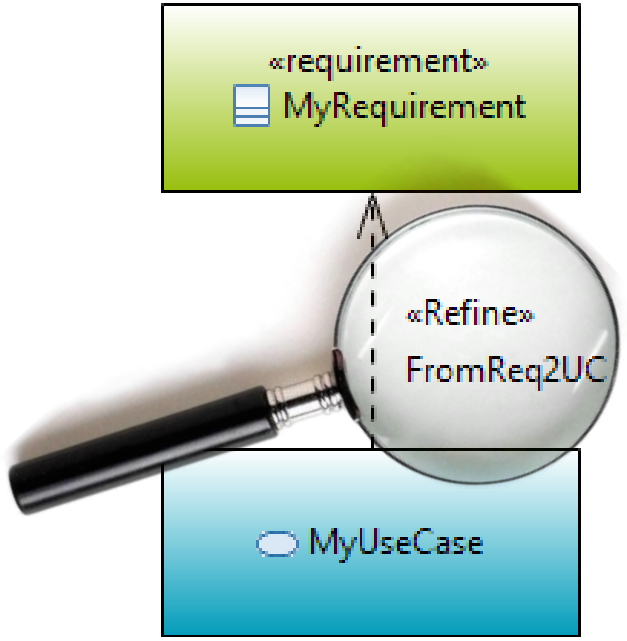


«Refine» Abstraction

- Denote how a model element or set of elements can be used to further refine a requirement.

Example

- Use case used to refine a requirement.



Requirements V&V: the «Verify» and «TestCase» concepts

«Verify» Dependency

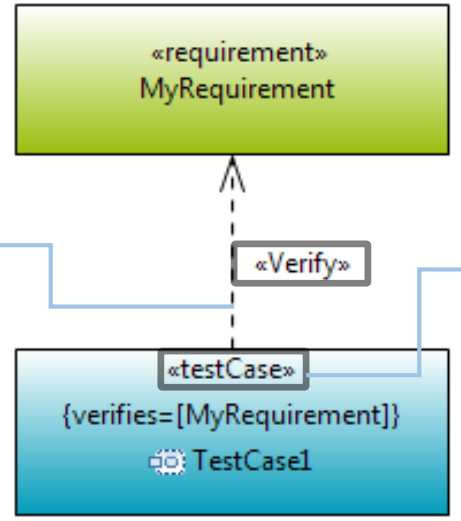
- Used for defining how a test case or other model element verifies a requirement.

«TestCase»

- Represent standard verification method for inspection, analysis, demonstration, or test.
- Extends UML::Operation and UML::Behavior.
- May be used to integrate both SysML and the UML testing profile (http://utp.omg.org/).

Modeling in Papyrus

Model a «Verify» relationship from the model element verifying the requirement (e.g., a test case) and the requirement.

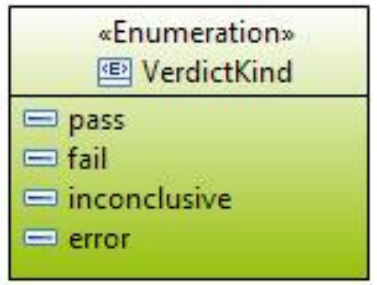


Apply the stereotype «TestCase» to a behavior (e.g., an activity, interaction…) or an operation.

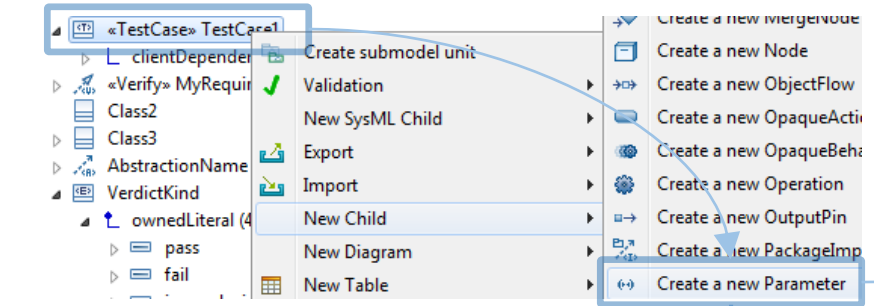
Requirements V&V (Seq.): the VerdictKind concept

VerdictKind (Enumeration)

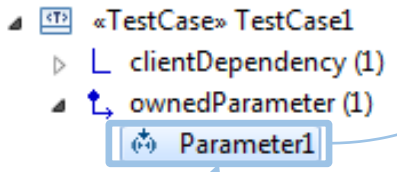
- Possible literals: pass, fail inconclusive or error
- Used to type the return values of the test case specification

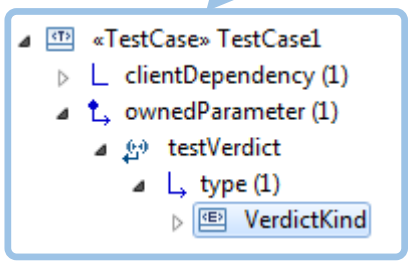


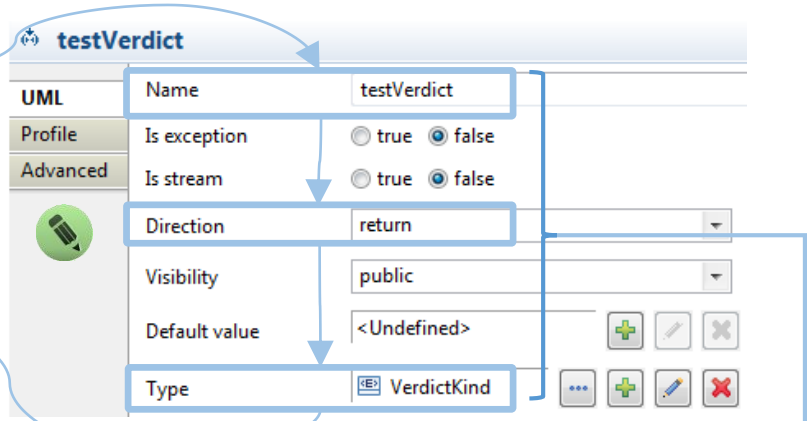
Modeling in Papyrus



Add a parameter to the behavior.







Set a name, select the return direction, set the type to be VerdictKind

«RequirementRelated» concept

Usage

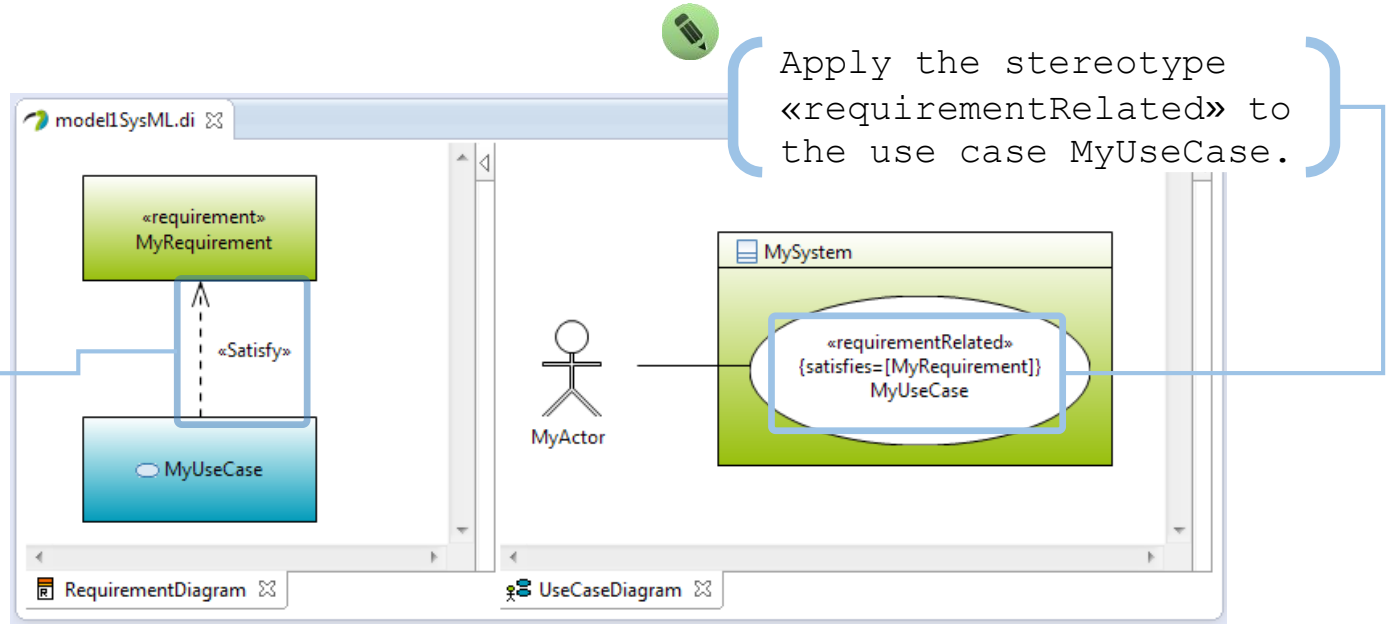
- Used to add properties to those elements that are related to requirements via the various SysML requirements relationships (e.g., verify and refine).
- Can be applied on UML::NamedElement (i.e., almost on all UML model elements)

Derived properties

- / tracedFrom: Requirement [*]
- / satisfies: Requirement [*]
- / refines: Requirement [*]
- / verifies: Requirement [*]

Modeling in Papyrus

Model a «Satisfy» relationship from the use case to the requirement.

