



SST

# Introduction to the SysML v2 Language Graphical Notation (2021-09-18)

This presentation of the SysML v2 language uses the proposed graphical modeling notation that is being formally specified in the SysML v2 specification, but is not yet fully complete or reflected in the SST pilot implementation. The graphical syntax is intended to complement the textual syntax.

.

*Copyright © 2021 by Sanford Friedenthal*

*Licensed under the Creative Commons Attribution 4.0 International License.*

*To view a copy of this license, visit <http://creativecommons.org/licenses/by/4.0/> or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.*



# Updates

*SST*

9-20-21: Slide 38 – removed @



# SysML v2 Functional Areas

SST

- |                             |  |
|-----------------------------|--|
| 1. Packages & Element Names | 13. Expressions & Calculations               |
| 2. Definition Elements      | 14. Quantities & Units                       |
| 3. Usage Elements           | 15. Constraints                              |
| 4. Part Decomposition       | 16. Requirements                             |
| 5. Part Interconnection     | 17. Use Cases                                |
| 6. Variability              | 18. Analysis Cases                           |
| 7. Actions                  | 19. Verification Cases                       |
| 8. Action Flow              | 20. Dependency & Allocation Relationships    |
| 9. States                   | 21. Annotations                              |
| 10. Interactions            | 22. Element Filters                          |
| 11. Individuals             | 23. View & Viewpoint                         |
| 12. Timeslices & Snapshots  | 24. Language Extension (not implemented yet) |



# General Caveats

- The graphical representation is intended to complement the textual notation
  - The textual notation has been formally specified
  - The graphical notation is being formally specified using graphical BNF
  - The graphical visualization in this presentation is captured in Visio, but will need to be rendered in a more precise way to reflect the specification in the future
  - Two visualization prototypes are under development (Tom Sawyer and PlantUML)
- SysML v2 Marker
  - Some of the more significant SysML v2 changes in functionality and terminology relative to SysML v1 are designated with the SysML v2 marker **v2**

# Overview



# SysML v2 Objectives

SST

Increase adoption and effectiveness of MBSE  
by enhancing...

- Precision and expressiveness of the language
- Consistency and integration among language concepts
- Interoperability with other engineering models and tools
- Usability by model developers and consumers
- Extensibility to support domain specific applications
- Migration path for SysML v1 users and implementors



# Key Elements of SysML v2

SST

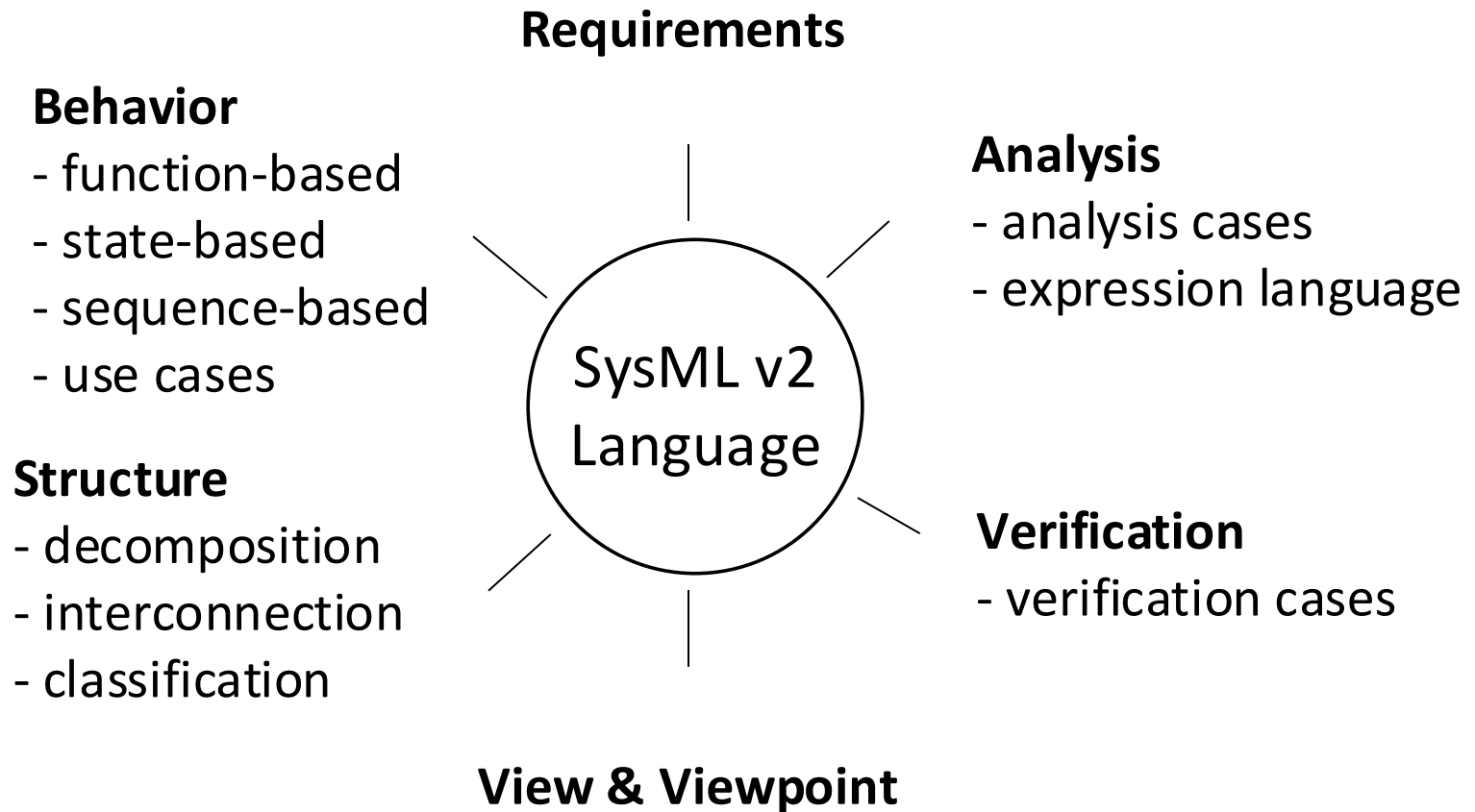
- New Metamodel that is not constrained by UML
  - Preserves most of UML modeling capabilities with a focus on systems modeling
  - Grounded in formal semantics
- Robust visualizations based on flexible view & viewpoint specification and execution
  - Graphical, Tabular, Textual
- Standardized API to access the model



# SysML v2 Language Capabilities

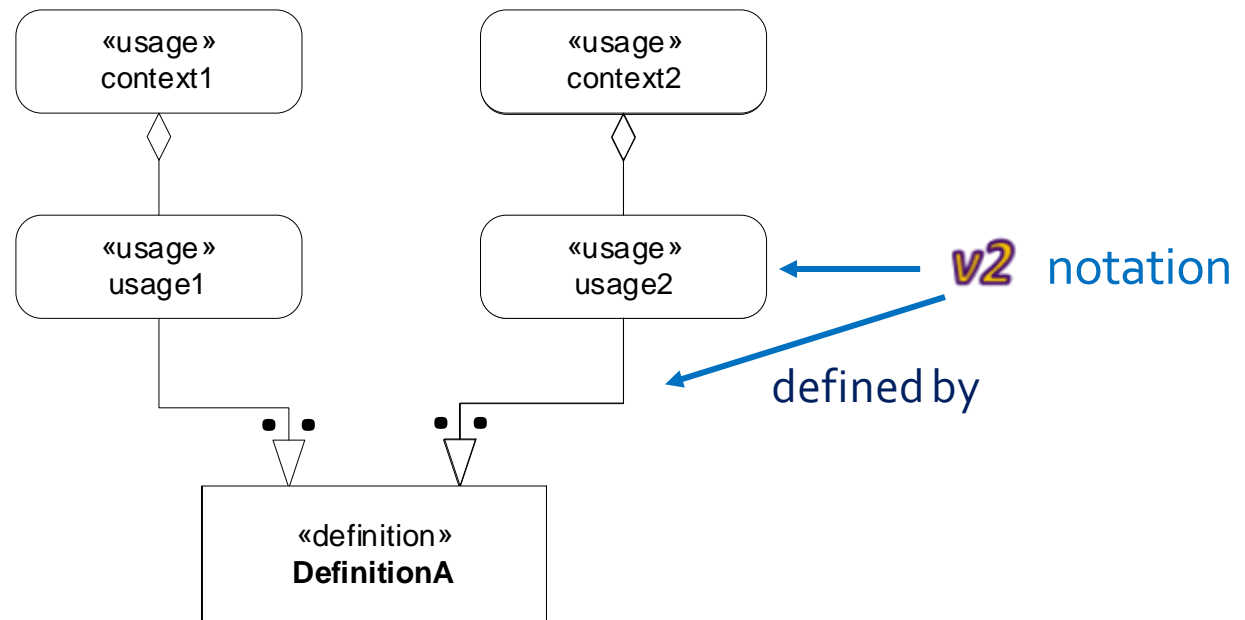
SST

Updated





- A definition element defines an element such as a part, action, or requirement
- A usage element is a usage of a definition element in a particular context
  - There can be different usages of the same definition element in either different contexts or the same context
- Pattern is applied consistently throughout the language **v2**





# SysML v2 to v1

## Terminology Mapping (partial)

SST

- SysML v2 includes consistent usage/definition terminology

SysML v2	SysML v1
part / part def	part property / block
attribute / attribute def	value property / value type
port / port def	proxy port / interface block
action / action def	action / activity
state / state def	state / state machine
constraint / constraint def	constraint property / constraint block
requirement / requirement def	requirement
connection / connection def	connector / association block

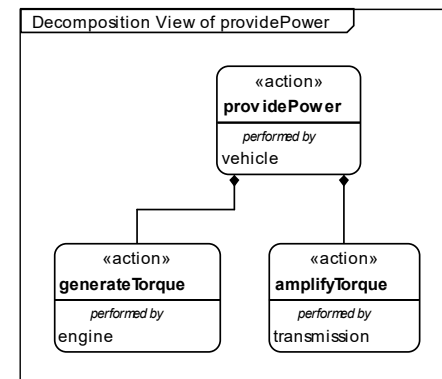
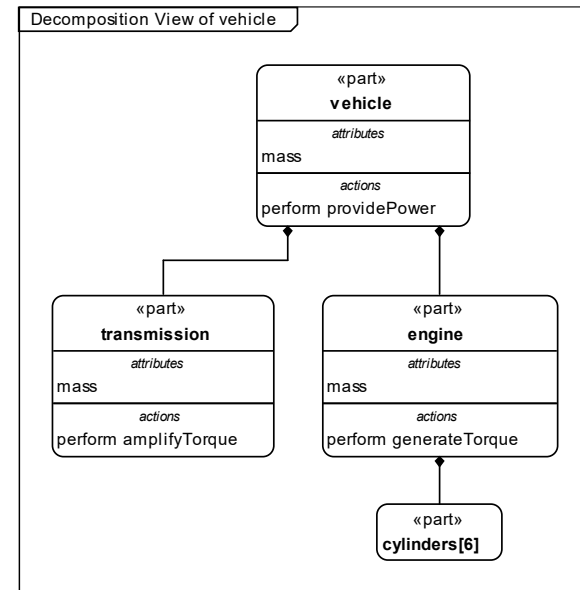


# SysML v2 Notation (1 of 2)

## Textual and Graphical

SST

```
package 'Vehicle Parts Tree' {  
    part vehicle {  
        attribute mass;  
        perform providePower;  
        part engine {  
            attribute mass;  
            perform generateTorque;  
            part cylinders [6];  
        }  
        part transmission {  
            attribute mass;  
            perform amplifyTorque;  
        }  
    }  
}  
  
package 'Vehicle Action Tree'{  
    action providePower {  
        action generateTorque;  
        action amplifyTorque;  
    }  
}
```



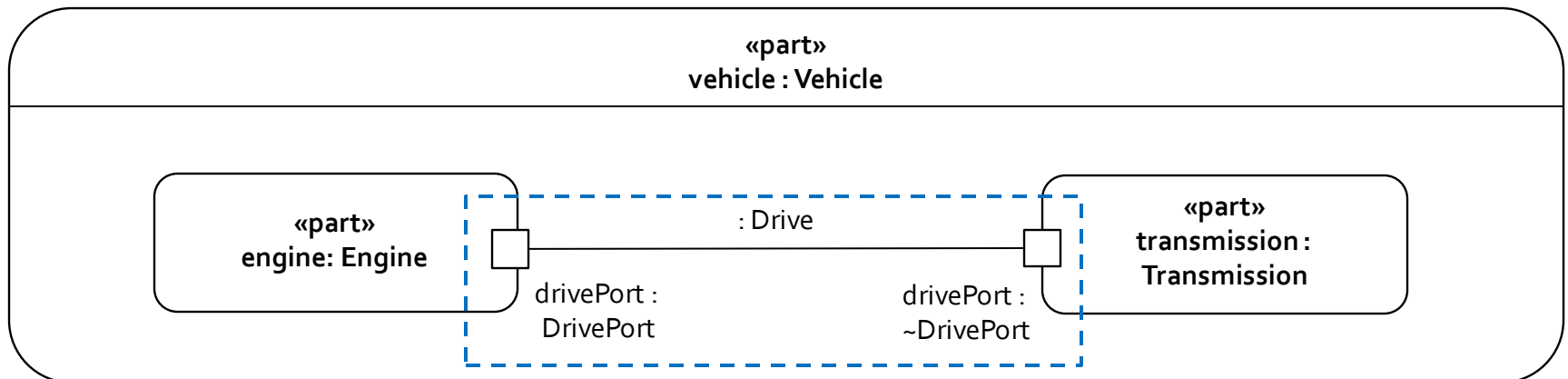


# SysML v2 Notation (2 of 2)

## Textual and Graphical

SST

```
interface def Drive {  
  end enginePort : DrivePort;  
  end transmissionPort : ~DrivePort;  
}  
  
part vehicle : Vehicle {  
  part engine : Engine { port drivePort : DrivePort; }  
  part transmission : Transmission { port drivePort : ~DrivePort; }  
  
  interface : Drive  
    connect engine.drivePort to transmission.drivePort;  
}
```

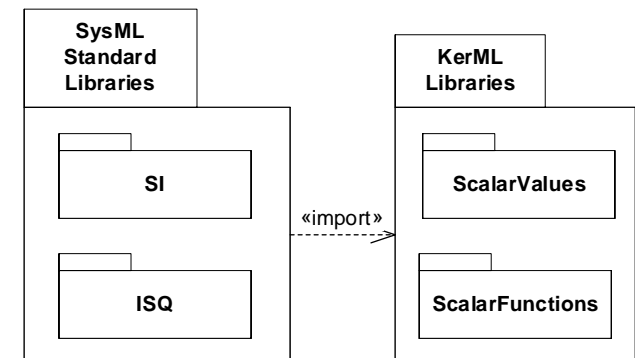
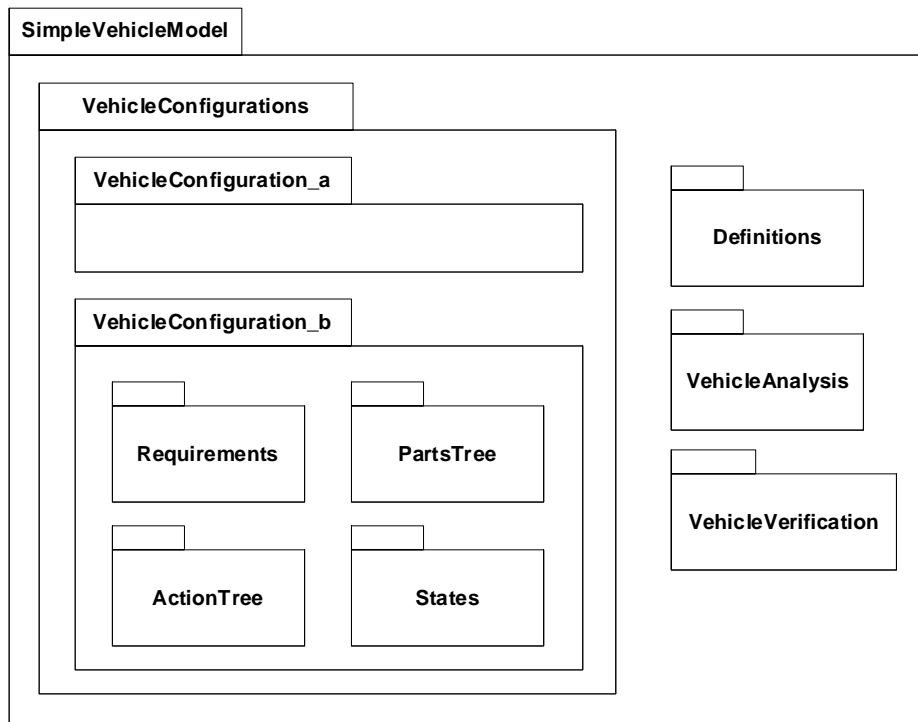


