SysML v2

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

Acknowledgments – contains material from Ed Seidewitz

Agenda – SysML v2

1. **Motivation, history, packages**
2. Selected packages – structure
3. Selected packages - actions
4. Selected packages – calculations & constraints
5. Tools

SysML v2 – History and Timeline

SysML v2 – next generation systems modeling language addressing SysML v1 limitations

- RFP: December 2017

Work on SysML v1 continues in parallel, v1.7 adopted 2022

SysML v2 Submission Team formed in December 2017

- Grew to 200+ members from 80+ organizations
- March 2024, Finalize Specifications, Establish Revision Task Forces
- Mid 2024, Publish Formal Specifications

Motivation

Problems of UML heritage

- Only small evolutions in UML
- No standardized diagram exchange format
- XML format does not work well with git => plant UML might become de-facto standard
- UML complexity

⇨Towards a new language with ***standardized textual format*** and automatically generated diagrams

- Meta model in KerML (Kernel Modeling Language), OMG standard (07/2023) https://www.omg.org/spec/KerML (400 pages)

Naming

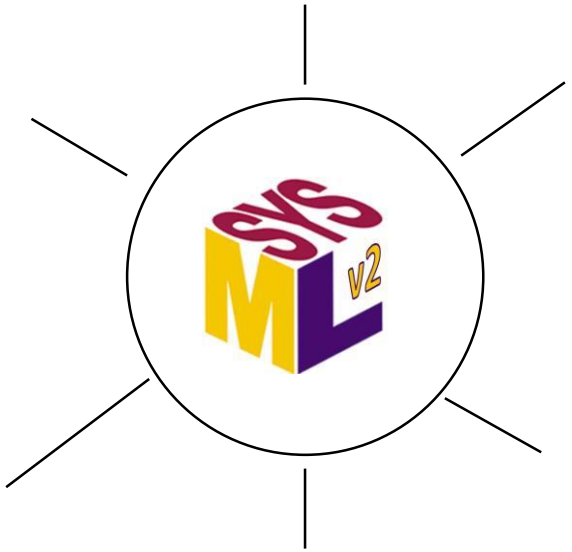
- Objective: more consistent naming
  - **“definition”** for reusable items
  - **“usage”** for references and context-dependent specifications
  - ⇨ Naming is (sometimes) quite different from SysML v1 and UML

- Example: connection definition and connection (in a couple of slides)

SysML v2 – Overview

Behavior
- Action-based
- State-based
- Sequence-based

Structure
- Decomposition
- Classification
- Interconnection



Analysis
- Expressions
- Constraints
- Analysis cases

View & Viewpoint
- rendering

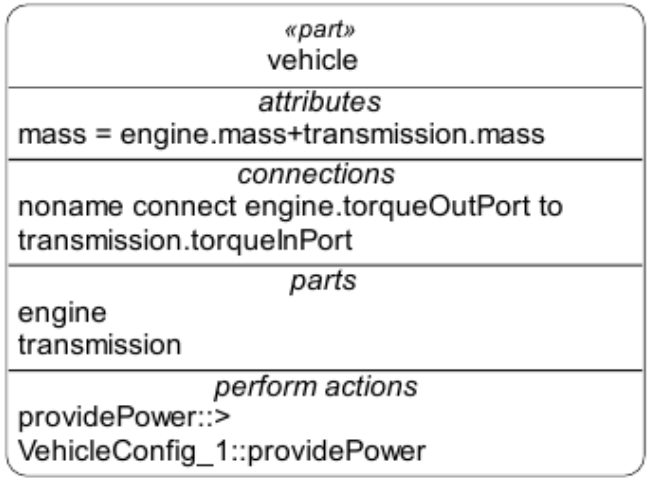
Verification
- verification cases

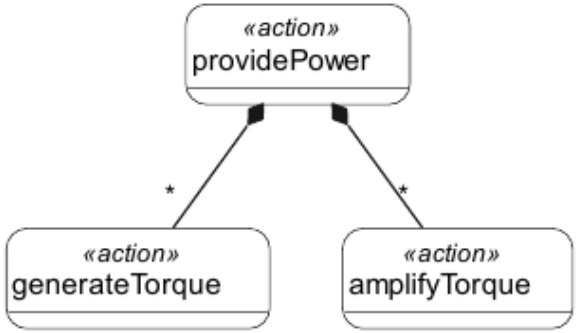
Requirements
- assumed / required constraints
- Stakeholders
- Concerns
- Use cases

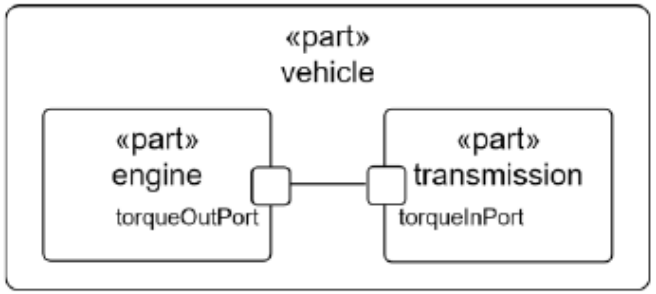
SysML v2

- Corresponding textual and graphical notations for each language construct.
- Comprehensive expression language.
- Textual notations can be used consistently on graphical diagrams.

```
part vehicle{
    attribute mass = engine.mass+transmission.mass;
    perform providePower;
    part engine{
        attribute mass;
        port torqueOutPort;
        perform providePower.generateTorque;
    }
    part transmission{
        attribute mass;
        port torqueInPort;
        perform providePower.amplifyTorque;
    }
    connect engine.torqueOutPort to transmission.torqueInPort;
}
action providePower{
    action generateTorque;
    action amplifyTorque;
}
```

