

DE LA RECHERCHE À L'INDUSTRIE



# STRUCTURAL ARCHITECTURE MODELING WITH SYSML

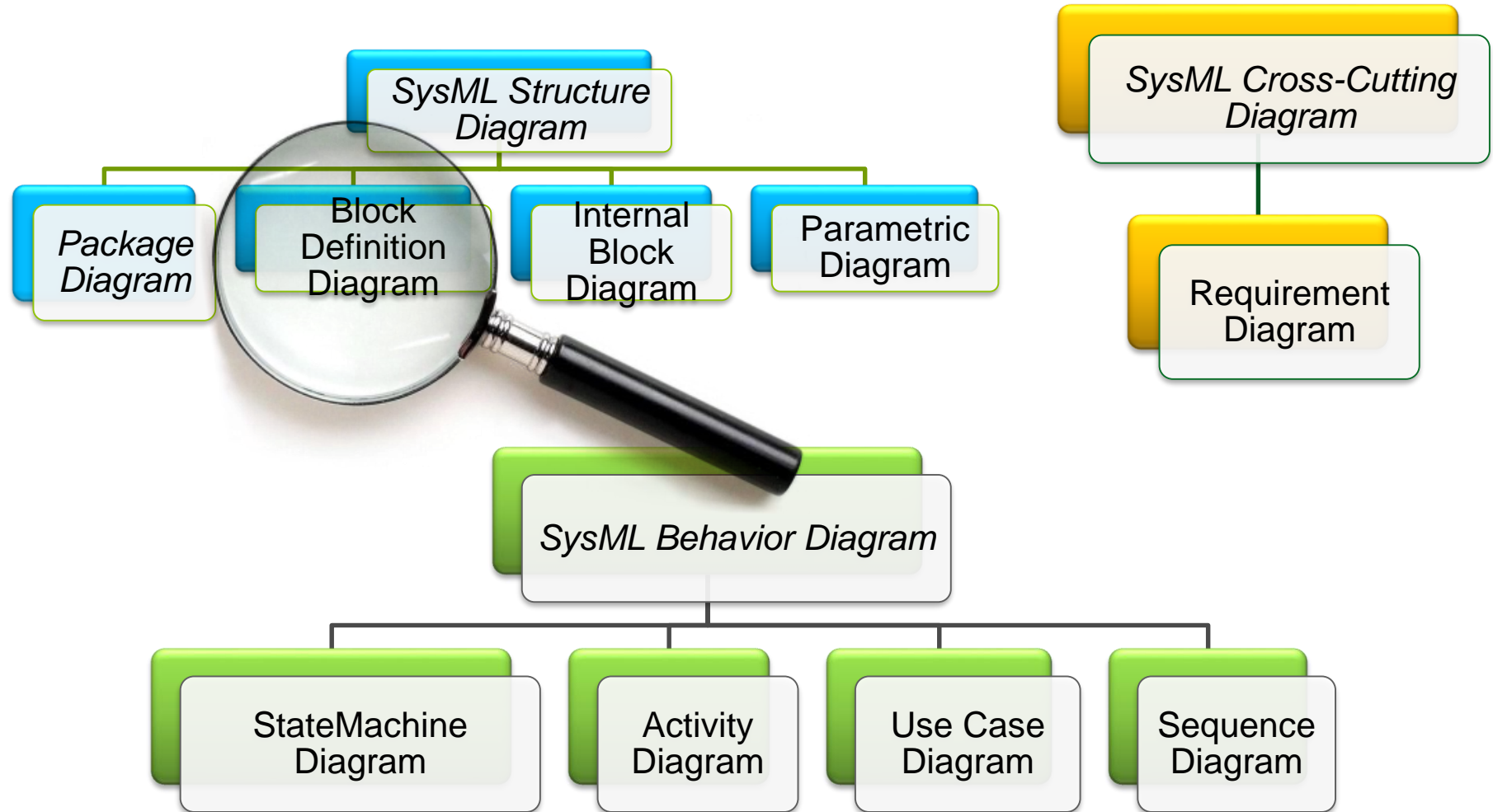
*Software and system engineering department (DILS)  
Laboratory of model driven engineering for embedded systems (LISE)*

[www.cea.fr](http://www.cea.fr)

Copyright (c) 2015, CEA LIST, All rights reserved.