Tom Sawyer Visualization Prototype

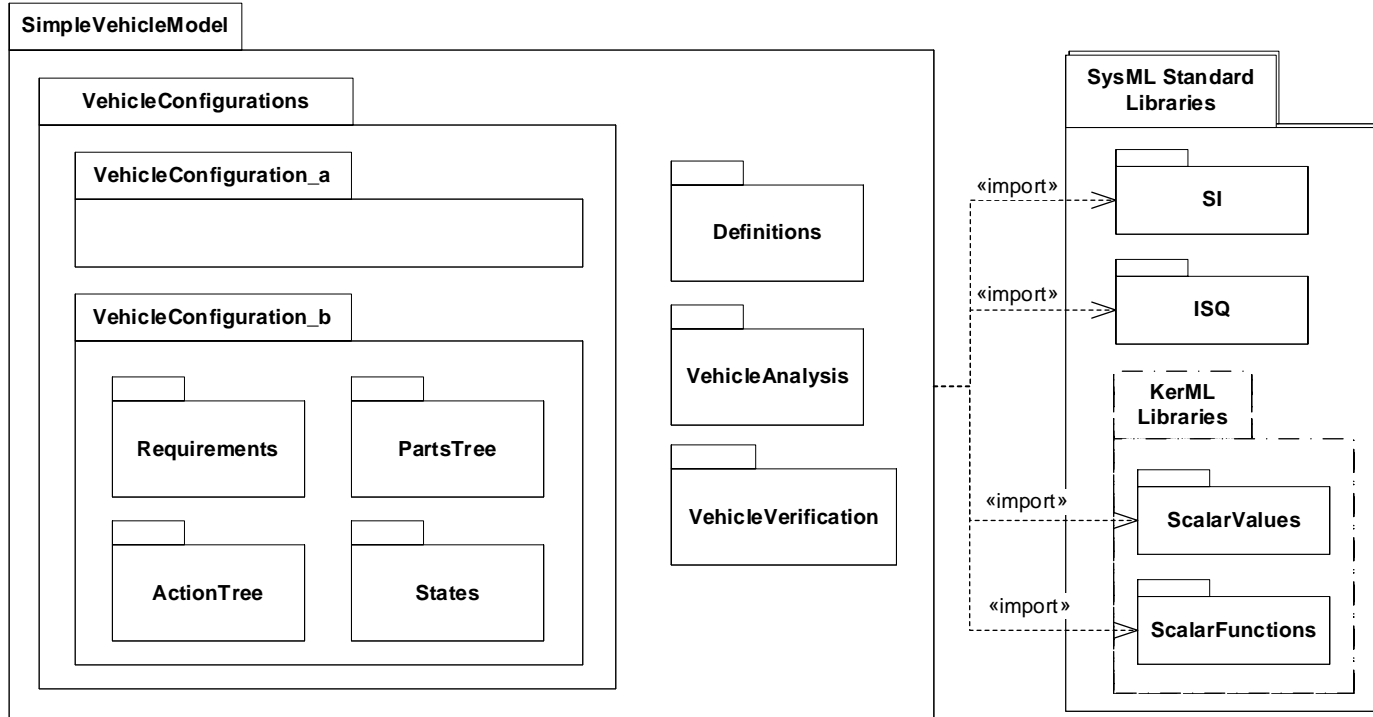
# **Module 1**

## **Packages & Element Names**

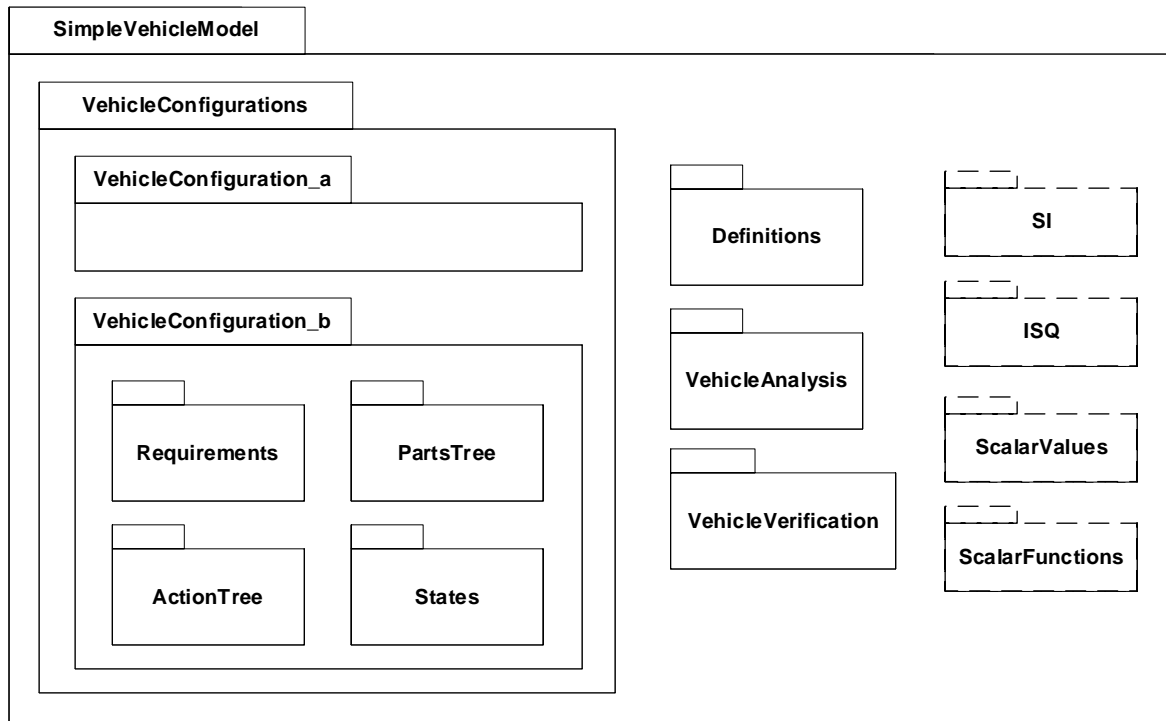
- A hierarchy of packages, where packages can contain other packages and elements
- Packages are namespaces that contain member elements that may be owned or unowned
- A package can import members of another package as unowned members, where deletion semantics do not apply, and they can be referred to by their local name



- A hierarchy of packages, where packages can contain other packages and elements
- Packages are namespaces that contain member elements that may be owned or unowned
- A package can import members of another package as unowned members, where deletion semantics do not apply, and they can be referred to by their unqualified name
- Nested import notation enables flexible organization **v2**

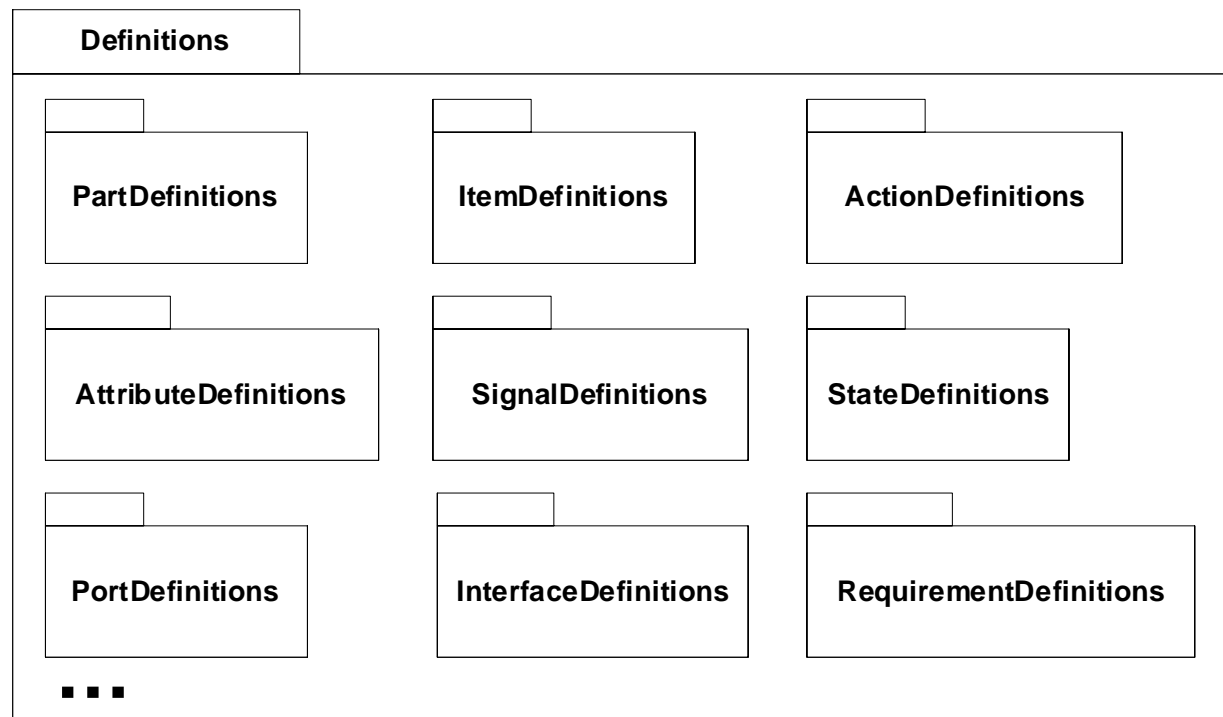


- A hierarchy of packages, where packages can contain other packages and elements
- Packages are namespaces that contain member elements that may be owned or unowned
- A package can import members of another package as unowned members, where deletion semantics do not apply, and they can be referred to by their unqualified name
  - Nested import notation enables flexible organization **v2**

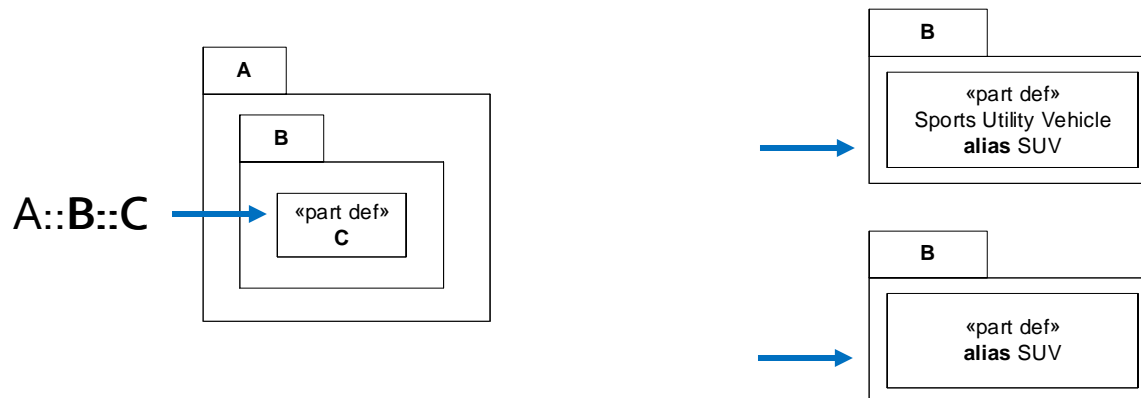




- Example: Definitions package contains packages to define different kinds of elements
  - Ellipsis indicates there is more content



- A qualified name of a model element is prepended by the name of its owner followed by a double colon (::)
- A fully qualified name is prepended by the concatenated names of its owners in its containment hierarchy separated by double colons (::)
- An element can have one or more aliases **v2**
- Each element includes a UUID and optionally other id's, such as a URI **v2**
  - UUID: universally unique identifier
  - URI: Uniform Resource Identifier



# Module 2

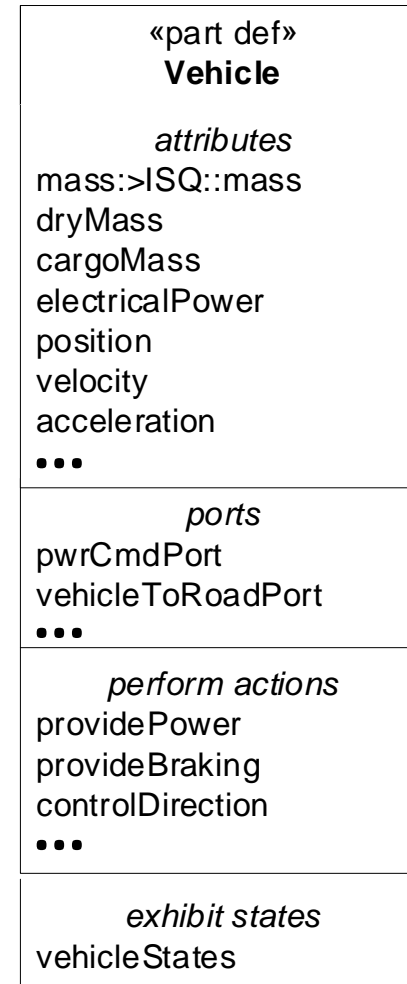
## Definition Elements



# Part Definition

SST

- A Part Definition can contain features such as attributes, ports, actions, and states
  - A modular unit of structure





# Other Kinds of Definition Elements

## Terminology v2

SST

- Each kind of definition element can contain specific kinds of features
  - Attribute definition
    - Data type that defines a set of data values
    - Primitive attribute definitions include integer, Real, Complex, Boolean, String, ....
    - Can include quantity kinds such as Torque with units
    - Can include complex data types such as vectors
  - Port definition
    - Defines a connection point on parts that enable interactions
    - Contain kinds of features including directed features with in, out, and in/out
    - Ports can be conjugated (i.e., reverse direction of directed features)
  - Item definition v2
    - Defines kind of entity that may be acted on, such as a flow, or a stored item
  - Action definition
    - Defines kind of behavior that transforms inputs and outputs
  - State definition
    - Defines kind of behavior that responds to events
    - Enables entry, exit, and do actions

```
«attribute def»  
Real
```

```
«attribute def»  
Torque
```

```
«port def»  
~ FuelCmdPort  
  
directed features  
in item fuelCmd:FuelCmd  
  
attributes  
x:Real
```

```
«item def»  
FuelCmd
```

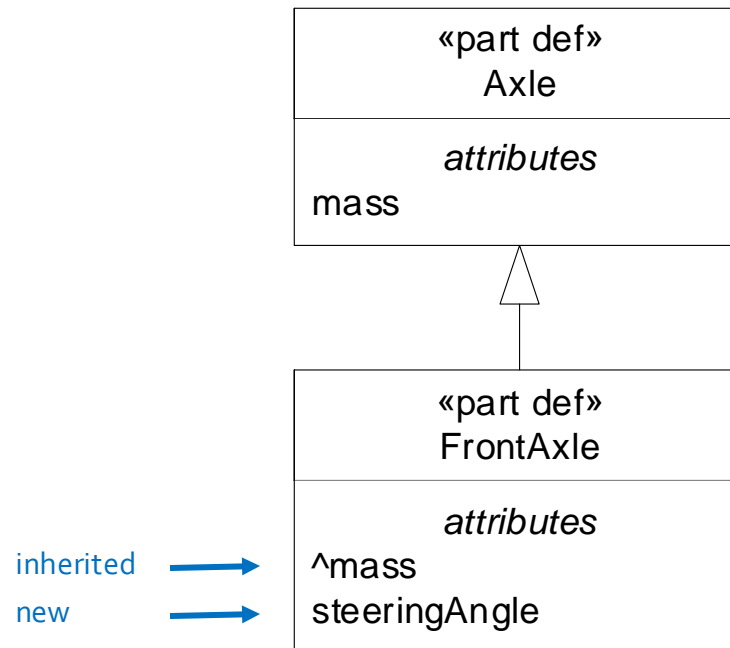
```
«item def»  
Fuel  
  
attributes  
fuelMass:>ISQ::mass
```

```
«action def»  
ProvidePower  
  
parameters  
in pwrCmd:PwrCmd  
out torqueToWheels:Torque [*]
```

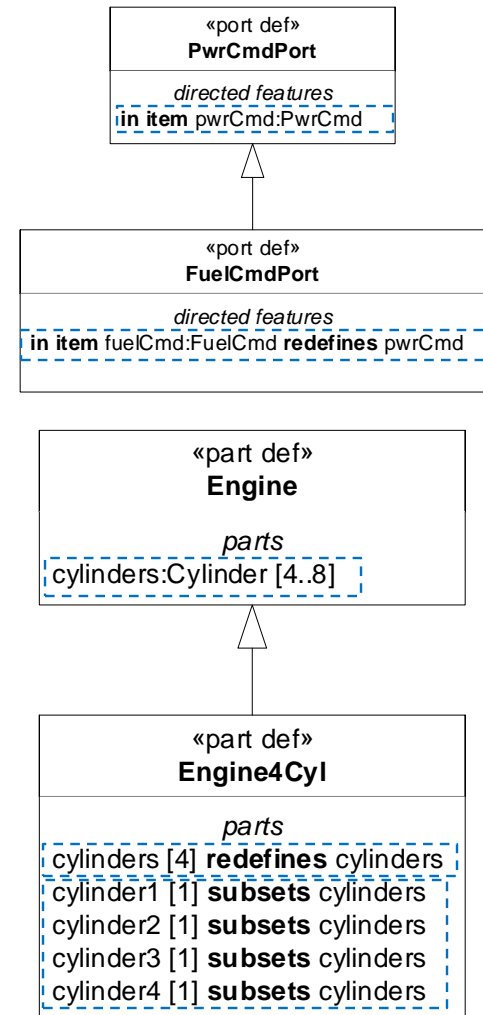
```
«state def»  
VehicleStates  
  
actions  
do providePower
```

# Specialization

- Definition elements can be specialized
  - Subclass inherits features of superclass and can add new features
  - Can be subclassed by more than one superclass (i.e., multiple inheritance)



- An inherited feature can be **redefined** by a feature whose definition is more specialized
  - The more specialized feature overrides the inherited feature
  - Symbol for redefines ( : >> ) **v2**
- An inherited feature can be **subsetting** by one or more features with a more constrained multiplicity
  - The Engine4Cyl contains a subset of the cylinders contained by Engine
  - First redefine multiplicity to 4
  - Then identify the subsetting features
  - Symbol for subsets ( : > ) **v2**

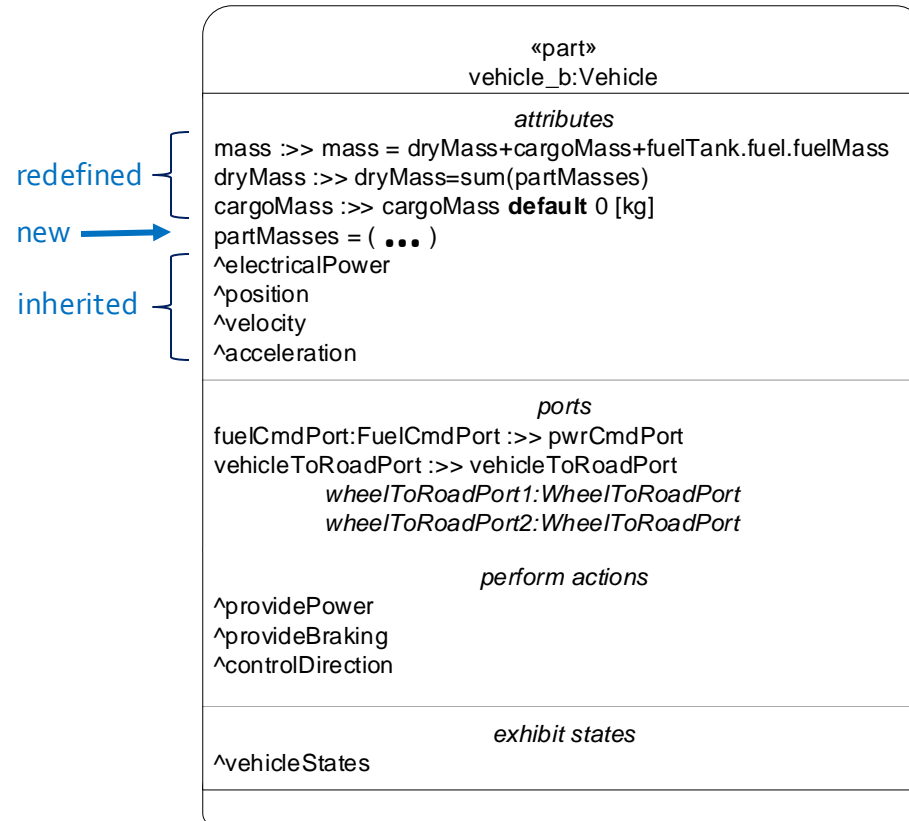


# Module 3

## Usage Elements



- A part is a usage of a part definition
- Part inherits its features from its definition **v2**
- Inherited features can be redefined or subsetted, and new features can be added





# Other Usage Elements

## Terminology v2

- Other kinds of usage elements are defined by specific kinds of definition elements
  - Attributes defined by attribute definition
  - Ports defined by port definition
  - Items defined by item definition v2
  - Actions defined by action definition
  - States defined by state definition

- A data type whose range is restricted to a set of discrete values

- Without units

- Definition: **enum def** Colors {red; blue; green;}
- Usage: **attribute** color1: Colors = blue;

«enum def» Colors
enums red blue green

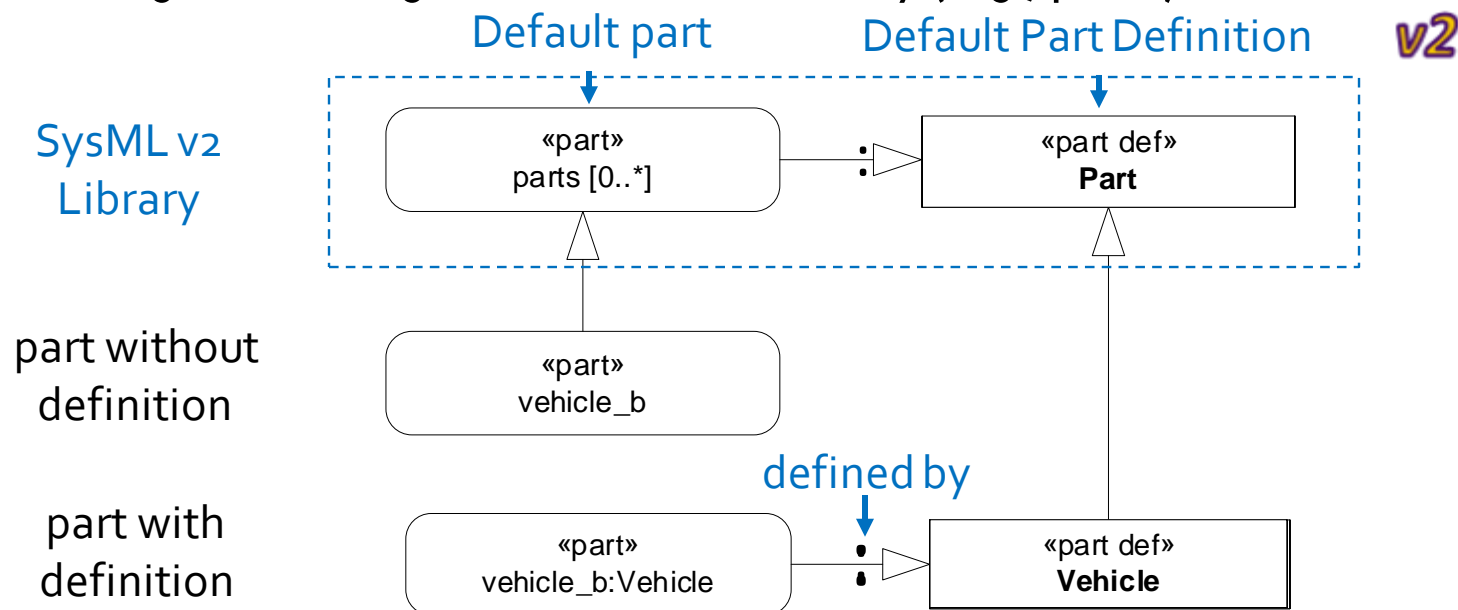
- With units:

- Definition: **enum def** DiameterChoices :> ISQ::LengthValue {  
     **enum** = 60 [mm];  
     **enum** = 70 [mm];  
     **enum** = 80 [mm];  
     }

«enum def» DiameterChoices
enums =60 [mm] =70 [mm] =80 [mm]

- Usage: **attribute** cylinderDiameter: DiameterChoices = 80 [mm];

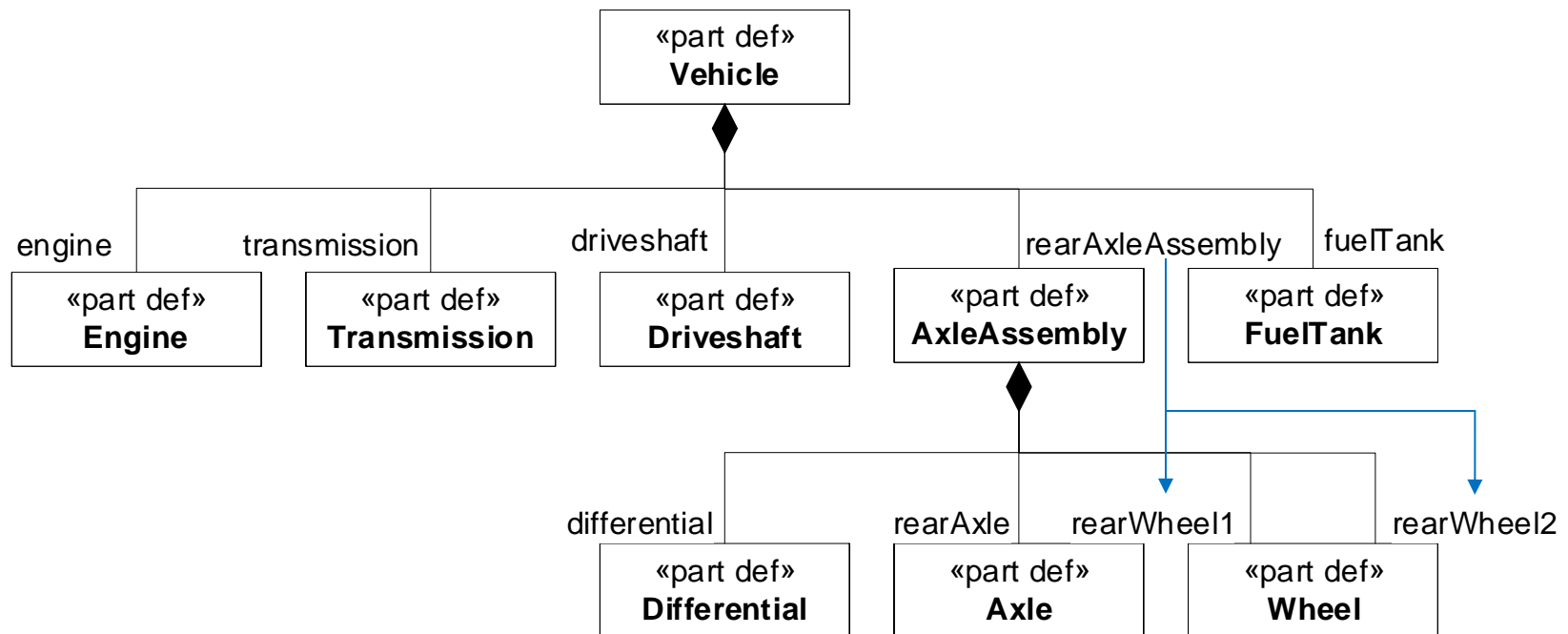
- A definition element is a subclass of the most general definition element in the SysML v2 library (e.g., Part) **v2**
- A usage element is defined by a definition element
  - *defined by* is a kind of specialization
- A usage element that is not provided a definition by the modeler is a subset of the most general usage element in the library (e.g., parts)



# Module 4

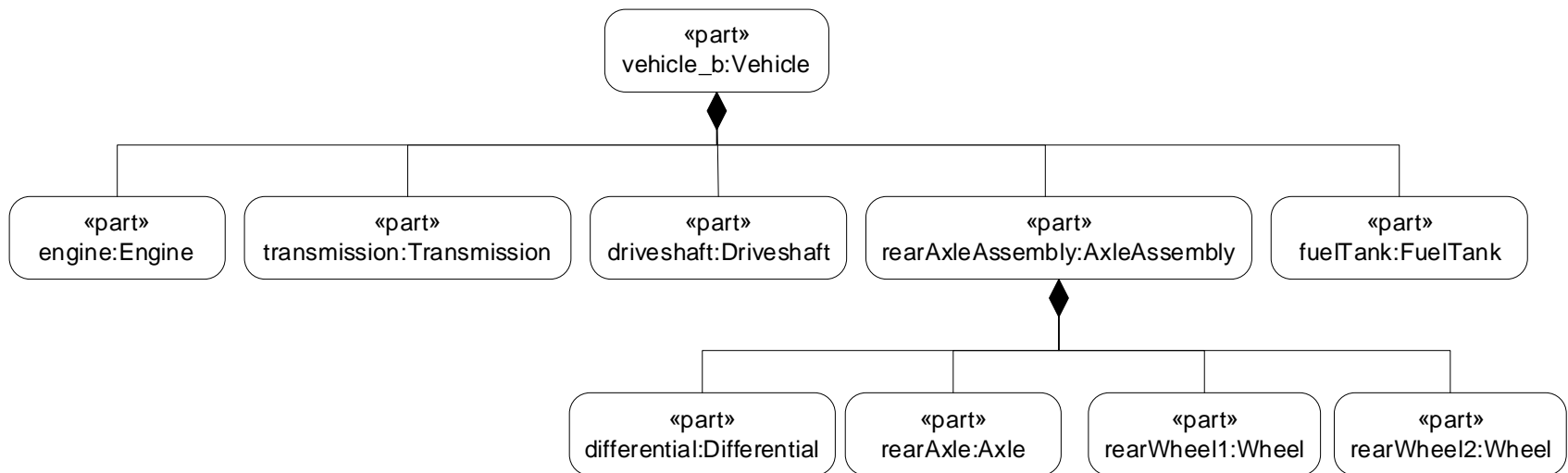
## Part Decomposition

- Can create the equivalent of SysML v1 block decomposition using part definitions
  - No part to part relationships in SysML v1



# Parts Decomposition

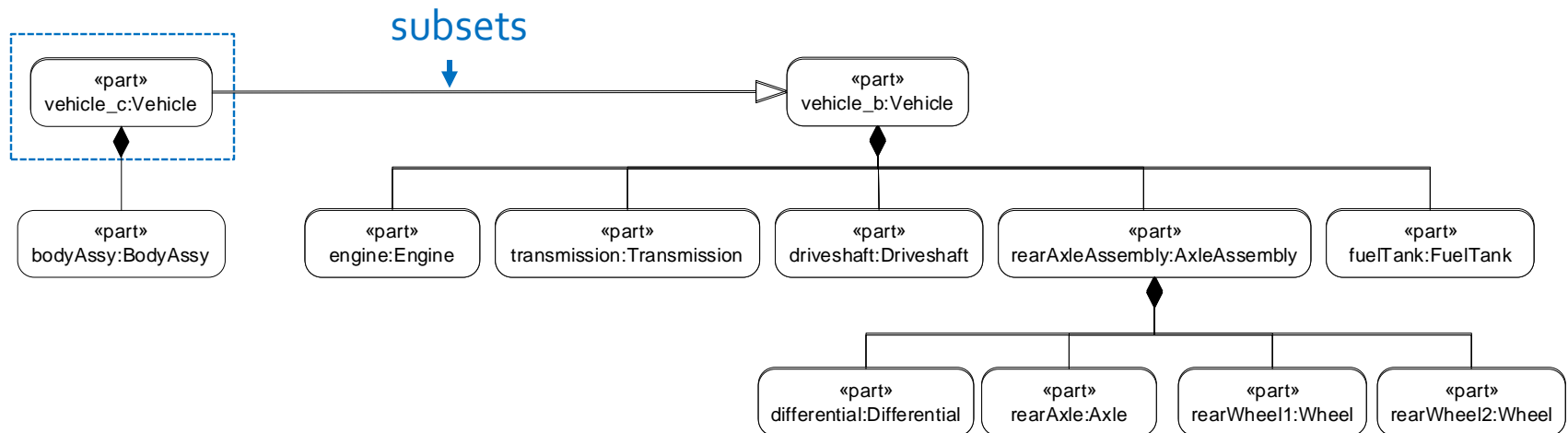
- A parts tree is a composition of a part from other parts **v2**
  - SysML v1 only supports block decomposition
  - Definition element can serve as black-box specification that is not decomposed
    - Does not commit to an internal structure (e.g., decomposition)
  - Can provide significant advantages (more intuitive, less ambiguous, easier to modify a design configuration)



# Parts Tree Reuse & Modification

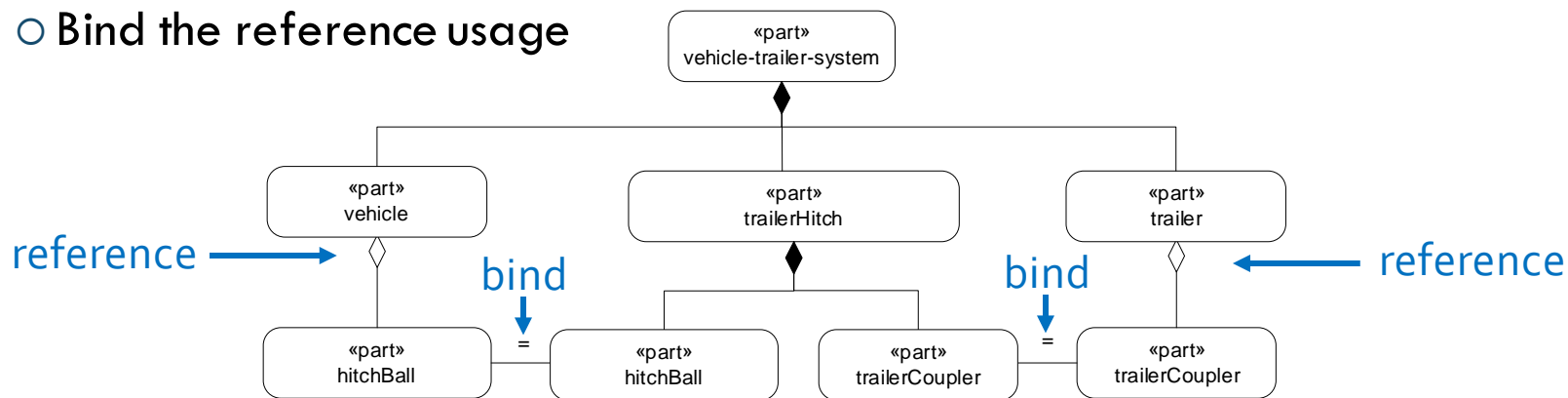
## Subsetting a Part

- A part can be a subset of another part
  - Equivalent to specializing a part **v2**
  - Inherits the part decomposition and other features
  - Can modify inherited parts and features through redefinition and subsetting
  - Can add new parts **v2**

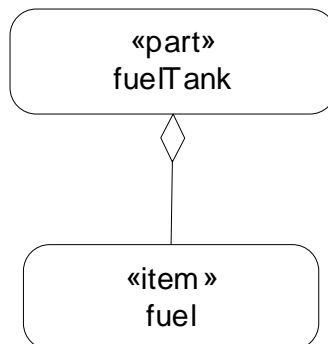




- A part can refer to other parts that are part of another decomposition **v2**
  - Not composite (i.e., lifetime semantics do not apply)
  - Bind the reference usage

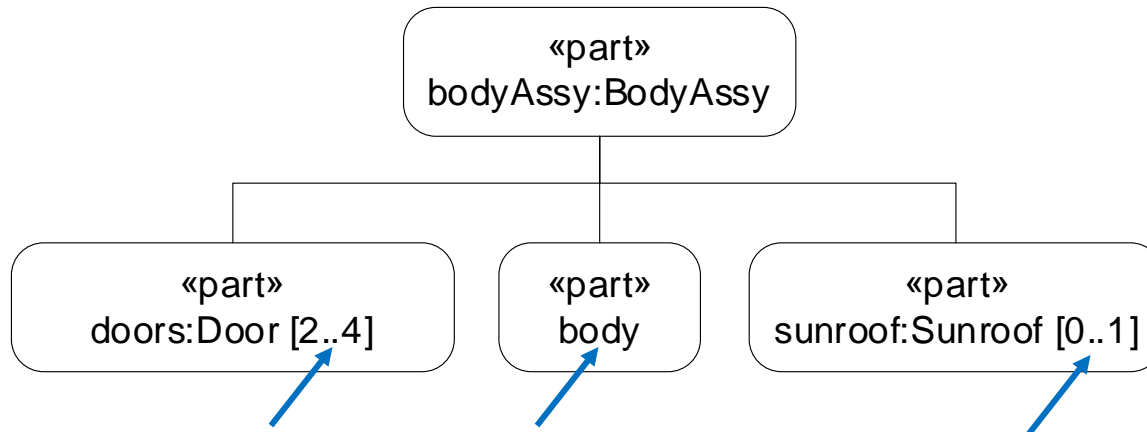


- A part can refer to an item that it stores **v2**



# Multiplicity

- Defines the number of features (e.g., parts) in a particular context
  - Defines a range with a lower bound and upper bound
  - Lower and upper bounds are integers
  - Default multiplicity is [1..1]
  - Optional multiplicity [0..1]

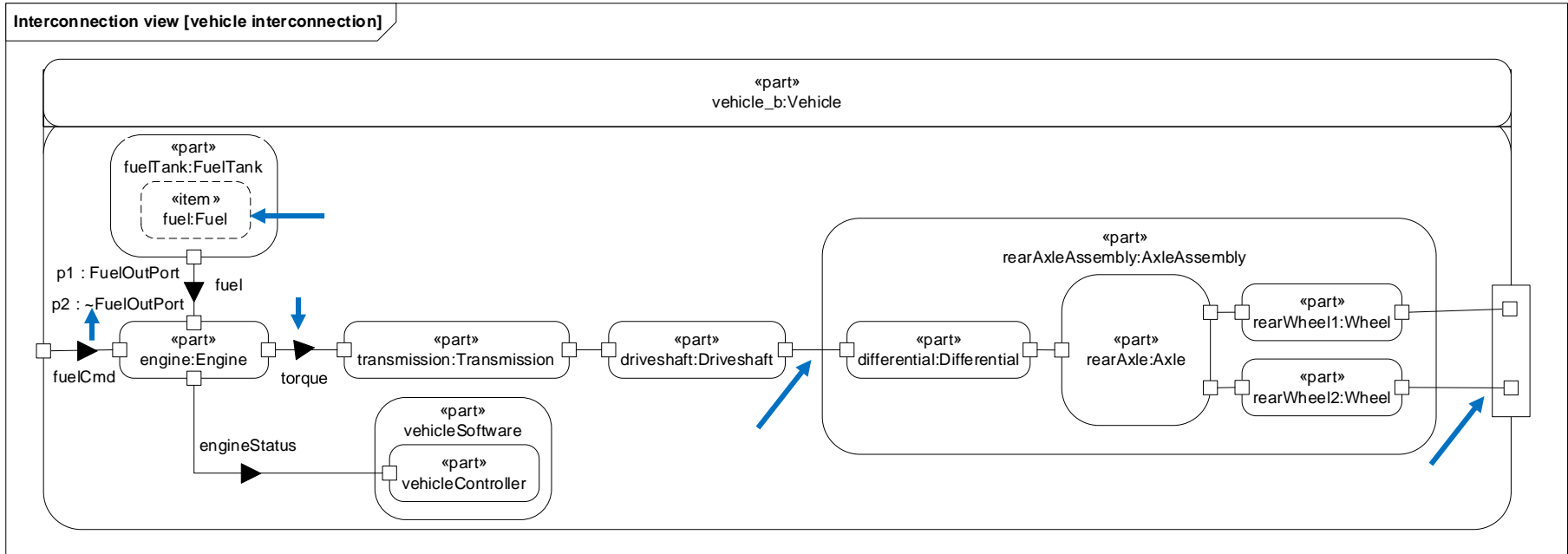


# Module 5

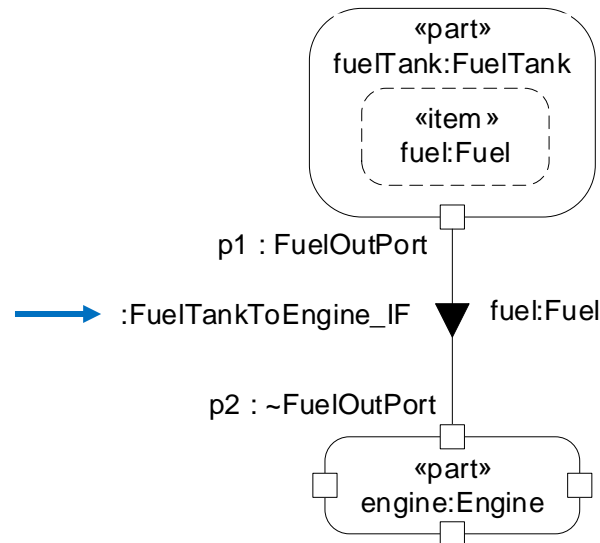
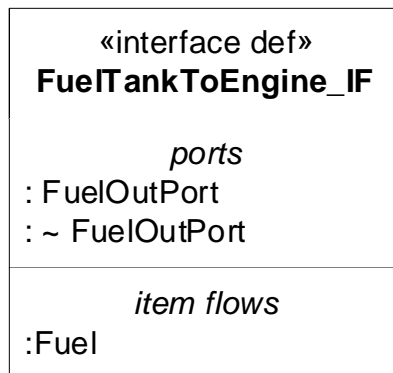
## Part Interconnection

# Connecting Parts

- Parts can be connected via their ports or without ports
  - Parts can be connected directly without connecting to the composite part
  - Ports can contain nested ports
  - Flows can be shown on connections
  - References are dashed
  - Conjugate port



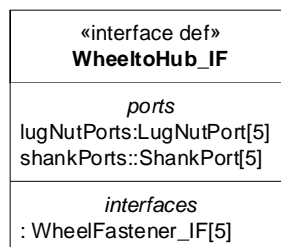
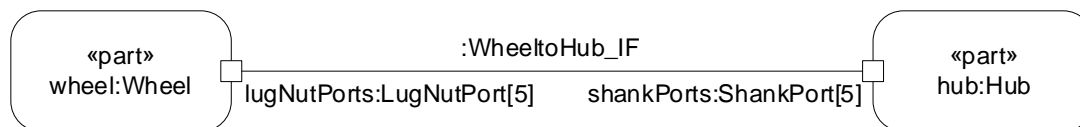
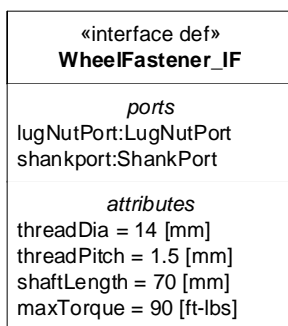
- A connection definition connects two usage elements (e.g., two parts)
- An interface definition is a connection definition whose ends are compatible ports
  - Can contain item flows
  - Can contain other kinds of features
- An interface defined by an interface definition is used to connect parts
  - Ports on parts must be compatible with the interface definition



