SST

Introduction to the SysML v2 Language Graphical Notation

(2023-03-07)

This presentation uses the graphical notation to complement the textual syntax (i.e., notation) to represent a SysML model. The graphical notation is generated manually with Microsoft Visio.

Agenda

SST

- Language Background and Overview
- SysML v2 Functional Areas
- Contrasting SysML v2 with SysML v1
- Summary

Refer to date in lower right on each slide to see when it was last modified

General Caveats

SST

- The graphical and textual notation (i.e., syntax) are different renderings of the same model
  - The textual syntax is formally specified using BNF
  - The graphical syntax is formally specified using graphical BNF
  - The graphical visualization in this presentation is captured in Visio
  - 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

Language Background & Overview

Systems Modeling Language™ (SysML®)

SST

Supports the specification, analysis, design, and verification and validation of complex systems that may include hardware, software, information, processes, personnel, and facilities

- SysML has evolved to address user and vendor needs
  - v1.0, adopted in 2006; v1.6, current version; v1.7
- SysML has facilitated awareness and adoption of MBSE
- Much has been learned from using SysML for MBSE

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

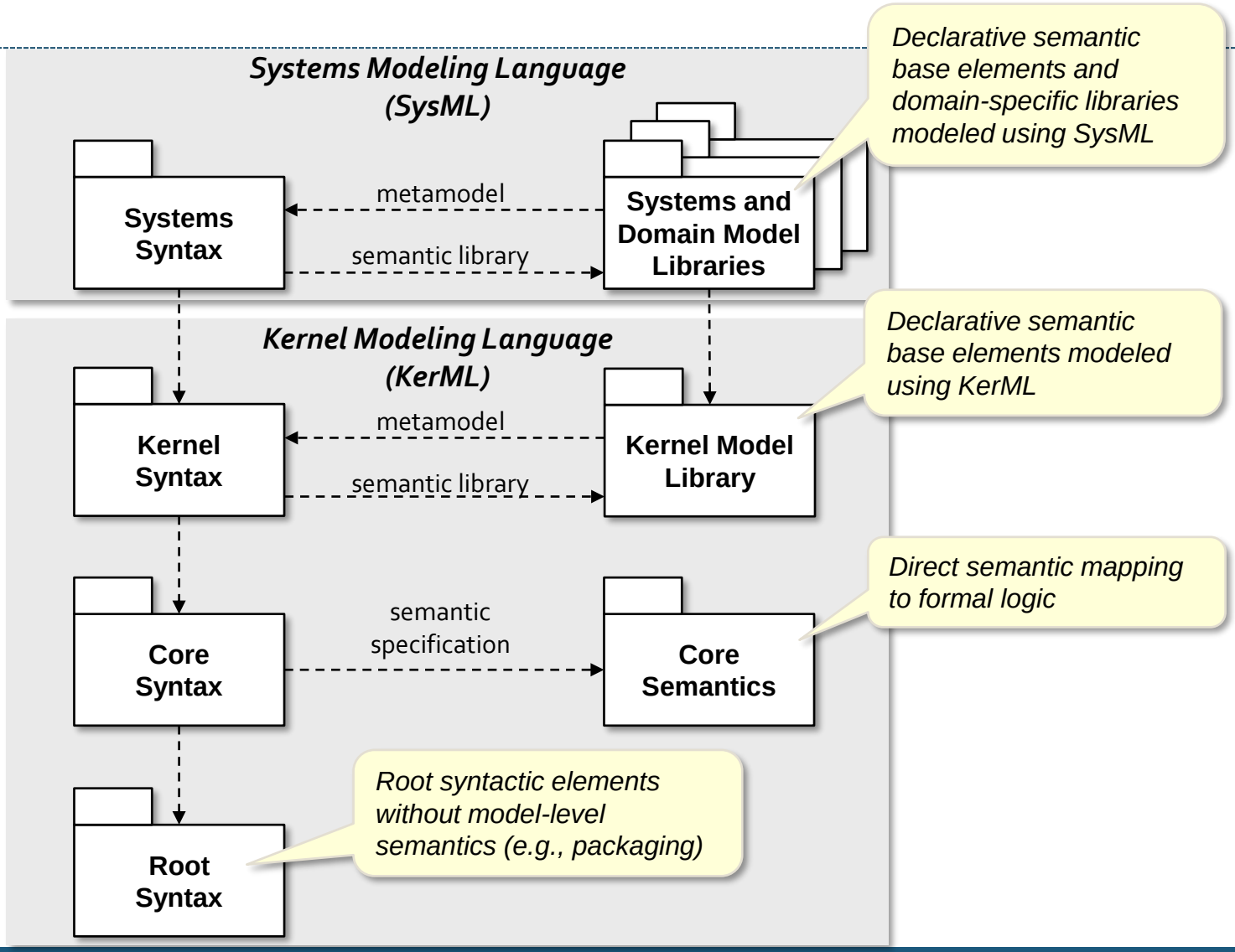
Key Elements of SysML v2

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
  - Graphical, Tabular, Textual
- Standardized API to access the model