Redistribution and use (commercial or non-commercial), of this presentation, with or without modification, is strictly forbidden without explicit and official approval from CEA LIST



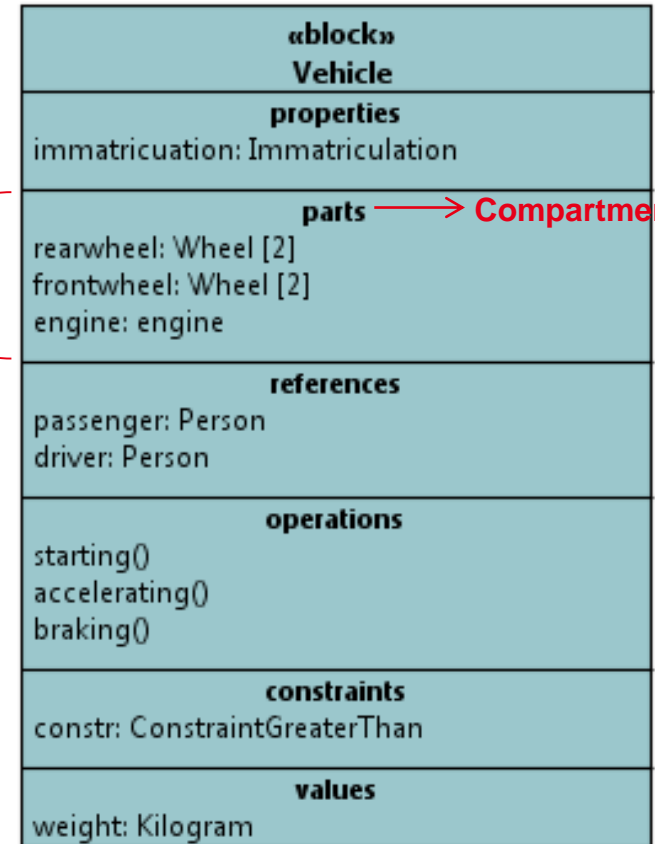
## Block Definition Diagram (BDD)

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

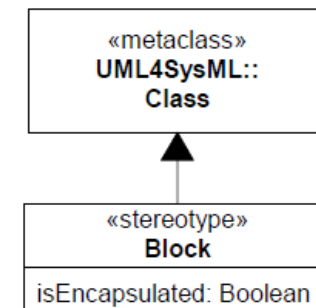
## Block

- 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.