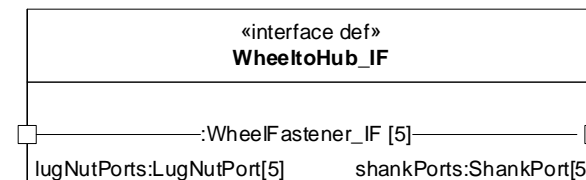
# Interfaces [2]

## Connecting a Wheel to a Hub

- Wheel Fastener interface species a compatible connection between a lug nut and shank
  - Specifies attributes needed for a compatible connection
- Wheel to Hub interface specifies a compatible connection between a wheel and hub (i.e., 5 lug nuts and 5 shanks)
  - Decomposes into 5 Wheel Fastener interfaces
- Usage of Wheel to Hub interface definition connects wheel to hub



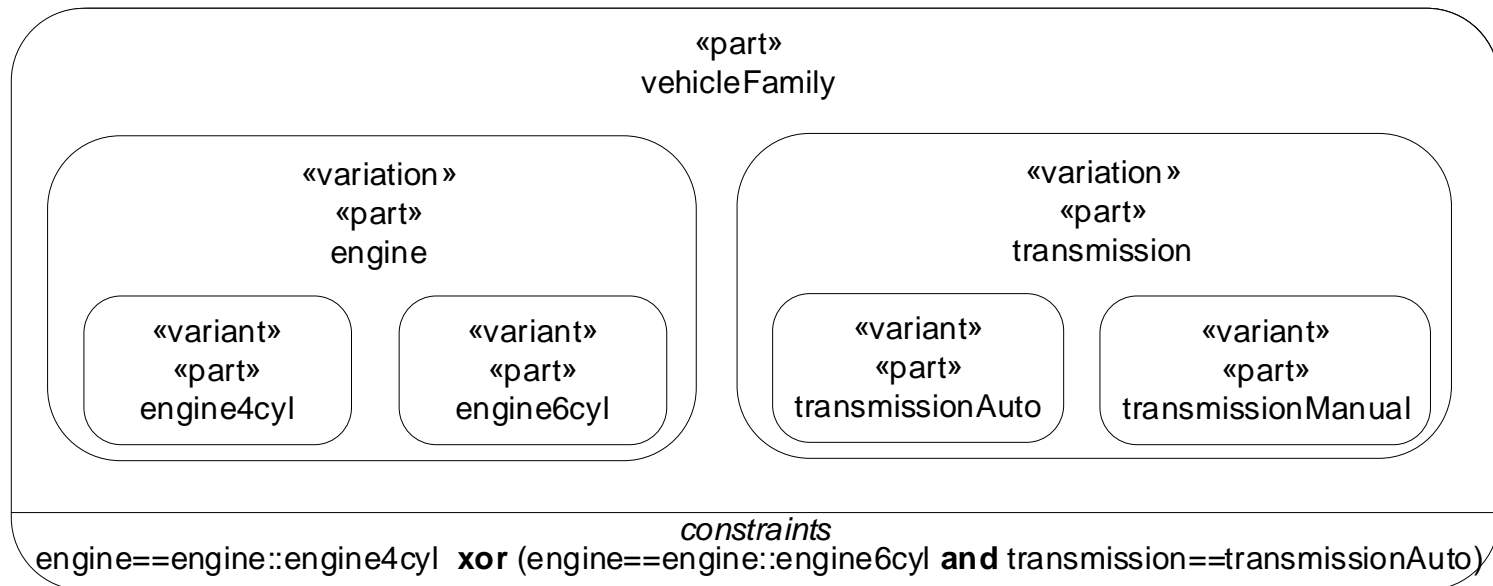
Shows connection  
in compartment



# Module 6

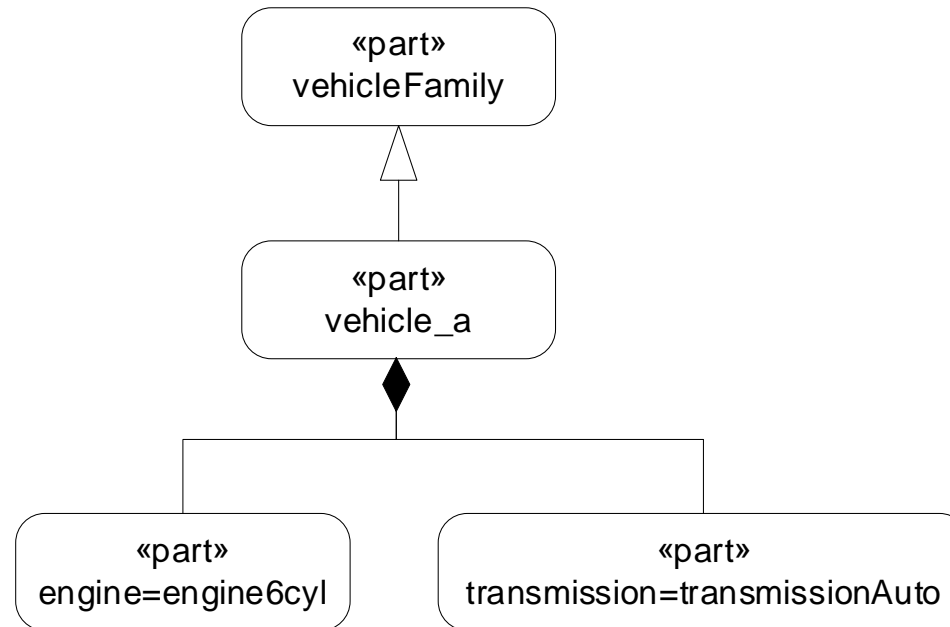
## Variability

- Vehicle family is the superset model (150%)
- *Variation* points represent elements that can vary
  - Can be applied to all definition and usage elements
- A variant represents a particular choice at a variation point
- A choice at one variation point can constrain choices at other variation points
- A system can be *configured* by making choices at each variation point





- vehicle\_a subsets vehicleFamily to represent a particular design configuration
- parts are selected that conform to variability constraints
  - model is invalid if constraints are not satisfied
  - Variability modeling applications can automate the selection of valid configurations

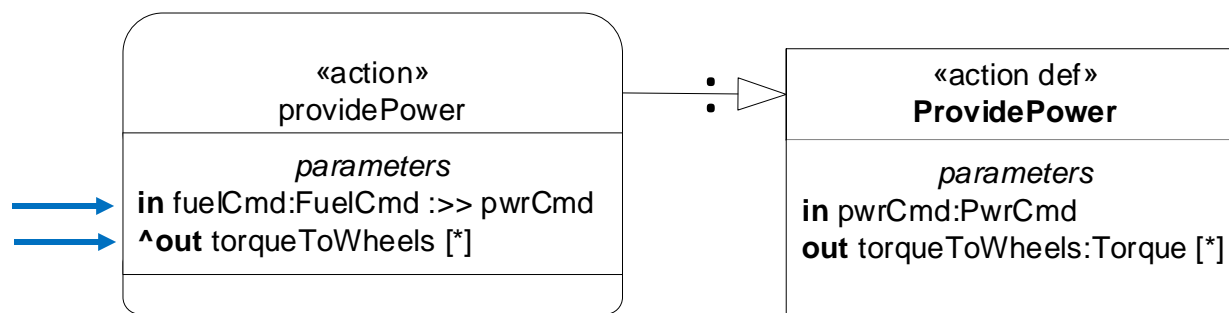


# Behavior

# Module 7

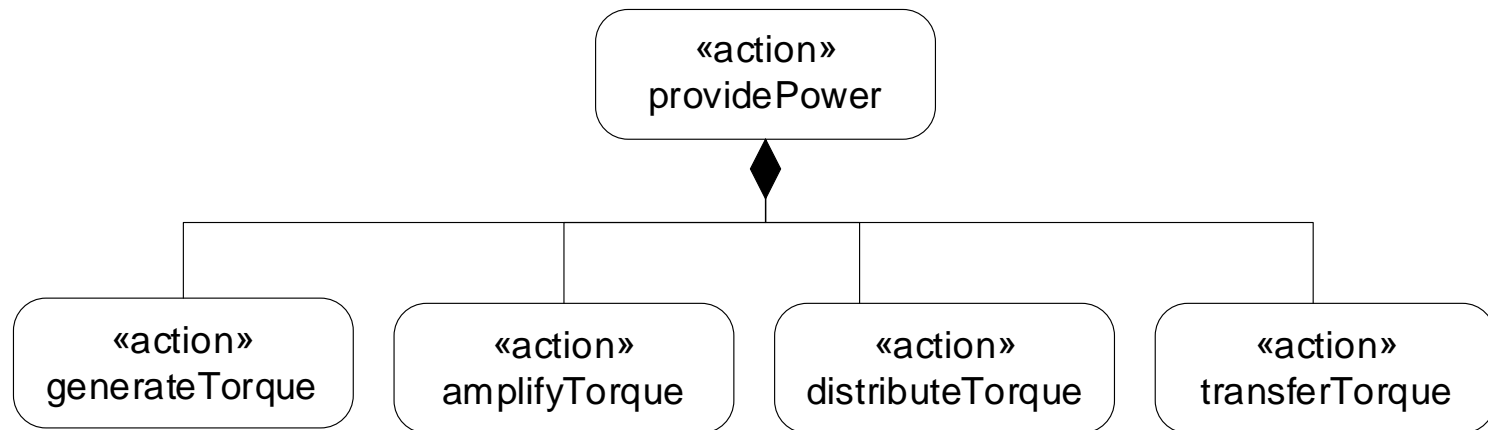
## Actions

- Actions are defined by action definitions **v2**
  - Inherit features (e.g., input and output parameters)
  - Features can be redefined



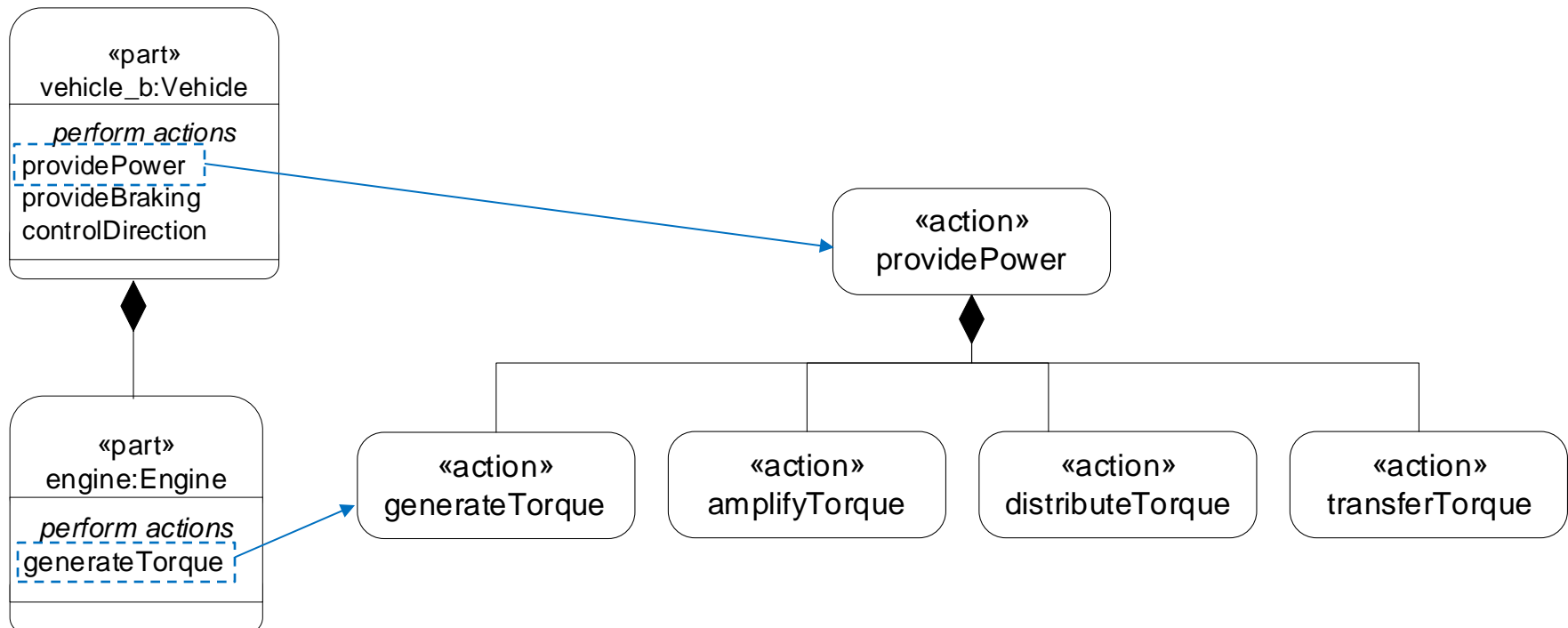
# Action Decomposition

- Actions can be decomposed in a similar way as parts v2



# Part Performs Actions

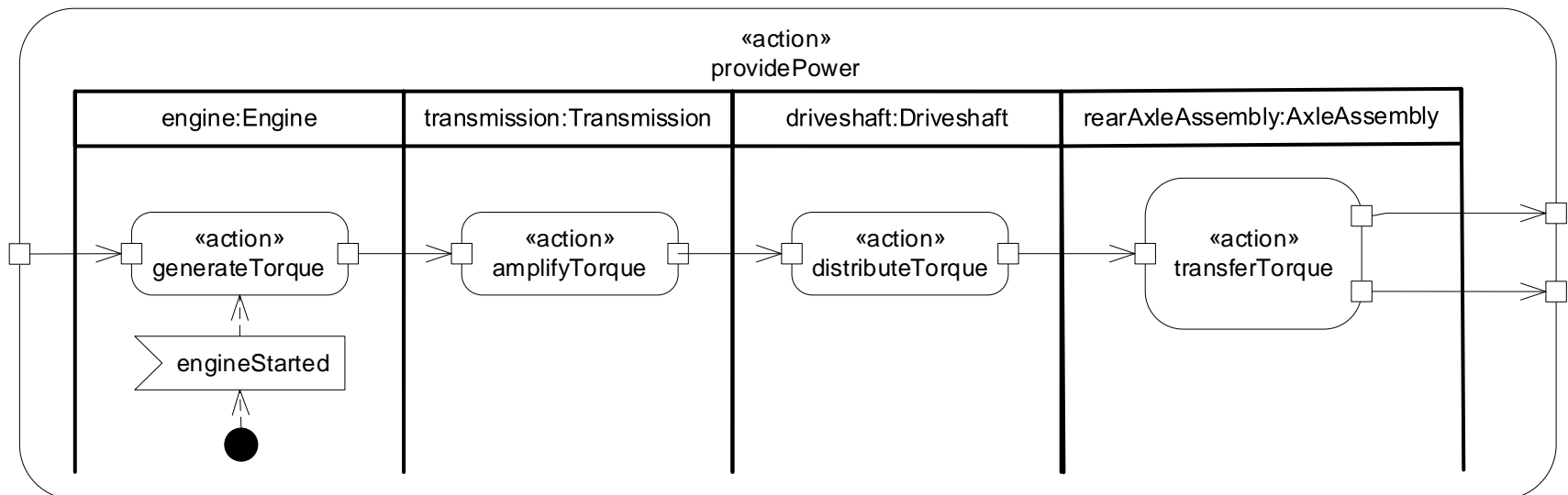
- A part that performs an action can refer to an action in an action tree



# Module 8

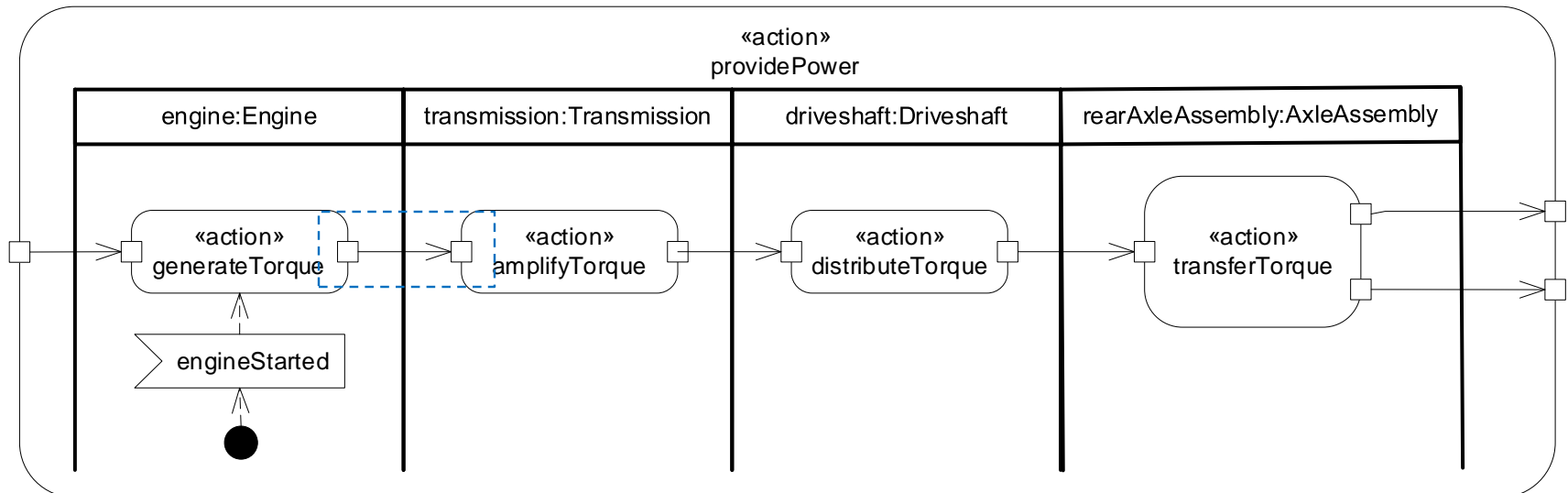
## Action Flow

- Action flows can include control flows and input/output flows
- Control nodes include decision, merge, fork, and join nodes
- Actions can include send and accept actions
- Swimlanes represent performers of actions