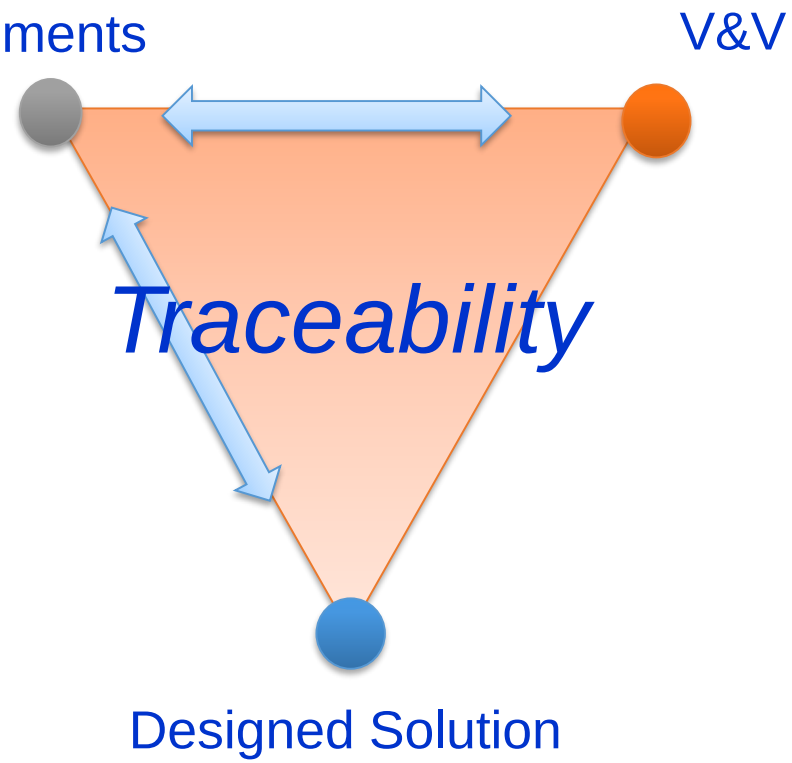


Apply the stereotype «requirementRelated» to the use case MyUseCase.

Requirement TRACEABILITY

- ✓ Requirements ⟺ Designed Solution
  Traceability between requirements and the designed solution demonstrates that the solution satisfies user needs.

- ✓ Requirements ⟺ V&V
  Traceability between requirements and test cases shows which requirements are covered by tests. Tests are a means to verify requirements.



Sample driver assistance system (ADAS)

- CruiseControl: keep set speed
  - Activate via set button
  - Deactivate via cancel button or brake pedal
- EmergencyBreak
  - Execute emergency break if obstacle (e.g. pedestrian) is detected within n meters (5 seconds x current speed)
- Environment
  - Speed sensor,
  - Pedals (acceleration & break)
  - ObstacleDetector: distance to obstacle (if detected)

Hands-On: SysML requirements

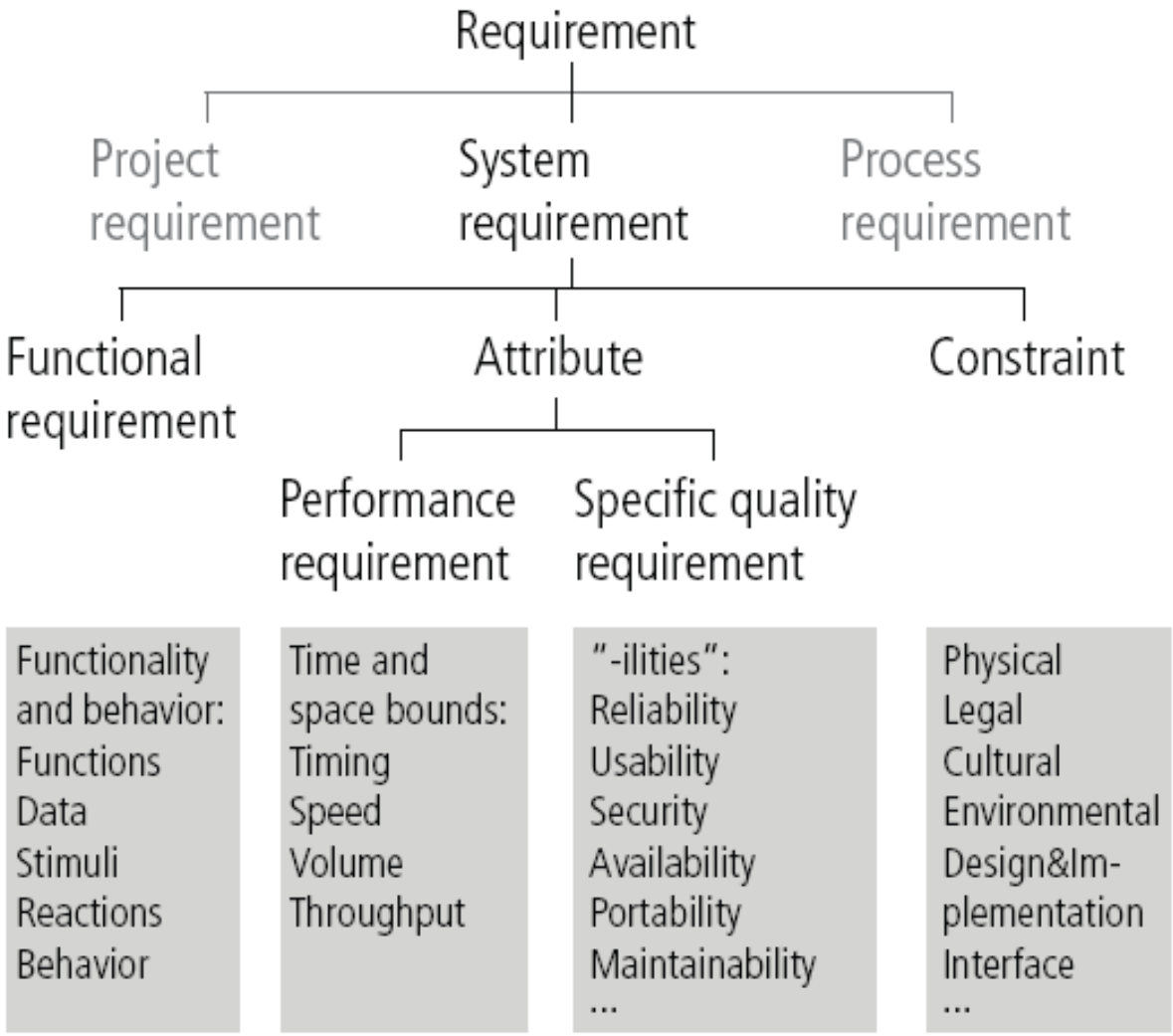
- Your turn
  - Capture the requirements using a SysML requirements diagram.
  - Identify the requirements and decompose them in logically independent requirements.
  - Should the requirements be semantically related to each other, use appropriate relationships (containment, derived requirements…).
  - Should you have to motivate some requirements definitions, use the Rationale stereotype

- Do it in Papyrus
  - Create a package “Requirements” at the root of the model and a requirement diagram in this package.
  - Create a requirement table corresponding to these requirements.

Hands-On: extend SysML requirements

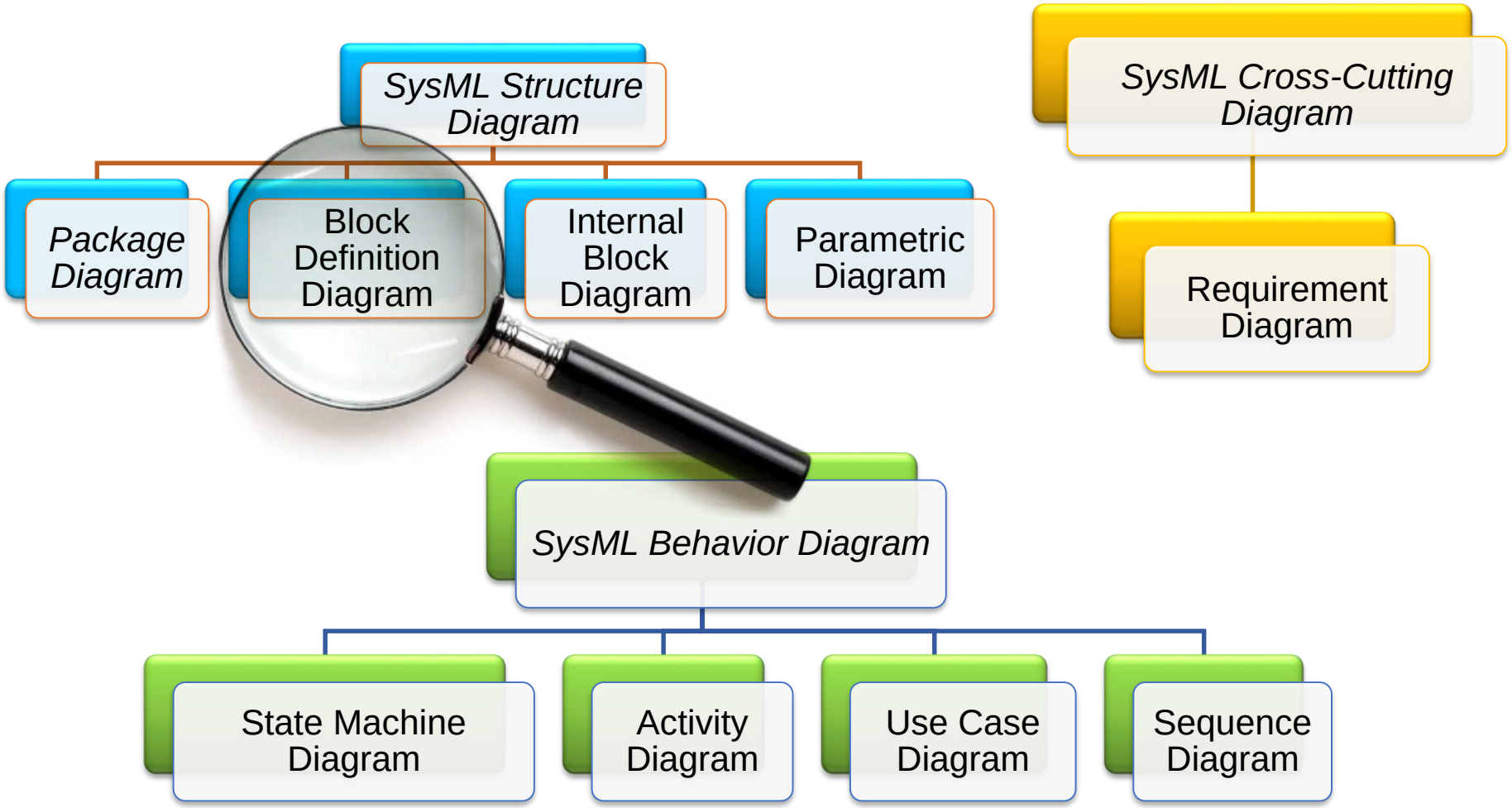
- Create a profile to capture the classification of the following slide
  - Create a Papyrus project « RequirementProfile »
    (choose Profile in the language wizard)
  - Create stereotypes for every class (FunctionalRequirement …) – extend the SysML::Requirements::Requirement stereotype
  - Create an Enumeration for every class and add literals (function, behavior…)
  - In each stereotype create a property "kind" typed with the corresponding Enumeration
  - Save the profile to define it.
  - Apply the profile to the former model and apply appropriate stereotypes to requirements.