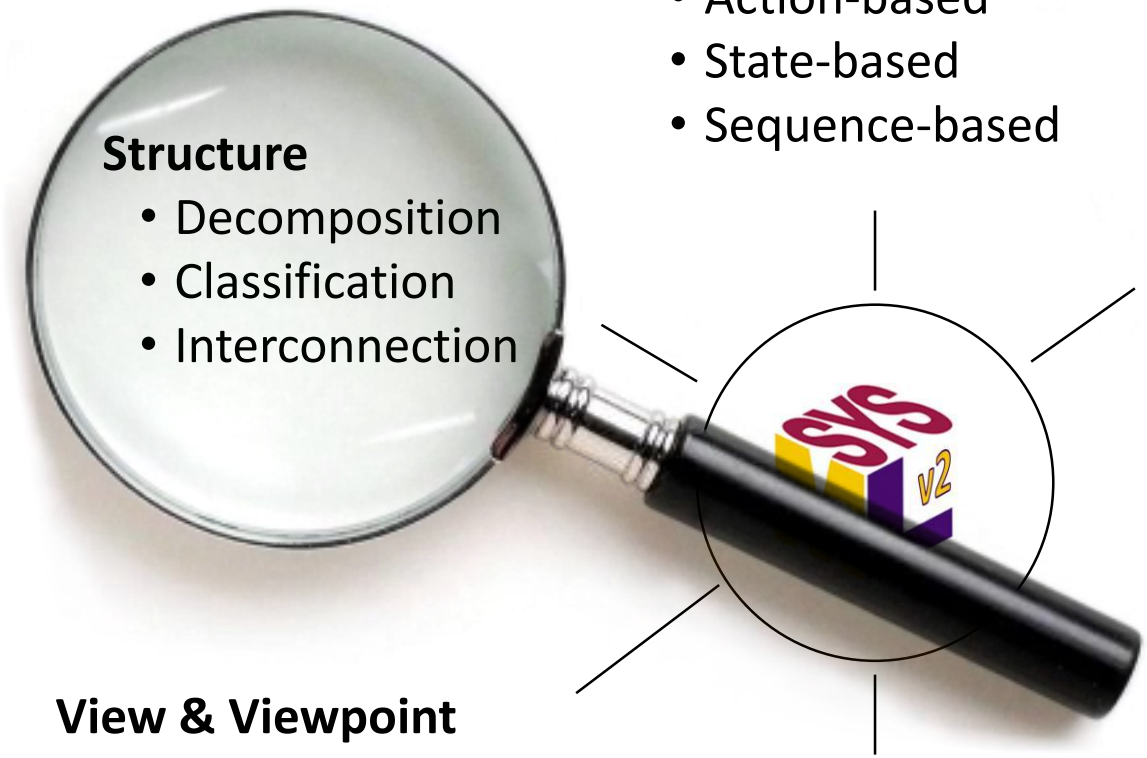






SysML v2 – Structure

Behavior
- Action-based
- State-based
- Sequence-based



Structure
- Decomposition
- Classification
- Interconnection

Analysis
- Expressions
- Constraints
- Analysis cases

View & Viewpoint
- rendering

Verification
- verification cases

Requirements
- assumed / required constraints
- Stakeholders
- Concerns
- Use cases

UML vs SysML v2 terminology

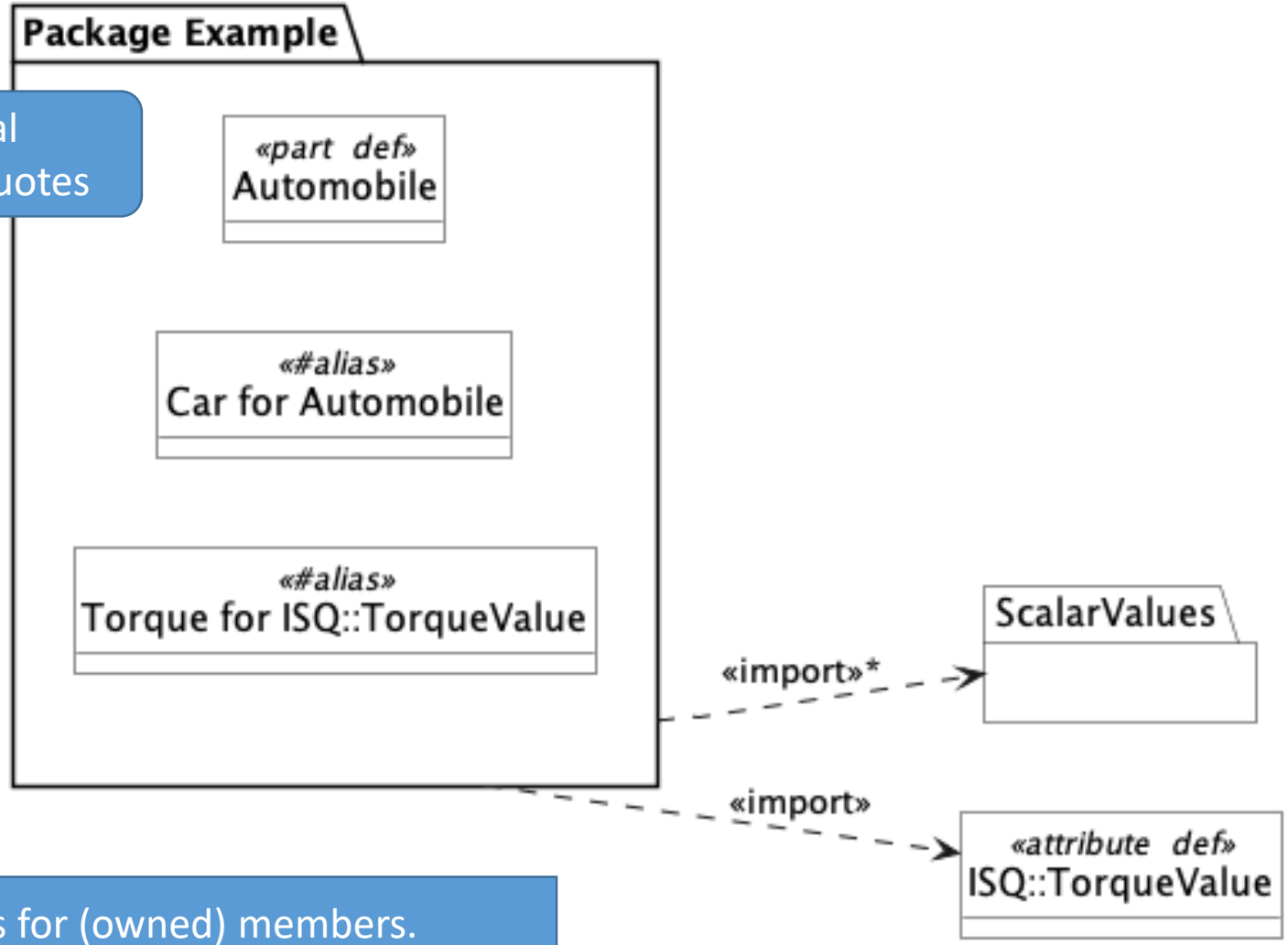
UML	SysML v2
package / member / visibility	package / member / visibility
element import / package import	membership import / namespace import
owned member / imported member	owned member / imported member / alias member
comment	comment / documentation