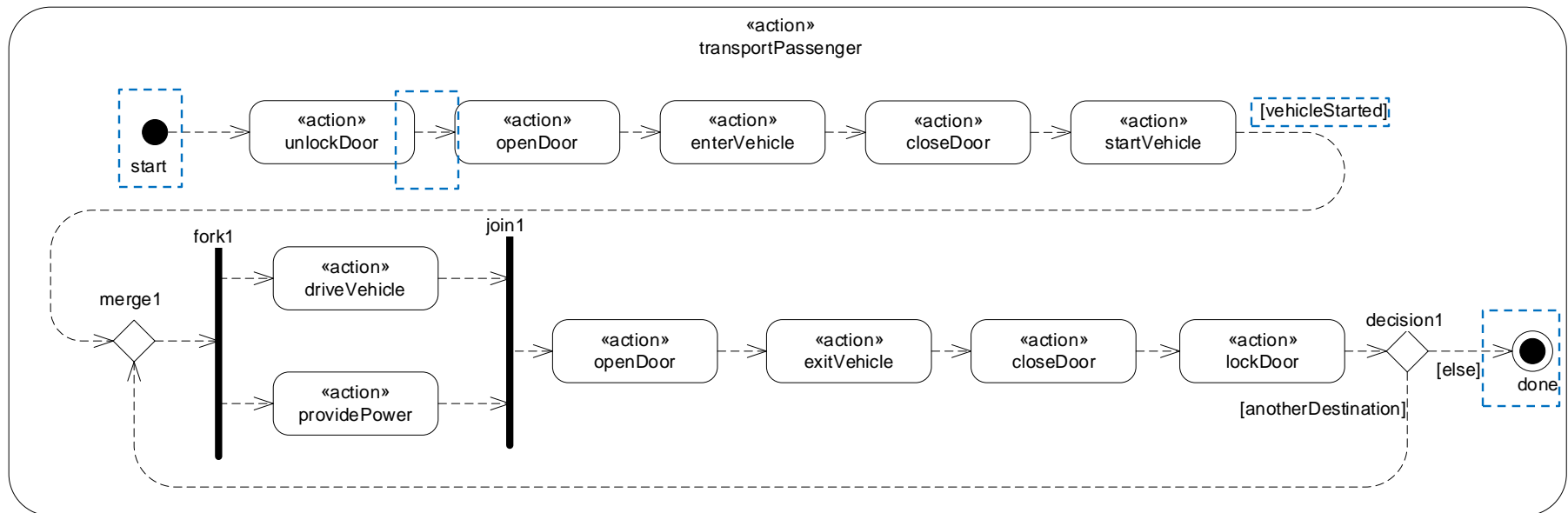




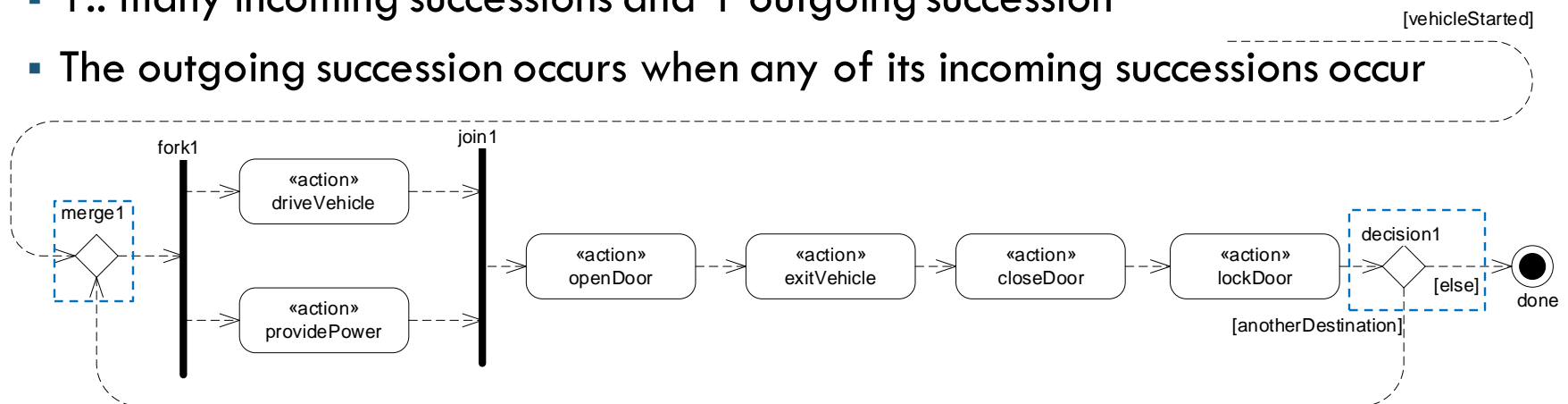
- An item flow is a kind of connection that transfers items from a source output to a target input
  - An item flow is streaming if the transfer continues while the source and target actions execute (this is the default) **v2**
  - An item flow is non-streaming if the transfer only occurs after the source action ends and before the target action starts



- A start action and done action represent the start and end of an action sequence
- A succession asserts that the target action can start execution only after the source action ends execution **v2**
- A conditional succession asserts that the target action can start only if the guard condition is true (if...then). If the guard is false, the target action cannot start



- Control nodes are kinds of actions (*with names*) that control outgoing successions in response to incoming successions **v2**
  - Decision node
    - 1 incoming succession and 1.. many outgoing successions
    - One outgoing succession is selected based on the guard condition that is satisfied (“else” condition is the complement of all other guard conditions)
  - Merge node
    - 1.. many incoming successions and 1 outgoing succession
    - The outgoing succession occurs when any of its incoming successions occur



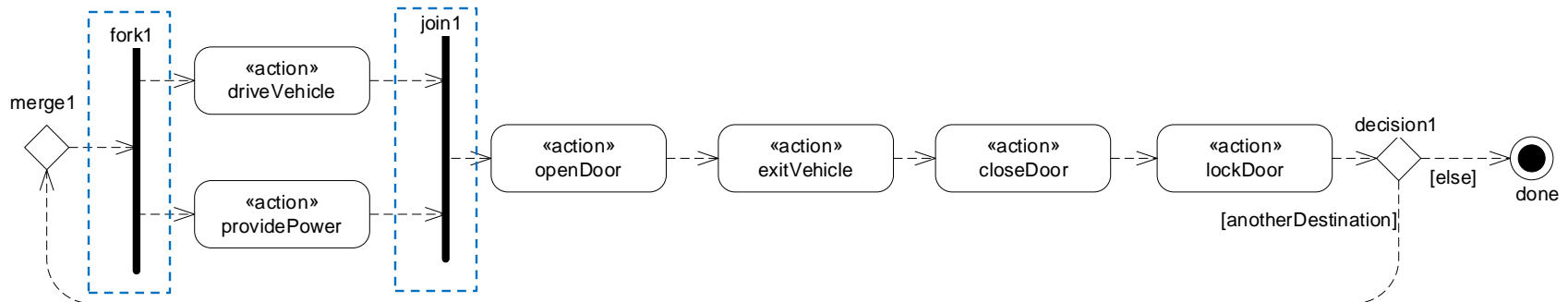
## ○ Fork node

- 1 incoming succession and 1.. many outgoing successions
- All outgoing successions occur when its incoming succession occurs

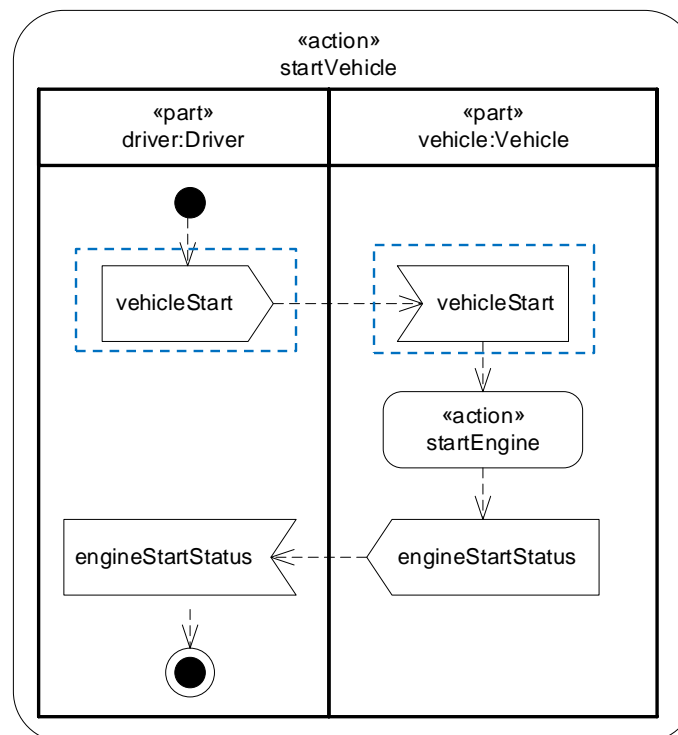
## ○ Join node

- 1.. many incoming successions and 1 outgoing succession
- Outgoing succession occurs when all its incoming successions occur

**Note: Control nodes cannot currently be applied to input/output flows**



- Accept actions and send actions
  - When a send action executes, it transfers an item to a target part
  - When a triggering input item is received, an accept action executes and can transfer the input item to another action

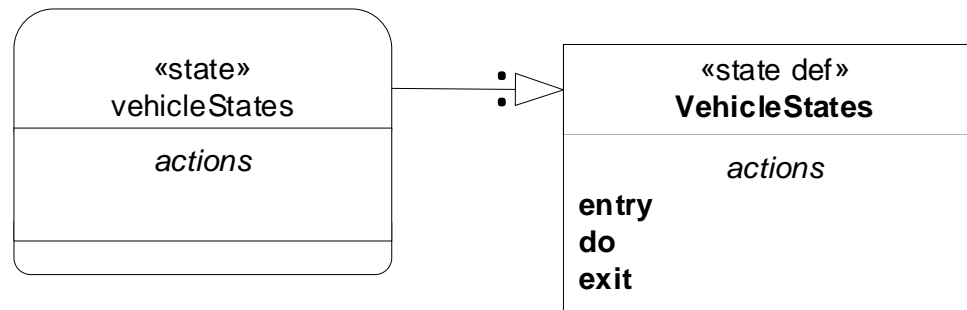


*Driver and vehicle are in a shared context*

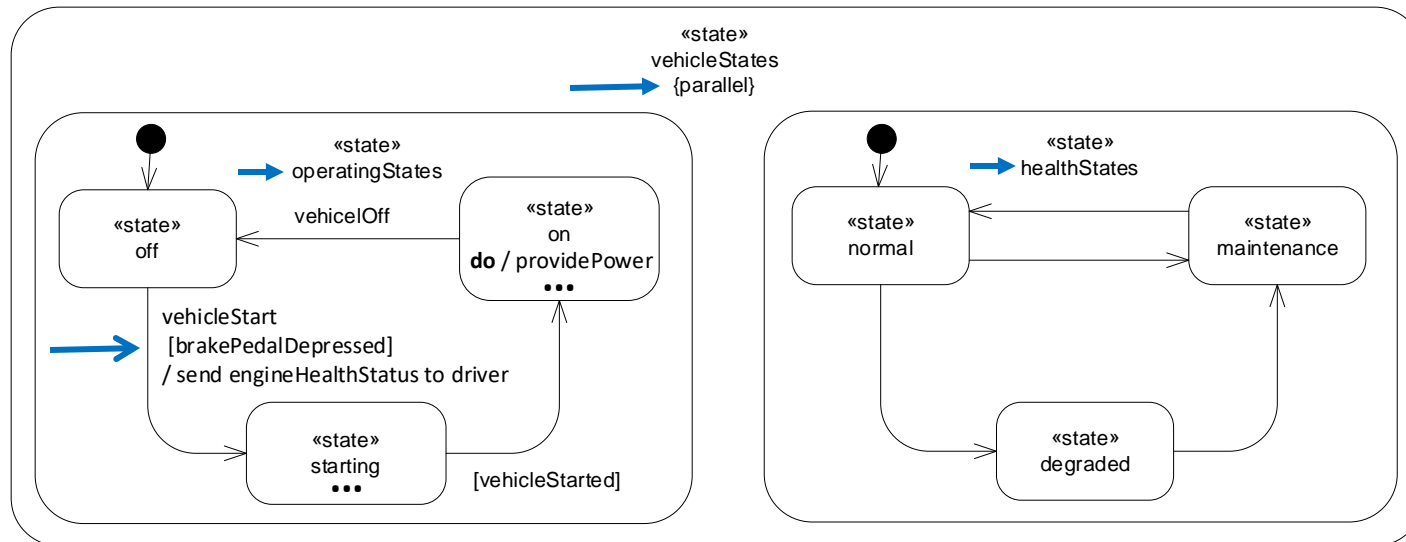
# Module 9

## States

- States are defined by state definitions v2
  - Inherit features (e.g., entry, exit, and do actions) from its state definition
  - Features can be redefined

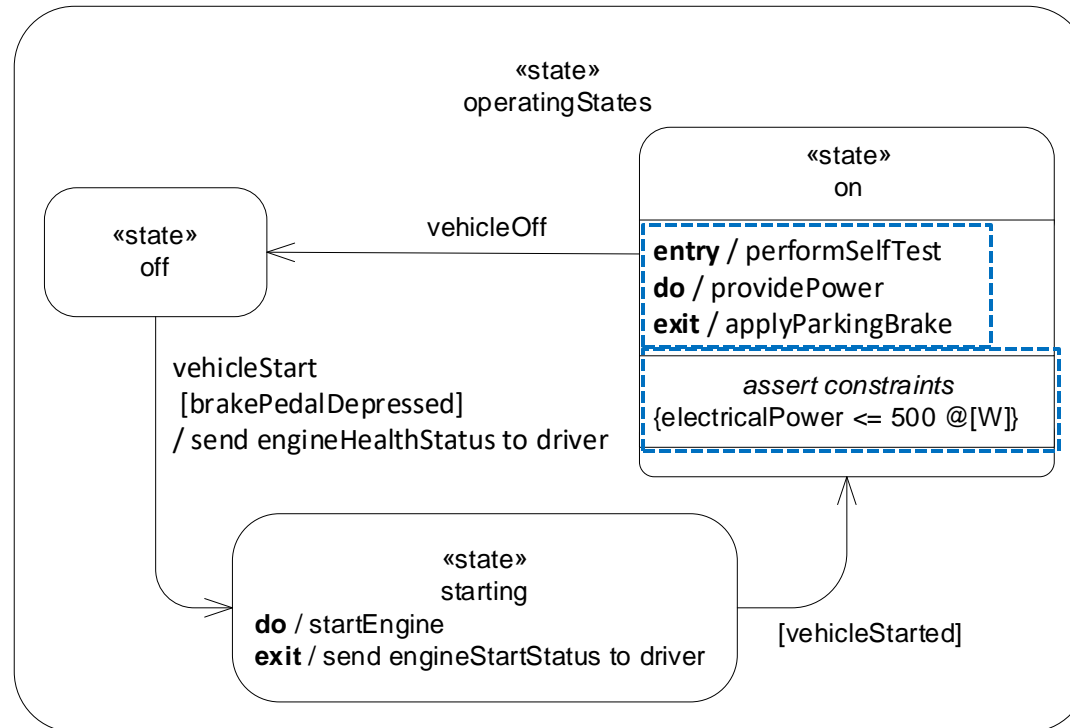


- States decompose into nested states (*without regions, which were in SysML v1*) v2
  - A parallel state decomposes into concurrent states
  - A non-parallel state (default) decomposes into sequential/exclusive states
- Transitions between sequential states
  - Triggered by events
  - Can include guard and action



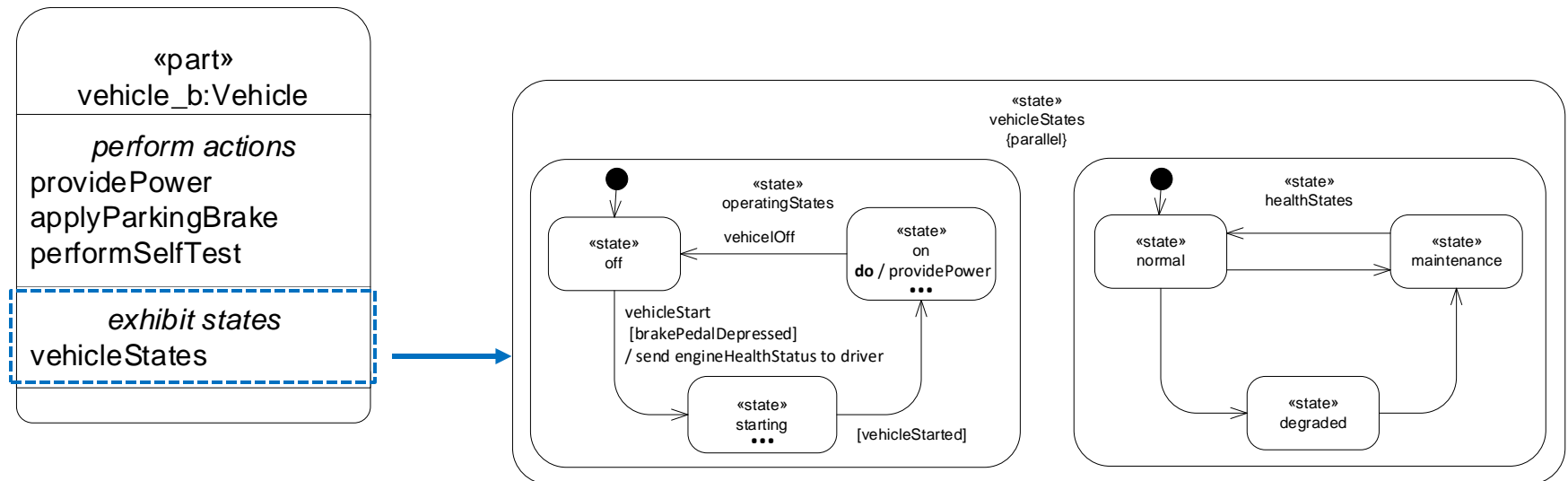


- States can have entry, exit, and do actions that can refer to actions in action trees or be defined locally
- States can include constraints (i.e. SysML v2 state invariants)



# Part Exhibits States

- A part that exhibits states can refer to state-based behavior v2



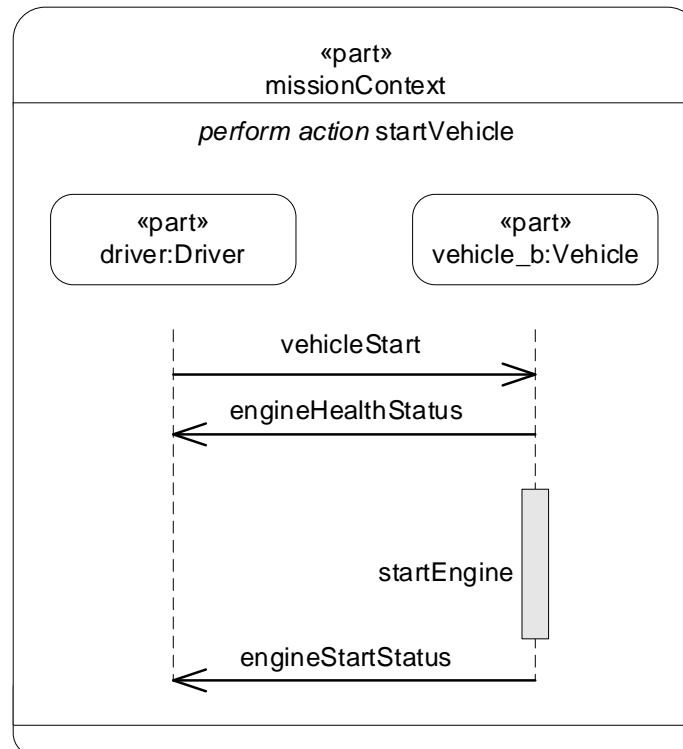
# Module 10

## Interactions

# Sequences

## In Process

- Represents a sequence of messages between send and accept actions
  - Asynchronous – sender does not wait for reply
  - Synchronous – sender waits for reply (**under discussion**)