SysML v2 Language Architecture

SST



SysML v2 API & Services

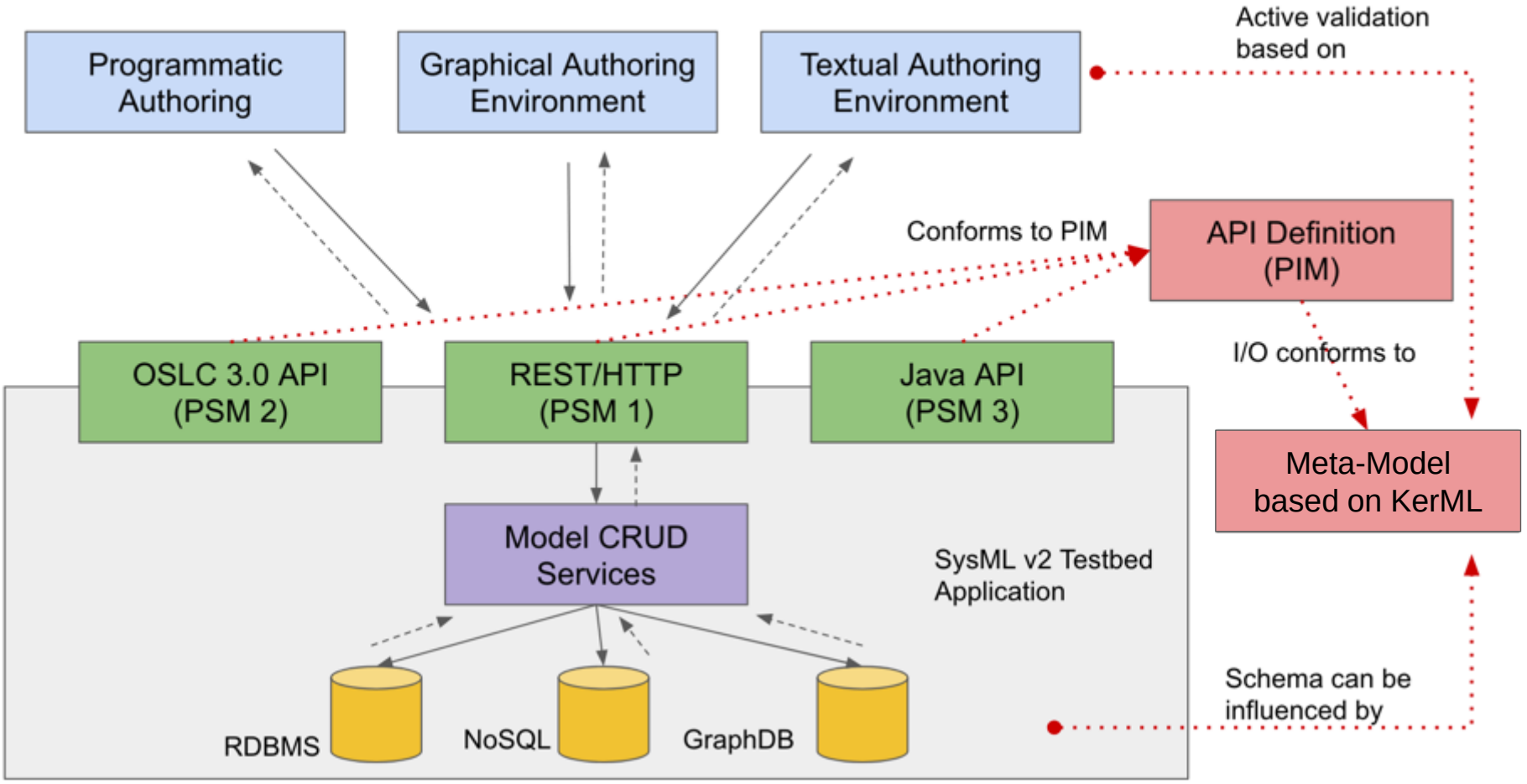
SST

- Enables other tools and applications to access SysML models in a standard way
- Provides services to:
  - Create, update, and delete elements
  - Query and navigate model
  - Other services including support for model management, analysis, transformation, and file export generation
- Supports common patterns called recipes (GitHub - Systems-Modeling/SysML-v2-API-Cookbook: Recipes for using the SysML v2 API)
  - Navigating a decomposition tree
  - Creating a branch
  - Query with multiple constraints
- Facilitates use of different implementation technologies such as REST/HTTP, Java, or OSLC

Pilot Implementation Using Standard API

SST

High-Level Architecture of SysML v2 Testbed



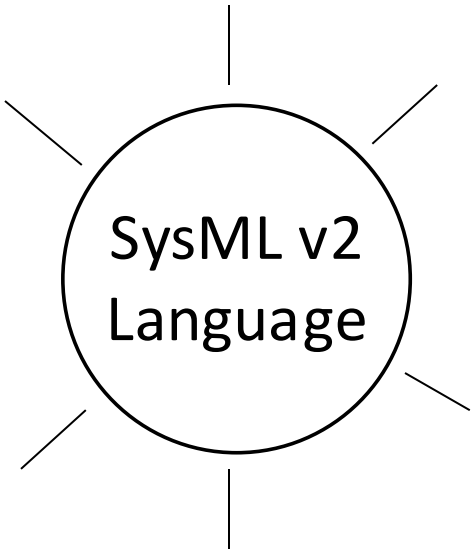
SysML v2 Language Capabilities

SST

Requirements

Behavior
- function-based
- state-based
- sequence-based
- use cases

Structure
- decomposition
- interconnection
- classification



Analysis
- analysis cases
- expression language

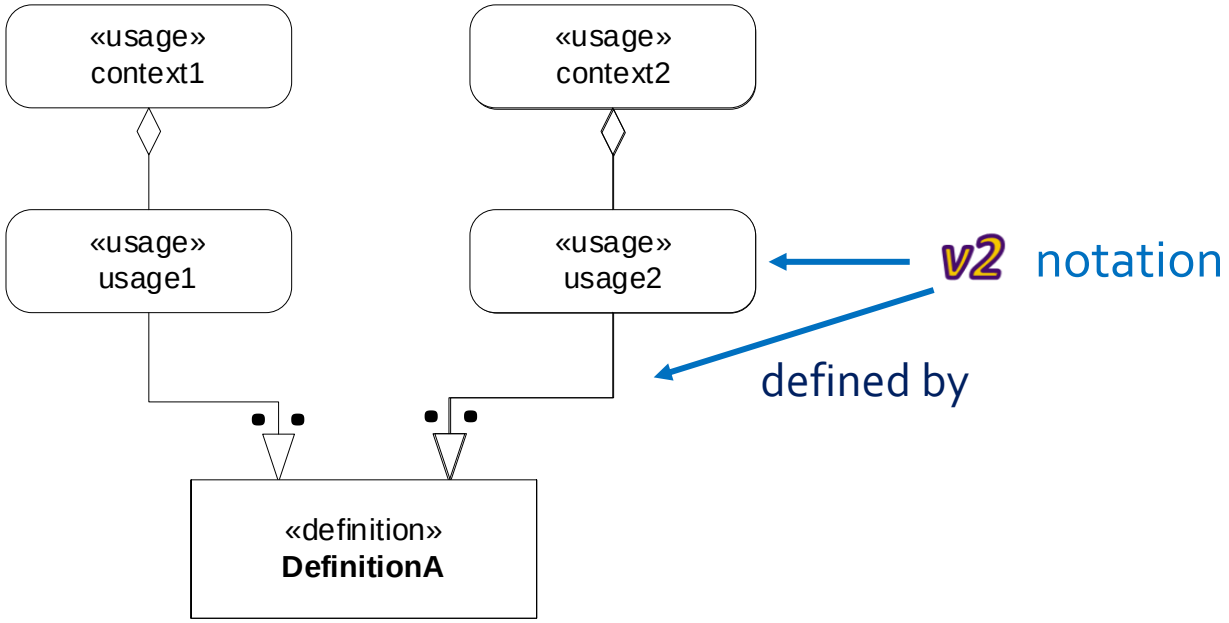
Verification
- verification cases

View & Viewpoint

Definition and Usage Reuse Pattern

SST

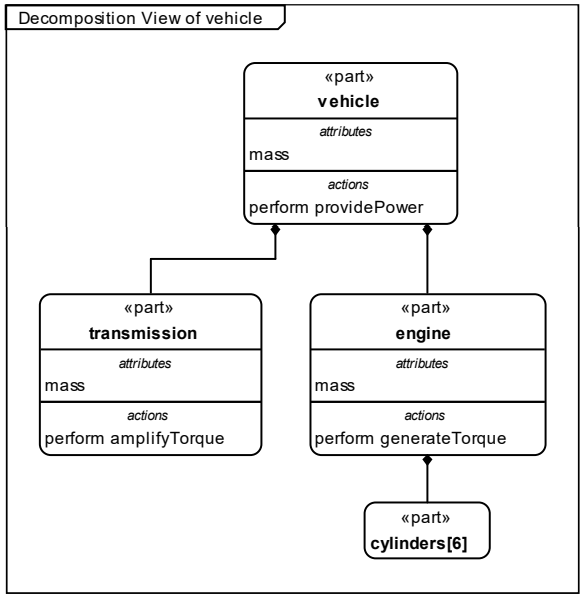
- A definition element defines an element such as a part, action, or requirement
- A usage element is a usage of a definition element in a particular context
  - There can be different usages of the same definition element in either different contexts or the same context
- Pattern is applied consistently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