SysML v2

As in UML : namespace for its members and container for its owned members.

spaces or other special characters ⇨ single quotes

```
package 'Package Example' {
    import ISQ::TorqueValue;
    import ScalarValues::*;
    part def Automobile;
}
alias Car for Automobile;
alias Torque for
ISQ::TorqueValue;
```

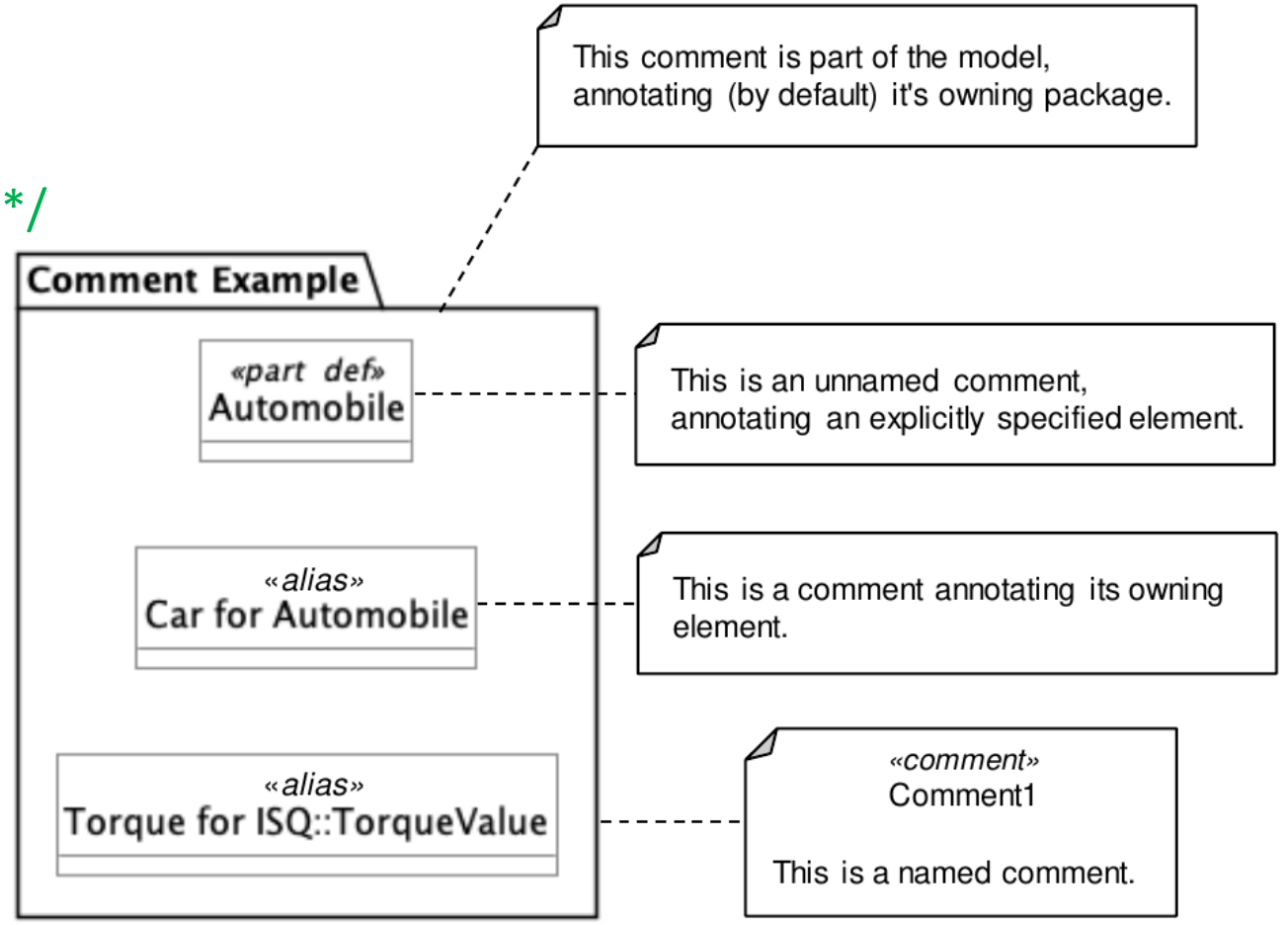


Import either single member or all members of an imported package

Introduce alias for (owned) members.
Use sparingly / with care, since all members are re-exported!

SysML v2 – Comments and Notes

```
package 'Comment Example' {
    /* This comment is part of the model,
     * annotating (by default) it's owning package */
    comment Comment1 /* This is a named comment */
    comment about Automobile
    /* This is an unnamed comment,
     * annotating an explicitly specified element. */
    part def Automobile;
    alias Car for Automobile {
        /* This is a comment annotating its owning
         * element. */
    }
    // This is a note. In the text, not part of the model
    alias Torque for ISQ::TorqueValue;
}
```