Further Requirement classification



Extract from M.Glinz. *On Non-Functional Requirements*. Proc. of the 15th IEEE Int. RE Conference, 2007.

Block Definition Diagram



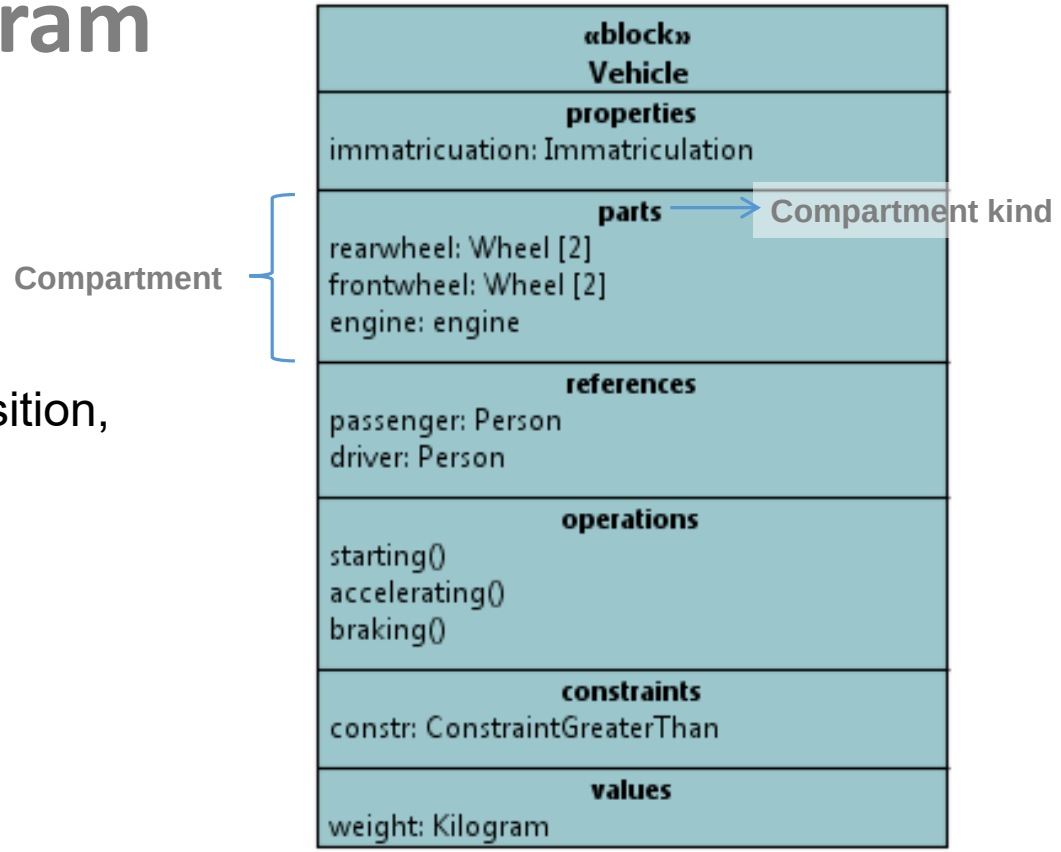
Block Definition Diagram

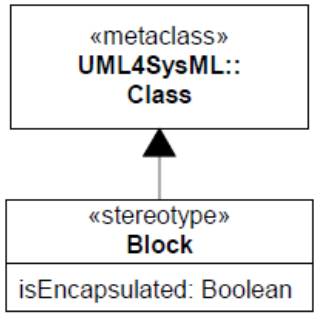
Block Definition Diagram (BDD)

- Represent blocks…
- …their properties and their relationships (decomposition, aggregation, generalization)

Block

- Extension of the UML meta-class Class.
- Basic entity, no restriction on its nature (hardware, software)
- Definition of a type, reusable in multiple contexts
- Notation: inside a BDD a block is represented by a rectangle divided in compartments (see figure). The only obligatory compartment is the compartment for the block name.