Compartment

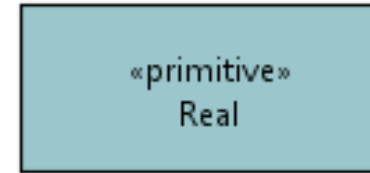


Compartment kind



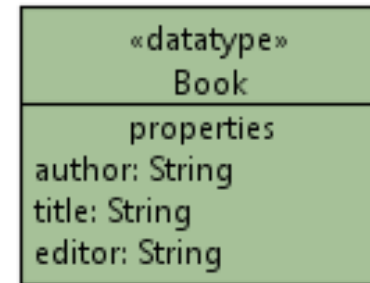
## PrimitiveType

- No properties / no operations



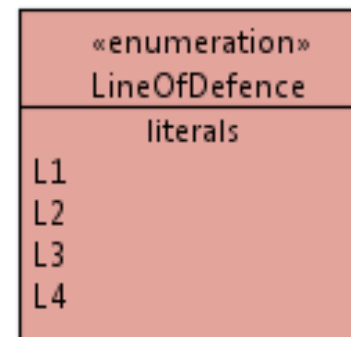
## DataType

- Structured type
- Properties
- May have operations



## Enumeration

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



## Block extends Class so it has...

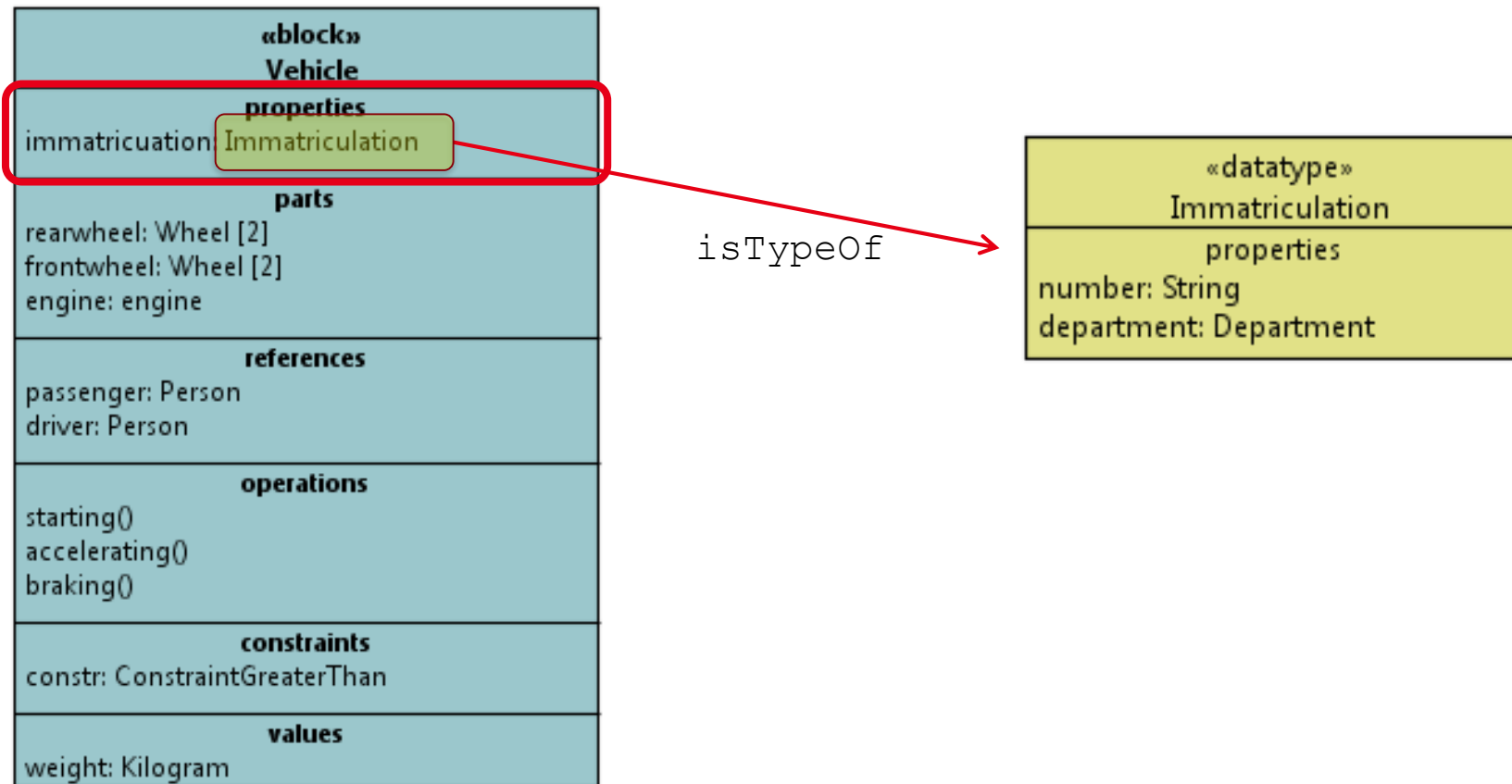
- Properties
- Operations

## Block specializes properties

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

«block» Vehicle
<b>properties</b> immatriculation: Immatriculation
<b>parts</b> rearwheel: Wheel [2] frontwheel: Wheel [2] engine: Engine
<b>references</b> passenger: Person [0..1] driver: Person [0..1]
<b>operations</b> starting() accelerating() braking()
<b>constraints</b> constr: ConstraintGreaterThan
<b>values</b> weight: Kilogram