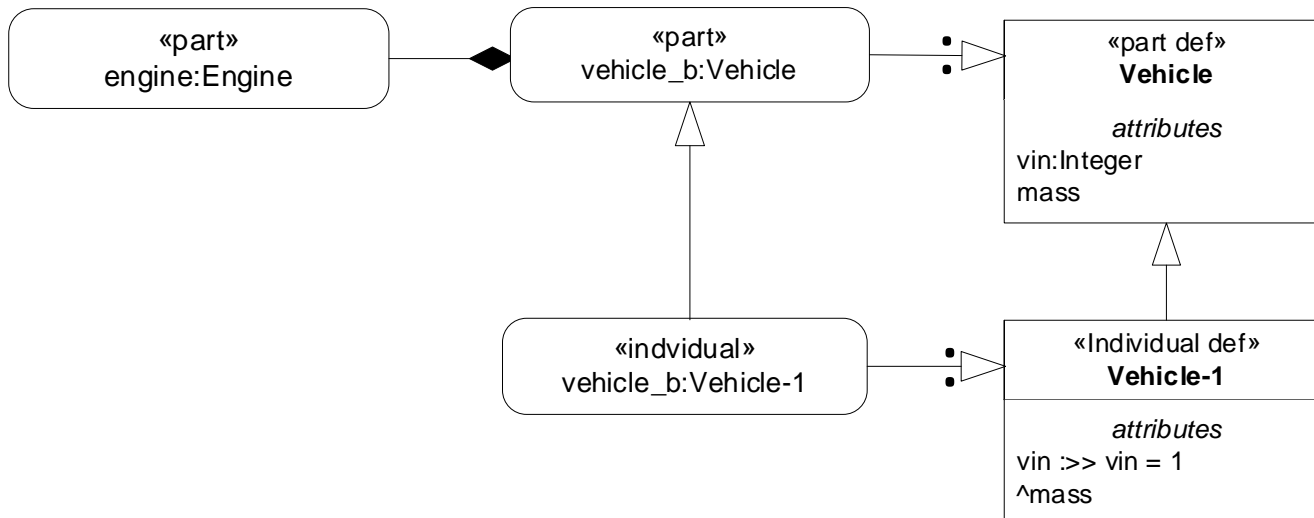
# Cruise Control Interaction

*SST*

# Module 11

## Individuals

- An individual definition is a unique member of a class with identity
  - A specialization of a class, having only one member
  - Has a unique lifetime
- An individual is a usage that is defined by an individual definition
  - Can subset a part to represent a particular configuration
  - Can have different configurations across Vehicle-1 lifetime

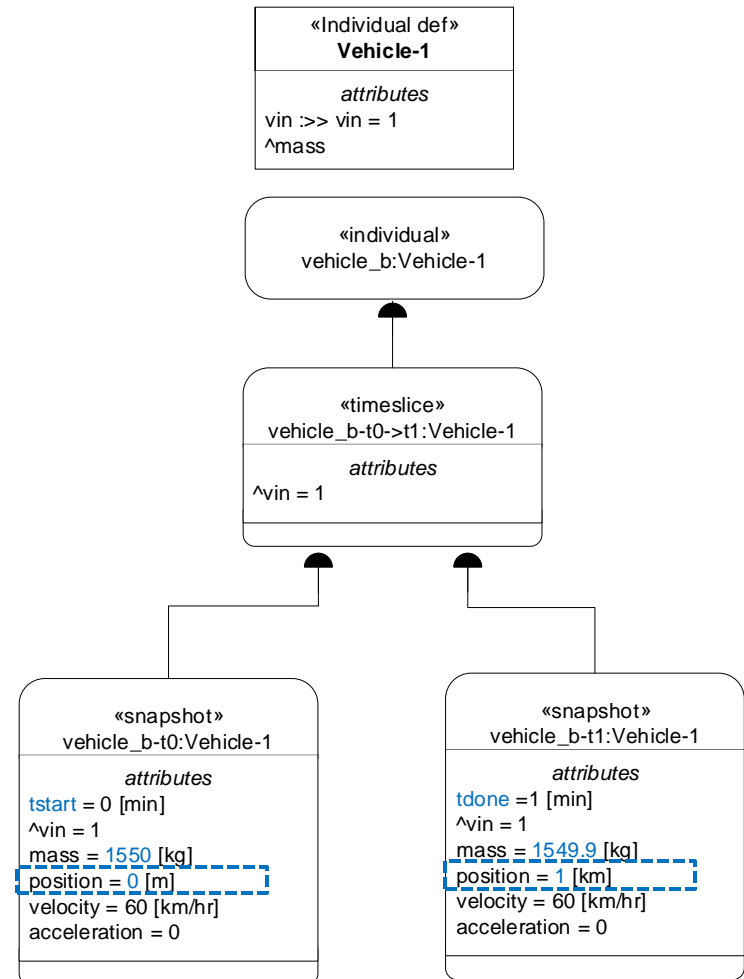


# Module 12

## Timeslices & Snapshots

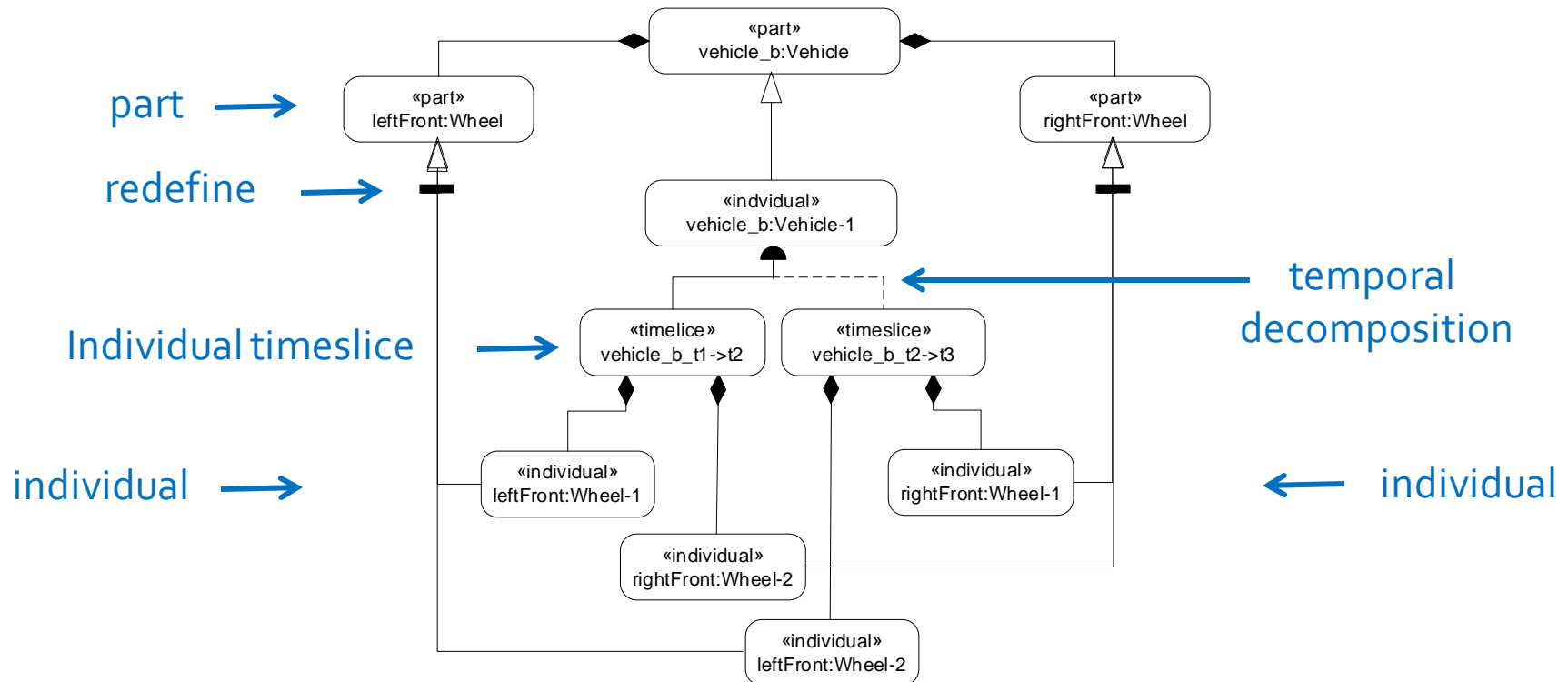


- Each individual definition has a unique lifetime
  - Begins when it is created
  - Ends when it is destroyed
- Individual usage exists over some portion of Vehicle-1 lifetime
- Individual lifetime can be segmented into timeslices (i.e., durations of time)
- A timeslice whose duration is zero is a snapshot
  - Beginning and ending snapshot of a time slice is designated as start and done snapshot
- The condition of an individual for a timeslice or snapshot can be specified in terms of the values of its features



# Individuals Playing Different Roles *v2* SST

- An individual can play different roles and have different configurations in different timeslices
  - Individual wheel (wheel-1) is left front wheel during vehicle\_b\_t1-t2 timeslice
  - Individual wheel (wheel-1) is right front wheel during vehicle\_b\_t2-t3 timeslice



# Module 13

## Expressions and Calculations



# Expressions v2

SST

- Used to compute a result
- Library of mathematical functions (e.g., sum, max, ...)
  - Includes arithmetic operators that also apply to vectors and tensors

«part»  
vehicle\_b:Vehicle

*attributes*

```
mass ::>> mass=dryMass+cargoMass+fuelTank::fuel::fuelMass
```

```
dryMass :>> dryMass=sum(partMasses)
```

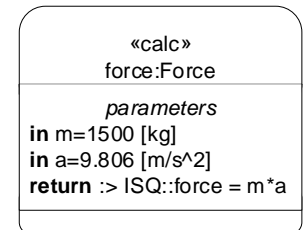
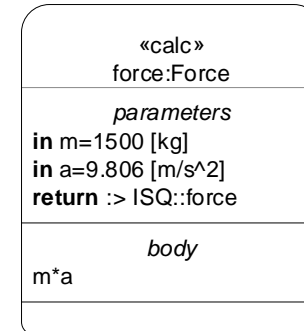
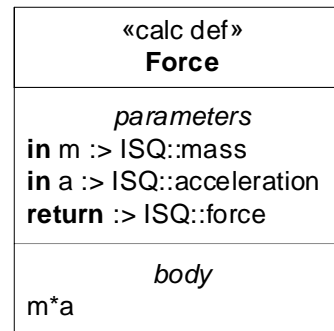
```
cargoMass :>> cargoMass default 0 [kg]
```

```
partMasses=(fuelTank.mass,frontAxleAssembly.mass,rearAxleAssembly.mass,engine.mass,transmission.mass,driveshaft.mass)
```

Update name force on 7/3/2021

- A kind of behavior generally used to represent reusable mathematical functions

- Inputs
- Expression
- Returns a single result



*Do not repeat expression  
in usage if in definition*

- Represent a calculation using the textual notation
  - **calc def** Force (m :>ISQ::mass, a :>ISQ::acceleration) :>ISQ::force = (m \* a);
- Evaluate usage
  - Bind values to input parameters. Then evaluate expression and return a result
  - **calc** force : Force (m=1 500 [kg], a=9.806 [m/s^2]) f;
  - f = Force (m=>1 500 [kg] , a => 9.806 [m/s ^2])
- Usage can be owned by a part, action, constraint, package, ....



# Binding Connection

- A binding is a kind of connection where both sides of the connection are asserted to be equal at all times
  - Symbol is «bind» or =
  - If both sides are not equal, the model is invalid (e.g.,  $3=4$ ) is invalid
  - A tool could make both sides equal if the values are not the same
    - In the above example, the values can be set to  $(3=3)$  or  $(4=4)$
    - For  $y = x+3$ , when  $x$  is 1, then  $y$  can be set to 4 so that both sides are equal
  - Different from the Boolean operator '==', which evaluates whether both sides of the '==' have the same value or not, and returns a value of true or false (e.g.,  $\{3==4\}$  returns a value of false)
- Binding is used more broadly to establish equality of any kind of feature
  - A part can be set to be the same value (e.g.,  $\text{engine} = \text{engine4cyl}$ )
  - A port on a composite part can be bound to a port on a nested part to constrain them to have equal values

# Module 14

## Quantities & Units



# Quantities and Units

SST

- Quantity is an attribute (e.g., mass) **v2**
- Quantity is defined by an attribute def of a quantity kind (e.g., MassValue)
- Units conform to the quantity kind
- Units are associated with the value **v2**
- A change in a unit will apply the unit conversion factor (if a tool supports this)
  - **attribute** *m:MassValue = 25 [kg] (if kg is changed to lbs, the value is changed to 55.1 [lbs])*
  - In SysML v1, a change in unit requires a change in the value type
- Complex quantities and units can be derived from primitive quantities and units
- Libraries
  - International System of Quantities (ISQ)
  - International System of Units (SI)
  - US Customary Units (USCustomaryUnits) **v2**



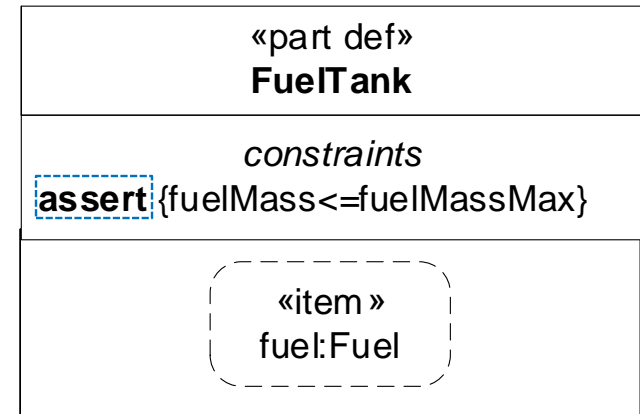
# Module 15

## Constraints

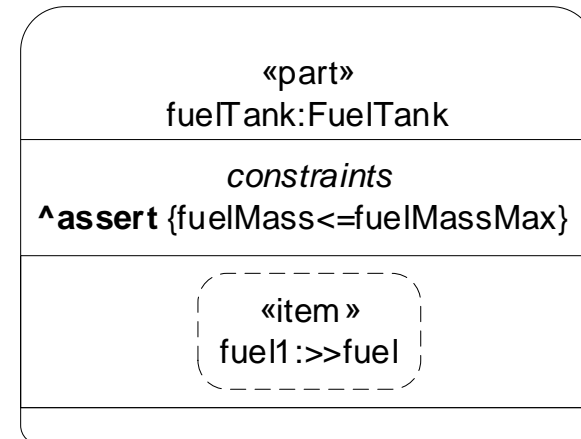
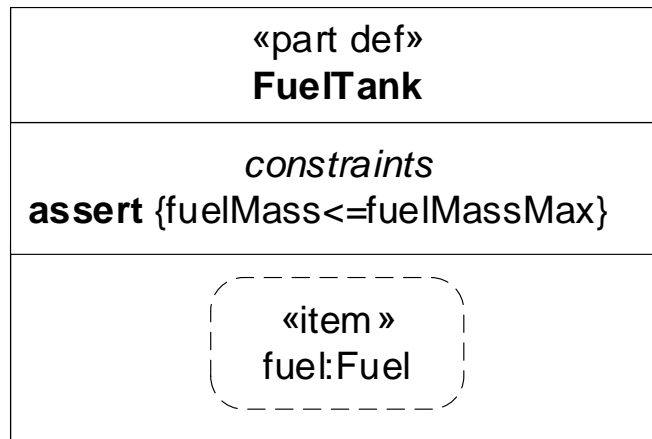
- A Boolean expression that evaluates to true or false, which includes v2
  - Constraint operators ( $=$ ), ( $>$ ), ( $\geq$ ), ( $<$ ), ( $\leq$ ), ( $\neq$ )
    - Used to compare expressions (e.g.,  $\{a < b\}$ )
  - Boolean operators include **and** ( $\&$ ), **or** ( $|$ ), **xor** ( $\wedge$ ), **not** ( $!$ )
    - Used to logically combine expressions (e.g.,  $\{A \text{ and } B \text{ or } C\}$ )
- Asserting a constraint to be true means the evaluation must be true for the model to be valid

```

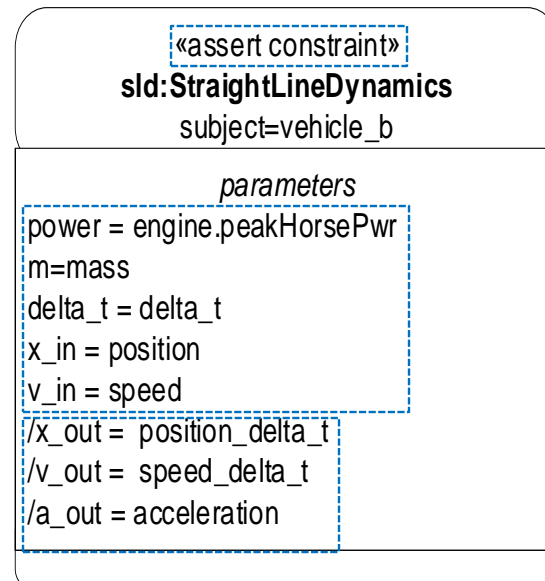
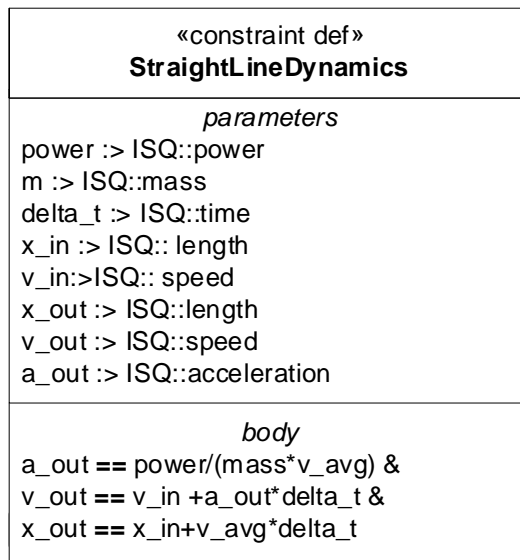
part def FuelTank {
    attribute mass :> ISQ::mass;
    ref item fuel:Fuel{
        attribute redefines fuelMass;
    }
    attribute fuelMassMax:>ISQ::mass;
    → assert constraint {fuel.fuelMass<=fuelMassMax}
}
  
```



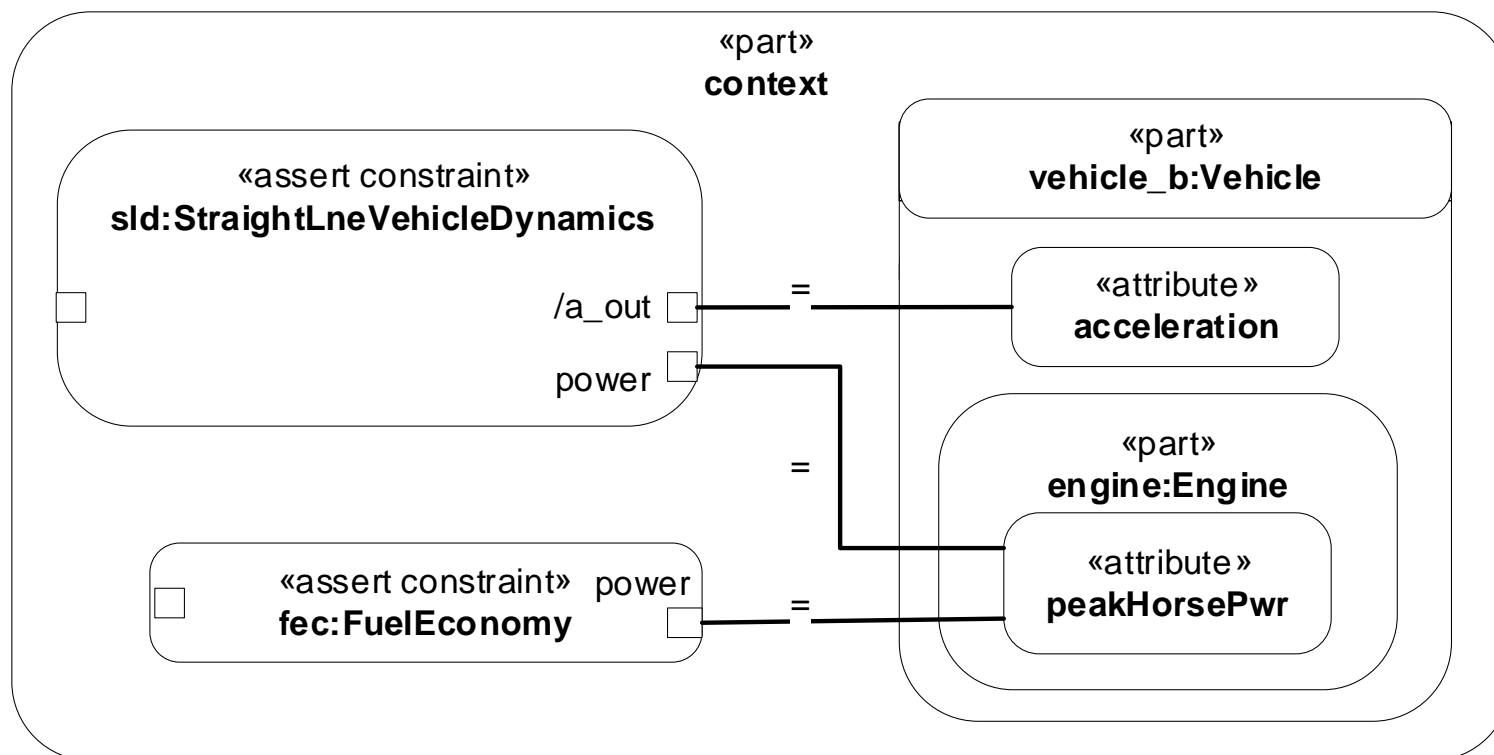
- Usage inherits constraint expression



- A reusable constraint expression
- Default parameter direction is 'in'
- Can assert a constraint on the usage
- Bind parameters on the usage
- Identify dependent parameters as derived