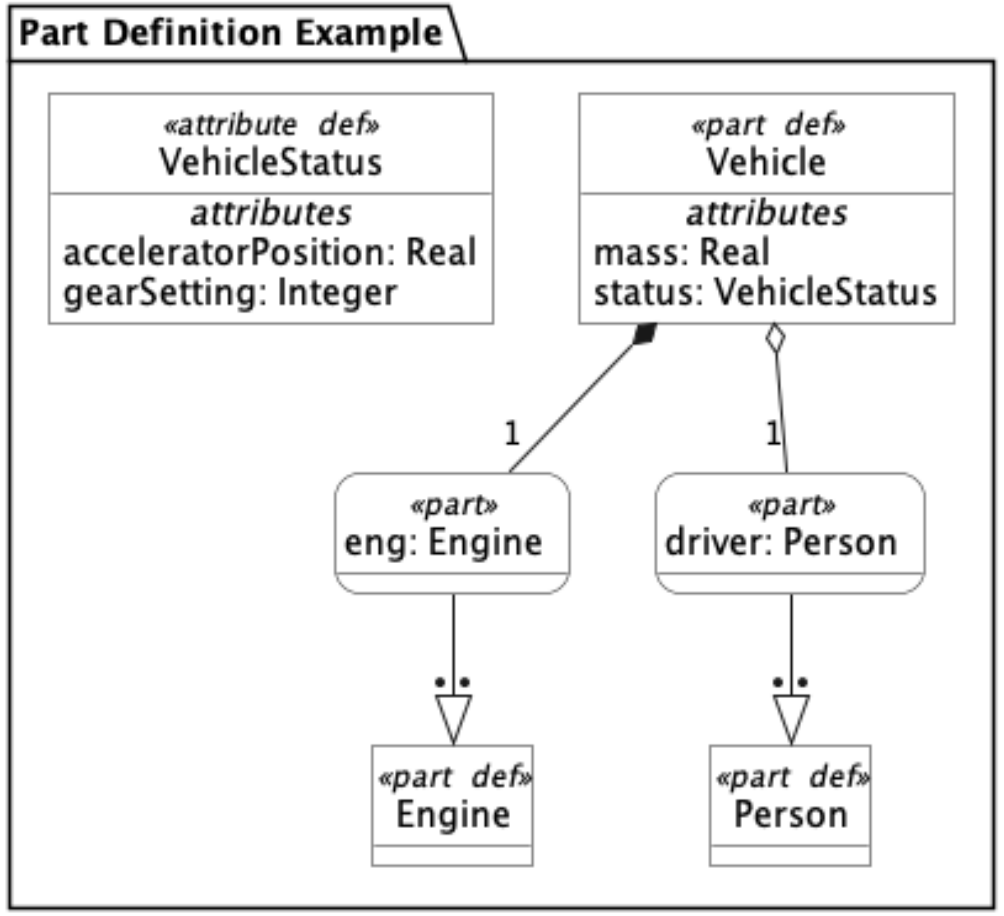


UML - SysML v2 – Terminology differences

UML / SysML v1	SysML v2
class / property	item definition / item usage
block / part property	part definition / part usage
value type / value property	attribute definition / attribute usage
interface block / proxy port (flow property)	port definition / port usage (directed usage)
association block / connector	connection definition / connection usage interface definition / interface usage
item flow	flow connection definition / flow definition usage

SysML v2 – part definition (blocks)

```
part def Vehicle {
        attribute mass : ScalarValues::Real;
        part eng : Engine;
        ref part driver : Person;
}
attribute def VehicleStatus {
        import ScalarValues::*;
        attribute gearSetting : Integer;
        attribute acceleratorPosition : Real;
}
part def Engine;
part def Person;
```



SysML v2 – Item definitions

```
item def Fuel;
item def Person;

part def Vehicle {
    attribute mass : Real;
    ref item driver : Person;
}

part fuelTank {
    item fuel: Fuel;
}
```

Defines class of things that exist in space and time but are **not** necessarily considered "parts“ of a system being modeled.

continuous, if any portion is the same kind of thing.
A portion of fuel is still fuel. Not true for a person.

All parts can be treated as items, but not all items are parts. The design of a system determines what should be modeled as its "parts".

SysML v2 – enumerations

```
enum def TrafficLightColor {
    enum green;
    enum yellow;
    enum red;
}
part def TrafficLight {
    attribute currentColor : TrafficLightColor;
}
part def TrafficLightGo specializes TrafficLight {
    attribute redefines currentColor = TrafficLightColor::green;
}
```

Values of an attribute usage are limited to the defined set of enumerated values.

This shows an attribute being bound to a specific value (more on binding later).

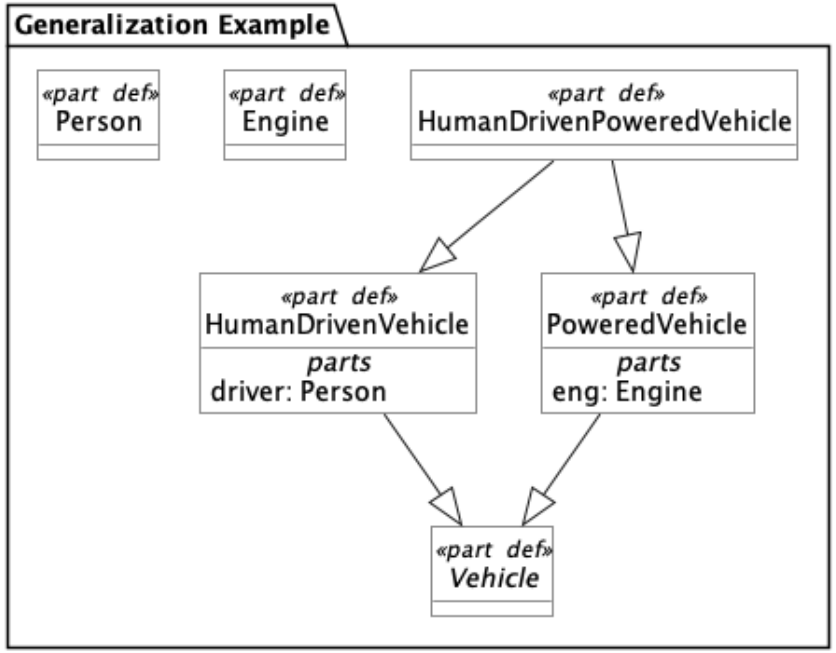
Specialization /Generalization

As in UML, defines a subset of the classification of its generalization.

```
abstract part def Vehicle;
part def HumanDrivenVehicle specializes Vehicle {
    ref part driver : Person;
}
part def PoweredVehicle :> Vehicle {
    part eng : Engine;
}
part def HumanDrivenPoweredVehicle :> HumanDrivenVehicle, PoweredVehicle;
part def Engine;
part def Person;
```

equivalent to specializes keyword.