## Simple properties: always typed properties



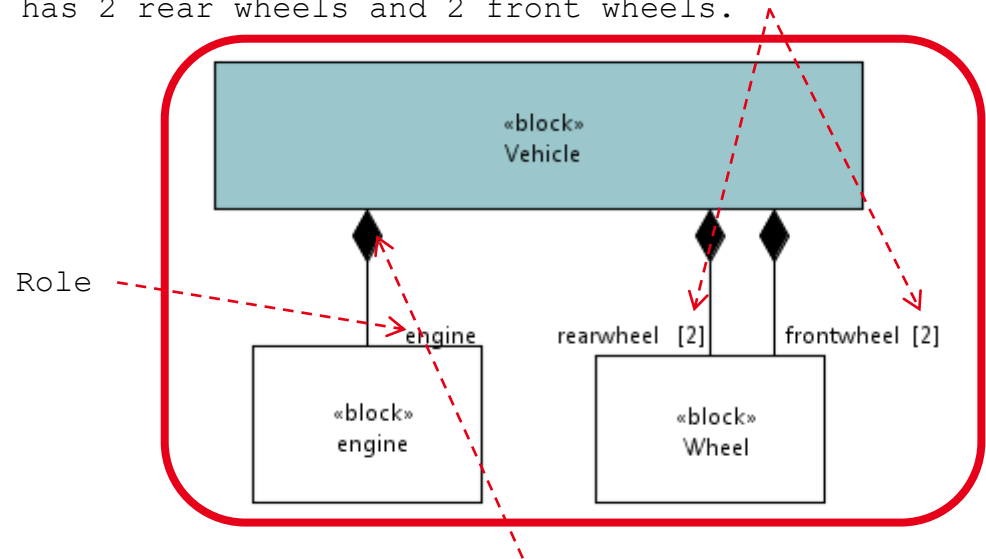
## Part properties: describe composition

«block» Vehicle
<b>properties</b> immatriculation: Immatriculation
<b>parts</b> rearwheel: Wheel [2] frontwheel: Wheel [2] engine: engine
<b>references</b> passenger: Person driver: Person
<b>operations</b> starting() accelerating() braking()
<b>constraints</b> constr: ConstraintGreaterThan
<b>values</b> weight: Kilogram

**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.

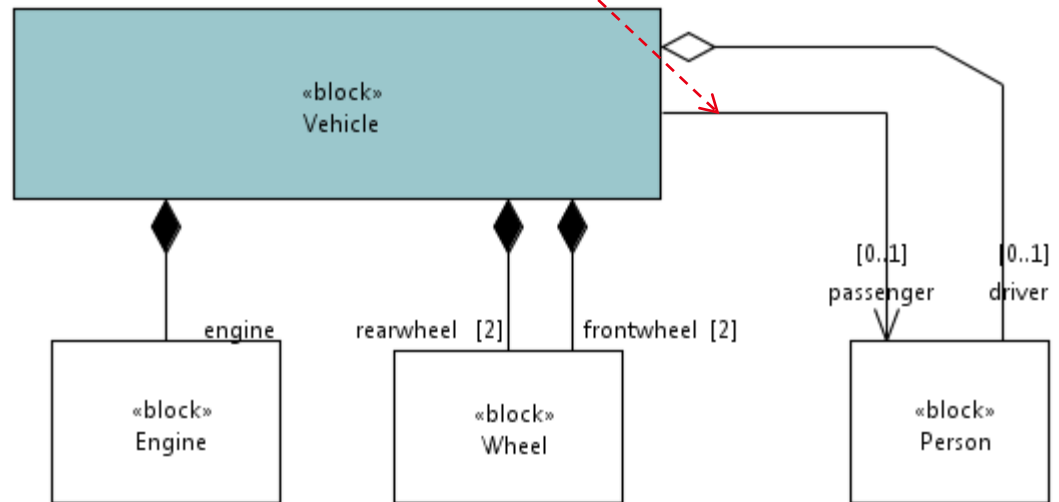


Filled diamond: if an instance of Vehicle is destroyed, the contained instances of Engine is also destroyed.

