SysML v2

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

Acknowledgments – contains material from Ed Seidewitz

Agenda – SysML v2

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

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

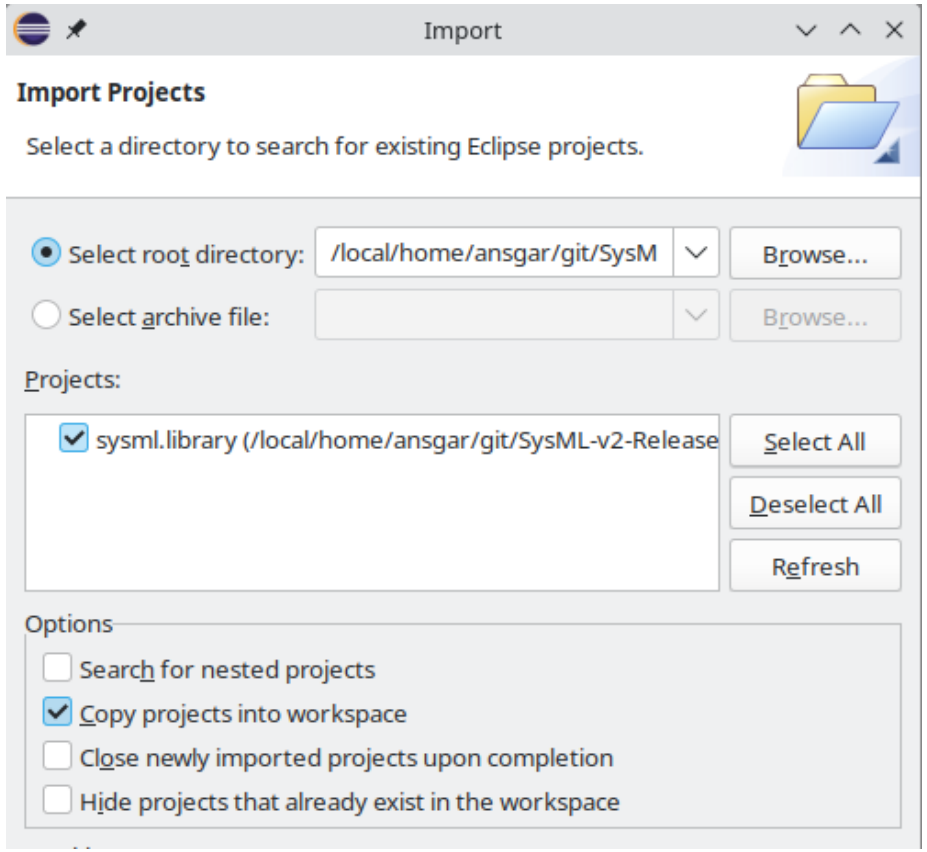
- Example: connection definition and connection (in a couple of slides)

Tools

- Eclipse-based reference implementation
  - Ed Seidewitz (Model-driven solutions)
  - https://github.com/Systems-Modeling/SysML-v2-Release/tree/master
- SysOn – Collaboration OBEO & CEA
  - Web-based (fronted/backend)
  - https://doc.mbse-syson.org/
- SysIDE for VS code
  - https://github.com/sensmetry/sysml-2ls

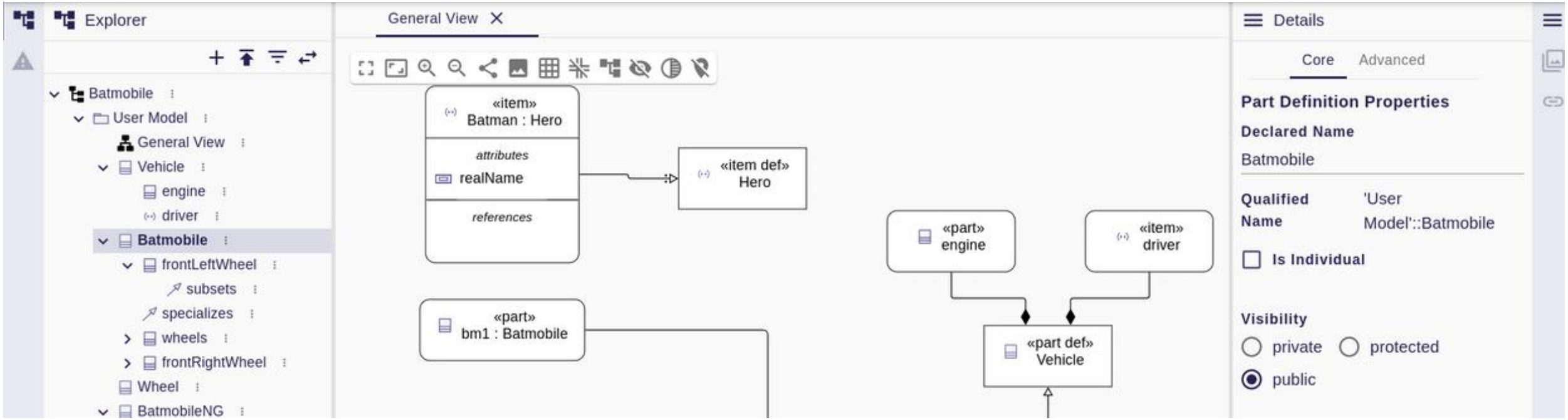
Installation instructions – Reference implementation

- We will mainly use the reference implementation (but you can install all three)
- Get an eclipse-modeling tools 2024-12 from here:
  - https://www.eclipse.org/downloads/packages/
  - Help > Install New Software > Add > archive “install/eclipse/org.omg.sysml.site.zip”
  - Import sysml.library via “Import > General > Existing projects into workspace
  - Check option “copy projects into workspace”
  - Import examples (copy option is not required)



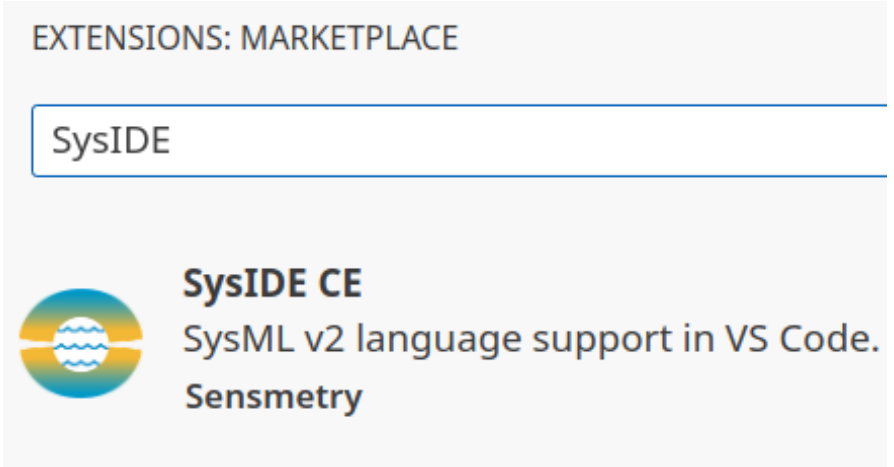
Installation instructions – SysON

- https://doc.mbse-syson.org/syson/v2025.1.0/installation-guide/index.html
- Install docker & docker compose first
- Download YAML file for single user installation https://github.com/eclipse-syson/syson/blob/v2025.1.0/docker-compose.yml



Installation instructions – SysIDE

- Get a current VS code installation
- Open Extensions > Type “SysIDE” in the filter
  - Install also the SysML library



Advantages/inconveniences

- Eclipse-based reference implementation
  - Quite complete, large set of examples (which have been copied to the instn git)
  - Diagrams are automatically generated, based on PlantUML
  - based on Eclipse + Xtext, (minor) performance issues
- SysON
  - Collaborative, built on same technology as PapyrusWeb
  - Text mode not yet integrated (likely based on SysIDE)
  - Graphical editor (arrange your boxes as needed, choose subset)
- SysIDE
  - Fast and easy to use
  - Only text, advanced features only in paid version

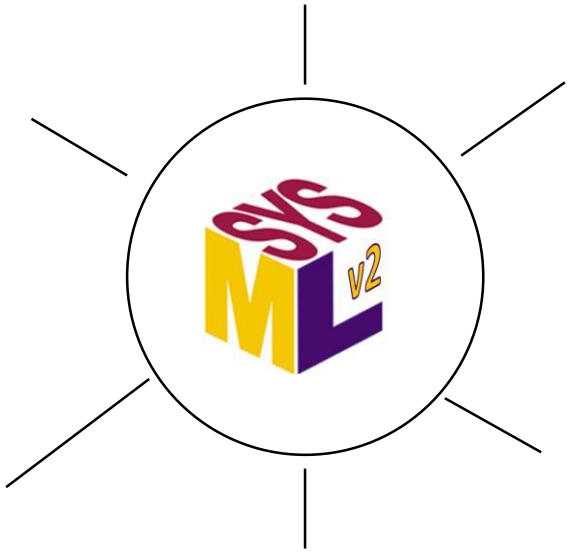
Tool conclusions

- Choose the tool that you like, textual specifications are exchangeable.
- The following slides use screenshots from SysIDE along with generated PlantUML diagrams via the reference implementation