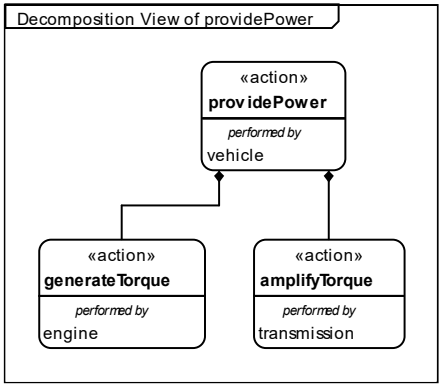


SysML v2 Notation (1 of 2) Textual and Graphical

SST

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

  action providePower {
    action generateTorque;
    action amplifyTorque;
  }
}
```

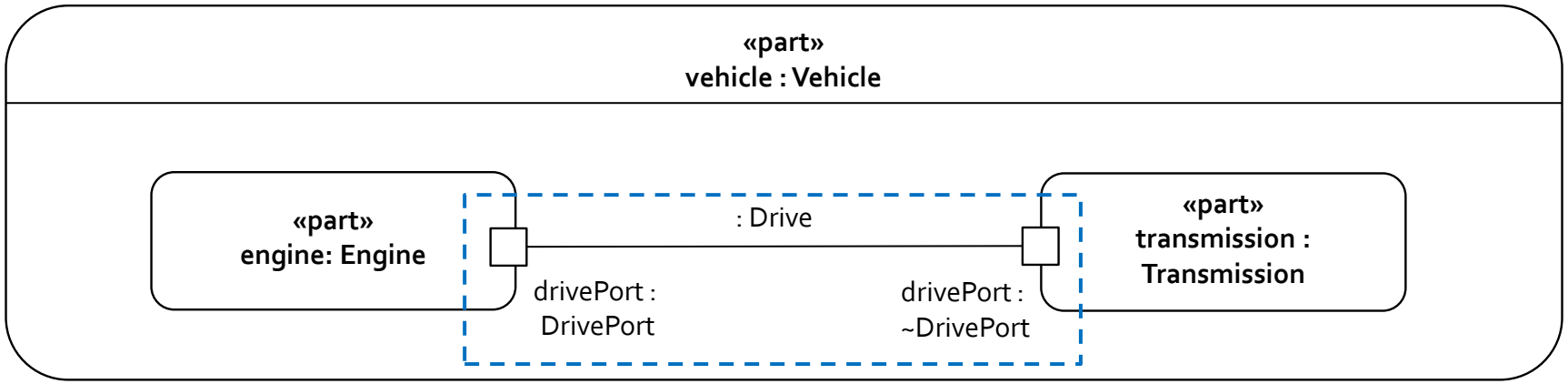




SysML v2 Notation (2 of 2) Textual and Graphical

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

SysML v2 Spec (Clause 7)
SysML v2 Language Description

SST

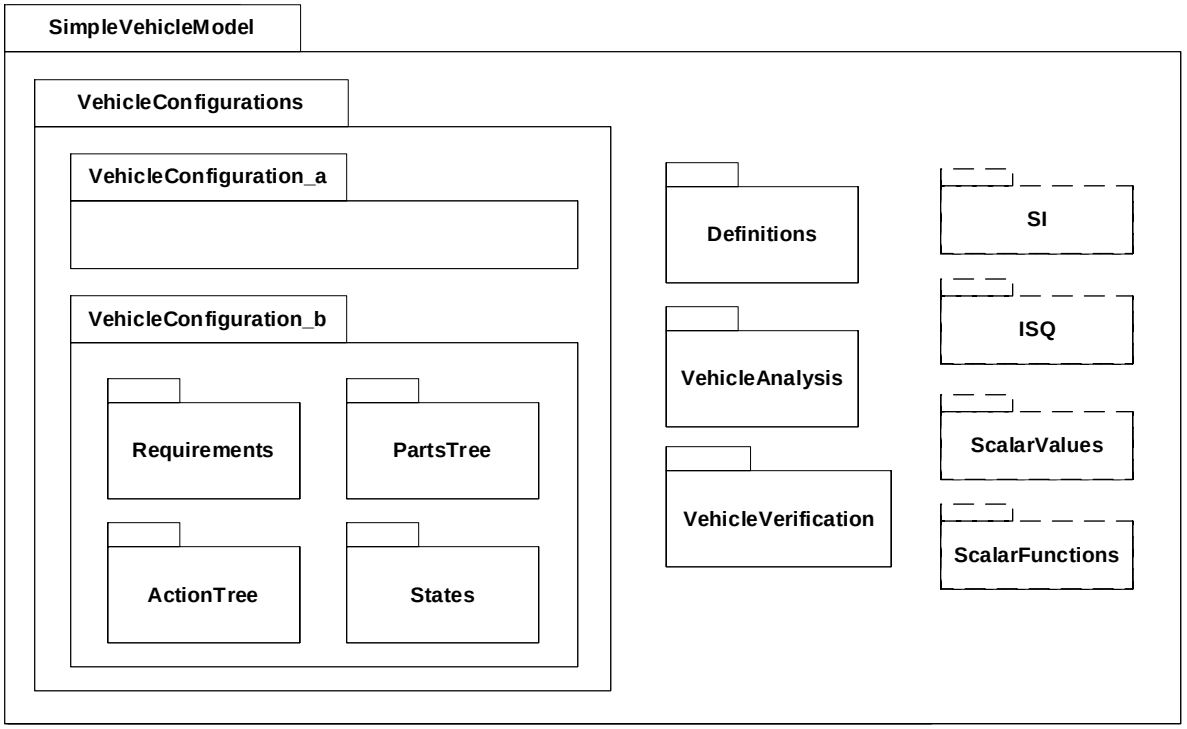
7.1 Language Overview
7.2 Elements and Relationships
7.3 Dependencies
7.4 Annotations
7.5 Namespaces and Packages
7.6 Definition and Usage
7.7 Attributes
7.8 Enumerations
7.9 Occurrences
7.10 Items
7.11 Parts
7.12 Ports
7.13 Connections
7.14 Interfaces
7.15 Allocations
7.16 Actions
7.17 States
7.18 Calculations
7.19 Constraints
7.20 Requirements
7.21 Cases
7.22 Analysis Cases
7.23 Verification Cases
7.24 Use Cases
7.25 Views and Viewpoints
7.26 Metadata (incl. User Defined Keywords)

Module 1
Packages & Element Names

Model Organization

SST

- 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
  - Nested import notation enables flexible organization v2



Import Notation

- no star is element import
- single star is package import (content of package)
- double star is recursive including outer package

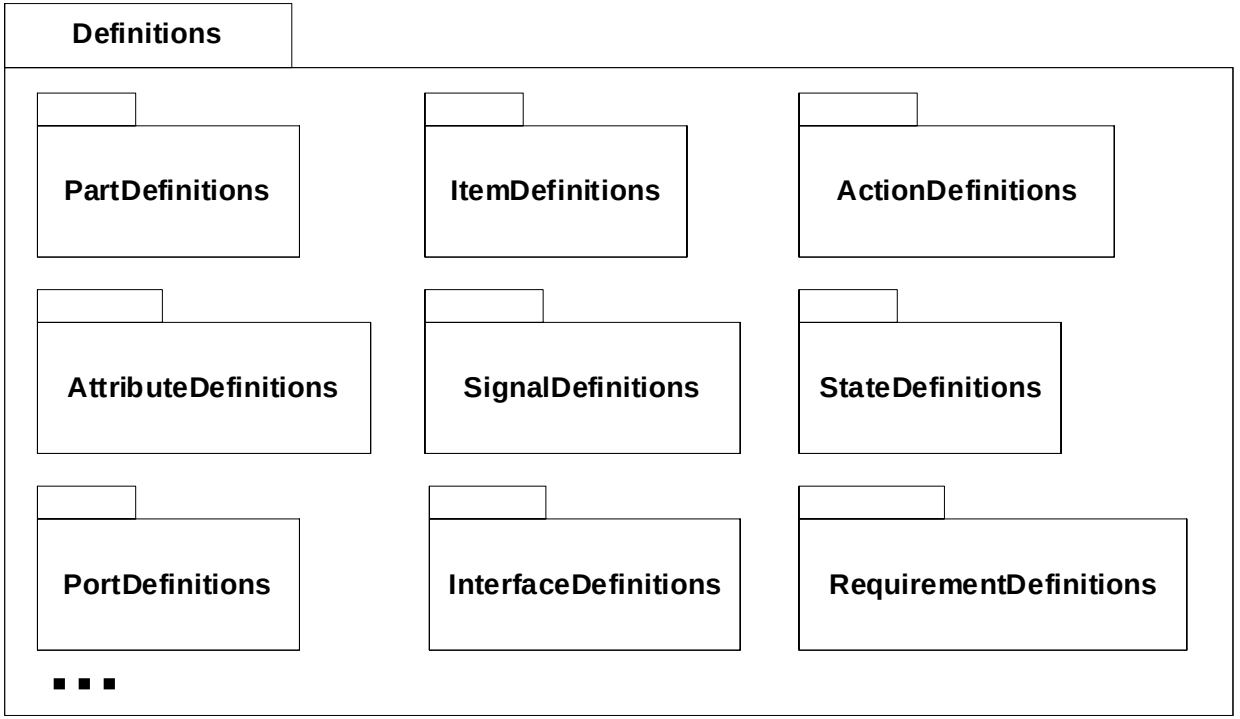
- Imports can apply to any namespace, but packages are most commonly used