## Part properties: describe associations and simple aggregations

<b>«block» Vehicle</b>
<b>properties</b> immatriculation: Immatriculation
<b>parts</b> rearwheel: Wheel [2] frontwheel: Wheel [2] engine: engine
<b>references</b> passenger: Person driver: Person
<b>operations</b> starting() accelerating() braking()
<b>constraints</b> constr: ConstraintGreaterThan
<b>values</b> weight: Kilogram

**Association:** simple reference (no containment relationship between blocks)

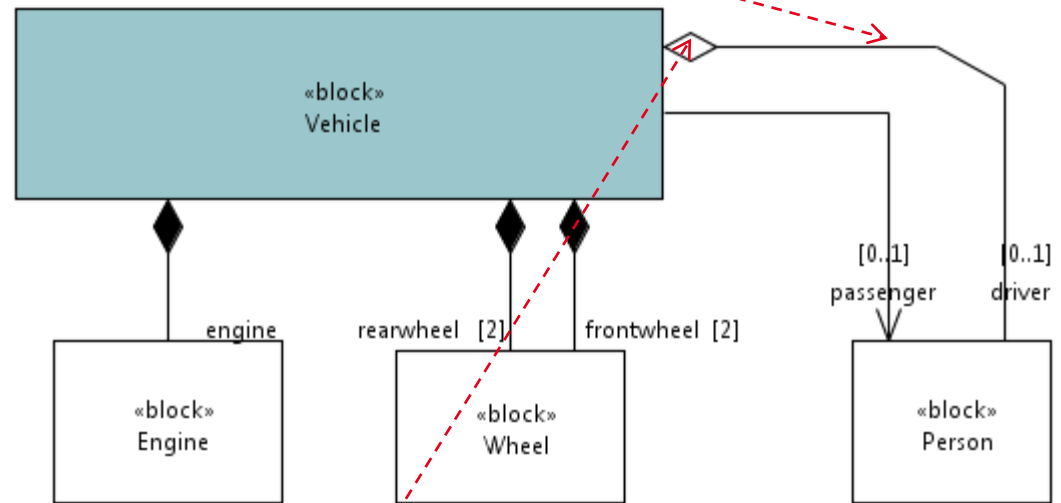




## Part properties: describe associations and simple aggregations

<b>«block» Vehicle</b>
<b>properties</b> immatriculation: Immatriculation
<b>parts</b> rearwheel: Wheel [2] frontwheel: Wheel [2] engine: engine
<b>references</b> passenger: Person driver: Person
<b>operations</b> starting() accelerating() braking()
<b>constraints</b> constr: ConstraintGreaterThan
<b>values</b> weight: Kilogram