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

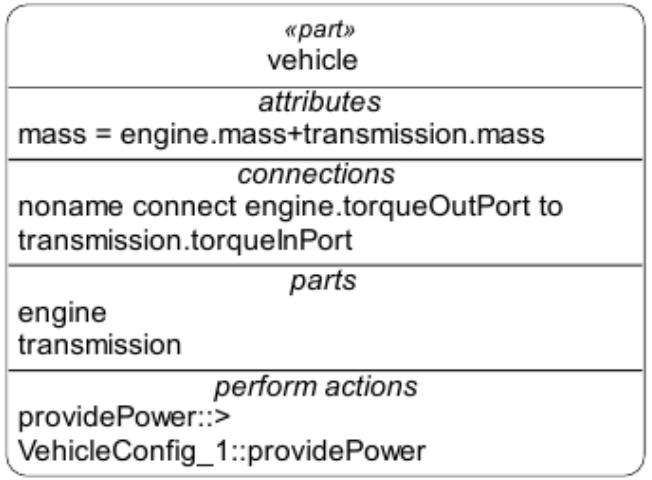
Requirements
- assumed / required constraints
- Stakeholders
- Concerns
- Use cases

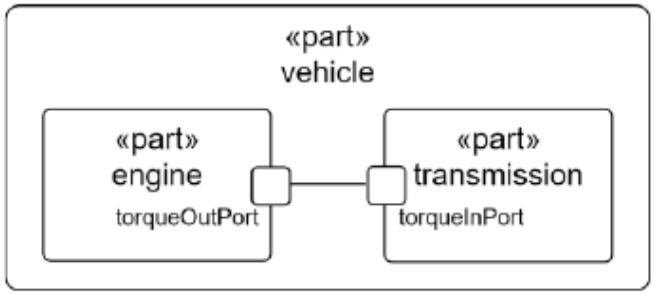
SysML v2

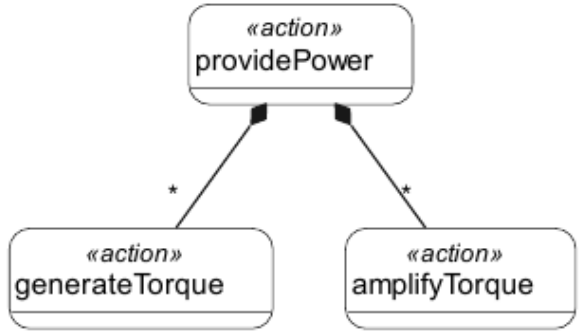
- Corresponding textual and graphical notations for each language construct.
- Comprehensive expression language.
- Textual notations can be used consistently on graphical diagrams.

```
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;
}
```

vehicle.sysml



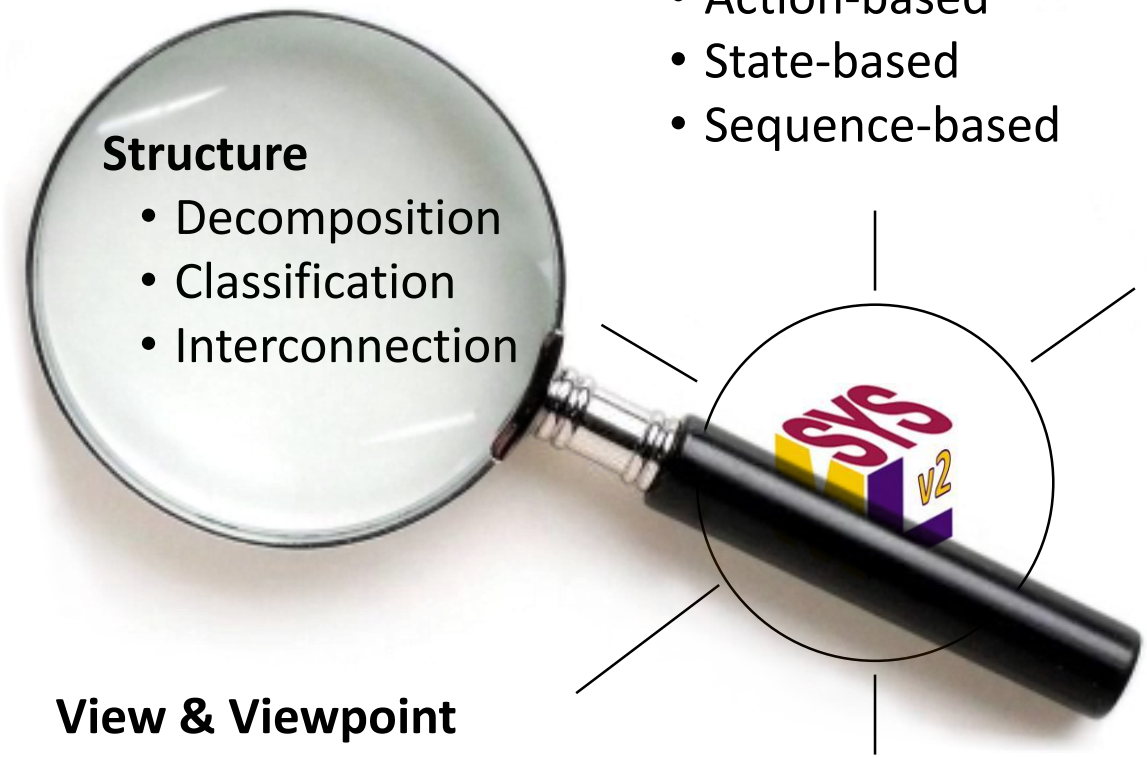




Generated PlantUML diagram

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

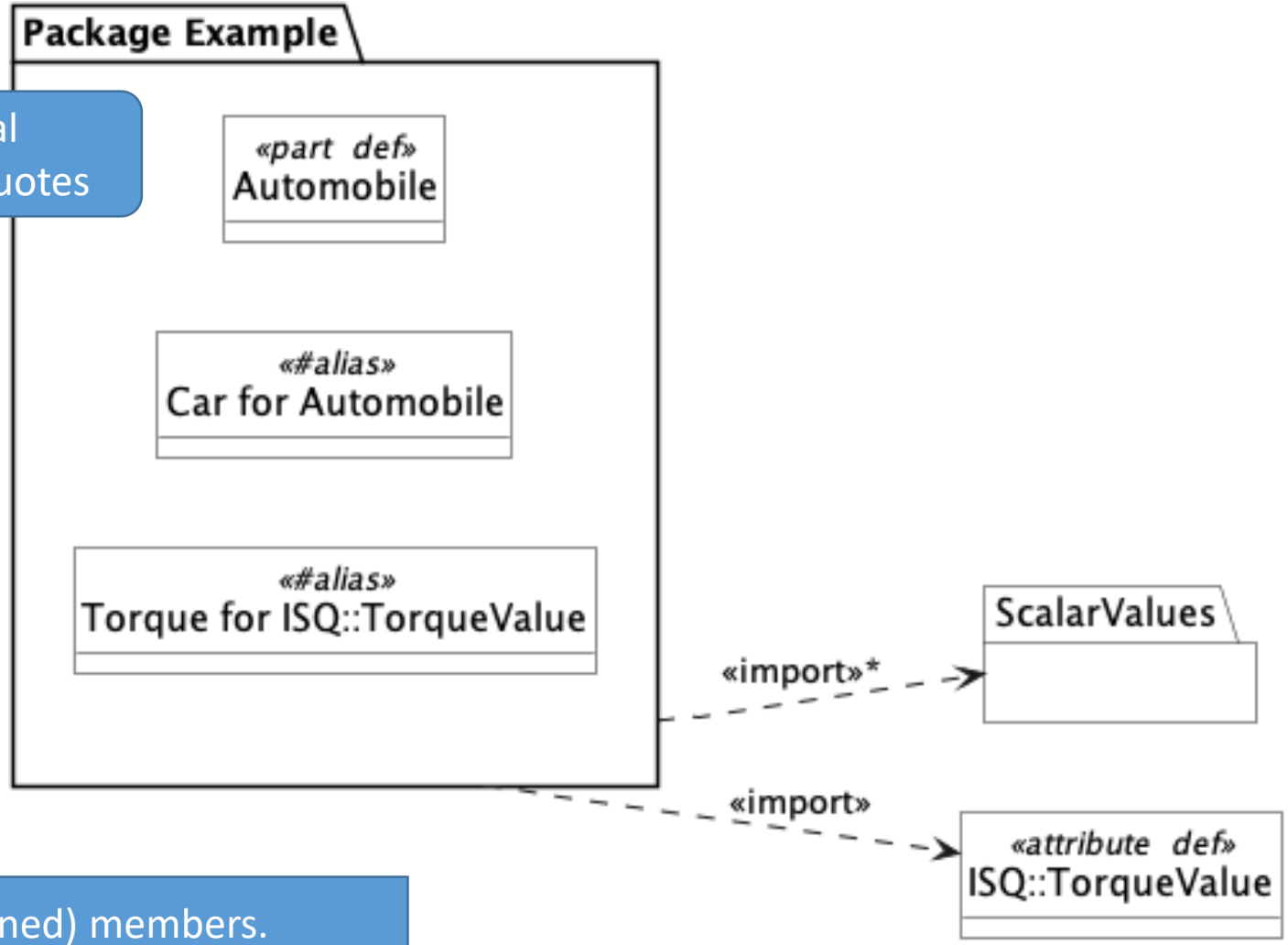
SysML v2

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

spaces or other special characters ⇨ single quotes

01. Packages/Package Example

```
package 'Package Example' {
    public import ISQ::TorqueValue;
    private import ScalarValues::*;

    private part def Automobile;

    public alias Car for Automobile;
    alias Torque for ISQ::TorqueValue;
}
```