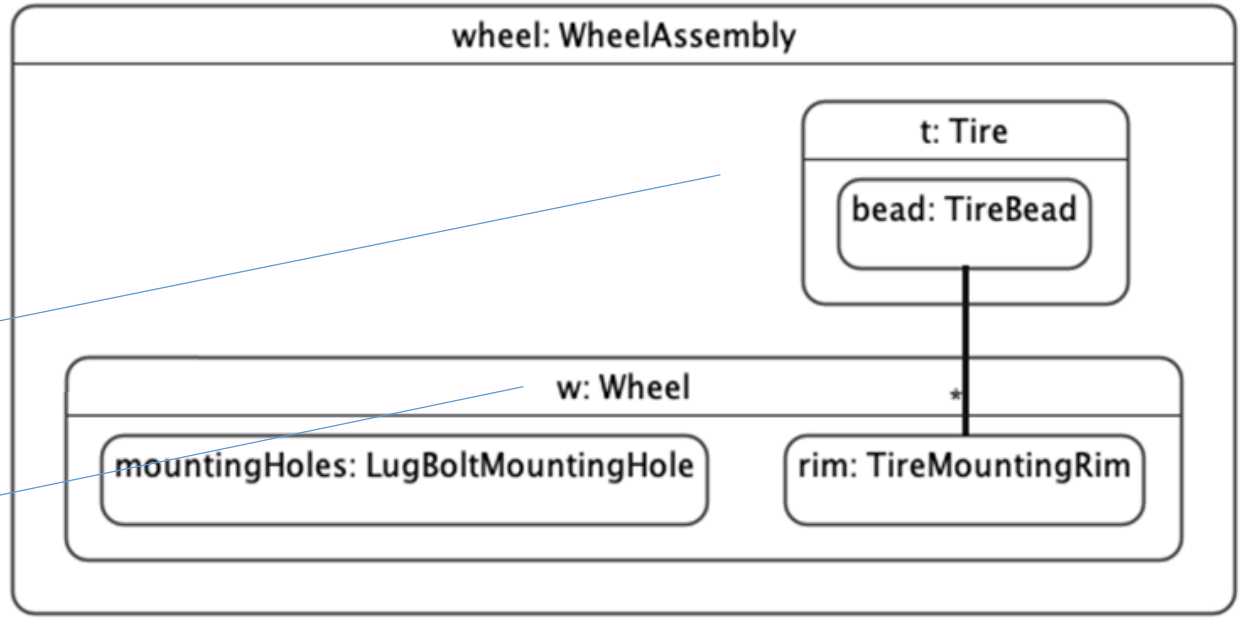
Can define additional features.



Structure – Connections and connection definitions

Association in UML

```
connection def PressureSeat {
    end bead : TireBead[1];
    end mountingRim: TireMountingRim[1];
}
part wheel : WheelAssembly[1] {
    part t : Tire[1] {
        part bead : TireBead[2];
    }
    part w: Wheel[1] {
        part rim : TireMountingRim[2];
        part mountingHoles : LugBoltMountingHole[5];
    }
    connection : PressureSeat connect bead references t.bead to mountingRim references w.rim;
}
```



Connector in UML

Ports and port definitions

```
port def FuelPort {
    attribute temperature : Temp;
    out item fuelSupply : Fuel;
    in item fuelReturn : Fuel;
}
part def FuelTankAssembly {
    port fuelTankPort : FuelPort;
}
part def Engine {
    port engineFuelPort : ~FuelPort;
}
```

Port definition has implicit ***conjugate*** port definition reversing input and output features. Name is prefixed with ~ (e.g. ~FuelPort)

Ports are ***compatible*** if they have directed features that match with inverse directions

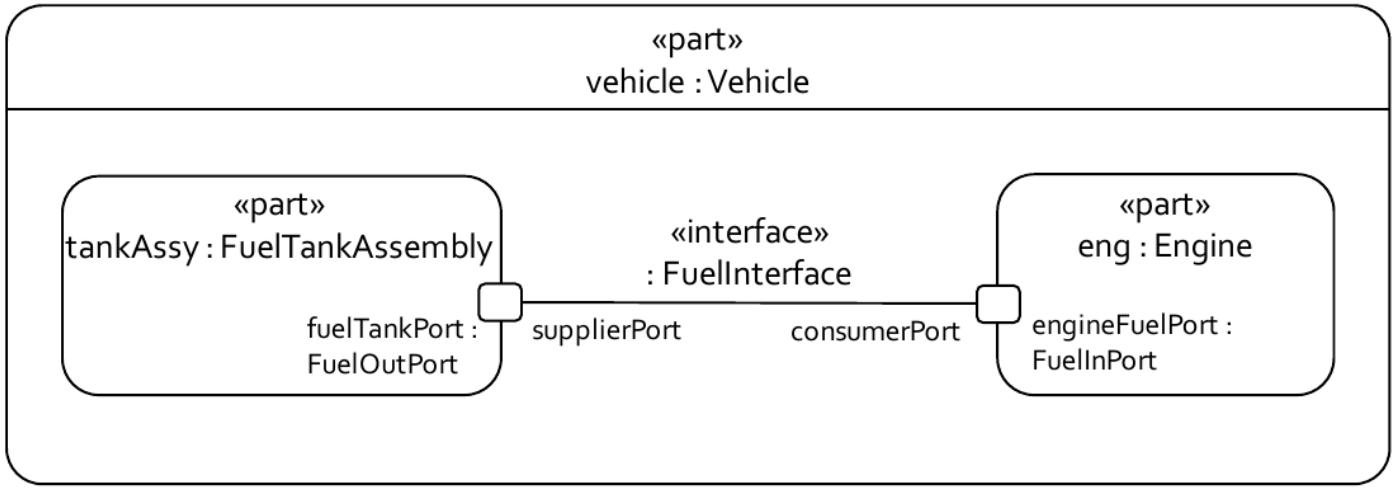
Directed features are always referential

Port = connection point through which a part definition makes some of its features available

Interfaces and interface definitions

```
interface def FuelInterface {
    end supplierPort : FuelOutPort;
    end consumerPort : FuelInPort;
}
part vehicle : Vehicle {
    part tankAssy : FuelTankAssembly;
    part eng : Engine;
}
interface : FuelInterface connect
    supplierPort ::> tankAssy.fuelTankPort to
    consumerPort ::> eng.engineFuelPort;
}
```