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

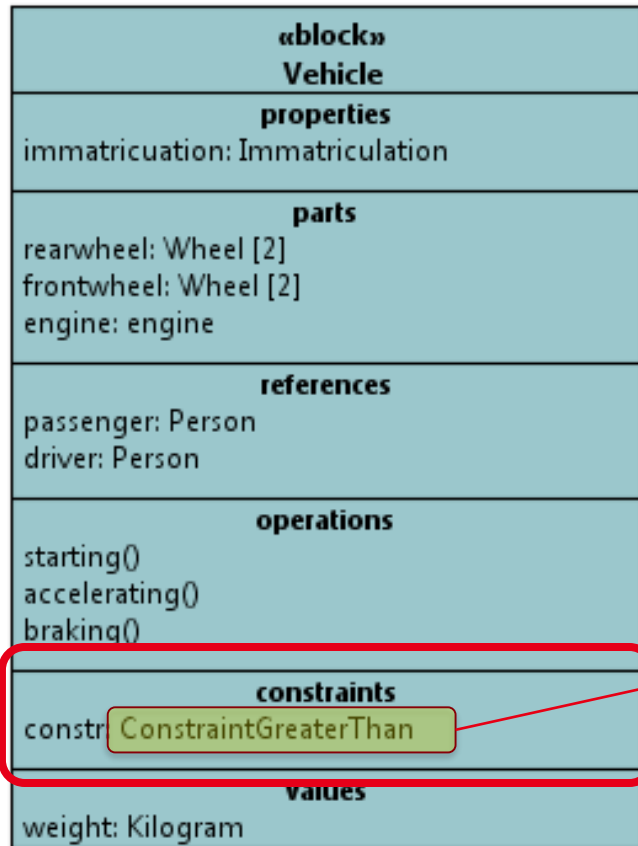


Empty diamond: if an instance of Vehicle is destroyed, the contained instances of Person are not destroyed.

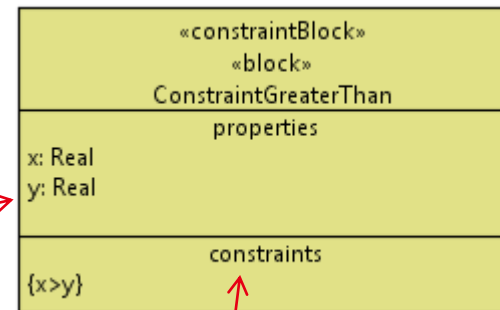
(Hopefully ☺)

## Constraint properties: typed by constraint block

**Constraint block:** can be used to specify a network of constraints that represent mathematical expressions which constrain the physical properties of a system  
They define generic forms of constraints that can be used in multiple contexts.



isTypeOf



Parameters

Constraint body in a formal or non formal language

## Constraint properties: typed by a ValueType

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

«block» Vehicle
<b>properties</b> immatriculation: Immatriculation
<b>parts</b> rearwheel: Wheel [2] frontwheel: Wheel [2] engine: engine
<b>references</b> passenger: Person driver: Person
<b>operations</b> starting() accelerating() braking()
<b>constraints</b> constr: ConstraintGreaterThan
<b>values</b> weight: <b>Kilogram</b>

isTypeOf

«DataType» «ValueType» Kilogram
«ValueType» quantityKind=Weight unit=KilogramUnit
attributes
operations «ControlOperator» + convertToLbs(): Lbs

QuantityKind: Weight

KilogramUnit: Unit
definitionURI = null description = The unit of kilogram quantityKind : QuantityKind = Weight symbol = kg

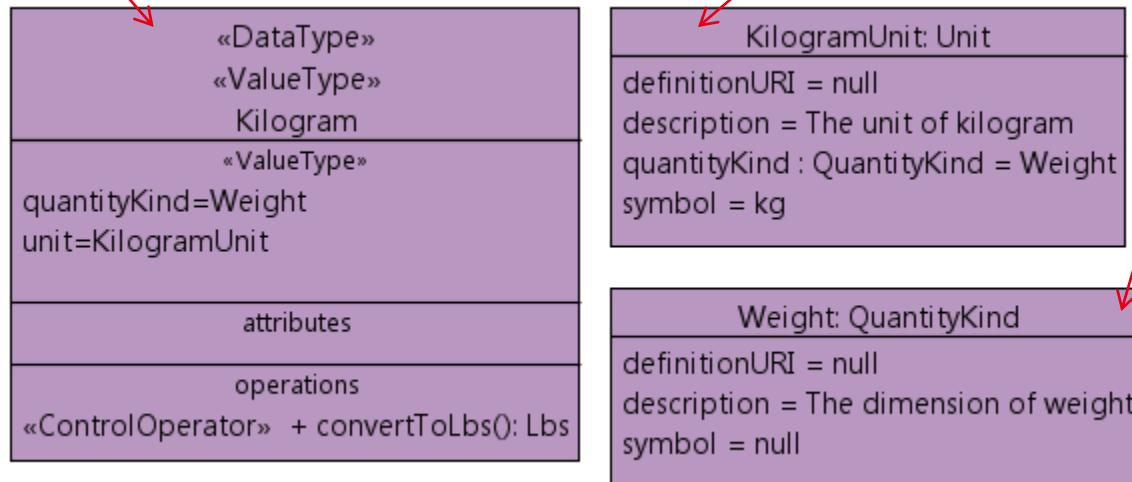
Weight: QuantityKind
definitionURI = null description = The dimension of weight symbol = null