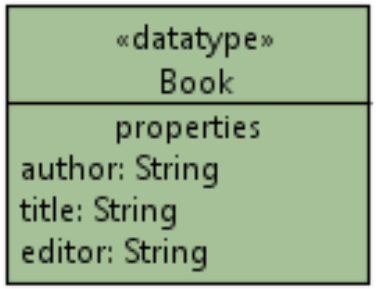
Basic Types

PrimitiveType

- No properties / no operations

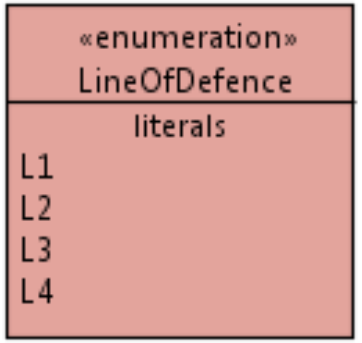
DataType

- Structured type
- Properties
- May have operations



Enumeration

- Finite number of possible values (literals)
- No properties / no operations



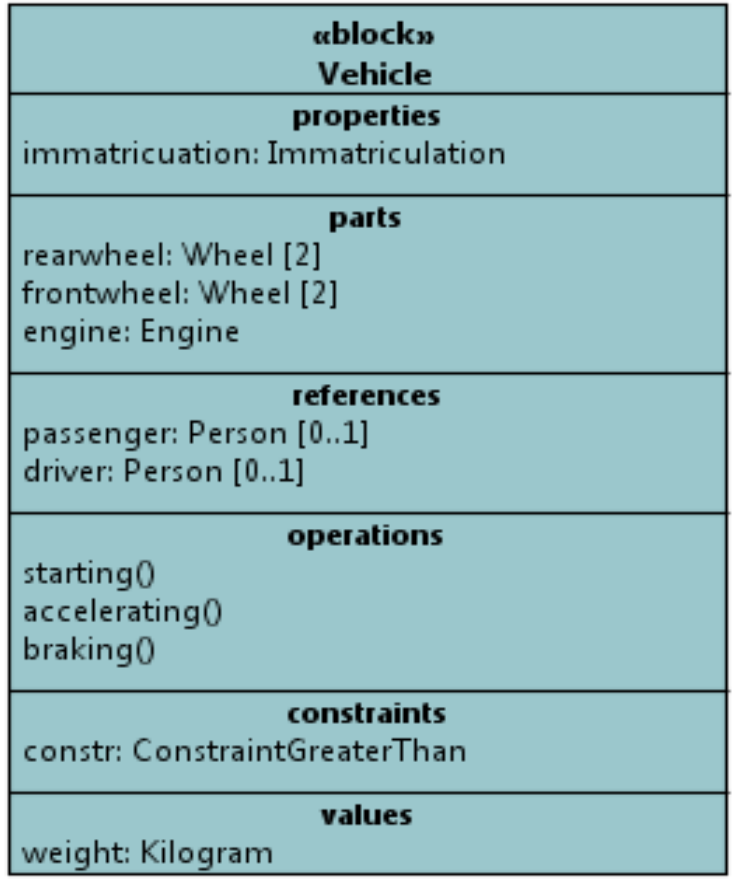
Block FEATURES

Block extends Class so it has…

- Properties
- Operations

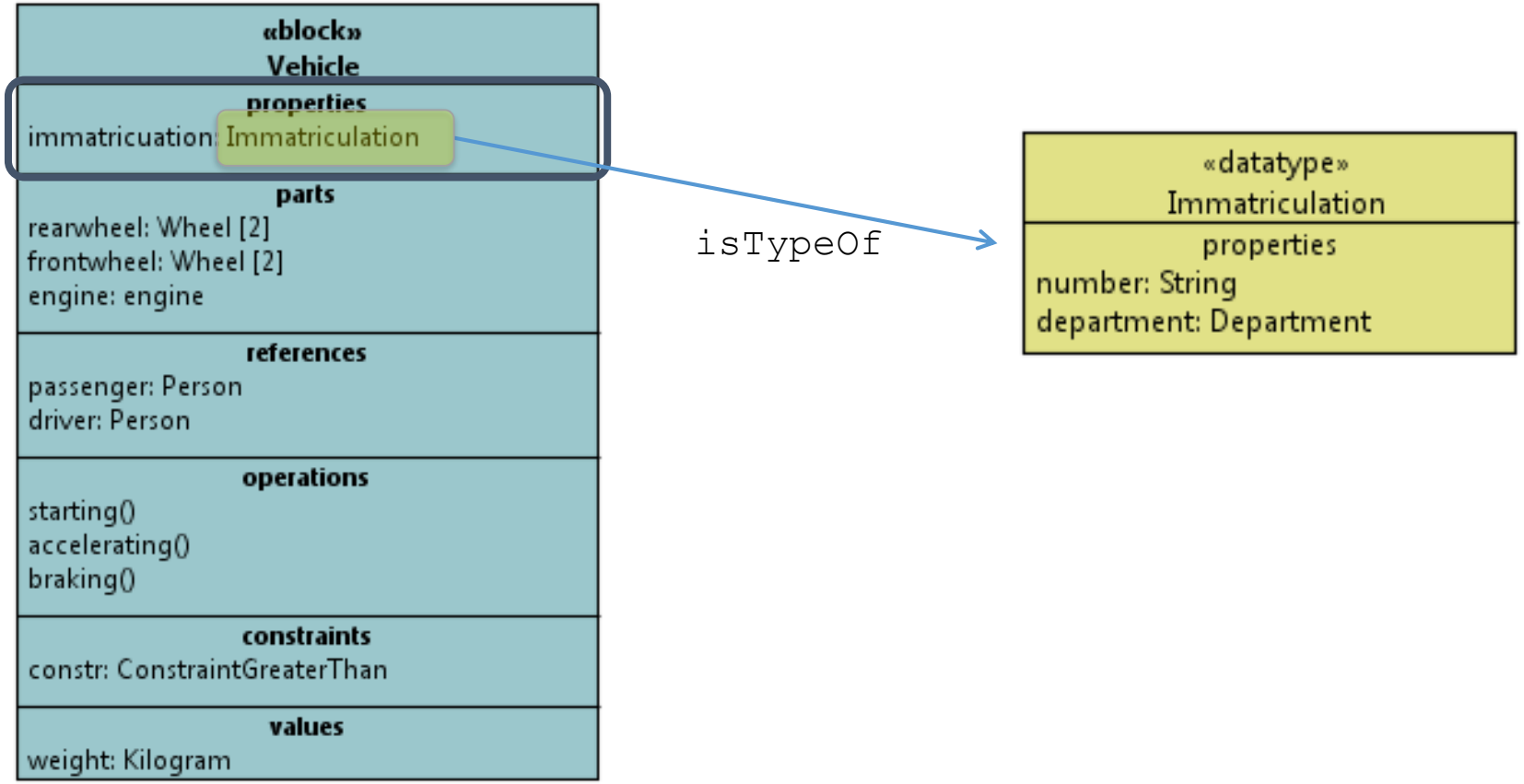
Block specializes properties

- Part properties
- Reference properties
- Constraint properties
- Values properties



Block FEATURES

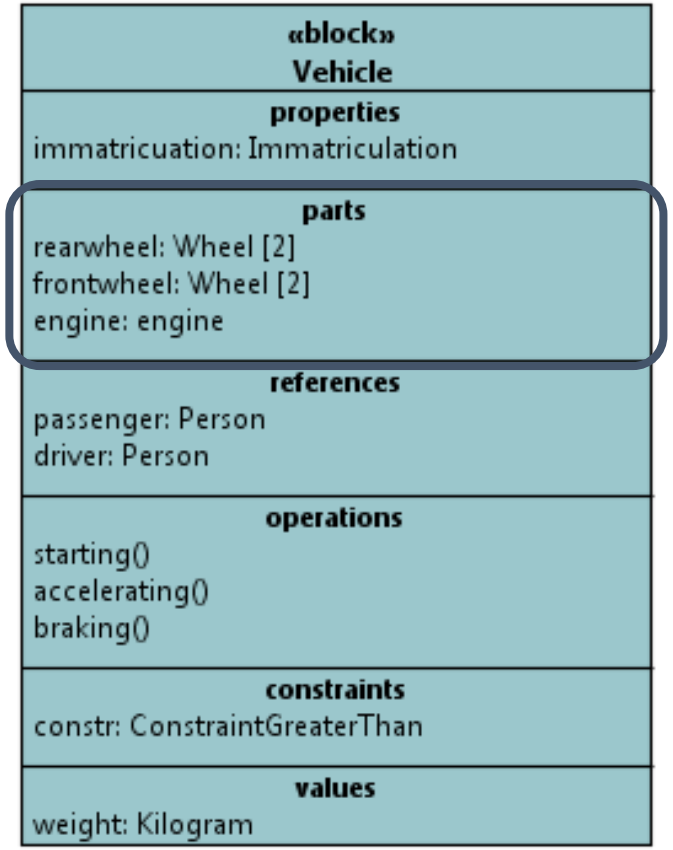
- **Simple properties: always typed properties**



Block FEATURES

As in standard UML

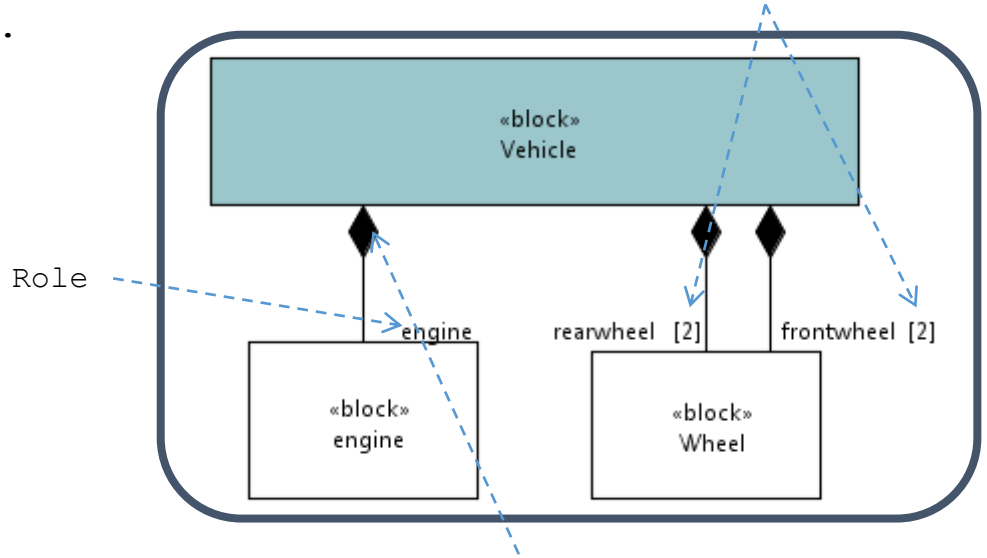
- **Part properties describe composition**



Composition: used to model decomposition of blocks (containment relationship)

- Specifies a multiplicity
- Specifies a role

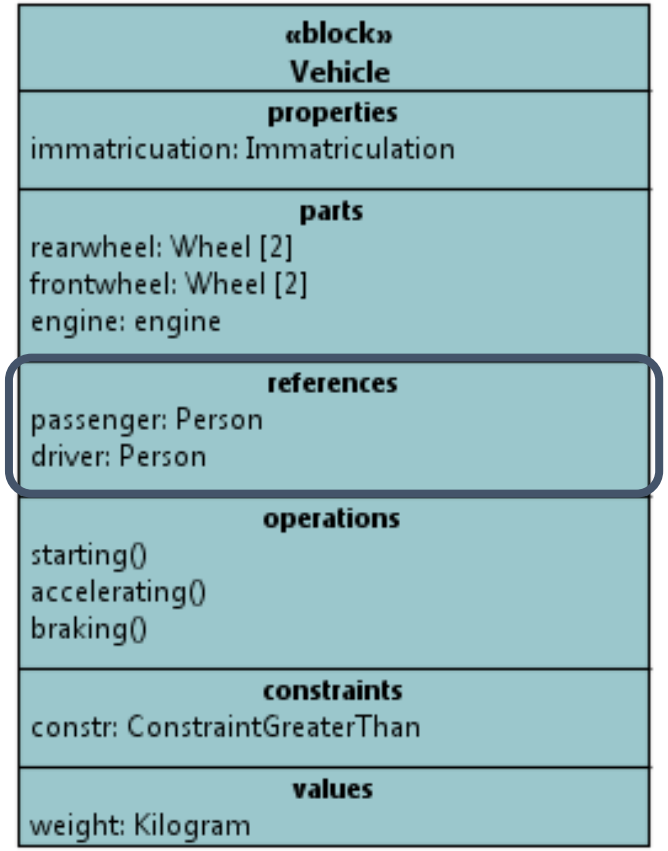
Multiplicity specifies, in the form of an interval, the number of instances of a block that can be contained in an instance of another block. Here, Vehicle has 2 rear wheels and 2 front wheels.



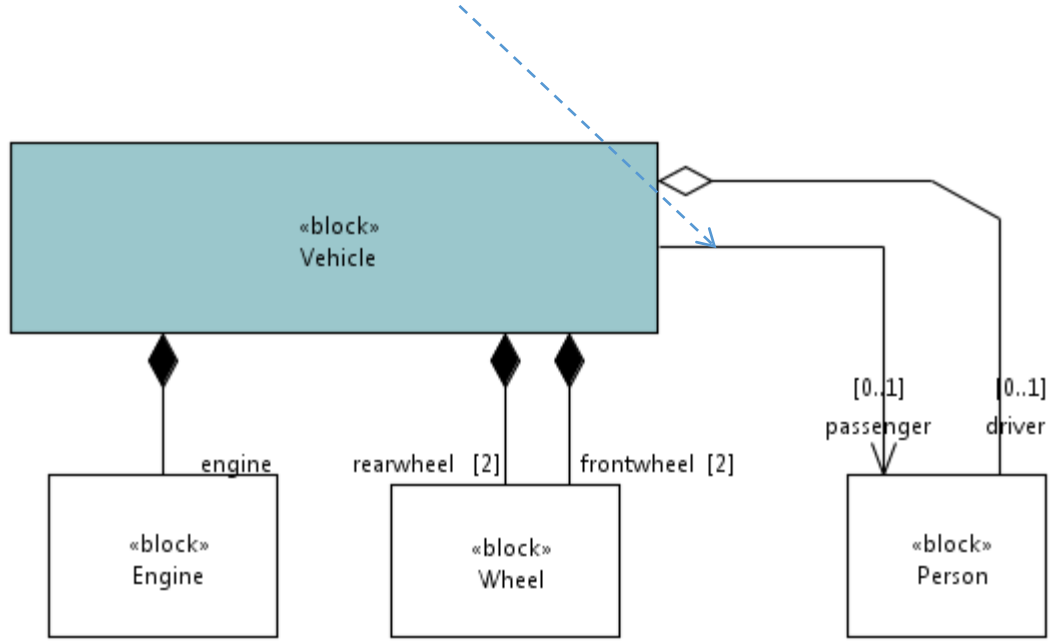
Filled diamond: if an instance of Vehicle is destroyed, the contained instances of Engine is also destroyed.