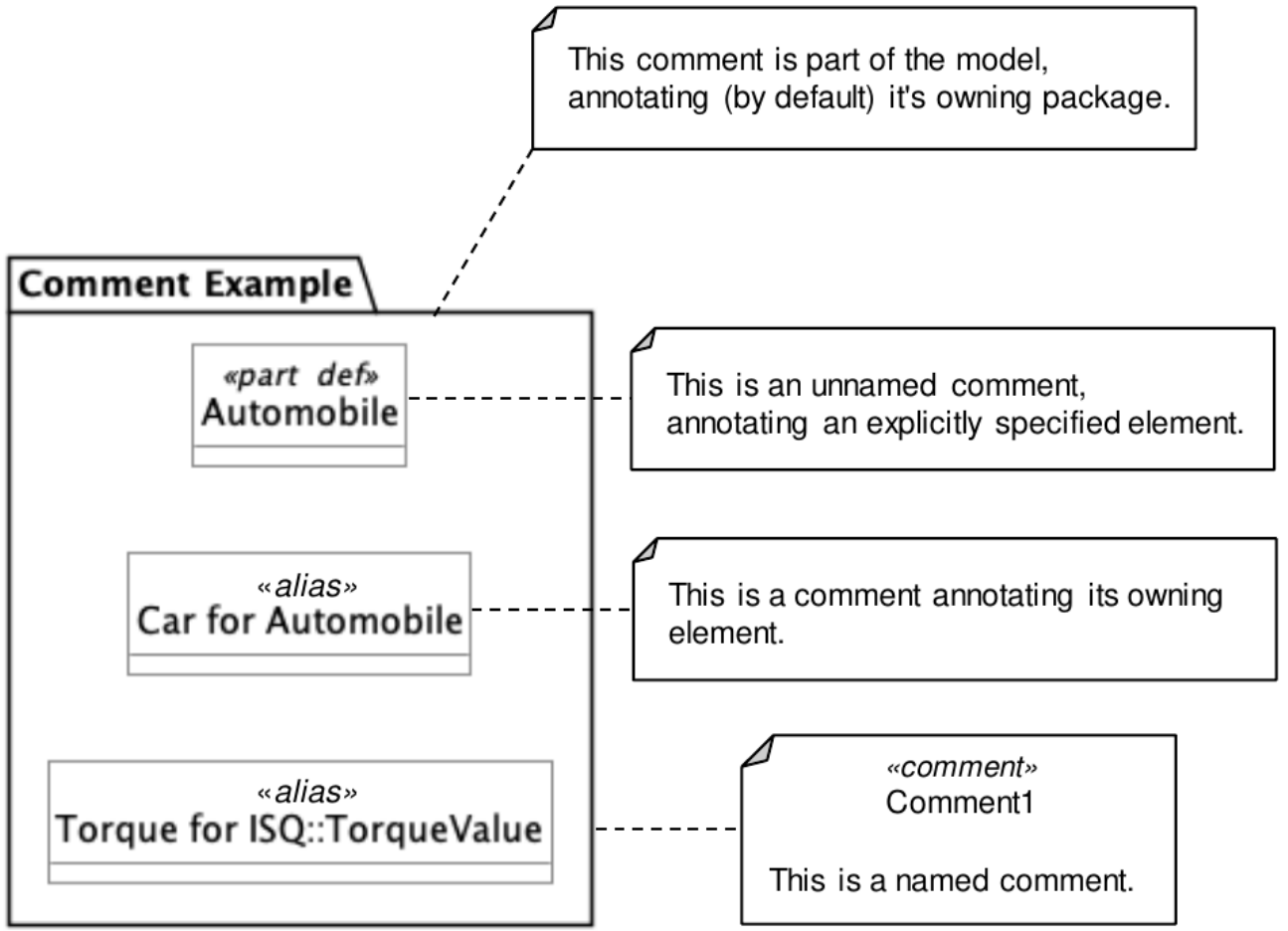
Import either single member or all members of an imported package

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

SysML v2 – Comments and Notes

01. Packages/Comment Example

```
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. It is in the text, but not part
    // of the model.
    alias Torque for ISQ::TorqueValue;
}
```

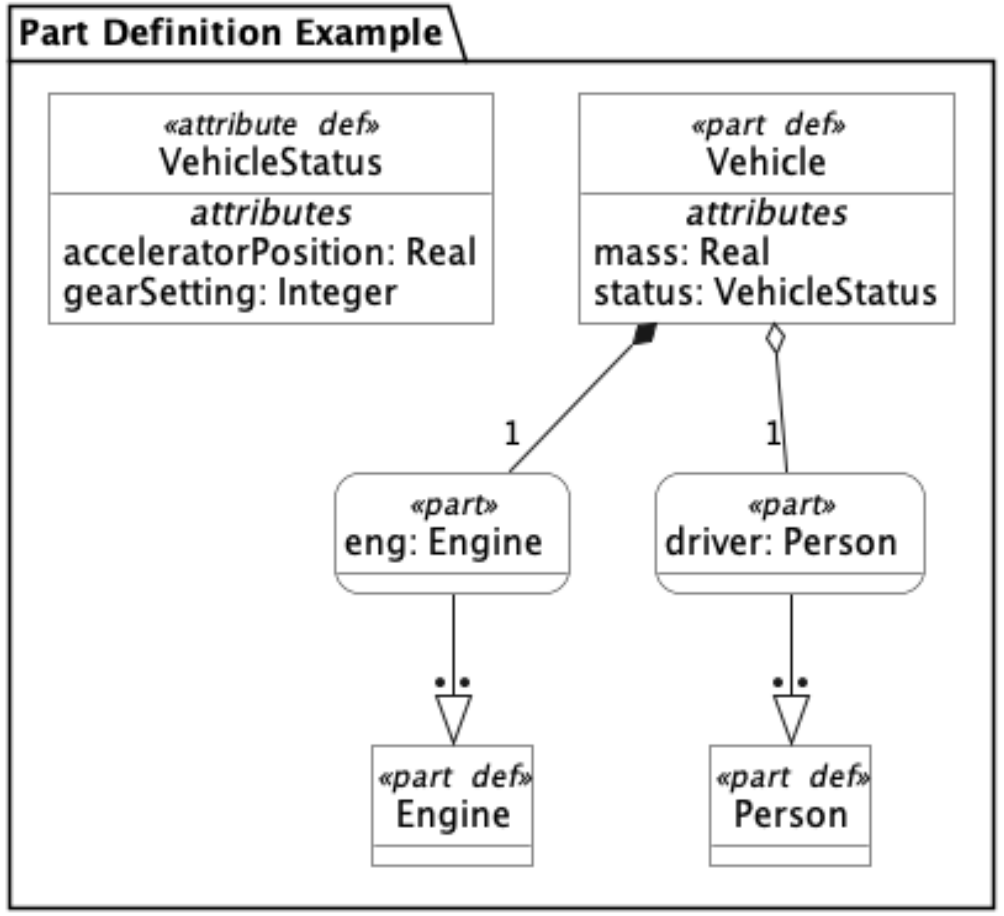


UML - SysML v2 – Terminology differences

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 : Real;
    attribute status : VehicleStatus;

    part eng : Engine;

    ref part driver : Person;
}

attribute def VehicleStatus {
    attribute gearSetting : Integer;
    attribute acceleratorPosition : Real;
}