Operation of  
Conversion

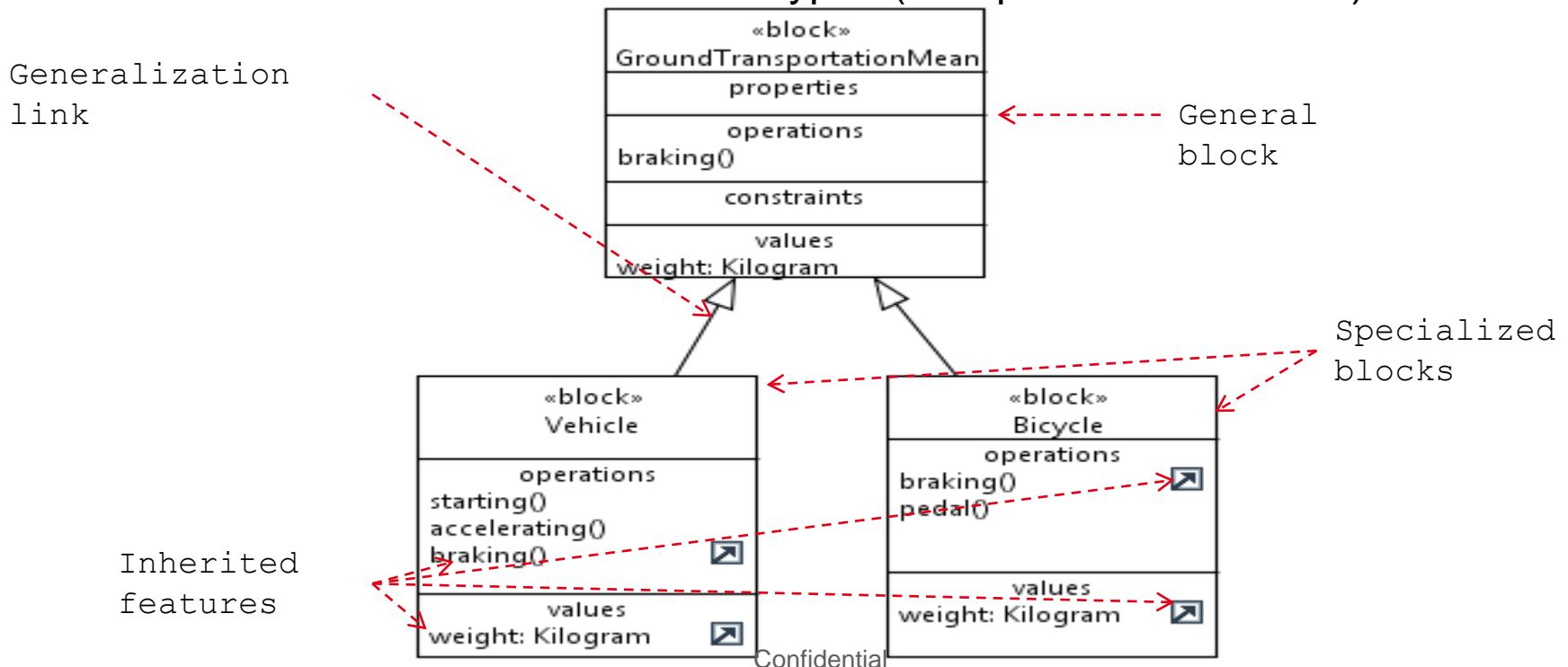
- **QuantityKind:** Kind of quantity that may be stated by means of defined units. For example, the quantity kind 'length' may be measured by units of meters, kilometers, or feet.
- **Unit:** Quantity in terms of which the magnitudes of other quantities that have the same dimension can be stated.

<<ValueType>>  
applied on a  
DataType

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



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



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.	« 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.	« use » ----->



## 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 “RoverSystem” and it contains two parts: Rover and Remote.
- Further refine this architecture.