- A package import can include filter criteria to select elements based on their metadata

v2

Definitions Package

SST

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

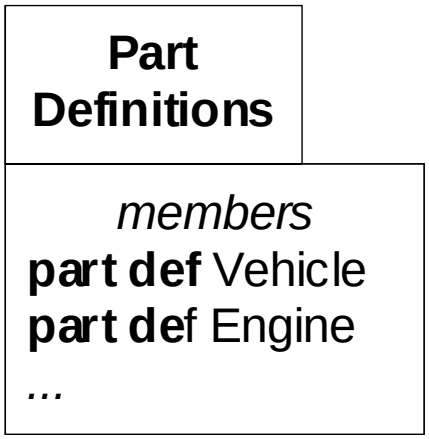


v2 →

Package Compartments

SST

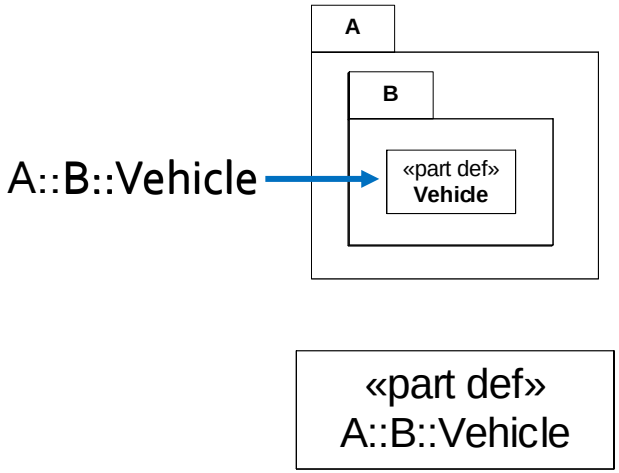
- Packages can include textual compartments v2
- Filter can be applied to select element kinds v2



Qualified Names

SST

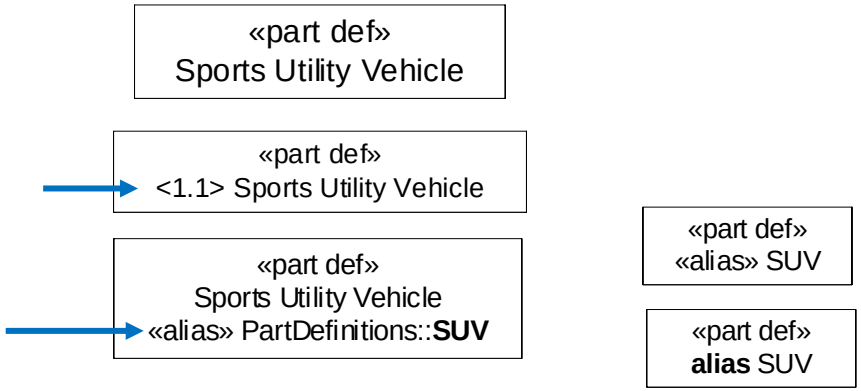
- A qualified name of a model element is prepended by the name of its owner followed by a double colon (::)
- A fully qualified name is prepended by the concatenated names of its owners in its containment hierarchy separated by double colons (::)



Element Names

SST

- Each element includes a universally unique identifier (UUID) that is not visible
  - Managed by the tool
  - Does not change over the life of the element
- Each element can include a name
  - Required for an element to be referenced in the textual notation
- Each element can optionally include a short name in angle brackets v2
  - Can be a user or tool selected id
- An element can have one or more aliases in the context of a namespace v2



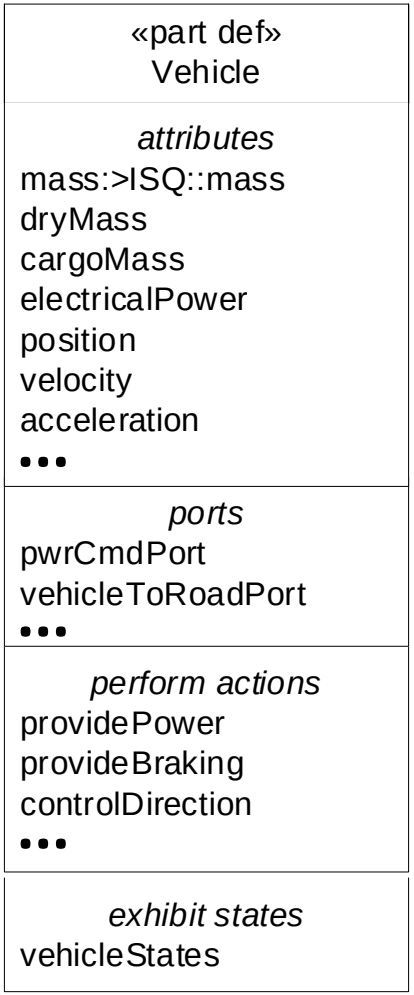
Note: reserved words can be shown in «guillemet» or **bold**

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

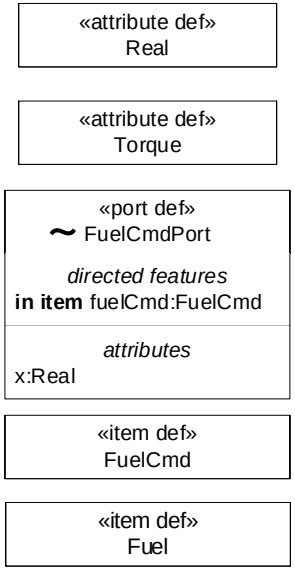


Other Kinds of Definition Elements

Terminology v2

SST