part def Engine;
part def Person;
```



02. Part definitions/Part Definition Example

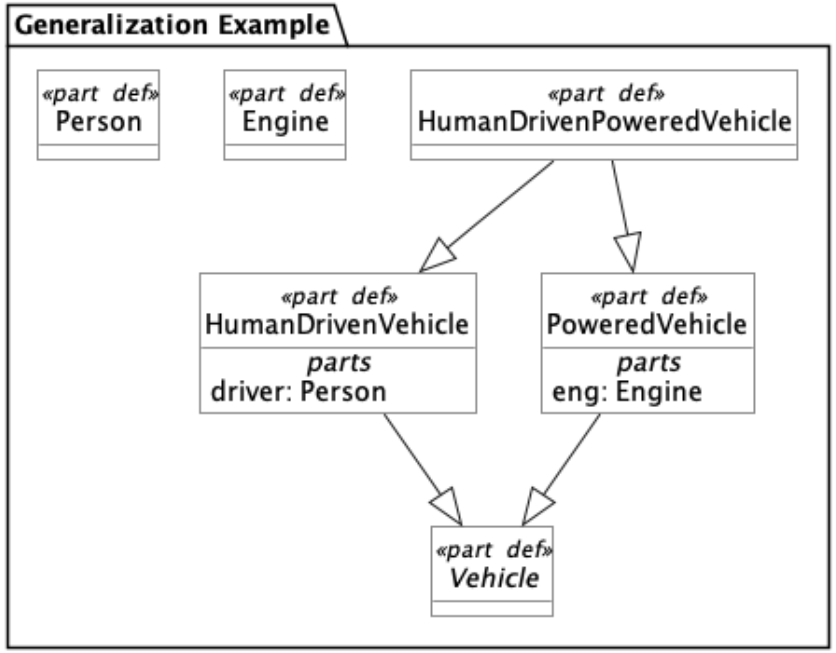
Specialization /Generalization

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.



03. Generalization/Generalization Example

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).

06. Enumeration Definitions/Enumeration Definitions-1

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".

08. Items/Items Example

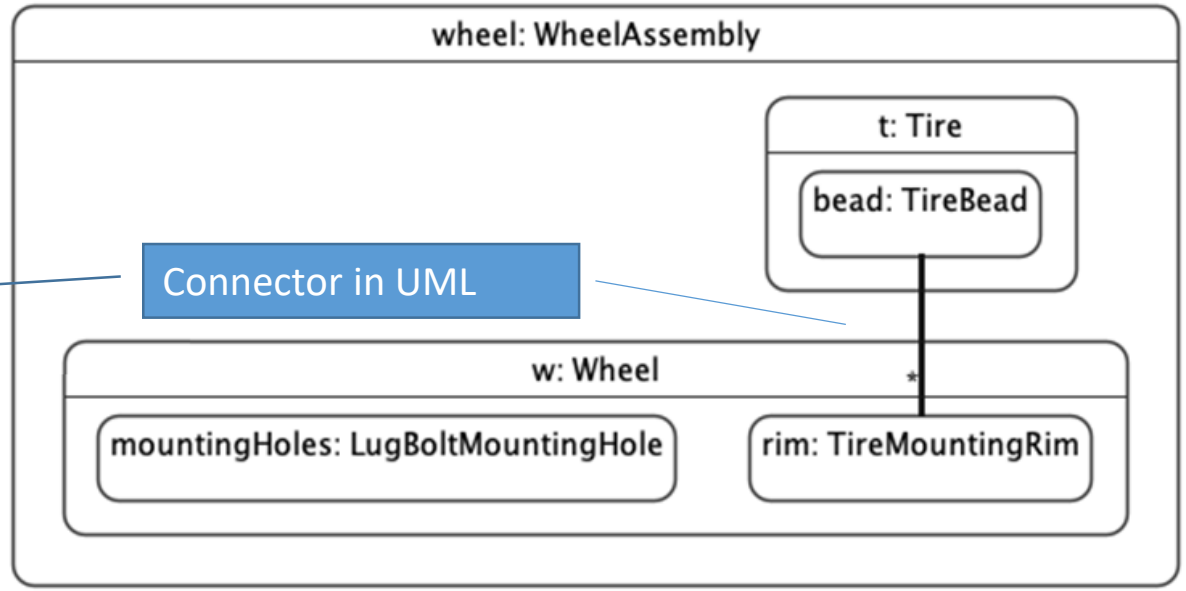
Structure – Connections and connection definitions

```
part wheelHubAssembly : WheelHubAssembly {

    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;
    }

    ...
```

```
connection def PressureSeat {
    end bead : TireBead[1];
    end mountingRim : TireMountingRim[1];
}
```

Association in UML
(typed connectors not really used in UML)

Connector in UML



09. Connections/Connections Example

Ports and port definitions

```
port def FuelPort {
    attribute temperature : Temp;
    out item fuelSupply : Fuel;
    in item fuelReturn : Fuel;
}

part def FuelTank {
    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

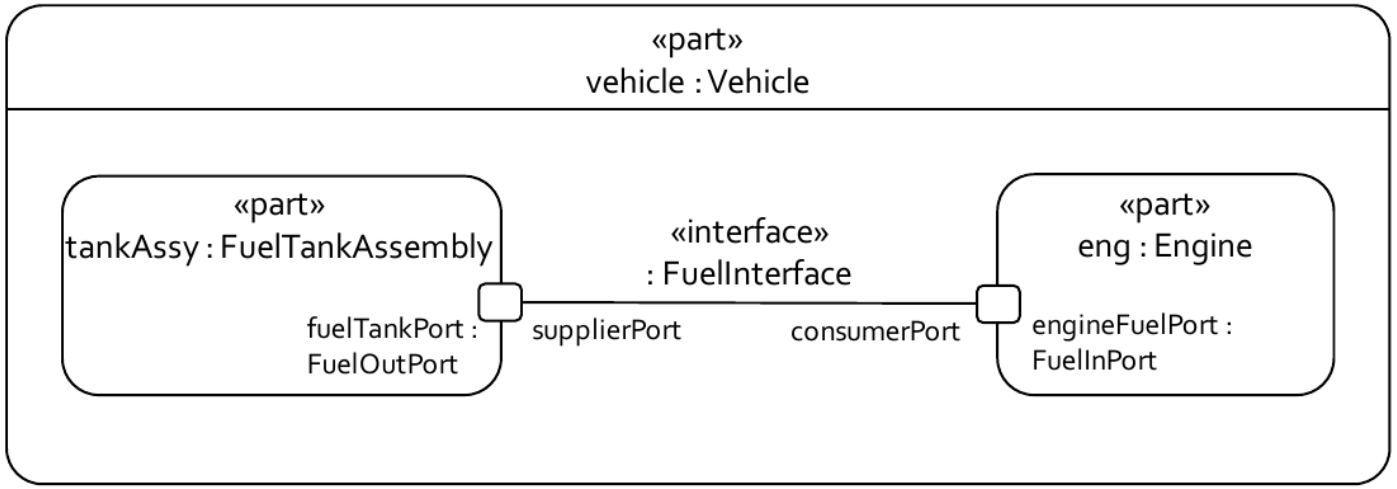
Directed features are always referential

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

Interfaces and interface definitions

```
part def Vehicle;

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



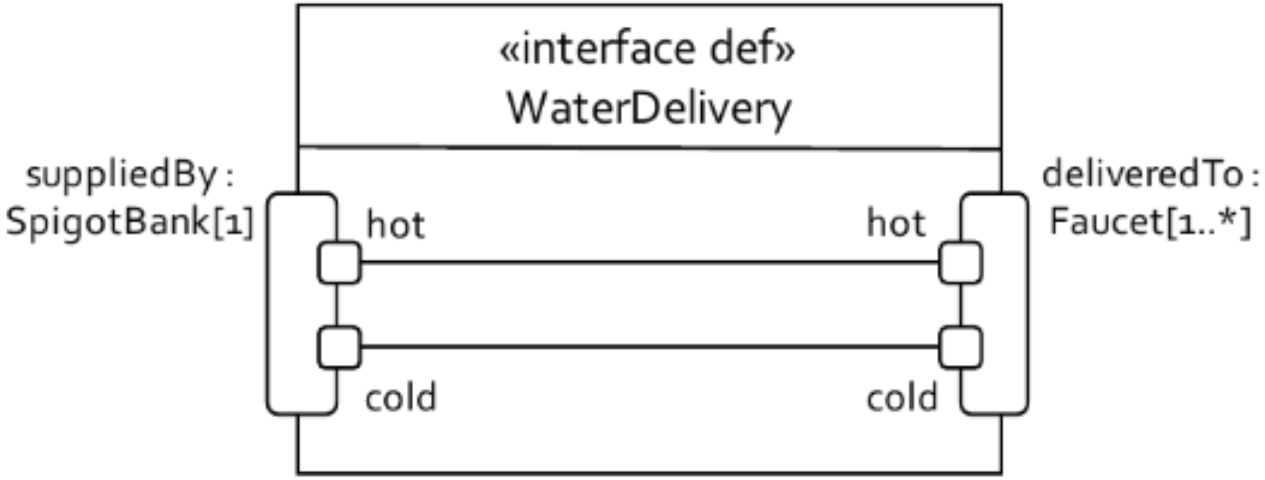
CAVEAT

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

11. Interfaces/Interface Example

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).

12. Binding Connectors/Binding Connectors Example-1

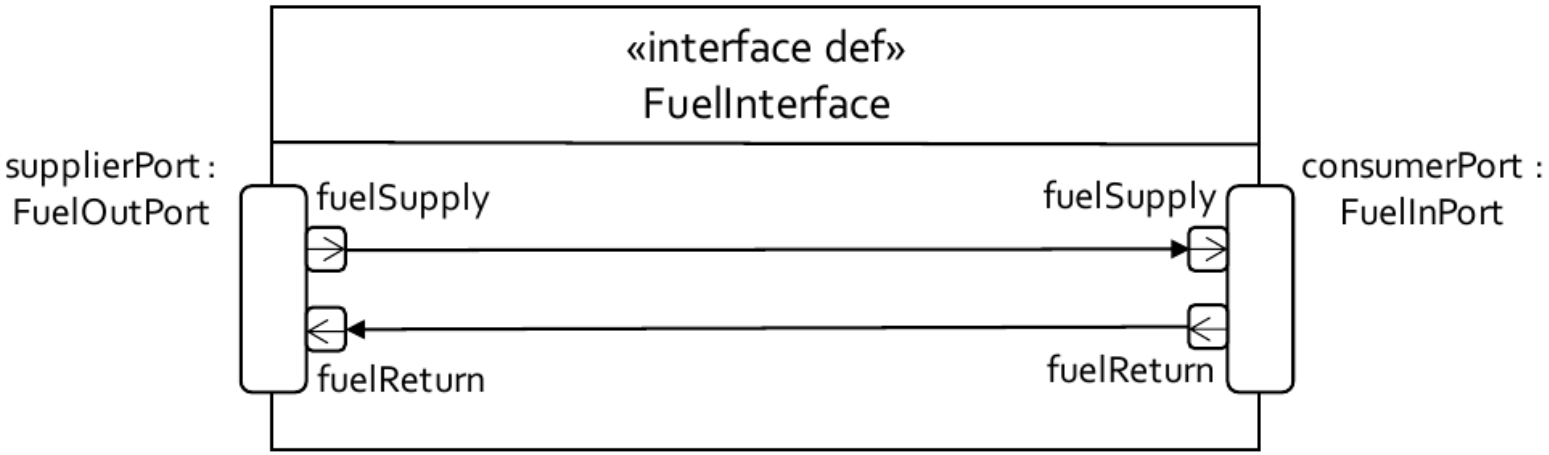
```
part vehicle : Vehicle {
    part tank : FuelTankAssembly {
        port redefines fuelTankPort {
            out item redefines fuelSupply;
            in item redefines fuelReturn;
        }