- Equivalent to SysML v1 parametric diagram



# Module 16

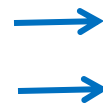
## Requirements

- A constraint definition that a valid design solution must satisfy that can include: v2
  - Identifier
  - Shall statement
  - Constraint expression that can be evaluated to true or false
    - can apply to performance, functional, interface and other kinds of requirements if desired
  - Assumed constraint expression that is asserted to be true for the requirement to be valid
  - Attributes of the constraint expressions

	«requirement def» MassRequirement
→	<b>doc</b> The actual mass shall be less than the required mass.
→	<i>attributes</i> massActual :> ISQ::mass massRequired :> ISQ::mass
→	<i>constraints</i> <b>require</b> {massActual<=massRequired}

- A tree of requirements that contains nested requirement usages v2
  - Requirement group can own or reference other requirements
  - Subject of nested requirements is the kind of entity being specified
  - Requirement features can redefine features of the requirement definition

requirement group  
subject



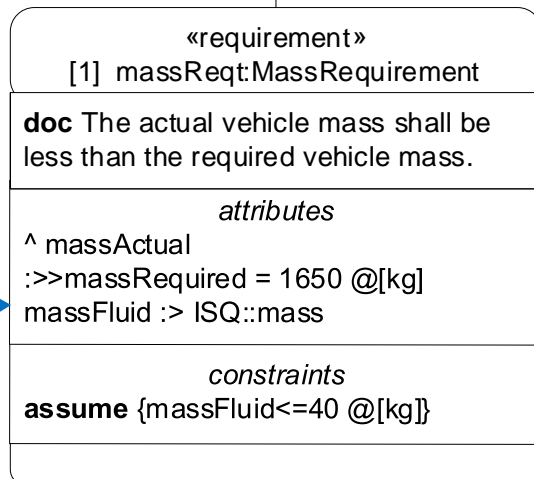
«requirement group»  
vehicleSpecification  
**subject** v:Vehicle

reqt id →

«requirement»  
[2.1] city:FuelEconomyRequirementI

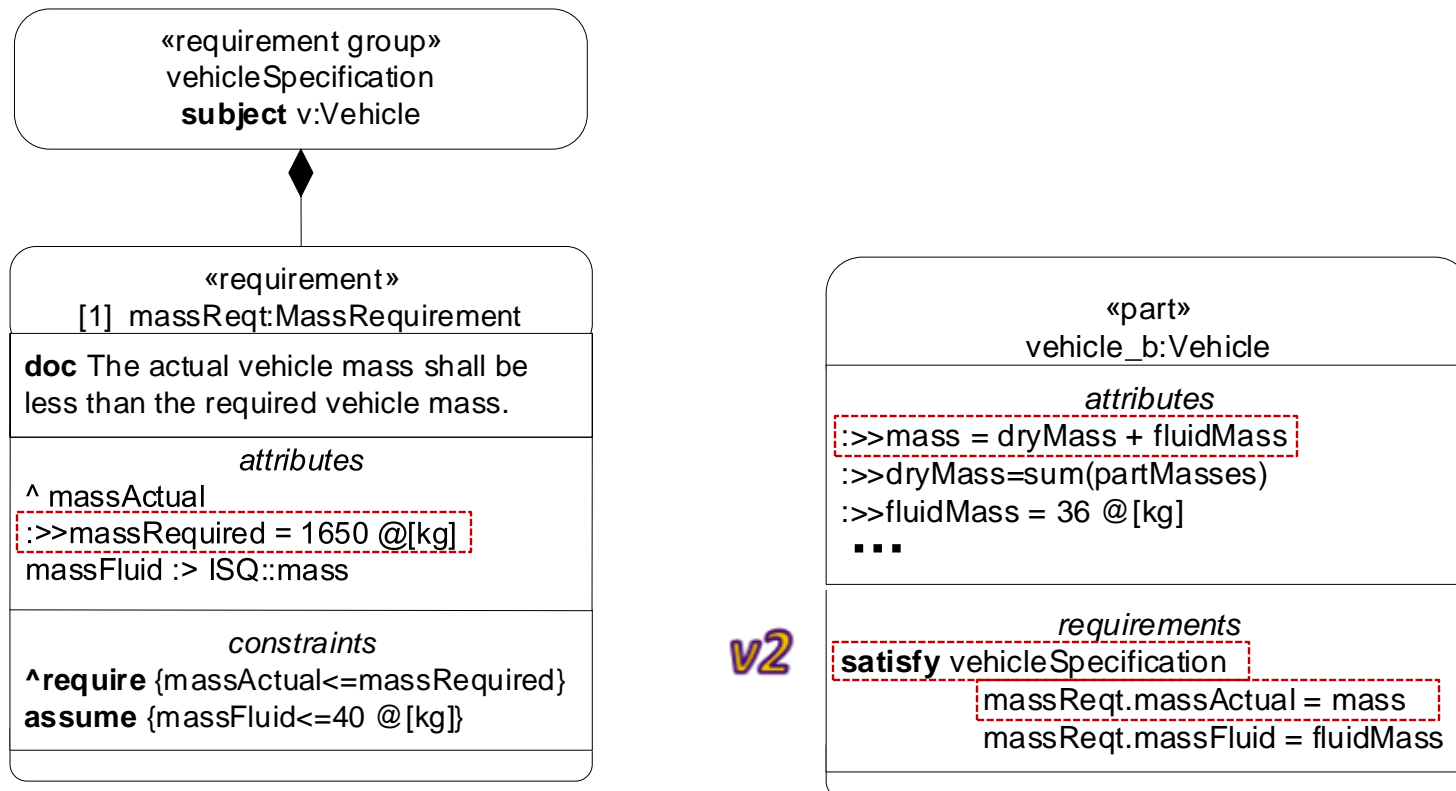
«requirement»  
[2.2] highway:FuelEconomyRequirement

← assumption





- The vehicleSpecification is used to impose constraints on the vehicle
  - The modeler can assert the vehicle mass satisfies the mass requirement
    - Bind the vehicle mass to massActual of the requirement

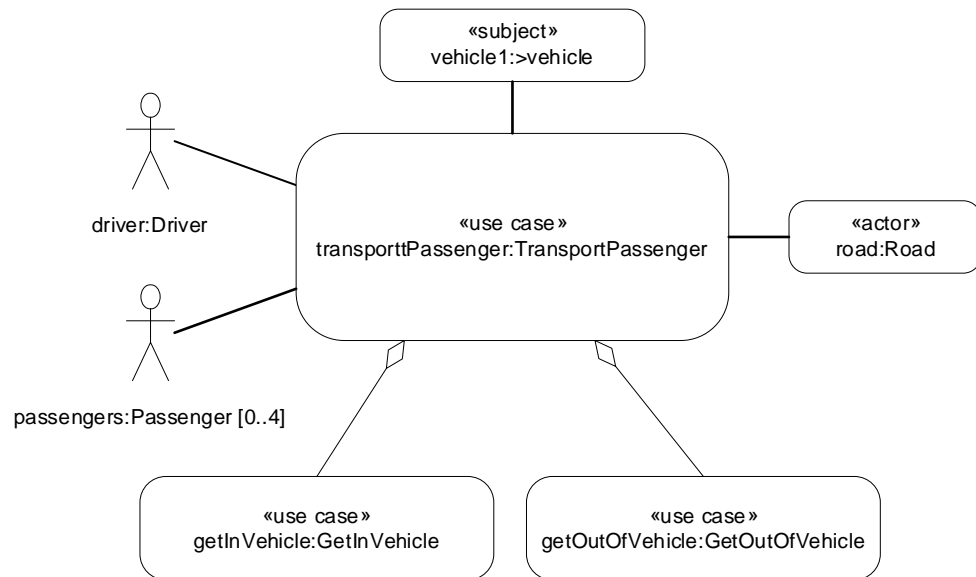


# Module 17

## Use Cases

- A kind of case
  - Subject of the use case (often the system of interest)
  - Objective is to provide value to one or more stakeholders
    - Stakeholders are typically external to the system of interest referred to as actors
  - Actions/steps performed by actors and subject are required to achieve the objective
  - Include use cases are always performed as part of the base use case

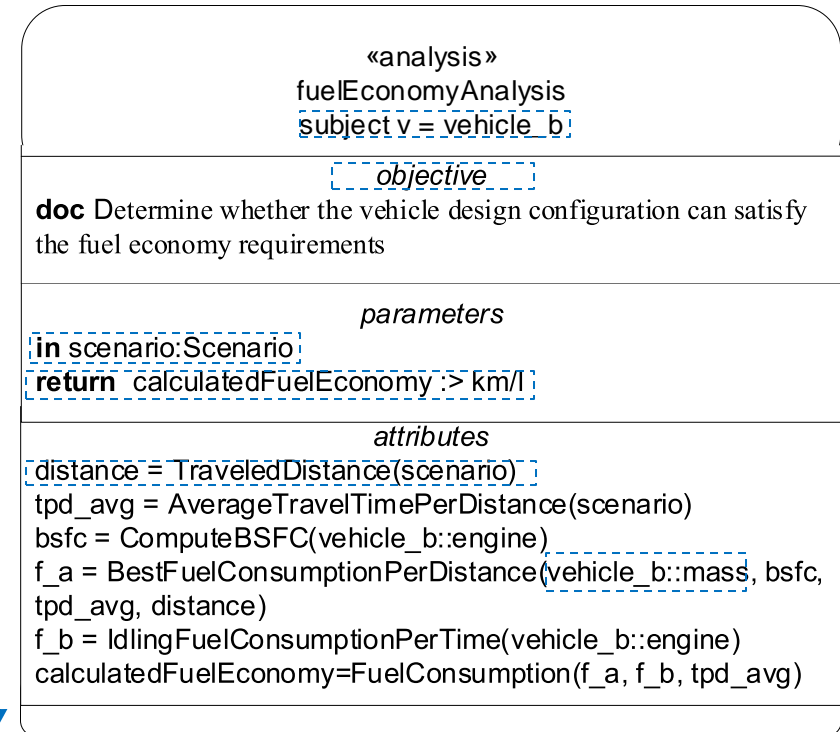
<p>«use case def» TransportPassenger subject vehicle:Vehicle</p>
<p><i>objective</i> <b>doc</b> Deiver passenger to destination safely, comfortably, and within acceptable time</p>
<p><i>actors</i> driver:Driver passengers:Passenger 0..4]</p>
<p><i>include use cases</i> getInVehicle:GetInVehicle getOutOfVehicle:GetOutOfVehicle</p>
<p><i>pre/post conditions</i> vehicle.position=startingLocation vehicle.position=endingLocation</p>



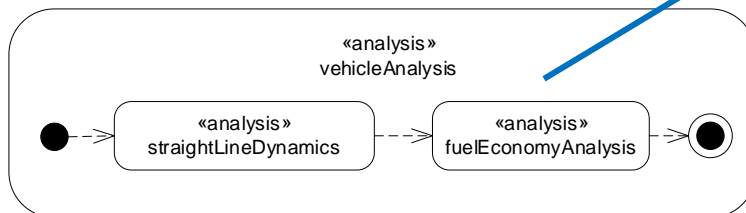
# Module 18

## Analysis Cases

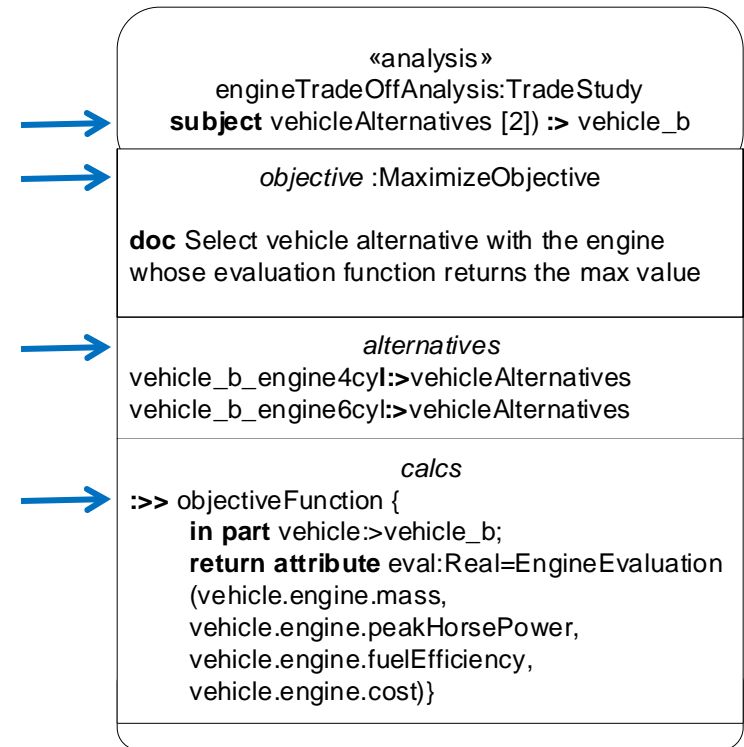
- A set of steps with an objective to evaluate a result about a subject of the analysis
  - A kind of behavior
  - Each step can be an action, calc, or an analysis case that can contain
    - subject
    - objective
    - input and output parameters
    - Single return parameter (i.e., the result)
    - Attributes bound to calculations and/or constraints to be solved by a solver
    - Bind input & output parameters to the subject



convert to liters / 100 km



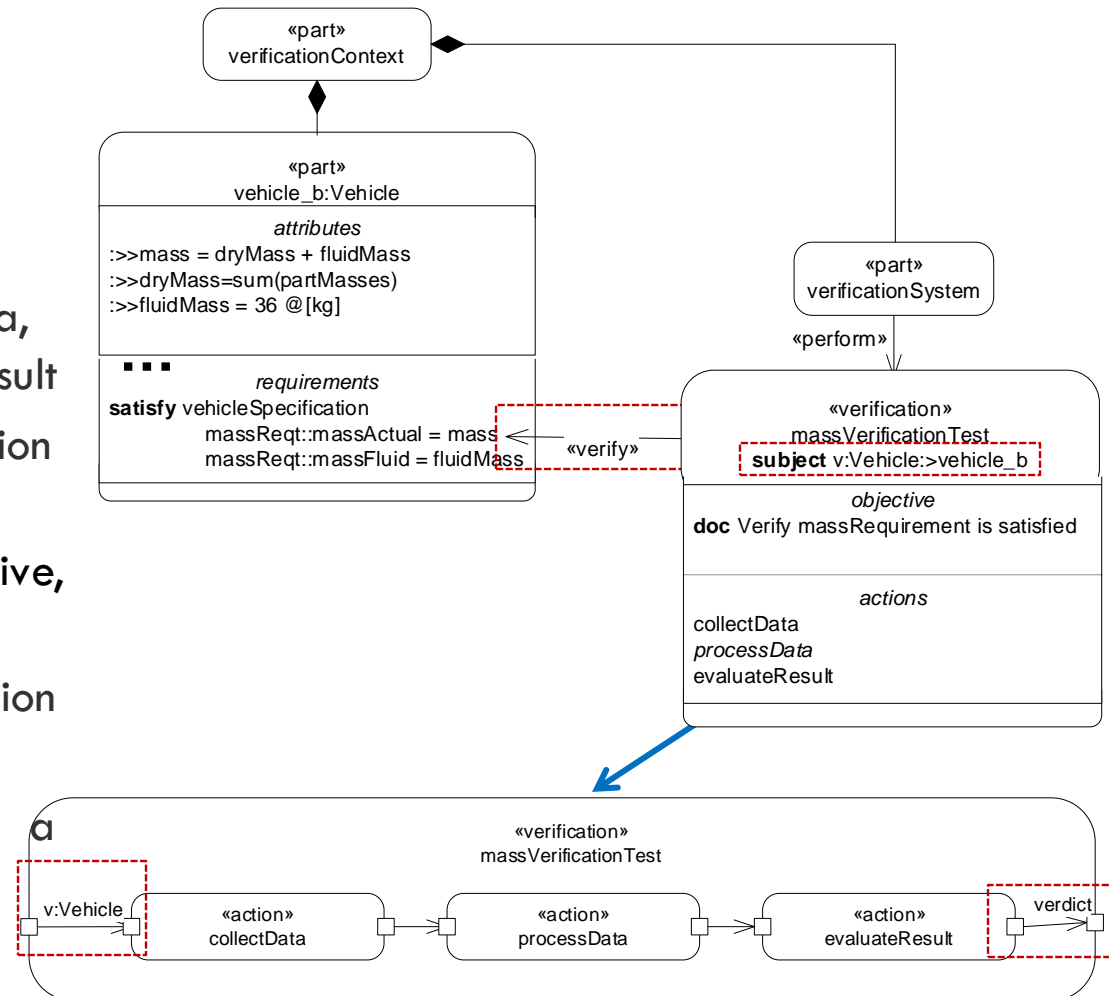
- A kind of analysis case
  - Trade study objective is to select preferred solution that maximizes or minimizes some objective function
  - Subject of the trade study are the vehicle alternatives with different engines
    - Alternatives can be modeled as variants
  - Evaluate objective function for each alternative
    - Criteria are parameters of objective function
    - Each criteria may require separate analysis
  - Rationale is a kind of annotation



# Module 19

## Verification Cases

- A set of steps with an objective to evaluate whether the subject of the verification satisfies one or more requirements
  - A kind of behavior
  - Steps typically include collect data, process/reduce data, evaluate result
  - Input is the subject of the verification
  - Returns a result called a verdict whose value is pass, fail, inconclusive, or error
  - Verify relationship binds verification case to subject
  - Verification case is performed by verification system in a context



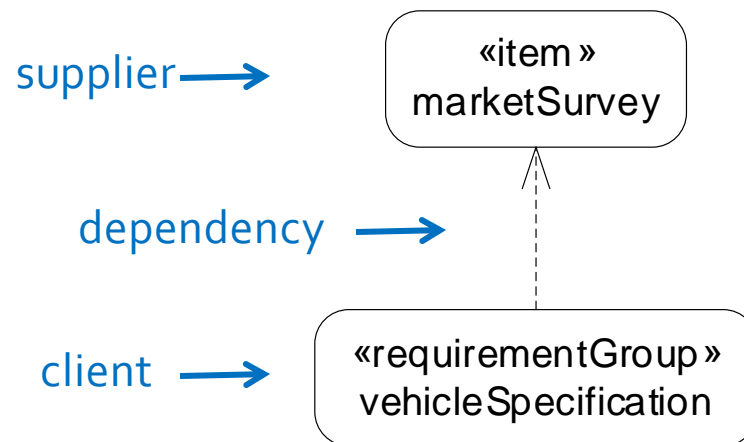


# **Module 20**

## **Dependency and Allocation Relationships**

# Dependency

- A directed relationship where the element on the client end may be impacted when the element on the supplier end changes
  - Example: Use of dependency to represent traceability between specification and source document
  - Can have 1.. many clients and 1.. many suppliers



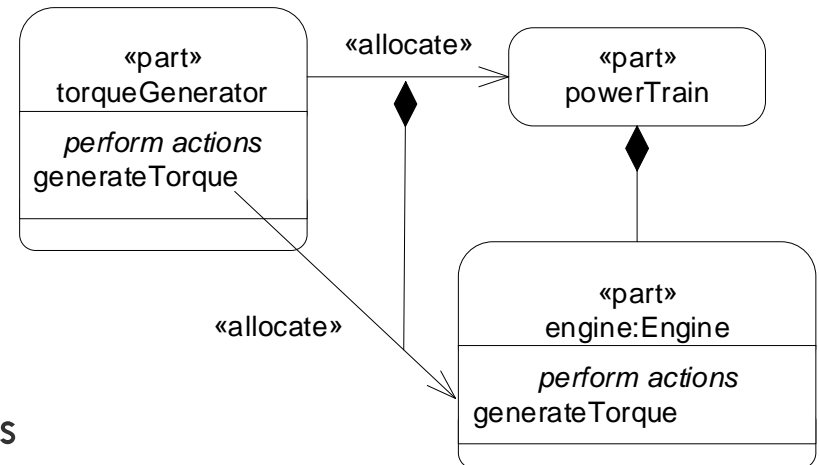
- A statement of intent that assigns responsibility from one set of elements to another set of elements

- A kind of mapping (1 to n)
- Includes definition and usage
- Can include suballocations

v2  
v2

- Examples

- Actions to components
- Logical components to physical components
- Logical connections to physical connections
- Budget allocation (e.g. weight budget)
- Software to hardware
- Requirements to design elements
- ...