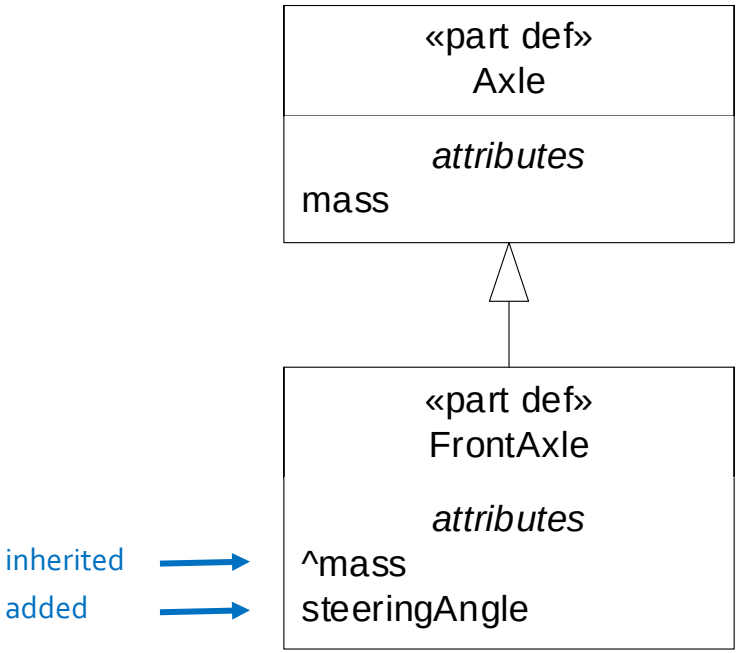
- Each kind of definition element can contain specific kinds of features
  - Attribute definition
    - Data type that defines a set of data values
    - Primitive attribute definitions include integer, Real, Complex, Boolean, String, ….
    - Can include quantity kinds such as Torque with units
    - 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 is acted on, such as an input, output 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



Specialization

SST

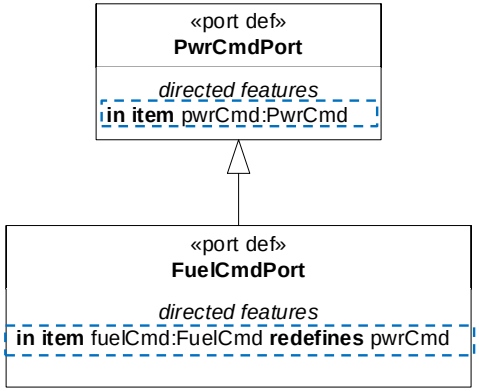
- Definition elements can be specialized
  - Subclass inherits features from its superclass that can be redefined or subsetted
  - Subclass can add new features
  - Subclass can be a specialization of more than one superclass (i.e., multiple inheritance)



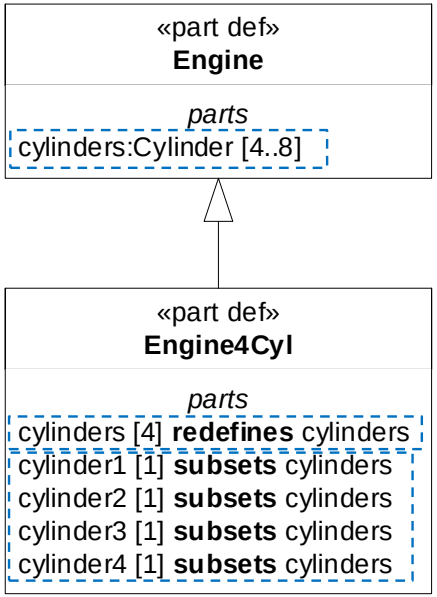
Redefinition and Subsetting

SST

- 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 **subsetted** 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

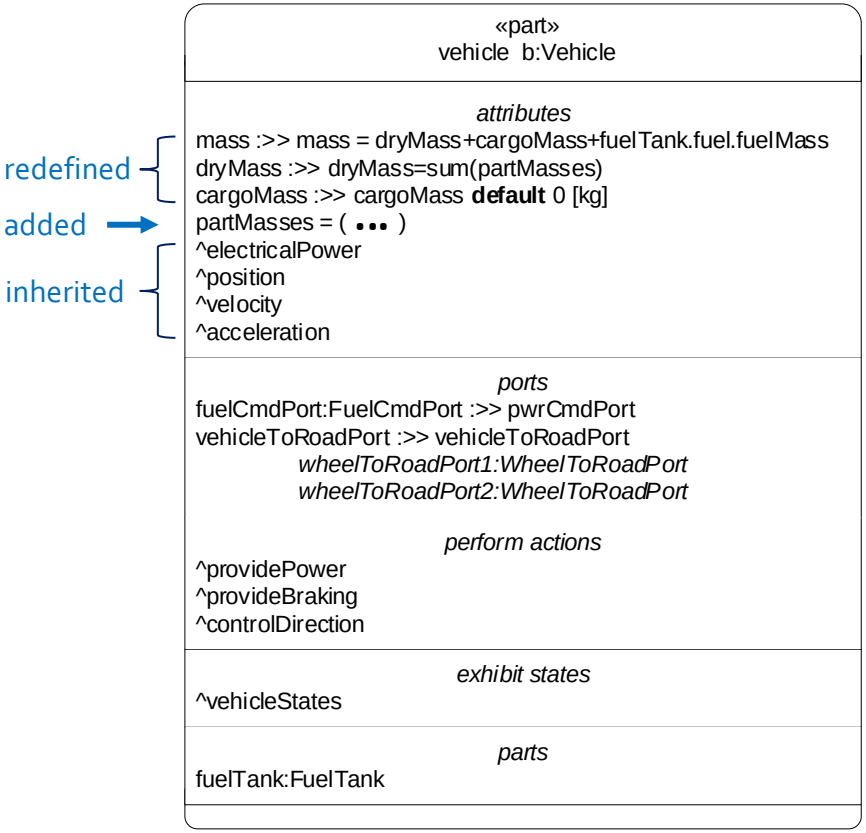


Module 3
Usage Elements

Part

SST

- 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

SST

- 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
  - …

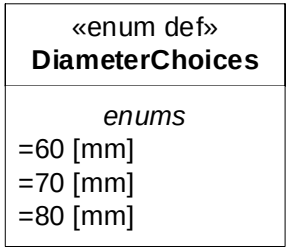
Enumeration

SST

- 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 = Colors::blue;

  - With units:
    - Definition: **enum def** DiameterChoices :> ISQ::LengthValue {