interface definition = connection definition whose ends are port definitions

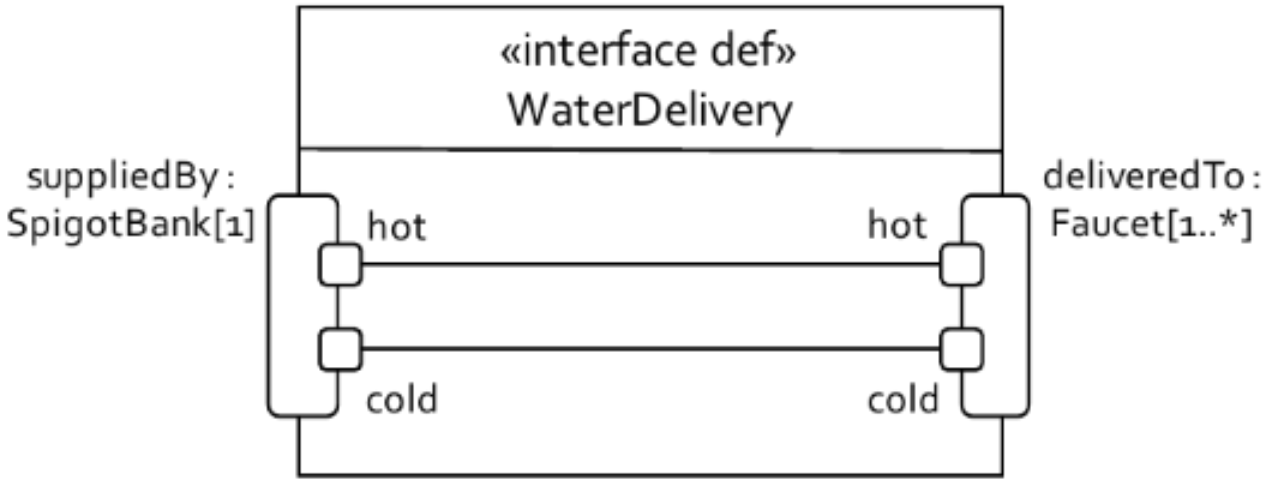


CAVEAT

Same term as in UML, but SysML v2 interfaces are very different from UML interfaces

Complex interface definitions

```
interface def WaterDelivery {
    end suppliedBy : SpigotBank[1] {
        port hot : Spigot;
        port cold : Spigot;
    }
    end deliveredTo : Faucet[1..*] {
        port hot : FaucetInlet;
        port cold : FaucetInlet;
    }
    connect suppliedBy.hot to deliveredTo.hot;
    connect suppliedBy.cold to deliveredTo.cold;
}
```



Beyond UML
enable identification of sub-connections, "cables"

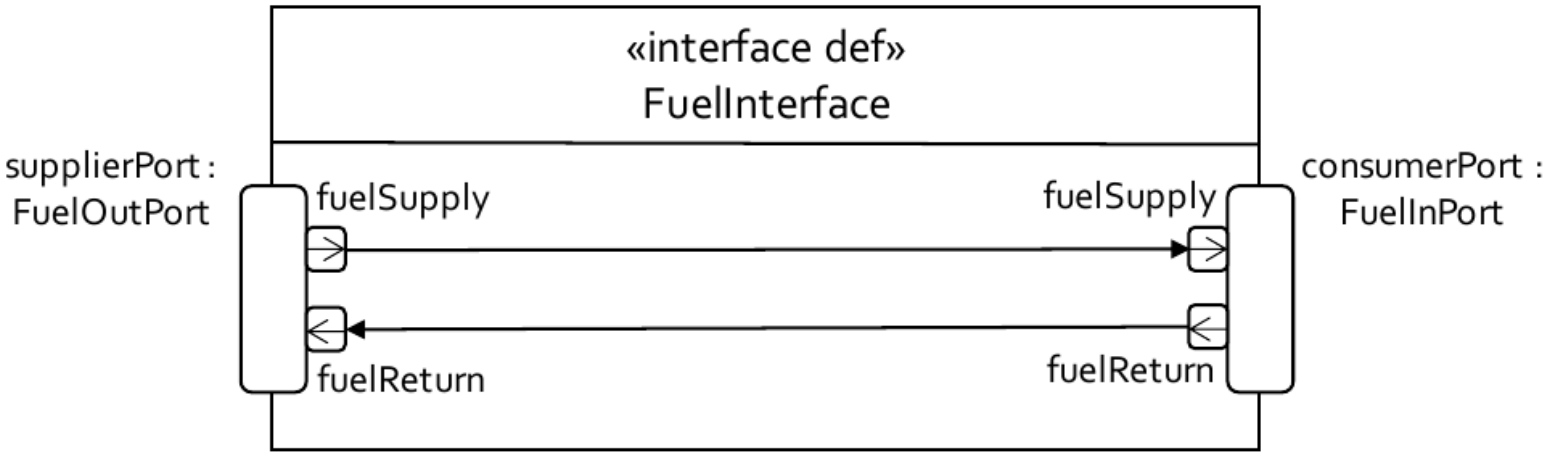
Binding connections vs. Flow connections

- A binding connection **asserts the equivalence** of the connected features (equal values in the same context).

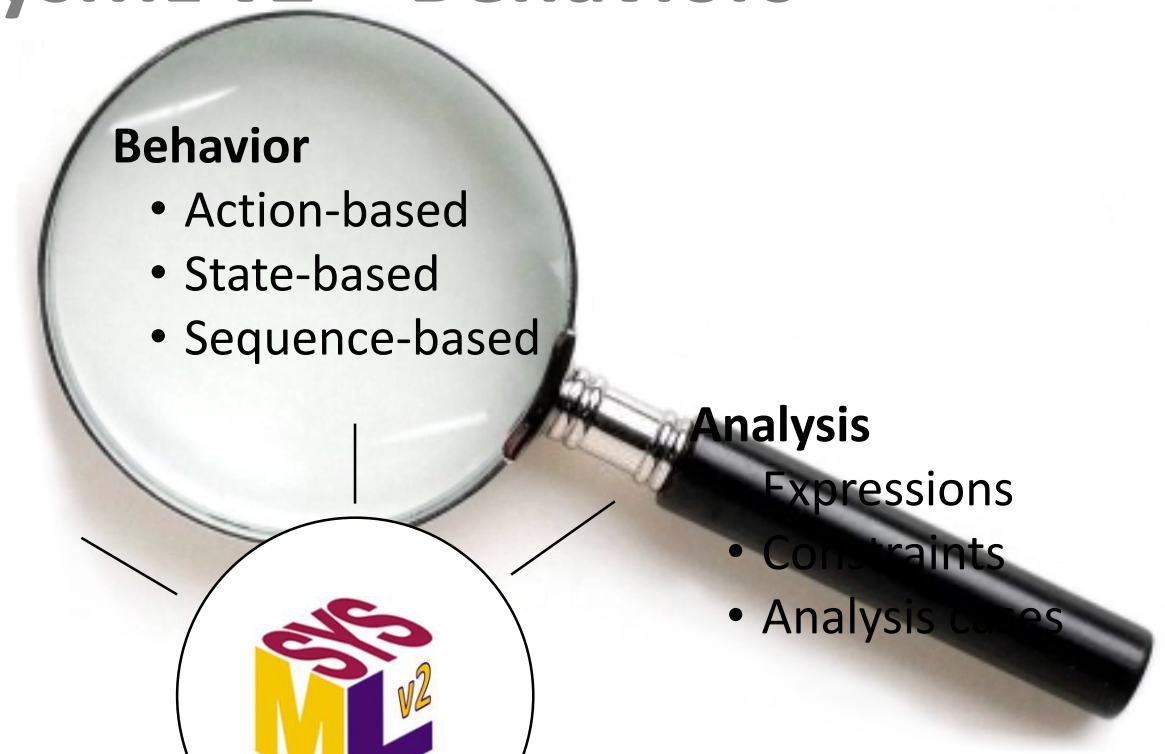
```
part tank : FuelTankAssembly {
    port redefines fuelTankPort {
    out item redefines fuelSupply;
    in item redefines fuelReturn;
}
bind fuelTankPort.fuelSupply = pump.pumpOut;
bind fuelTankPort.fuelReturn = tank.fuelIn;
```