Block FEATURES

As in standard UML

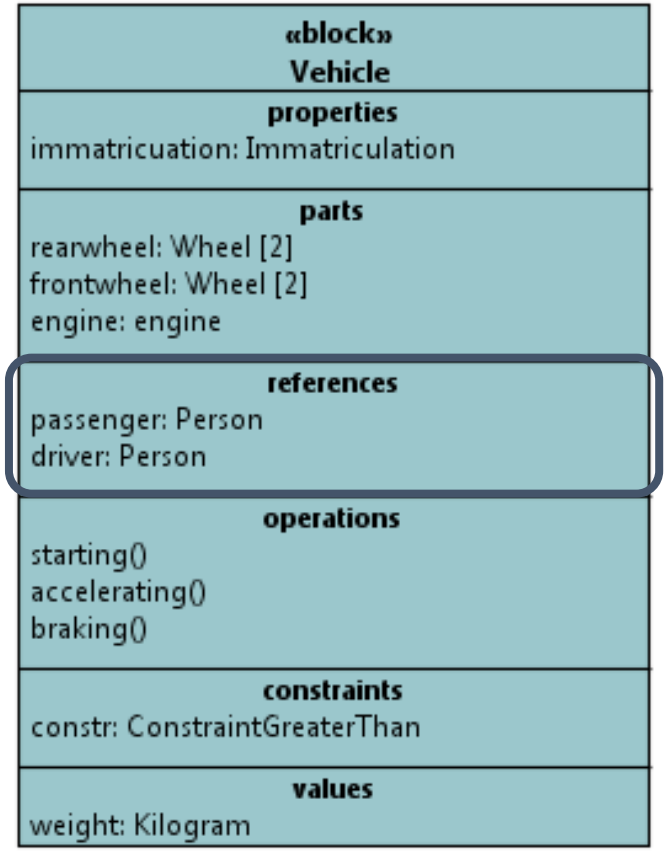


Association: simple reference (no containment relationship between blocks)



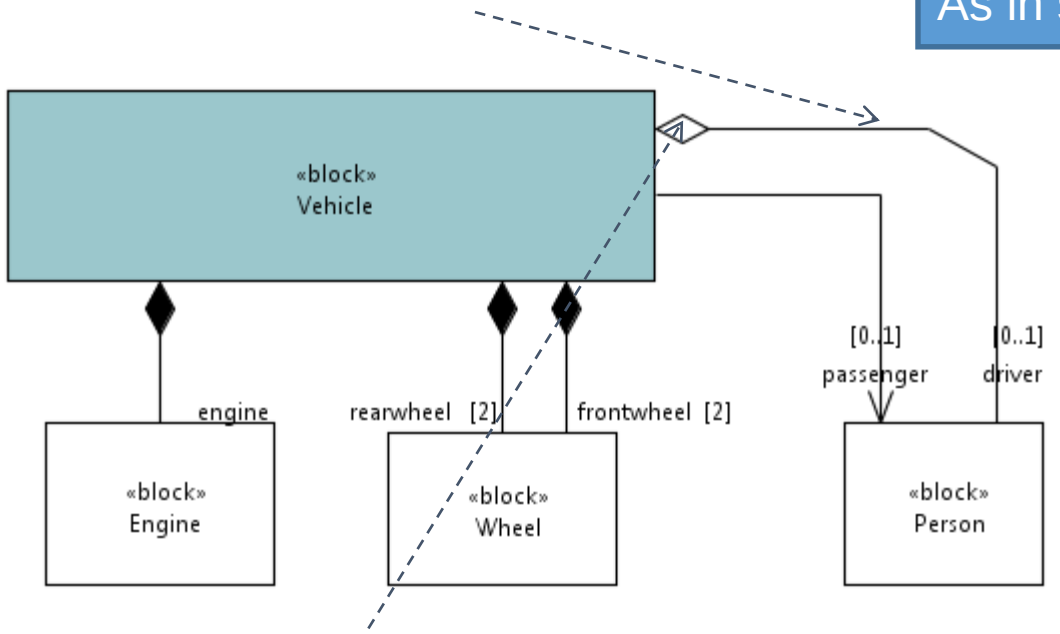
Block FEATURES

Part properties: describe associations and simple aggregations



Aggregation: kind of virtual "composition" (notion of containment), i.e. the referenced block is shared with other blocks

As in standard UML

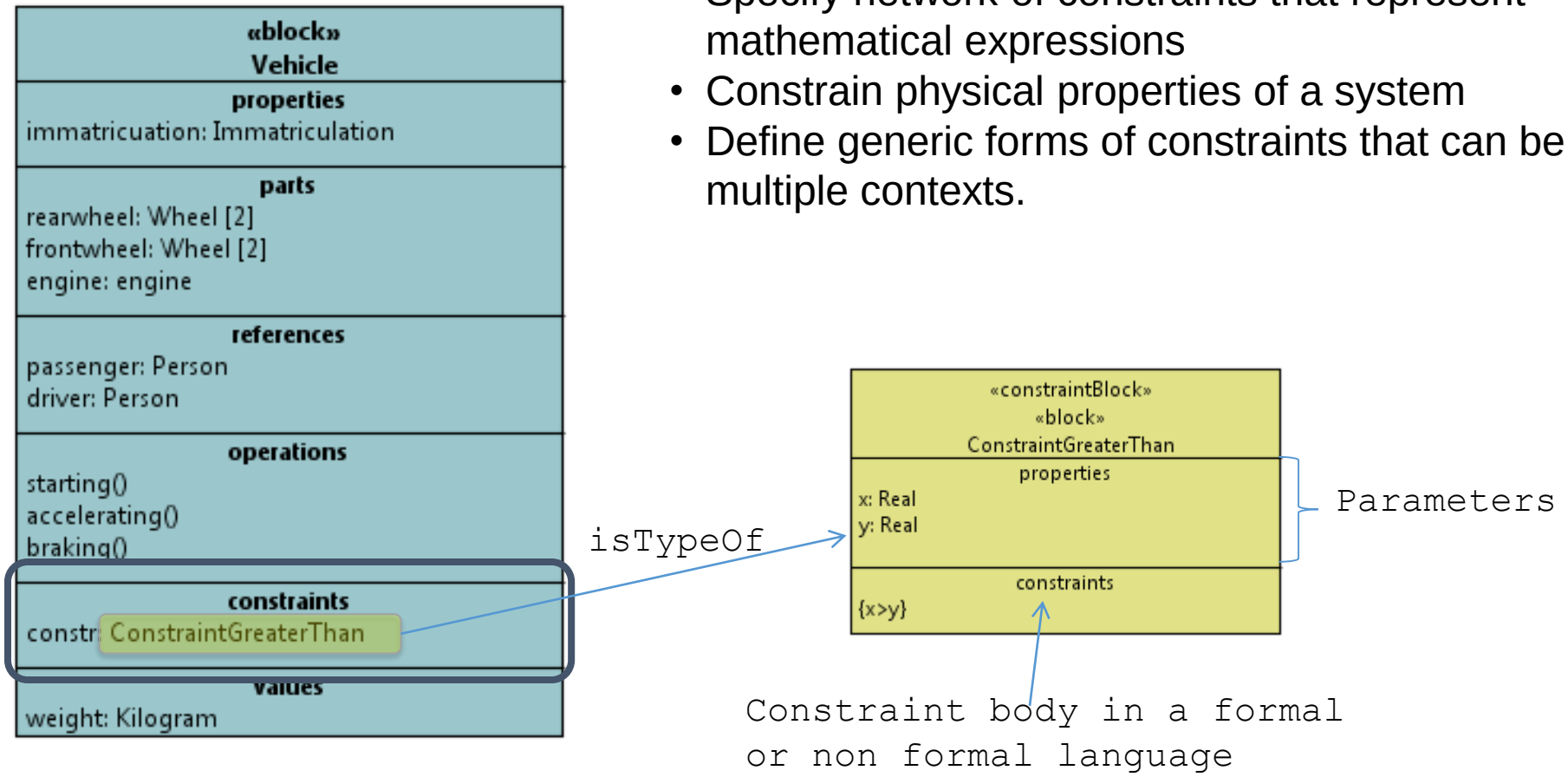


Empty diamond: if an instance of Vehicle is destroyed, the contained instances of Person are not destroyed.

(Hopefully ☺)

Block FEATURES

Constraint properties: typed by constraint block



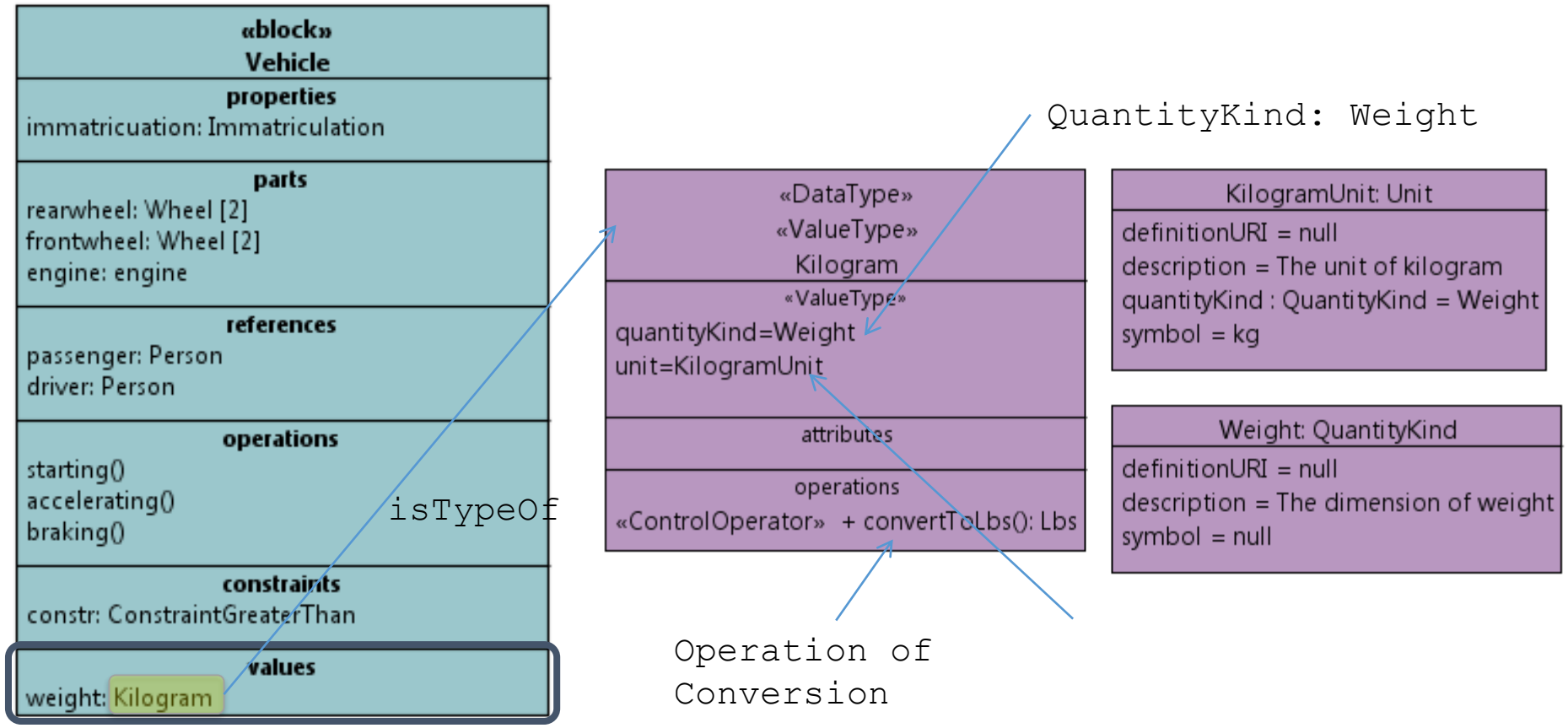
Constraint block:

- Specify network of constraints that represent mathematical expressions
- Constrain physical properties of a system
- Define generic forms of constraints that can be used in multiple contexts.

Block FEATURES

Constraint properties: typed by a ValueType

ValueType: describes the type of values; may have an associated Unit and Dimension

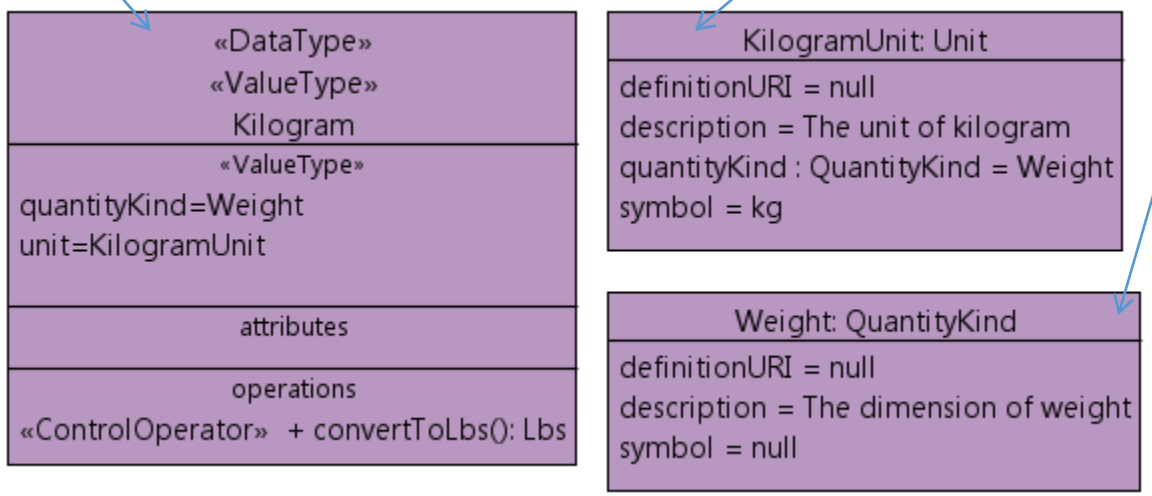


Dimensions and Units

- QuantityKind: Kind of quantity that may be stated by means of defined units. For example, the quantity kind ‘length” may be measured by units of meters, kilometers, or feet.
- Unit: Quantity in terms of which the magnitudes of other quantities that have the same dimension can be stated.

<<ValueType>> applied on a DataType

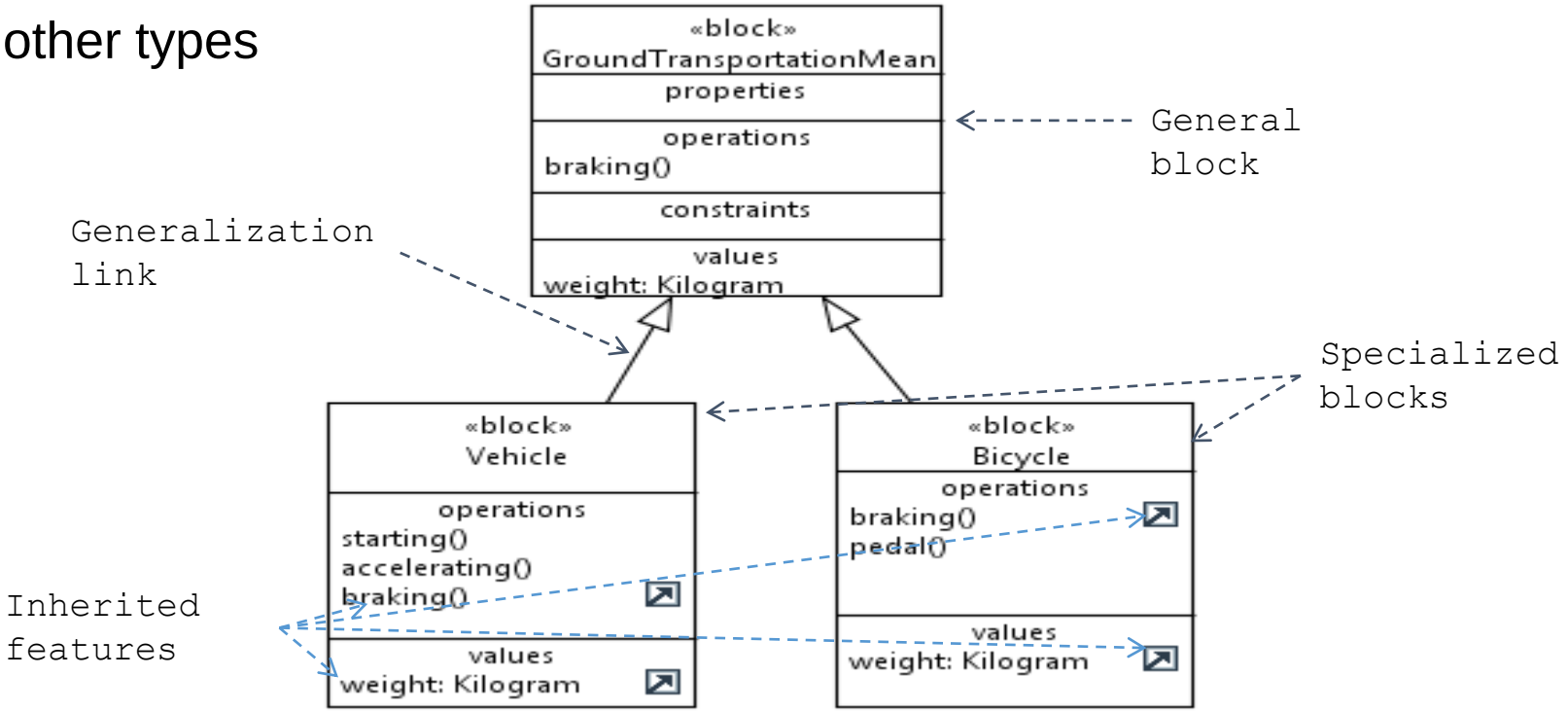
InstanceSpecifications of blocks “Unit” and “QuantityKind” defined in SysML model libraries.



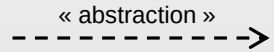
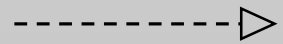
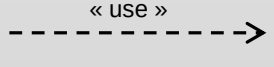
Generalization

As in standard UML

- Like a Class a Block may be a specialization of other Block(s)
- Specialized Blocks inherit features (properties, operations) and add their own
- Each instance of the specific block is also an instance of the general block
- Means for factorization of features (reuse)
- Reminder: works also for other types (except for Enumeration)



Other Relationships

Name	Description	Notation
Dependency	Relationship meaning that a single or a set of model elements requires other model elements for their specification/implementation.	
Abstraction	Relationship that relates two elements or sets of elements that represent the same concept at different levels of abstraction.	
Realization	Specialization of abstraction relationship between two sets of model elements, one representing a specification (the supplier) and the other representing an implementation of the latter (the client).	
Usage	Relationship in which one element requires another element (or set of elements) for its full implementation or operation.	