        bind fuelTankPort.fuelSupply = pump.pumpOut;
        bind fuelTankPort.fuelReturn = tank.fuelIn;

        part pump : FuelPump {
            out item pumpOut : Fuel;
            in item pumpIn : Fuel;
        }
        ...
```

Not a data-flow, rather an alias

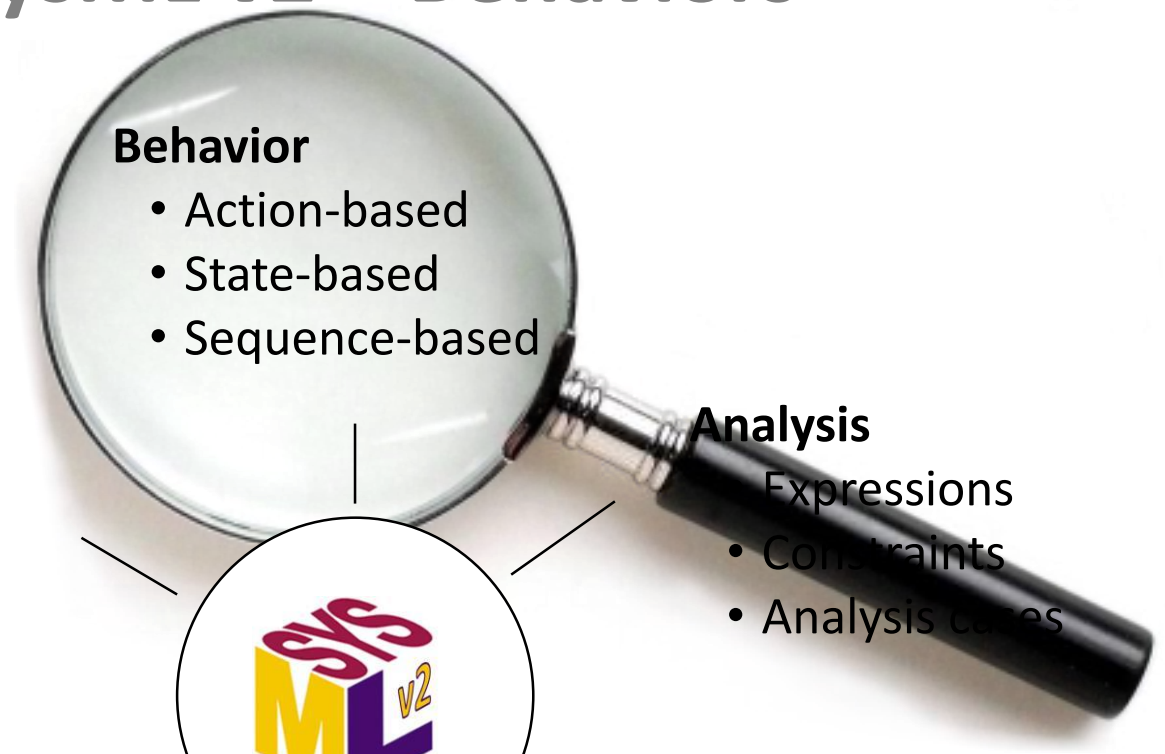
Flow connections in interfaces

```
interface def FuelInterface {
    end supplierPort : FuelOutPort;
    end consumerPort : FuelInPort;

    flow supplierPort.fuelSupply to consumerPort.fuelSupply;
    flow consumerPort.fuelReturn to supplierPort.fuelReturn;
}
```

13. Flow Connections/Flow Connection Interface Example



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

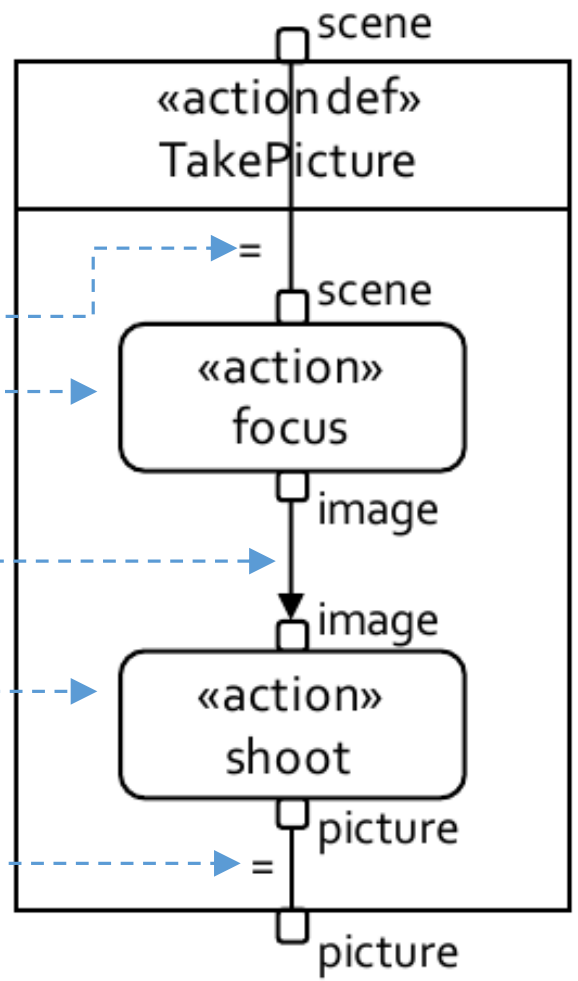
Requirements
- assumed / required constraints
- Stakeholders
- Concerns
- Use cases

SysML v2 – Action definition

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

14. Action Definitions/Action Definition Example

```
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 from focus.image to shoot.image;

    action shoot: Shoot { in image; out picture; }

    bind shoot.picture = picture;
}
```

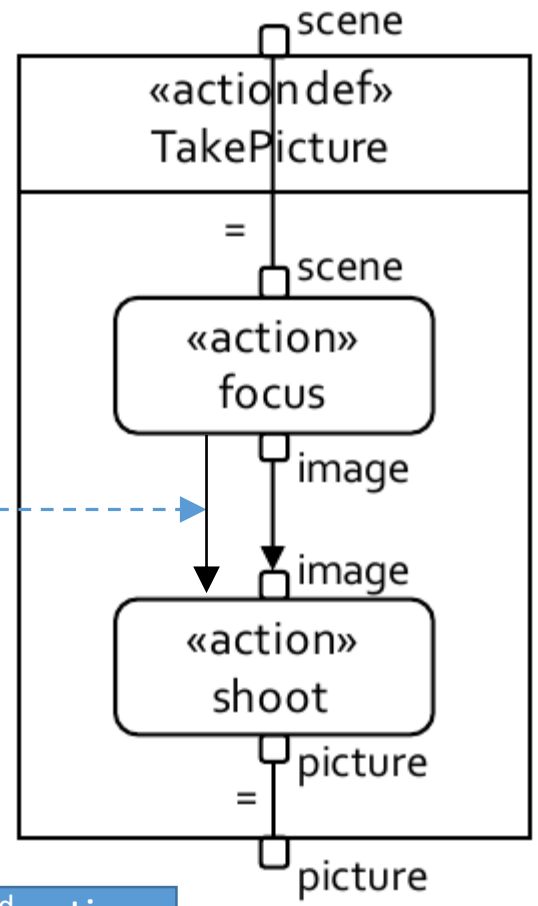


transfer items between actions via flow connection

SysML v2 – succession & succession flow

14. Action Definitions/Action Succession Example-1