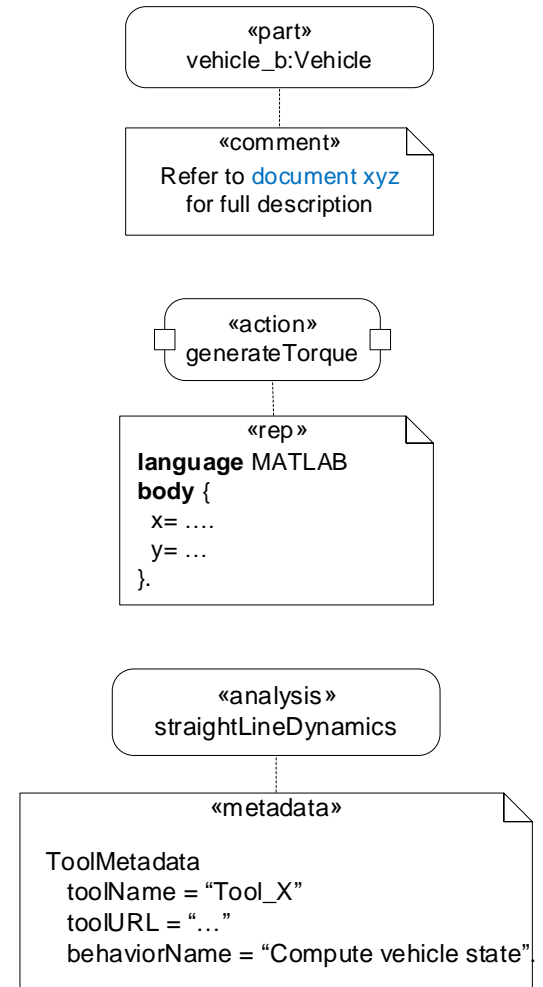


The torque generator  
Is allocated to the powertrain,  
and realized by the engine

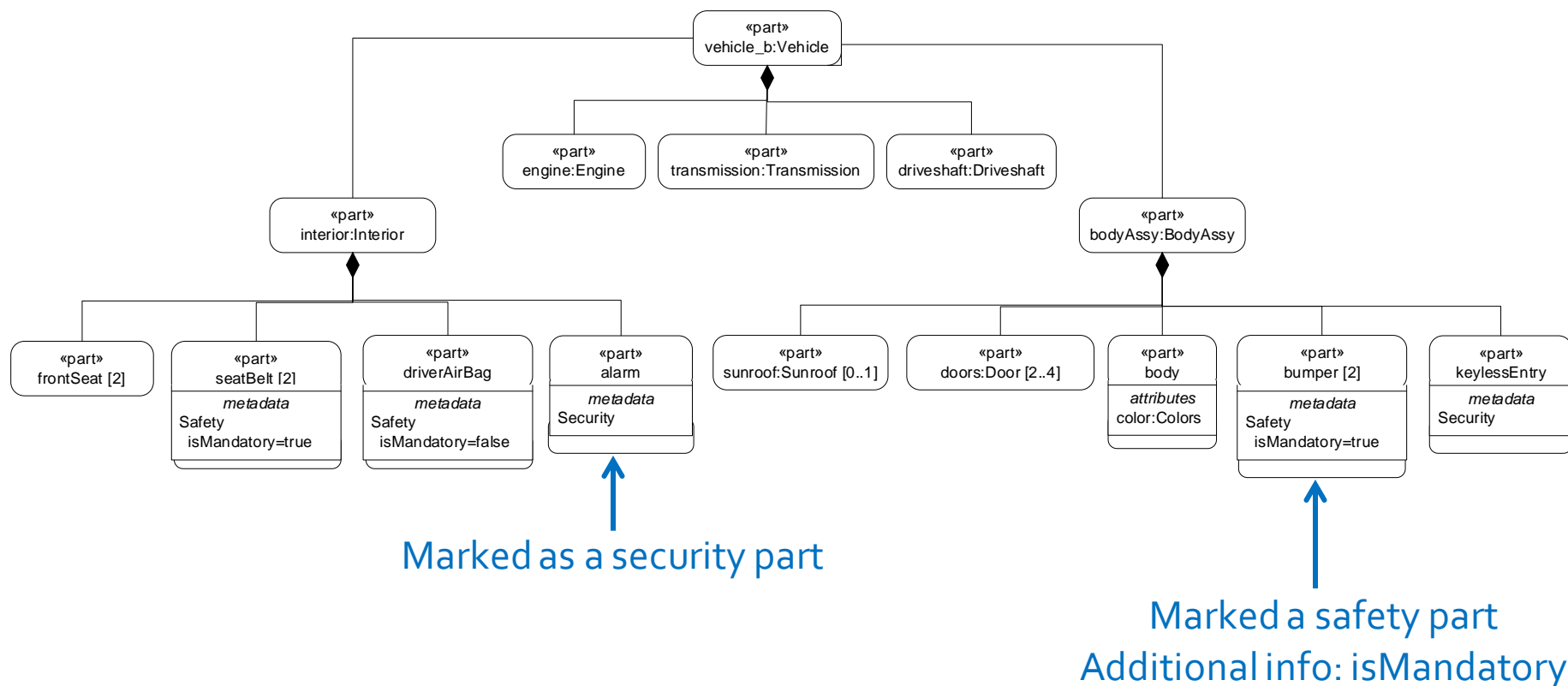
# Module 21

## Annotations

- An Annotating Element is an Element that is used to provide additional information about other elements
  - Can be named v2
  - Can apply to 0.. many elements
- An annotation is a kind of relationship that relates the v2
  - Comment
  - Documentation is an owned comment
  - Textual representation (e.g., opaque expression) v2
    - Includes name of language and body
  - Annotating feature v2
    - Used to add structured metadata



- Use metadata to mark elements **v2**
  - Identify parts that are Security or Safety parts (e.g., members of Security or Safety group)
  - Can add additional security and safety information

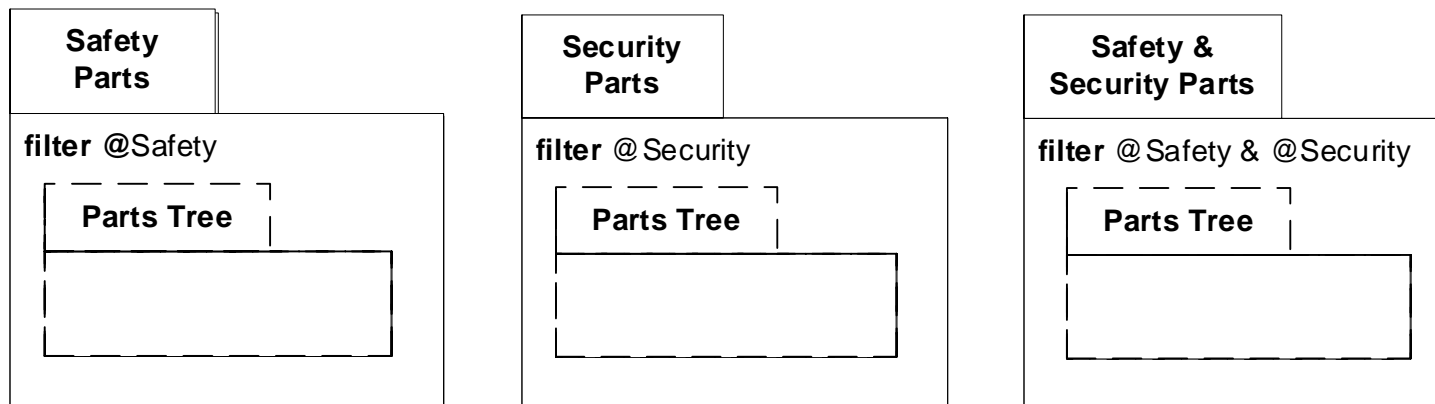


# Module 22

## Element Filters

# Element Filters

- Used to select elements from a group of elements
  - Create a package
  - Import the part of the model that contains the elements of interest
    - Use a recursive import to include all nested elements
  - Define the filter expression to select the imported elements based on the selection criteria



Filter selects parts marked with Safety metadata

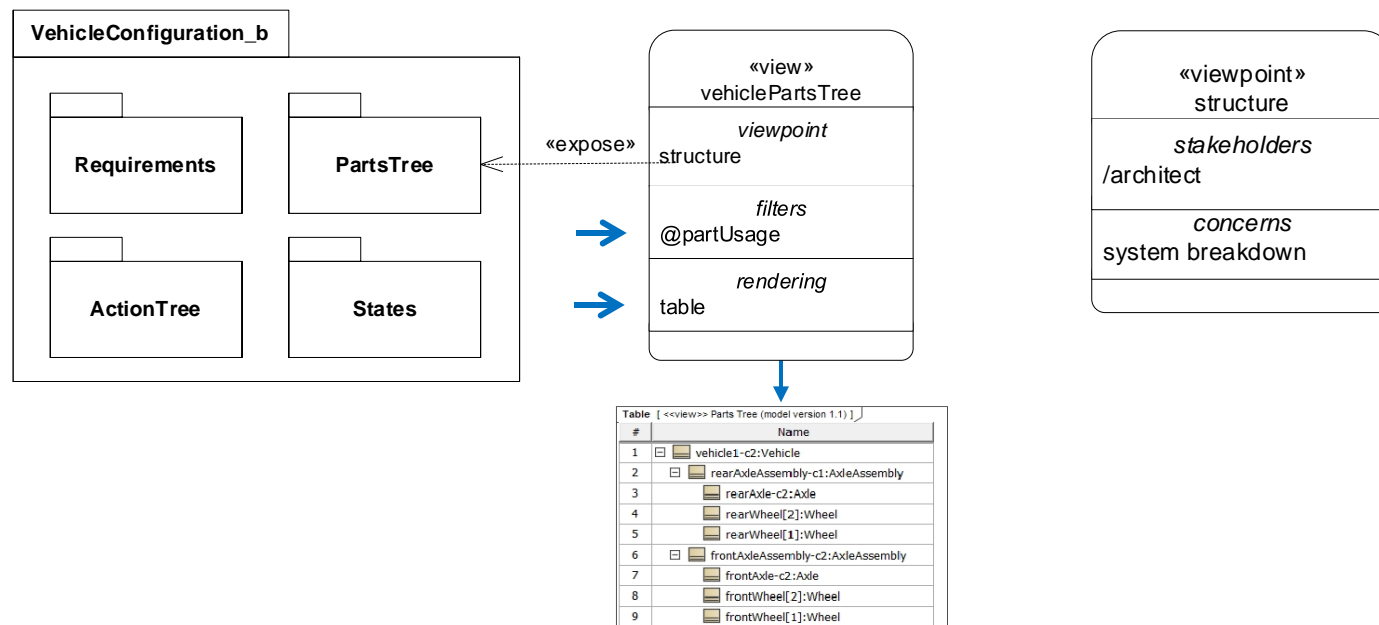


# Module 23

## View & Viewpoint

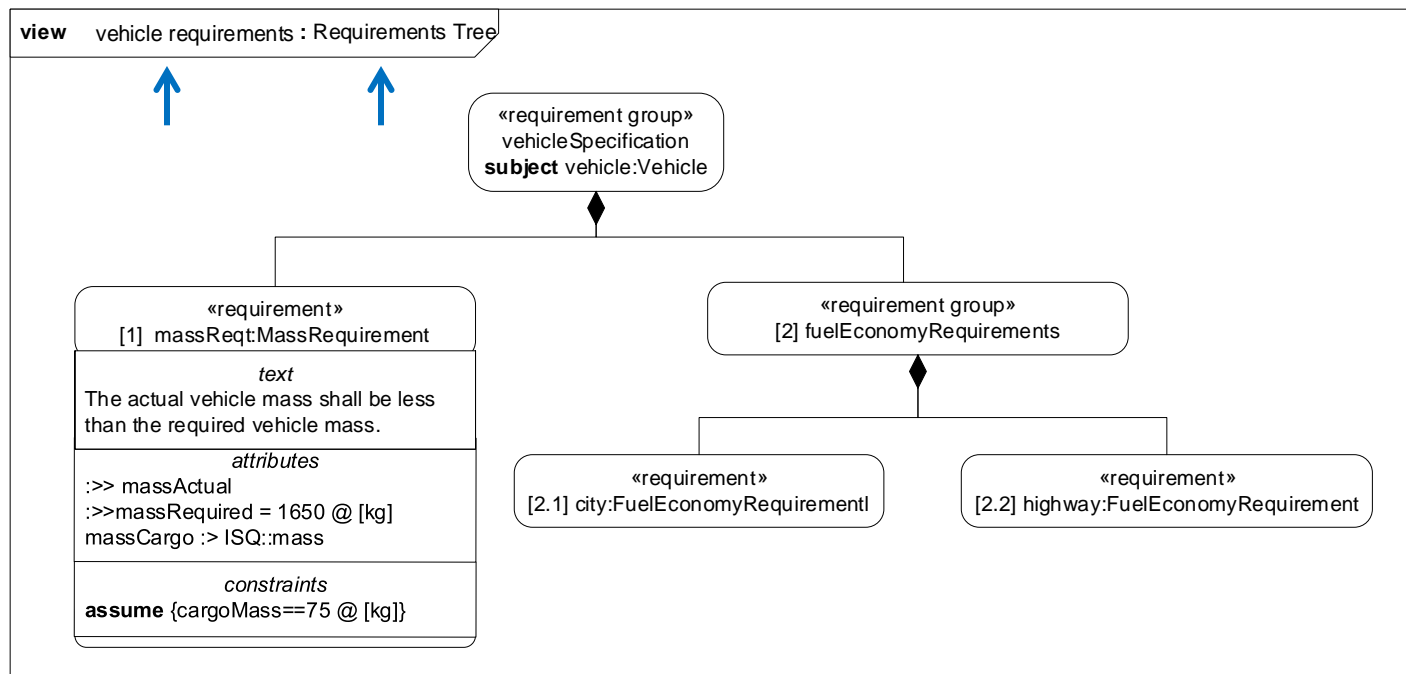
- A Viewpoint reflects the concerns of one more stakeholders in terms of what they care about relative to a domain of interest
- A View specifies an artifact to present to a stakeholder to satisfy their viewpoint
  - Exposes the scope of the model to be viewed
  - Filters the model to select a particular subset based on a scope and filter criteria
  - Renders the filtered results using a particular presentation form and style
  - Intended to support document generation (e.g., OpenMBEE)

v2



- Diagram is a View **v2**

- Frame represents a view that renders model elements which are exposed and filtered
- View definition (e.g., diagram kind)
- View name (e.g., diagram name)
- View content





- Standard diagram kinds / view definitions
  - Tree (hierarchy) views
  - Nested (interconnection) views
  - Relationship view (navigable nodes and edges)
  - Tabular/Matrix views
- A diagram/view can contain nested diagrams/views (e.g., sub views)
- User defined views
  - Users are not constrained to the standard views



# SysML v2 Compartments

SST

- Compartments are views of the element represented by the node
- Compartment names represent view definitions
- Standard compartments / view definitions
  - List view of contained elements of a particular kind (e.g., attributes, actions, ports)
  - List view of elements at the other end of relationships (e.g., allocation, specialization, ...)
- A compartment can be rendered using textual (e.g., list view) or graphical notation
- Nested view is a kind of graphical compartment where elements are represented as graphical symbols

<b>«part def» Vehicle</b>
<i>attributes</i> mass:>ISQ::mass dryMass cargoMass electricalPower position velocity acceleration ...
<i>ports</i> pwrCmdPort vehicleToRoadPort ...
<i>perform actions</i> providePower provideBraking controlDirection ...
<i>exhibit states</i> vehicleStates

# Module 24

## Language Extension

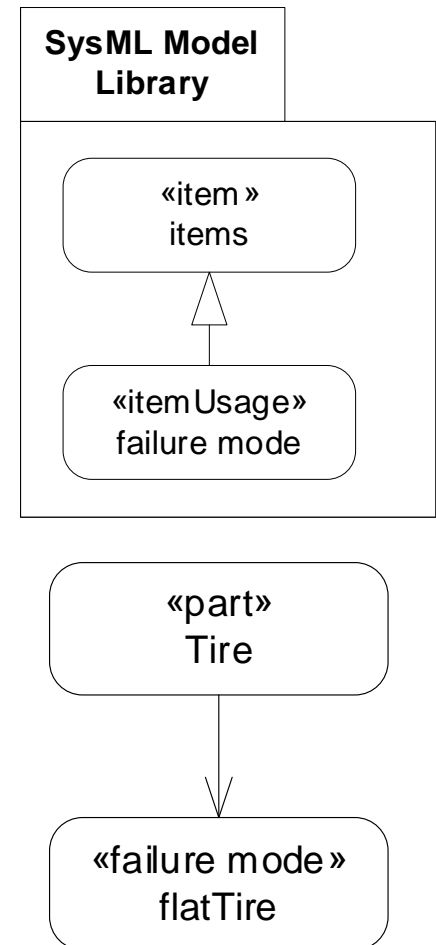


# Language Extension

In Process

SST

- Provides ability to extend concepts in SysML v2 to address domain-specific concepts and terminology
- Define concept by subclassing an element in the SysML Model Library **v2**
- Apply the concept to an element **define the concept** →



**apply the concept** →

# Summary





# Contrasting SysML v1 with SysML v2 *SST*

- **Simpler to learn and use**

- Systems engineering concepts designed into metamodel versus added-on
- Consistent definition and usage pattern
- More consistent terminology
- Ability to decompose parts, actions, ...

- **More precise**

- Textual syntax and expression language
- Formal semantic grounding
- Requirements as constraints
- Reified relationships (e.g., membership, annotation)

- **More expressive**

- Variant modeling
- Analysis case
- Trade-off analysis
- Individuals, snapshots, time slices
- More robust quantitative properties (e.g., vectors, ..)
- Filter expressions

- **More extensible**

- Simpler language extension capability
  - Based on model libraries

- **More interoperable**

- Standardized API



# References

SST

- Monthly Release Repository (release 2021-08)
  - <https://github.com/Systems-Modeling/SysML-v2-Release>
- SysML v2 Specification (revised submission)
  - Note: refer to Annex B for Example Vehicle Model using textual notation
- Introduction to the SysML v2 Language Textual Notation (release 2021-08)
- Friedenthal S., Seidewitz E., A Preview of the Next Generation System Modeling Language (SysML v2), Project Performance International (PPI), [Systems Engineering Newsletter, PPI SyEN 95 27 November, 2020](#)