Hands-On: Block Definition Diagram

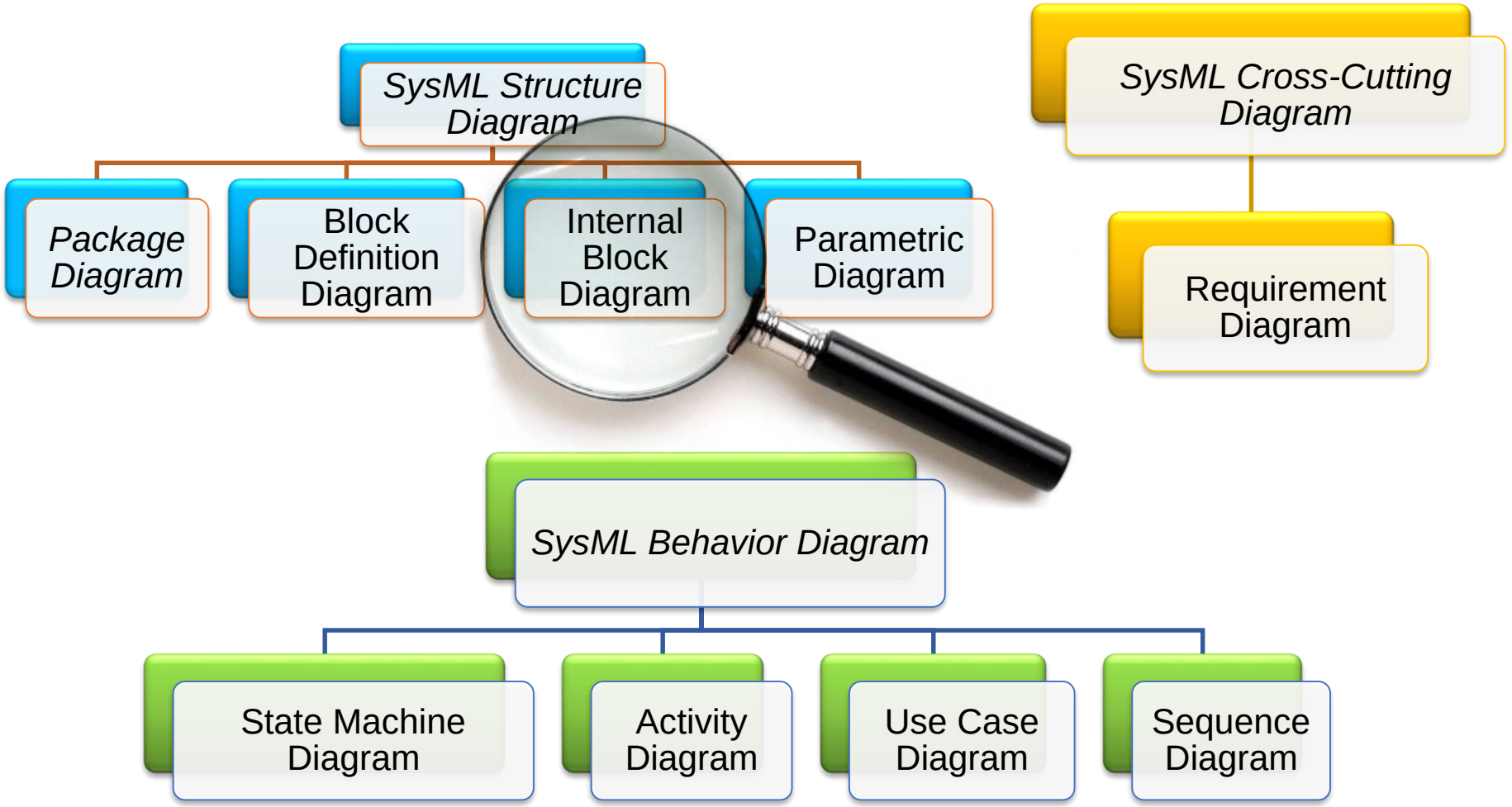
Your turn

- Design the architecture of the system.
- The objective is to identify the different parts of the systems and to describe how they are related to each other.

Do it in Papyrus

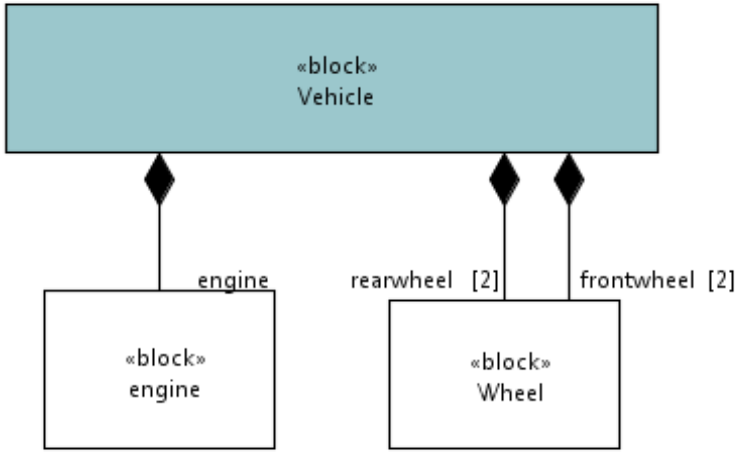
- Create a package “Architecture” at the root of the model and create a BDD in this package.
- The root of the architecture is “ADASsystem”, it contains two parts: CruiseControl and EmergencyBrake.
- Further refine this architecture.

Block Definition Diagram

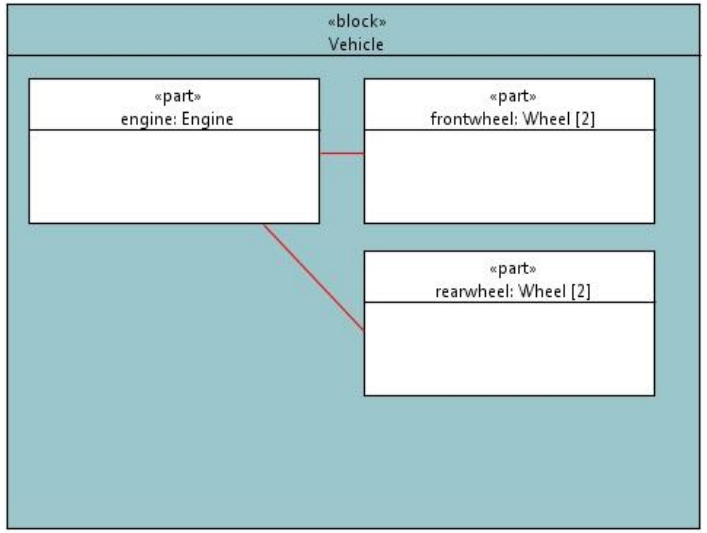


Internal Block Diagram (IBD)

- Describe internal structure (architecture) of a block
- Quite similar to composite structures in standard UML
- BDD: defined block properties (part, references, etc)



BDD ⟺ IBD



(1) different view, in which properties are represented as squares inside the Block

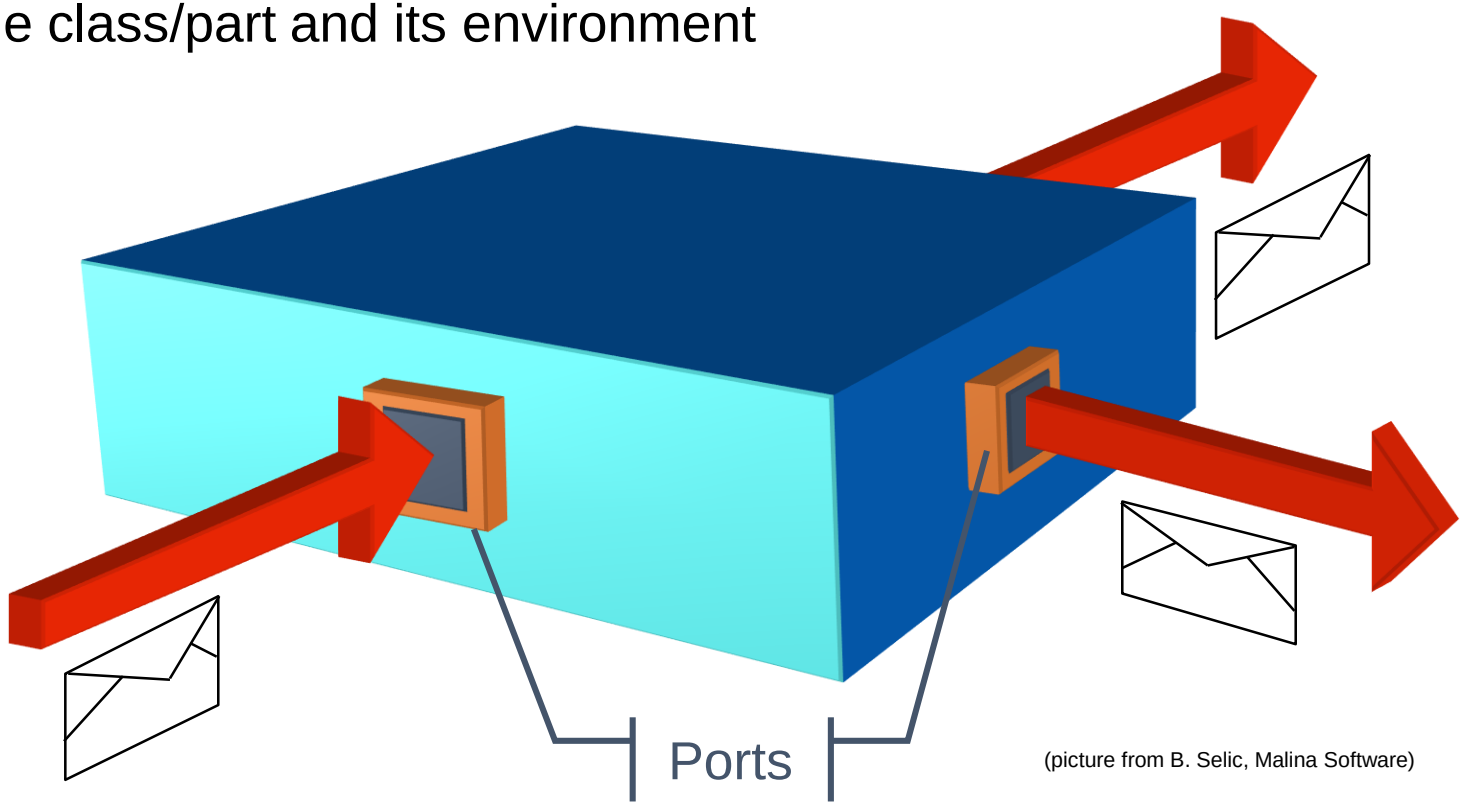
(2) add information about logical or physical wires (connectors) connecting these properties

IDB Ports

- *Services* are provided and required through standard UML Ports.
- Ports specify services the owning block provides (offers) to others and services that the owning block expects (requires) from others
- *Flows* are produced and consumed through SysML Flow Ports.
- A flow port specifies the input and output items that may flow between a block and its environment.
- Flow ports are interaction points through which **data, material, or energy can enter or leave the owning block**.
- The specification of what can flow is achieved by typing the flow port with a specification of things that flow.