```
action def Focus { in scene : Scene; out image : Image; }
action def Shoot { in image: Image; out picture : Picture; }

action def TakePicture {
    in item scene : Scene;
    out item picture : Picture;

    bind focus.scene = scene;

    action focus: Focus { in scene; out image; }

    flow from focus.image to shoot.image;

    first focus then shoot;

    action shoot: Shoot { in image; out picture; }

    bind shoot.picture = picture;
}
```



Succession asserts 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>

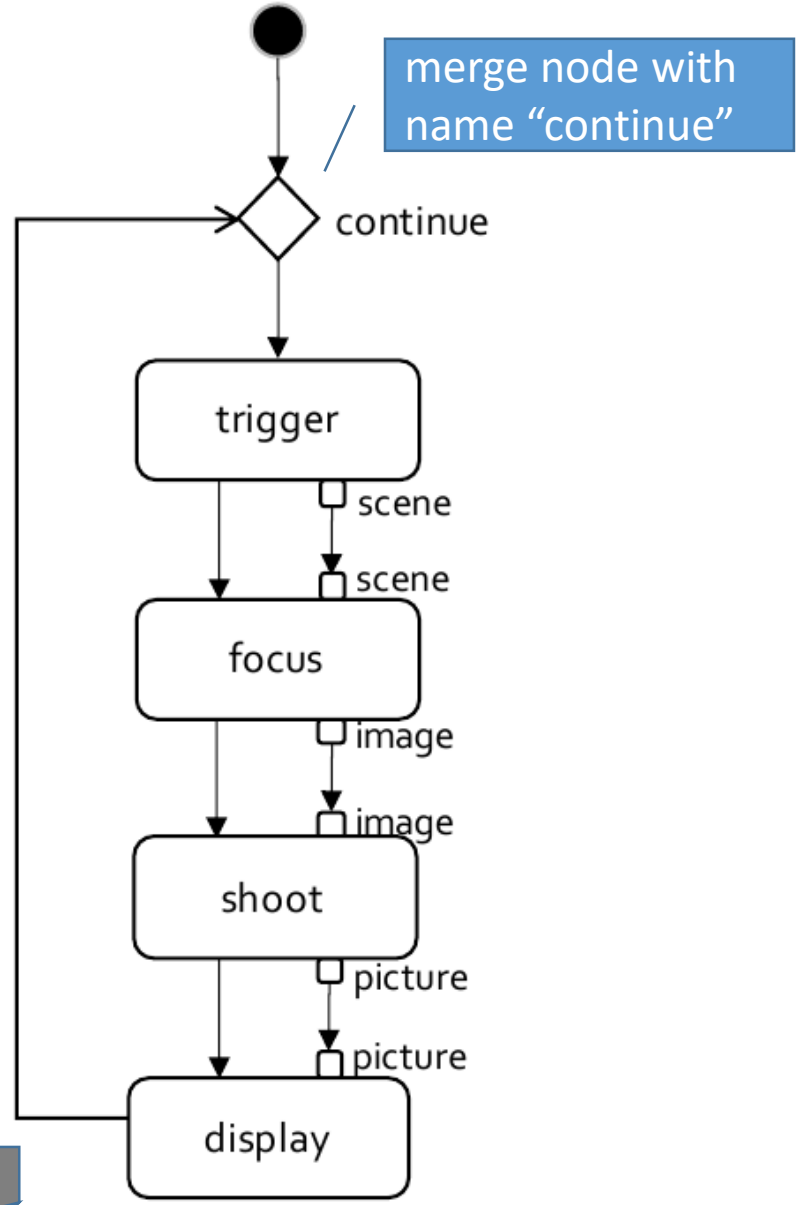
  - Asserts that second action follows first only if a guard condition is true

16. Conditional Succession/Conditional Succession Example-1/-2

SysML v2 – merge nodes / loops

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

Wait for start to complete



17. Control/Merge Example

SysML v2 – merge nodes / loops

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

Exit, if condition becomes false

or

loop action takePicture {
    …
} **until** battery.charge < 0.05

Exit, if condition becomes true

17. Control/Control Structure Examples

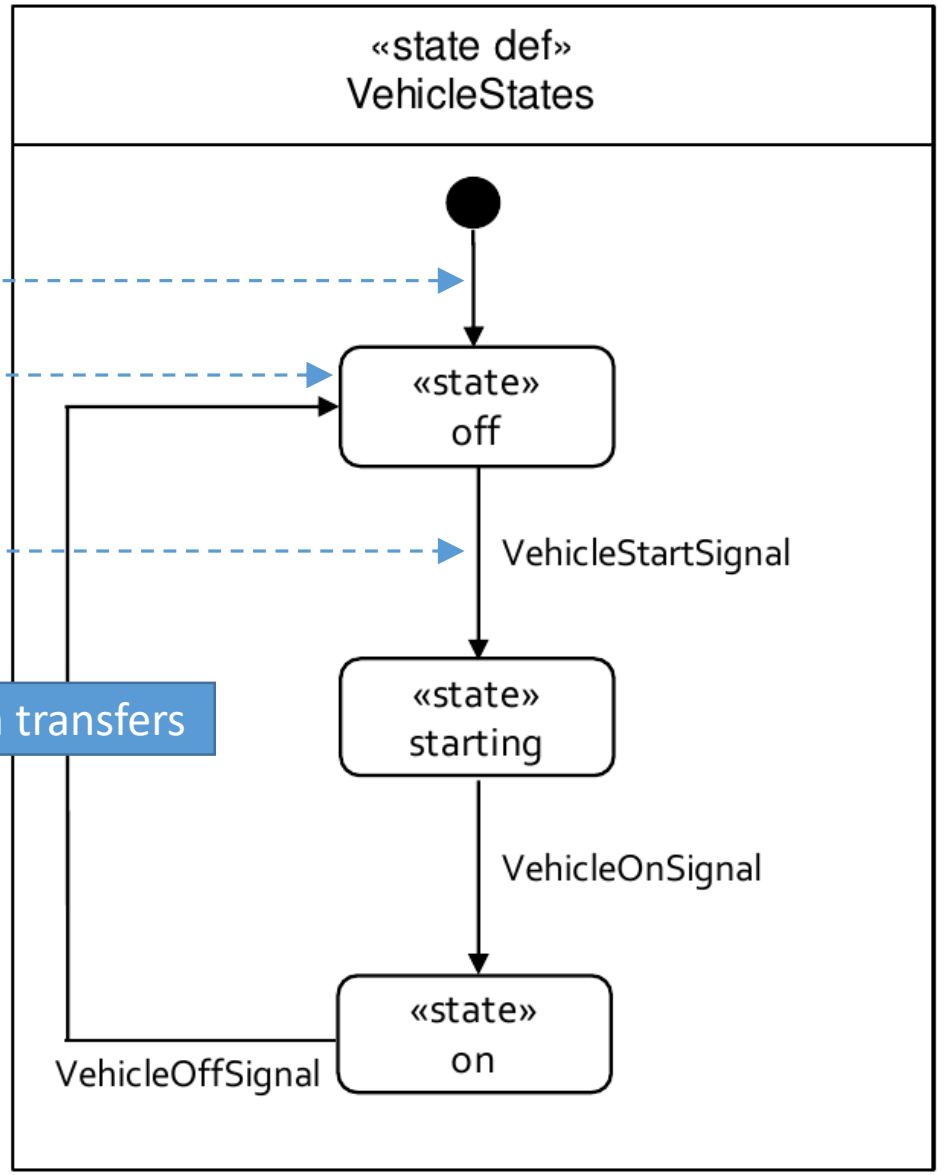
Beyond UML
Control structures as in programming languages

States

```
package 'State Definition Example-2' {

    attribute def VehicleStartSignal;
    attribute def VehicleOnSignal;
    attribute def VehicleOffSignal;

    state def VehicleStates {
        entry; then off;

        state off;
        accept VehicleStartSignal
            then starting;

        state starting;
        accept VehicleOnSignal
            then on;