Flow connections in interfaces

```
interface def FuelInterface {
    end supplierPort : FuelOutPort;
    end consumerPort : FuelInPort;
    flow supplierPort.fuelSupply to consumerPort.fuelSupply;
    flow consumerPort.fuelReturn to supplierPort.fuelReturn;
}
```



SysML v2 – Behaviors



Behavior
- Action-based
- State-based
- Sequence-based

Structure
- Decomposition
- Classification
- Interconnection

Analysis
- Expressions
- Constraints
- Analysis cases

View & Viewpoint
- rendering

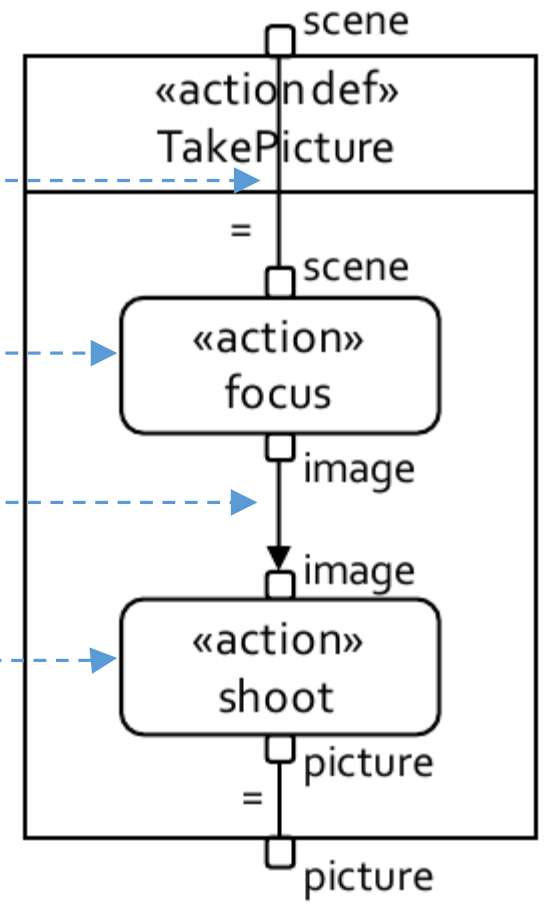
Verification
- verification cases

Requirements
- assumed / required constraints
- Stakeholders
- Concerns
- Use cases

SysML v2 – Actions definition

Directed features of an action definition are considered to be action parameters.

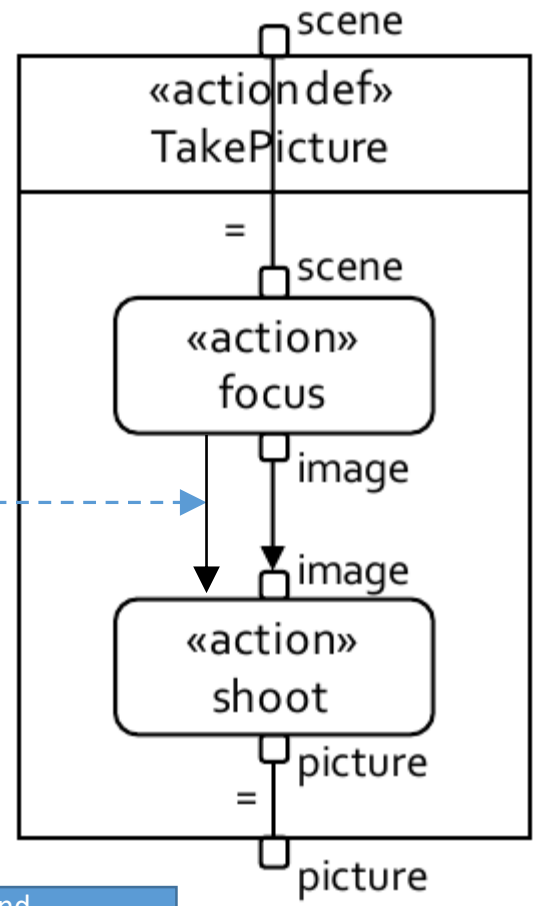
```
action def Focus{ in scene : Scene; out image : Image; }
action def Shoot{ in image : Image; out picture : Picture; }
action def TakePicture {
    in scene : Scene;
    out picture : Picture;
    bind focus.scene = scene;
    action focus : Focus { in scene; out image; };
    flow focus.image to shoot.image;
    action shoot : Shoot { in image; out picture; }
    bind shoot.picture = picture;
}
```



transfer items between actions via flow connection

SysML v2 – succession & succession flow

```
action def Focus{ in scene : Scene; out image : Image; }
action def Shoot{ in image : Image; out picture : Picture; }
action def TakePicture {
    in scene : Scene;
    out picture : Picture;
    bind focus.scene = scene;
    action focus : Focus { in scene; out image; };
    first focus then shoot;
    succession flow focus.image to shoot.image;
    action shoot : Shoot { in image; out picture; }
    bind shoot.picture = picture;
}
```