```
    enum = 60 [mm];
    enum = 70 [mm];
    enum = 80 [mm];
}
```



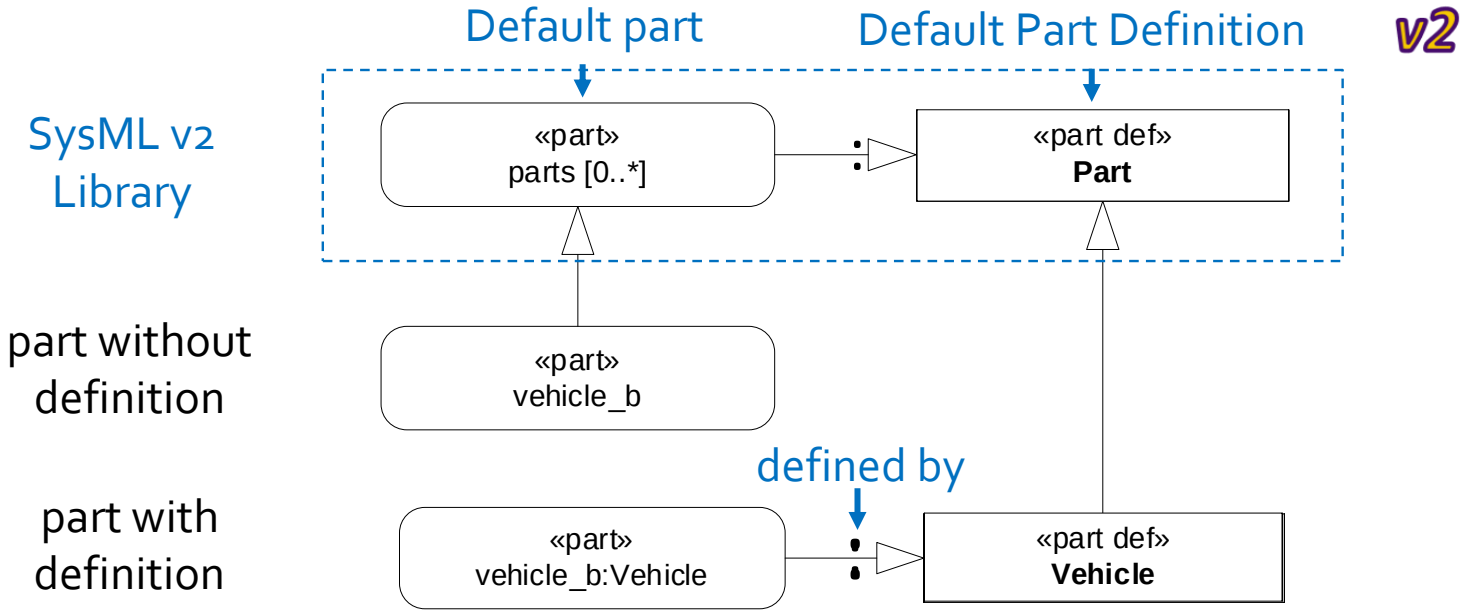
v2

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

Usage Elements Defined By Default Definition Elements

SST

- 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)

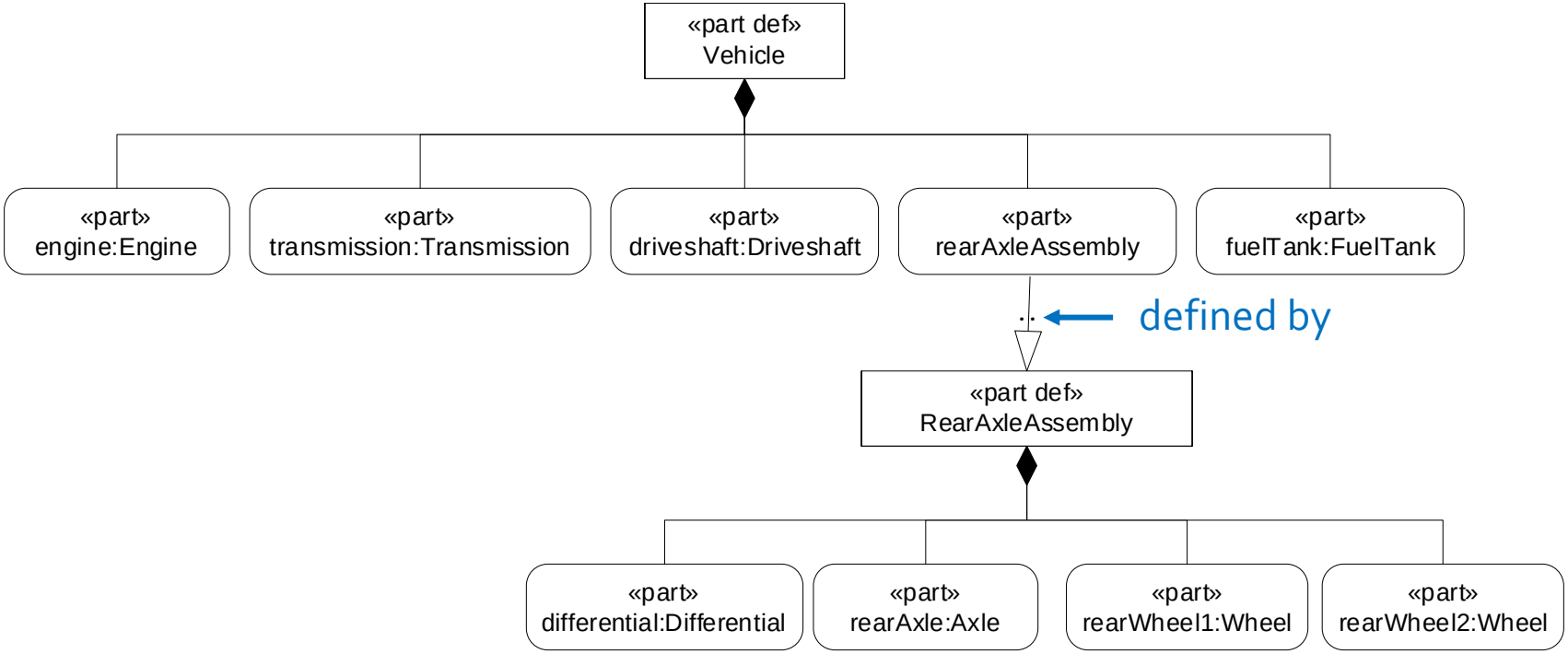


Module 4
Part Decomposition

SysML v1 Decomposition in SysML v2

SST

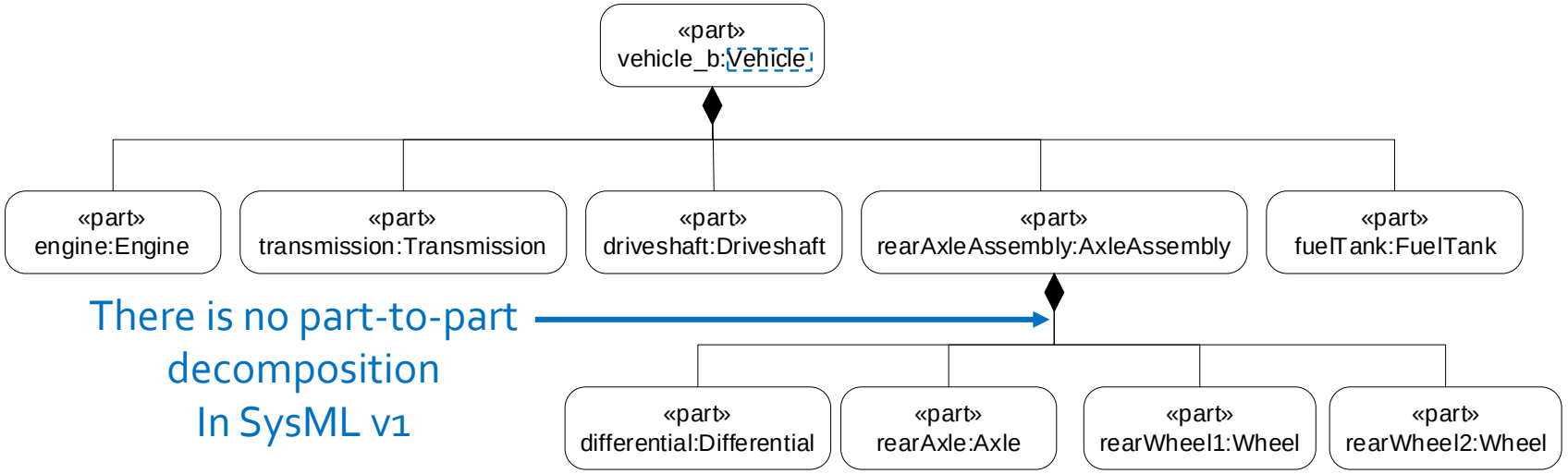
- *defined by* relationship is used to create next level of decomposition of a part def



Parts Decomposition

SST

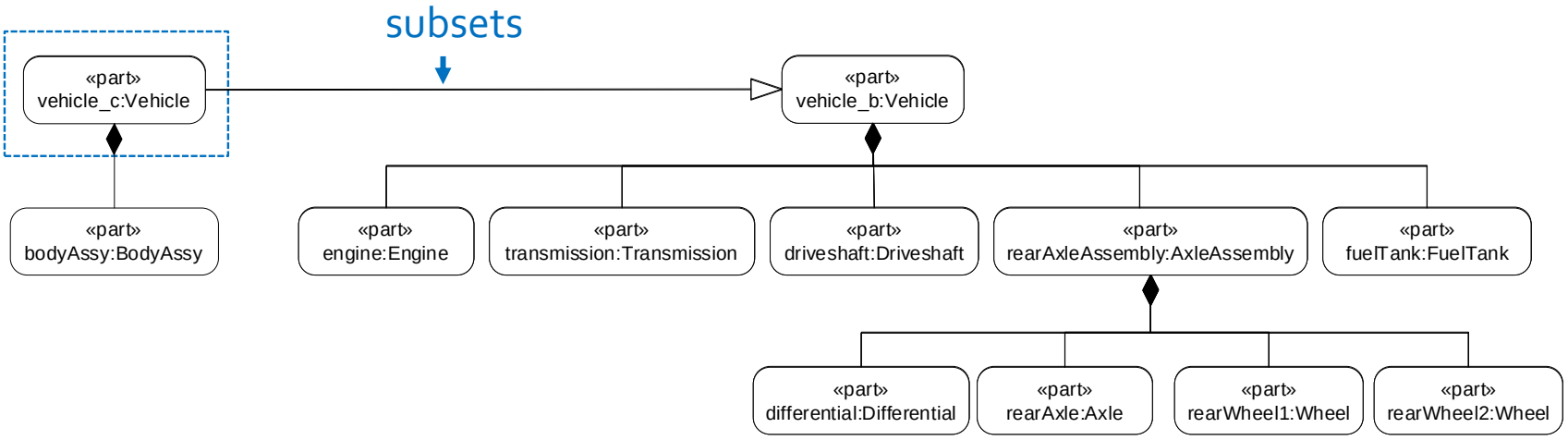
- A parts tree is a composition of a part from other parts v2
  - SysML v1 only supports block decomposition