Ports

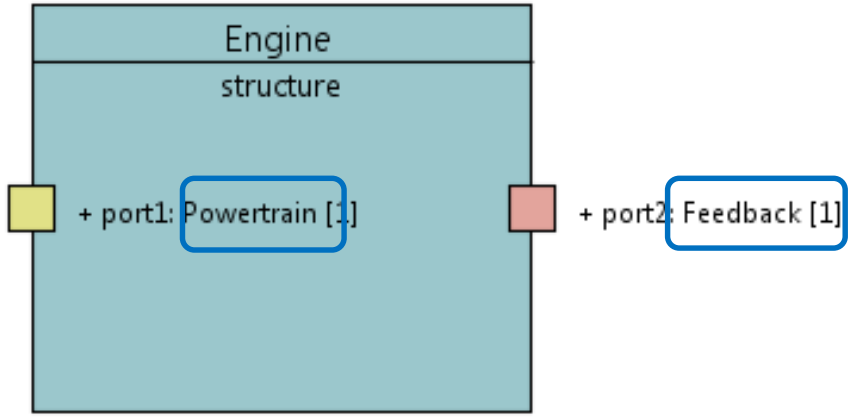
- Ports are interaction points
  - Between the class/part and its internal structure
  - Between the class/part and its environment

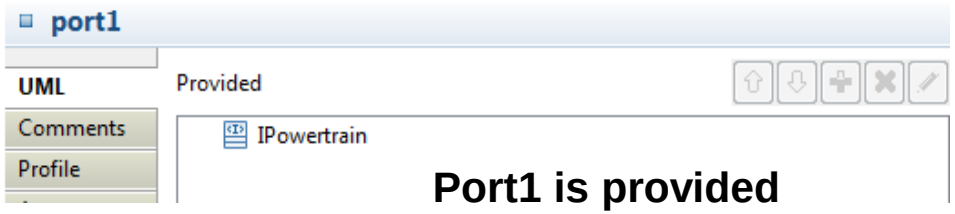


(picture from B. Selic, Malina Software)

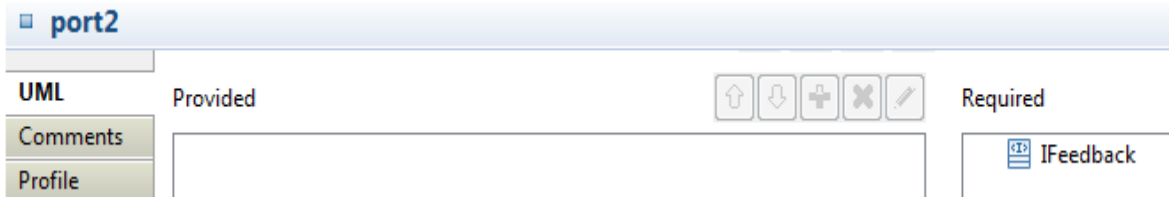
Required/Provided Services (Standard Ports)

The Engine block has two ports typed by Powertrain and Feedback.



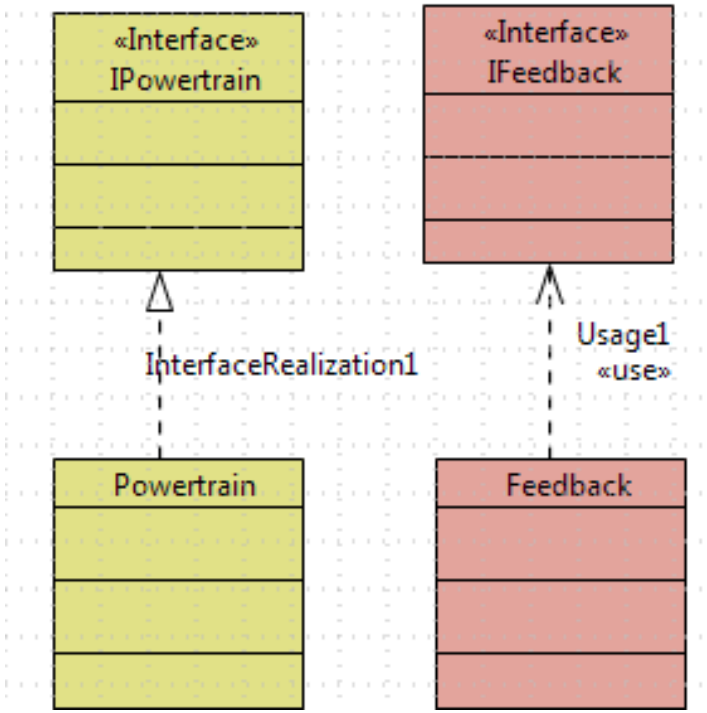


Port1 is provided



Port2 is required

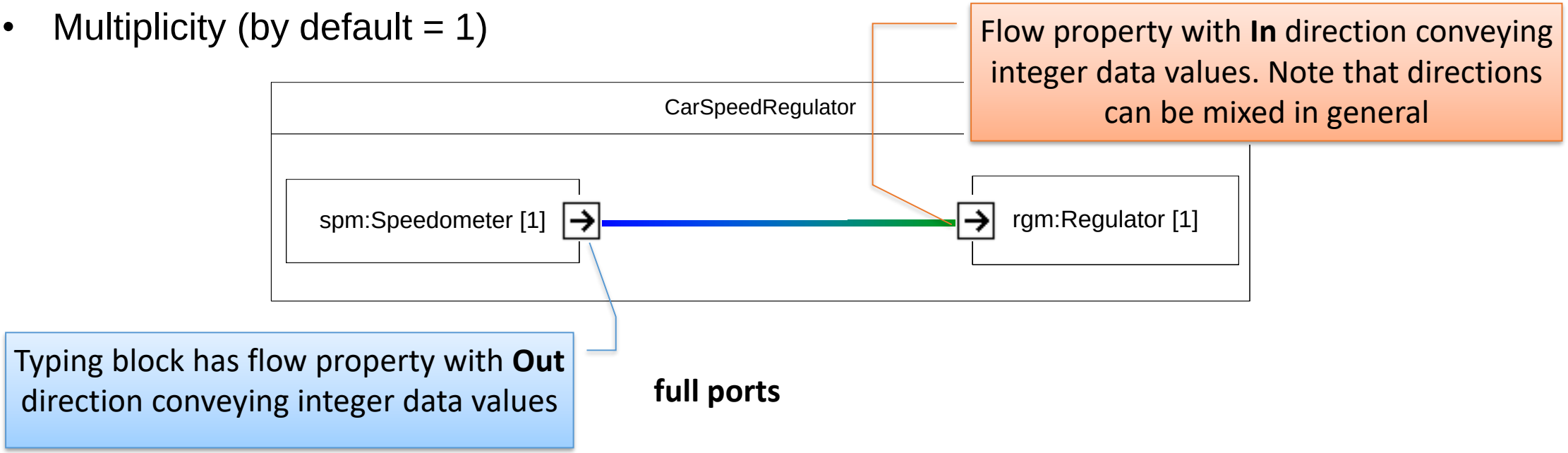
Powertrain realizes the IPowertrain interface
Feedback uses the IFeedback interface



As in standard UML

Full Ports

- Part of the system, typed with a block (port itself can interact)
- The block has set of **flow-properties** (stereotyped attribute)
- Each flow property has a
  - Name and type (since being a UML attribute)
  - Direction = {IN, OUT, INOUT}
  - Multiplicity (by default = 1)



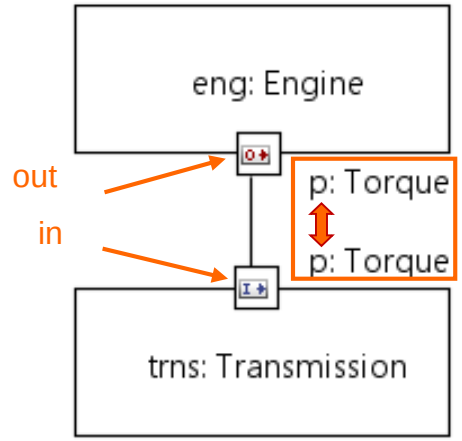
full ports

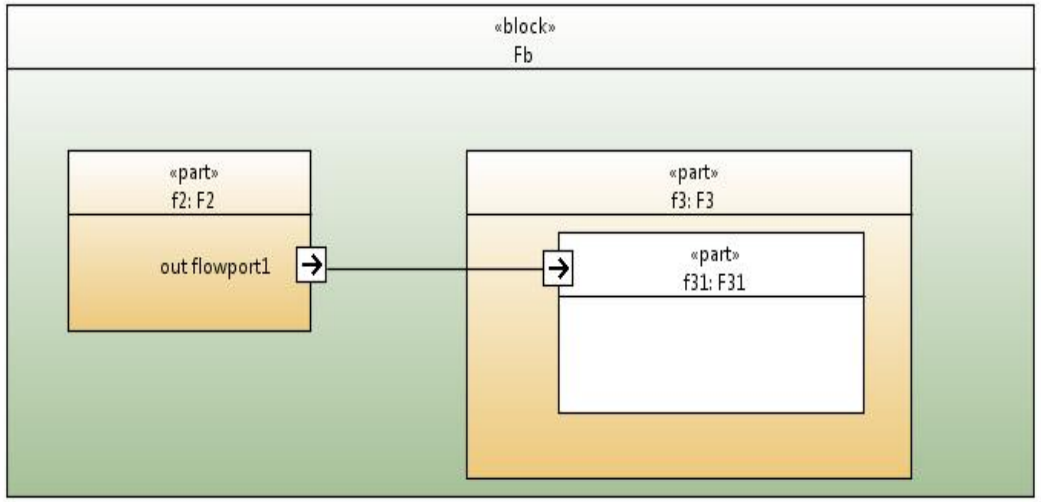
Full Port vs. Proxy Port

- Act as **proxy** for owning blocks or its internal parts
- Always typed with an **interface block**
- Interface block exposes features of owning block

Connector

- **Connector** specifies links that enables communication between two or more ports or parts.
- **Connected pPorts must have a compliant definition**: opposite direction on each side (or inout on both sides)
- **Connectors are owned by the block (Fb)**. The connector **can cross** a part boundary if the encapsulating block (F3) is not a black box (isEncapsulated attribute of Block equal to false)





Hands-On: Internal Block Diagram

Your turn

- Design the internal architecture of the parts of the system. This implies the definition of internal block diagram for each part that is worth a refinement.
- Add the port that enable the exchange of information between part.
- Those ports must be typed.
- Use connectors to relate the ports that will exchange information.

Do it in Papyrus

- Create an IBD diagram for the ADAS system