Succession – assert that 2nd action waits for completion of 1st

SysML v2 – shorthand / conditional successions

- **Shorthand**
  Only prefix action with “then” (i.e. omit “first” – defaults to previous action)
  e.g. **then action** shoot : Shoot

- **Conditional successions**

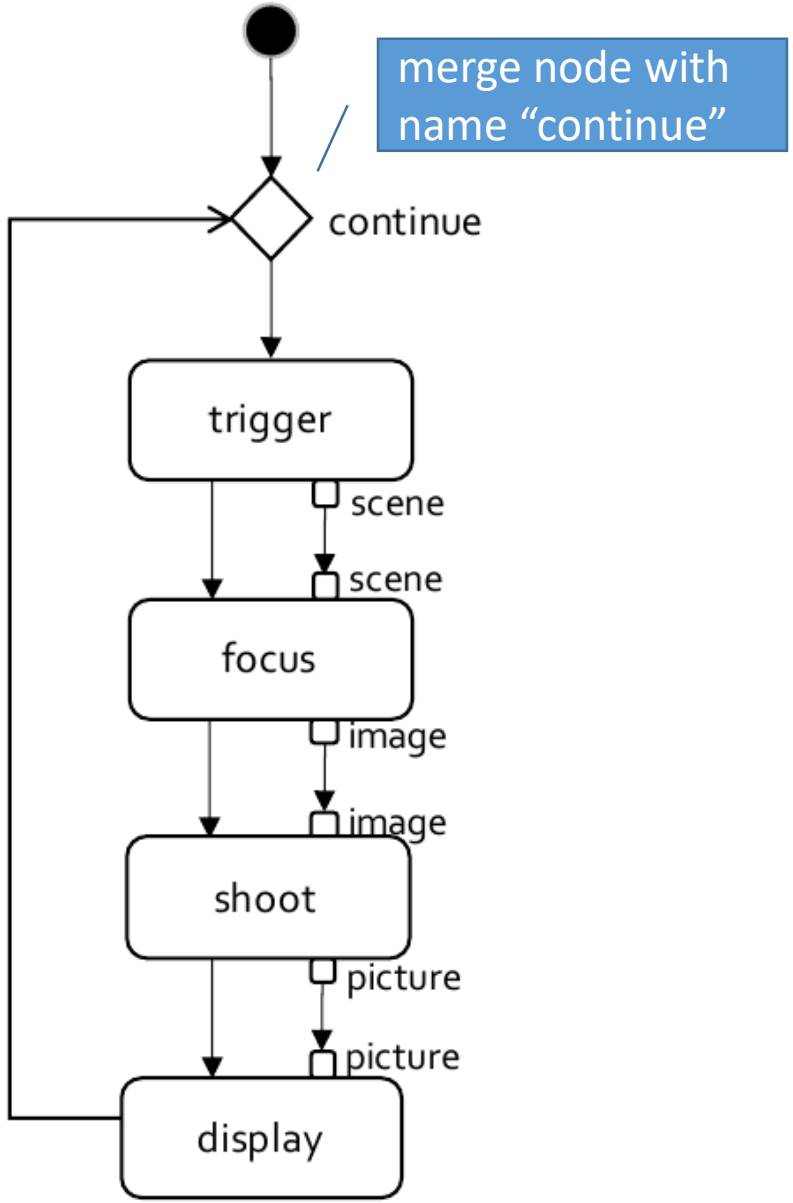
  if <guard expr> **then** <action>

  - Asserts that second action follows first only if a guard condition is true

SysML v2 – merge nodes / loops

```
action takePicture : TakePicture {
    first start;
    then merge continue;
    then action trigger { out item scene; }
    flow trigger.scene to focus.scene;
    then action focus : Focus {
        in scene; out image; }
    flow focus.image to shoot.image;
    then action shoot : Shoot {
        in image; out picture; }
    flow shoot.picture to display.picture;
    then action display { in picture; }
    then continue;
}
```

Wait for start to complete



SysML v2 – merge nodes / loops

```
while takePicture >= 0.05 action takePicture {
    …
}
```

Exit, if condition becomes false

or

```
loop action takePicture : TakePicture {
    …
} until battery.charge < 0.05
```

Exit, if condition becomes true

Beyond UML
Control structure as in programming languages

UML - SysML v2 action terminology

UML / SysML v1	SysML v2
activity / action (parameter / pin)	action definition / action usage (directed usage)
control flow	succession
object flow	flow connection usage
state machine / state	port definition / port usage (directed usage)
association block / connector	state definition / state usage
transition	transition usage

SysML v2 – Expressions and Constraints

Behavior
- Action-based
- State-based
- Sequence-based

Structure
- Decomposition
- Classification
- Interconnection

Analysis
- Expressions
- Constraints
- Analysis cases

Requirements
- assumed / required constraints
- Stakeholders
- Concerns
- Use cases

View & Viewpoint
- rendering

Verification
- verification cases

SysML v2 – Calculations definitions (expressions)

Reusable, *parameterized* expression

directed features are parameters (as in actions), defaults to “in”

```
calc def DeriveAcc {
    in tp: PowerValue; in m : MassValue; in v: SpeedValue;
    attribute intermediate = m * v;
    return : AccelerationValue = tp / intermediate;
}
calc def Velocity {
    in dt : TimeValue; in v0 : SpeedValue; in a : AccelValue;
    return : SpeedValue = v0 + a * dt;
}
```

Beyond UML
Calculations based on KerML

Calculation Usage

```
action straightLineDynamics { in delta_t : TimeValue;
    in v_in : SpeedValue; in x_in : LengthValue;
    calc acc : Acceleration {
        return DeriveAcc(wheelPower, mass, v_in);
    }
```

Constraint definition

```
import ISQ::*;
import SI::*;
import ScalarFunctions::*;
constraint def MassConstraint {
    in partMasses : MassValue[0..*];
    in massLimit : MassValue;
    sum(partMasses) <= massLimit
}
```

Beyond UML
Own constraint language
(not a separate language as OCL)

SysML v2 tools

- Pilot implementation from Ed Seidewitz
  https://github.com/Systems-Modeling/SysML-v2-Pilot-Implementation
  (based on Eclipse + Xtext, performance issues)

- OBEO/CEA, web based tool (to appear)
  https://mbse-syson.org/