  - Definition element can serve as black-box specification without committing to an internal structure (e.g., part decomposition)
  - Can provide significant advantages (more intuitive, less ambiguous, easier to modify a design configuration)



Parts Tree Reuse & Modification
Subsetting a Part

SST

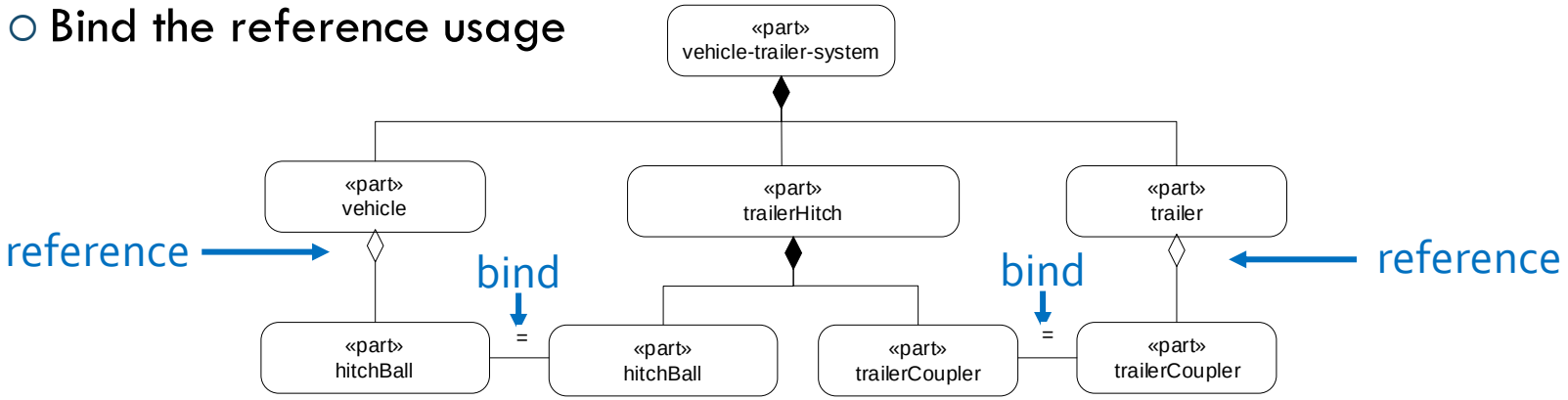
- A part can be a subset of another part
  - Equivalent to specializing a part v2
  - Inherits the part decomposition and other features
  - Can modify inherited parts and features through redefinition and subsetting
  - Can add new parts v2



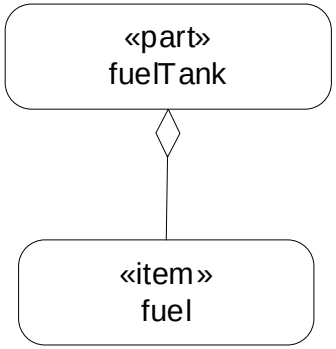
References

SST

- 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