        state on;
        accept VehicleOffSignal
            then off;
    }
}
```

initial state after entry is ”off”

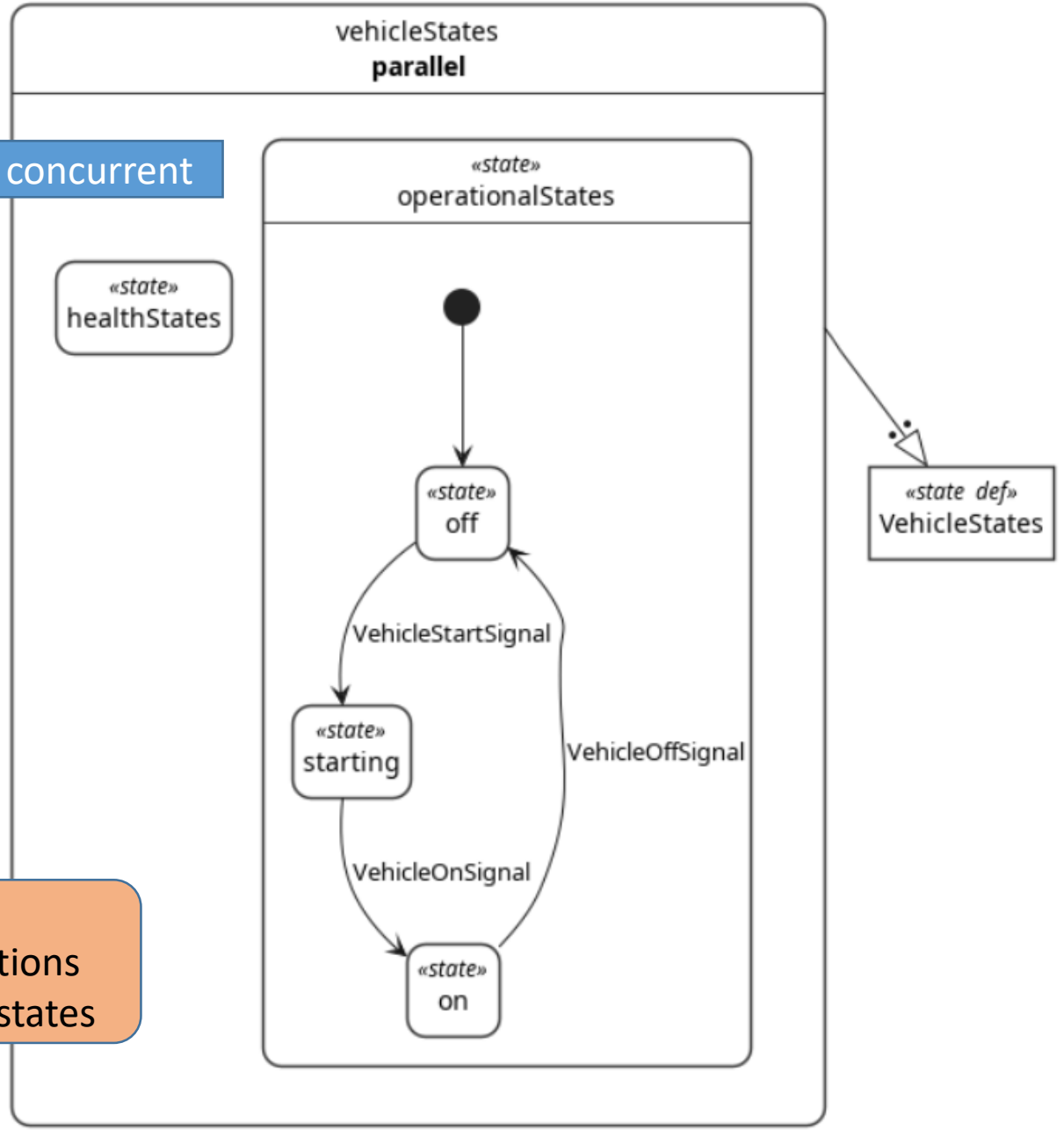
fire on *acceptance* of item transfers



Nested and concurrent states

parallel state ⇨ nested states are concurrent

```
state def VehicleStates;

state vehicleStates : VehicleStates parallel {
    state operationalStates {
        entry; then off;

        state off;
        accept VehicleStartSignal then starting;

        state starting;
        accept VehicleOnSignal then on;

        state on;
        accept VehicleOffSignal then off;
    }

    state healthStates {
        // ...
    }
}
```



CAVEAT
As in UML, no transitions between concurrent states

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	state definition / state usage
Transition / trigger	attribute definition (signal), accept

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 Power { in whlpwr : PowerValue; in Cd : Real; in Cf : Real; in tm : MassValue; in v : SpeedValue;
    attribute drag = Cd * v;
    attribute friction = Cf * tm * v;

    return : PowerValue = whlpwr - drag - friction;
}

calc def Acceleration { in tp: PowerValue; in tm : MassValue; in v : SpeedValue;
    return : AccelerationValue = tp / (tm * v);
}

calc def Velocity { in dt : TimeValue; in v0 : SpeedValue; in a : AccelerationValue;
    return : SpeedValue = v0 + a * dt;
}

calc def Position { in dt : TimeValue; in x0 : LengthValue; in v : SpeedValue;
    return : LengthValue = x0 + v * dt;
}
```

Beyond UML
Calculations based on KerML

Calculation Usage

```
part def VehicleDynamics {
    attribute C_d : Real;
    attribute C_f : Real;
    attribute wheelPower : PowerValue;
    attribute mass : MassValue;

    action straightLineDynamics {
        in delta_t : TimeValue;
        in v_in : SpeedValue;
        in x_in : LengthValue;
        out v_out : SpeedValue = vel.v;
        out x_out : LengthValue = pos.x;

        calc acc : Acceleration {
            in tp = Power(wheelPower, C_d, C_f, mass, v_in);
            in tm = mass;
            in v = v_in;
            return a;
        }
```

30. Calculations/Calculation usages-1

Reference Acceleration function definition

Binds input parameters to values, with nested function call

...

Constraint definition/usage

```
constraint def MassConstraint {
    in partMasses : MassValue[0..*];
    in massLimit : MassValue;

    sum(partMasses) <= massLimit
}

part def Vehicle {
    constraint massConstraint : MassConstraint {
        in partMasses = (chassisMass, engine.mass, transmission.mass);
        in massLimit = 2500[kg];
    }

    attribute chassisMass : MassValue;

    part engine : Engine {
        attribute mass : MassValue;
    }

    part transmission : Engine {
        attribute mass : MassValue;
    }
}
```

31. Constraints/Constraints Example-1

Convention: usage with lowerCase, Definition with upper case

Assign values to attributes of definition

Part usage adds an attribute

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

Requirements definition

```
package 'Requirement Definitions' {
    private import ISQ::*;
    private import SI::*;

    requirement def MassLimitationRequirement {
        doc /* The actual mass shall be less than or equal to the required mass. */

        attribute massActual: MassValue;
        attribute massReqd: MassValue;

        require constraint { massActual <= massReqd }
    }

    part def Vehicle {
        attribute dryMass: MassValue;
        attribute fuelMass: MassValue;
        attribute fuelFullMass: MassValue;
    }
```

32. Requirements/Requirement Definitions

Requirement definition is generic, concrete values are filled in Usage

Defines at least one constraint

Beyond UML
Can evaluate values within constraints

Requirements usage

```
requirement <'1.1'> fullVehicleMassLimit : VehicleMassLimitationRequirement {
    subject vehicle : Vehicle;
    attribute :>> massReqd = 2000[kg];

    assume constraint {
        doc /* Full tank is full. */
        vehicle.fuelMass == vehicle.fuelFullMass
    }
}

requirement <'1.2'> emptyVehicleMassLimit : VehicleMassLimitationRequirement {
    subject vehicle : Vehicle;
    attribute :>> massReqd = 1500[kg];

    assume constraint {
        doc /* Full tank is empty. */
        vehicle.fuelMass == 0[kg]
    }
}
```

Define what the requirement is about

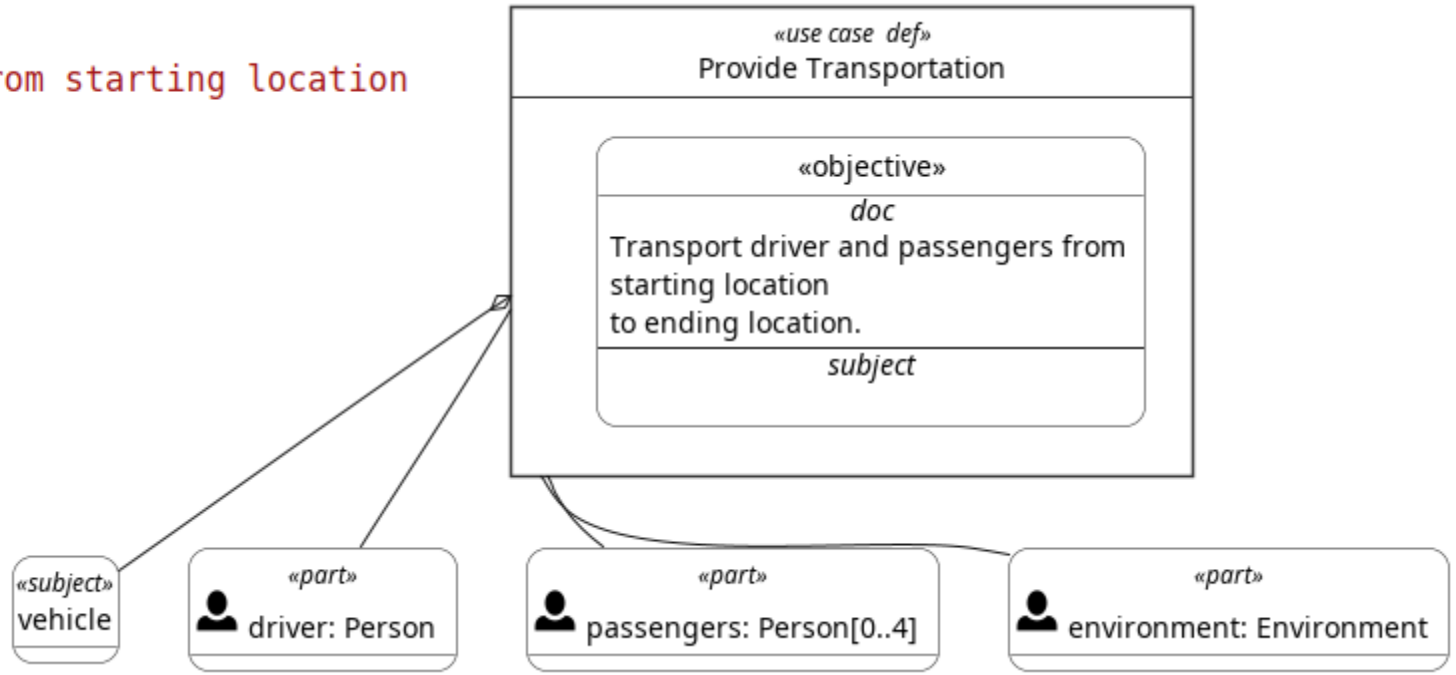
Shortcut for redefinition

32. Requirements/Requirement Usage

Use cases

```
use case def 'Provide Transportation' {
    subject vehicle : Vehicle;

    actor driver : Person;
    actor passengers : Person[0..4];
    actor environment : Environment;

    objective {
        doc
        /* Transport driver and passengers from starting location
         * to ending location.
         */
    }
}
```

35. Use cases/Use Case Definition Example



Use cases

```
use case 'provide transportation' : 'Provide Transportation' {
    first start;

    then include use case 'enter vehicle' : 'Enter Vehicle' {
        actor :>> driver = 'provide transportation'::driver;
        actor :>> passengers = 'provide transportation'::passengers;
    }

    then use case 'drive vehicle' {
        actor driver = 'provide transportation'::driver;
        actor environment = 'provide transportation'::environment;

        include 'add fuel'[0..*] {
            actor :>> fueler = driver;
        }
    }

    then include use case 'exit vehicle' : 'Exit Vehicle' {
        actor :>> driver = 'provide transportation'::driver;
        actor :>> passengers = 'provide transportation'::passengers;
    }

    then done;
}
```

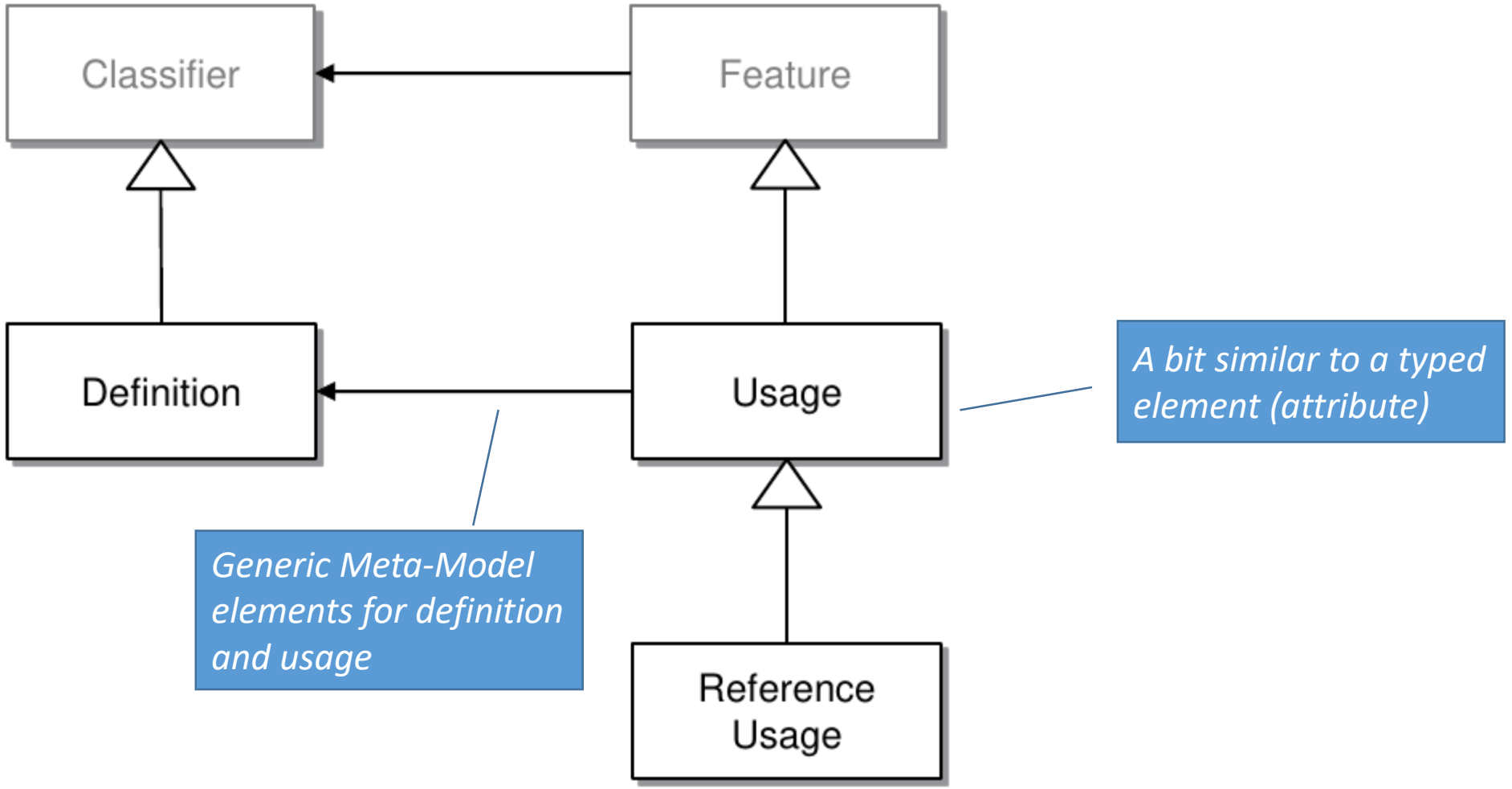
Sequencing as in actions

35. Use cases/Use Case Usage Example

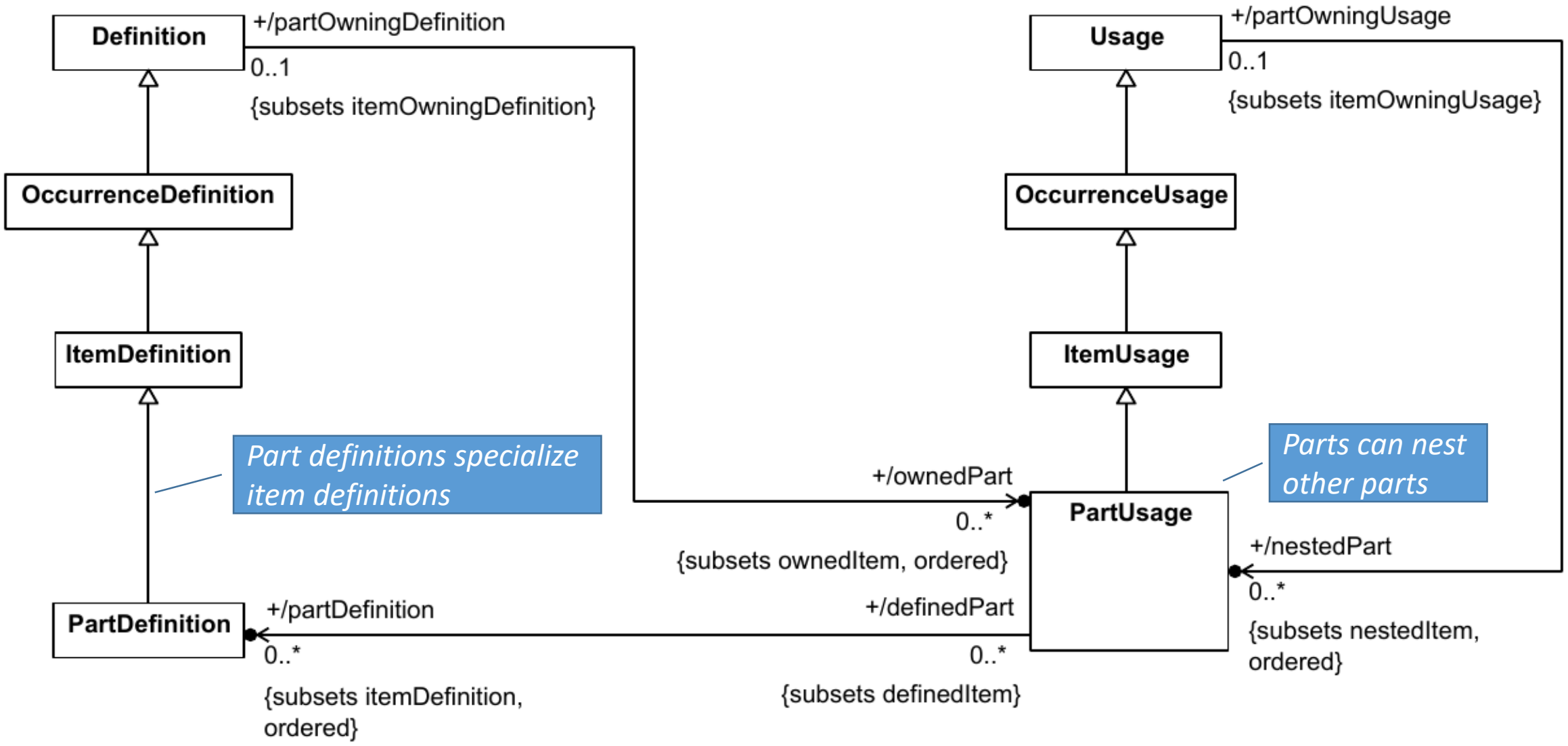
Results in quite complex diagram, not shown

Beyond UML
combines actions with use cases

Meta-Model excerpts (KerML)



Meta-Model excerpts



Functions/Expressions