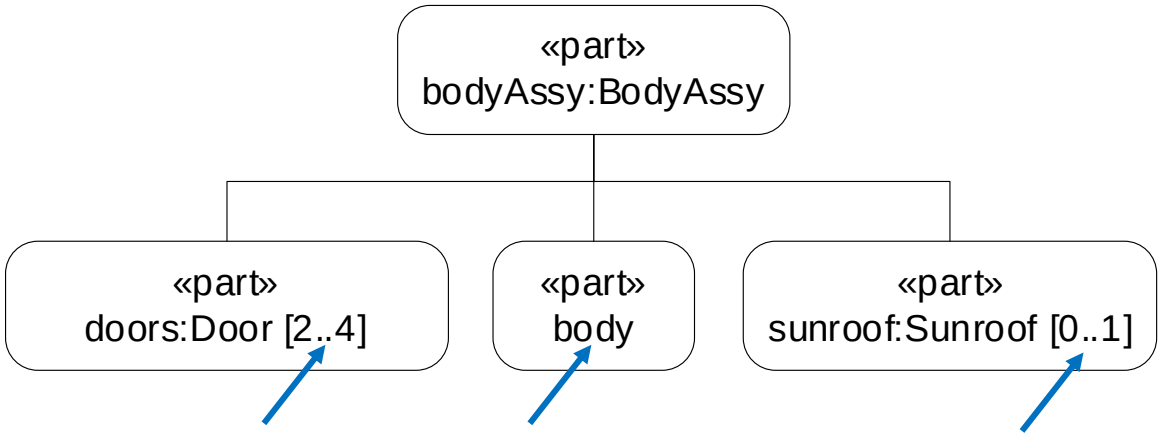
- A part can refer to an item that it stores v2



Multiplicity

SST

- 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
    - Can be parameterized [x..y] where x and y are expressions v2
  - Default multiplicity is [1..1]
  - Optional multiplicity [0..1]

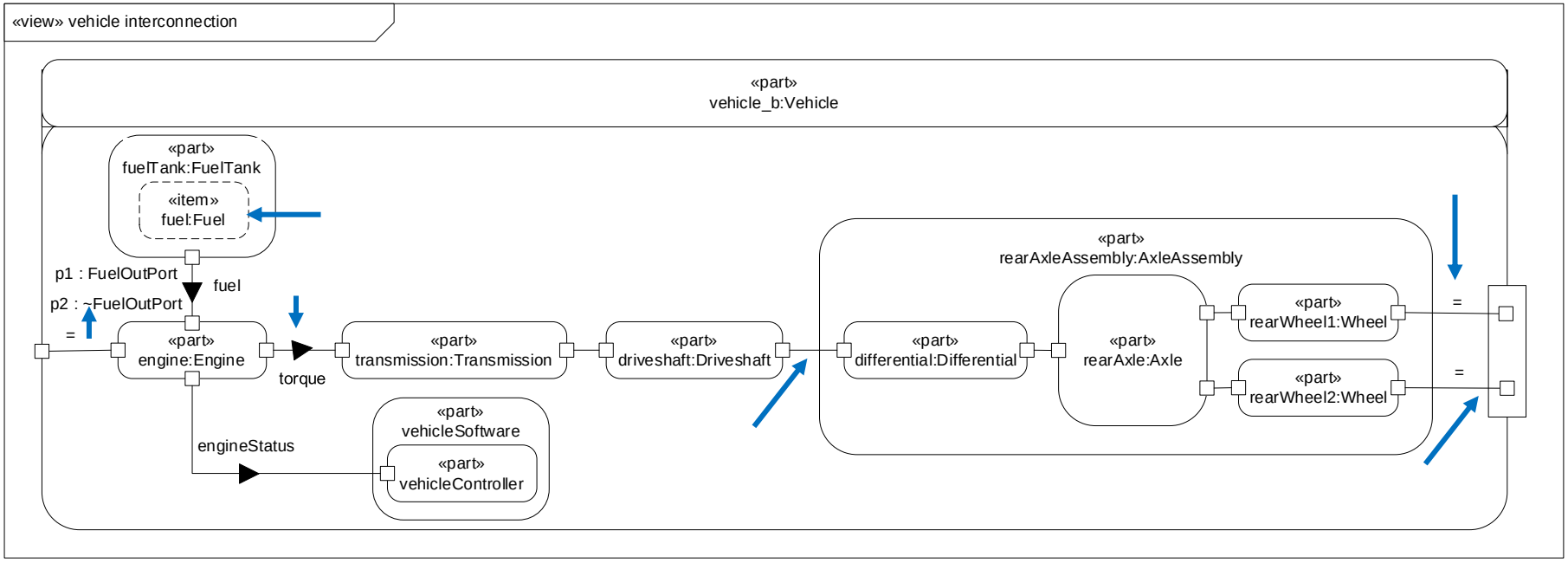


Module 5
Part Interconnection

Connecting Parts

SST

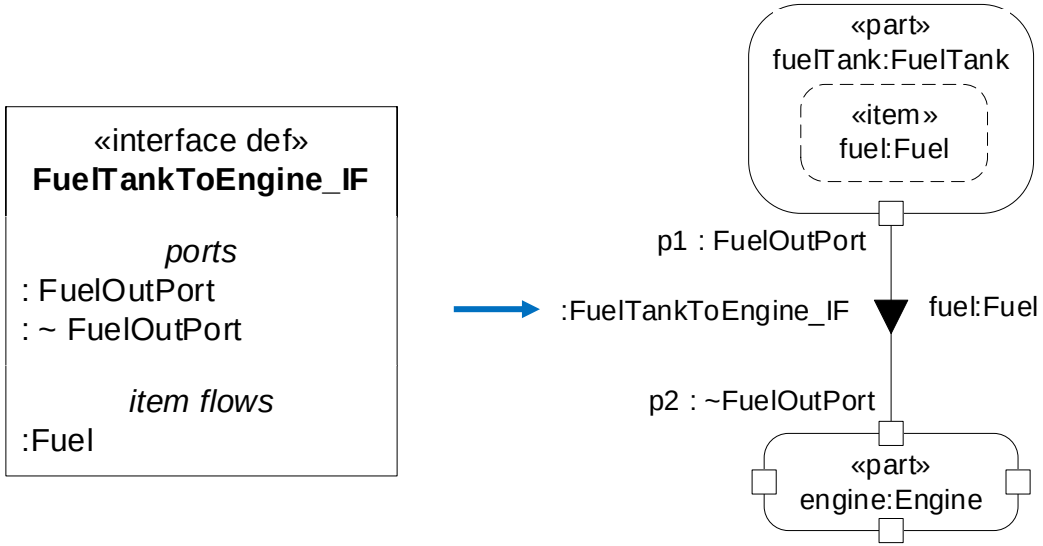
- Parts can be connected via their ports or without ports
  - Parts can be connected directly without connecting to the composite part
  - Ports can contain nested ports
  - Flows can be shown on connections
  - References are dashed
  - Conjugate port
  - Binding connection



Interfaces

SST

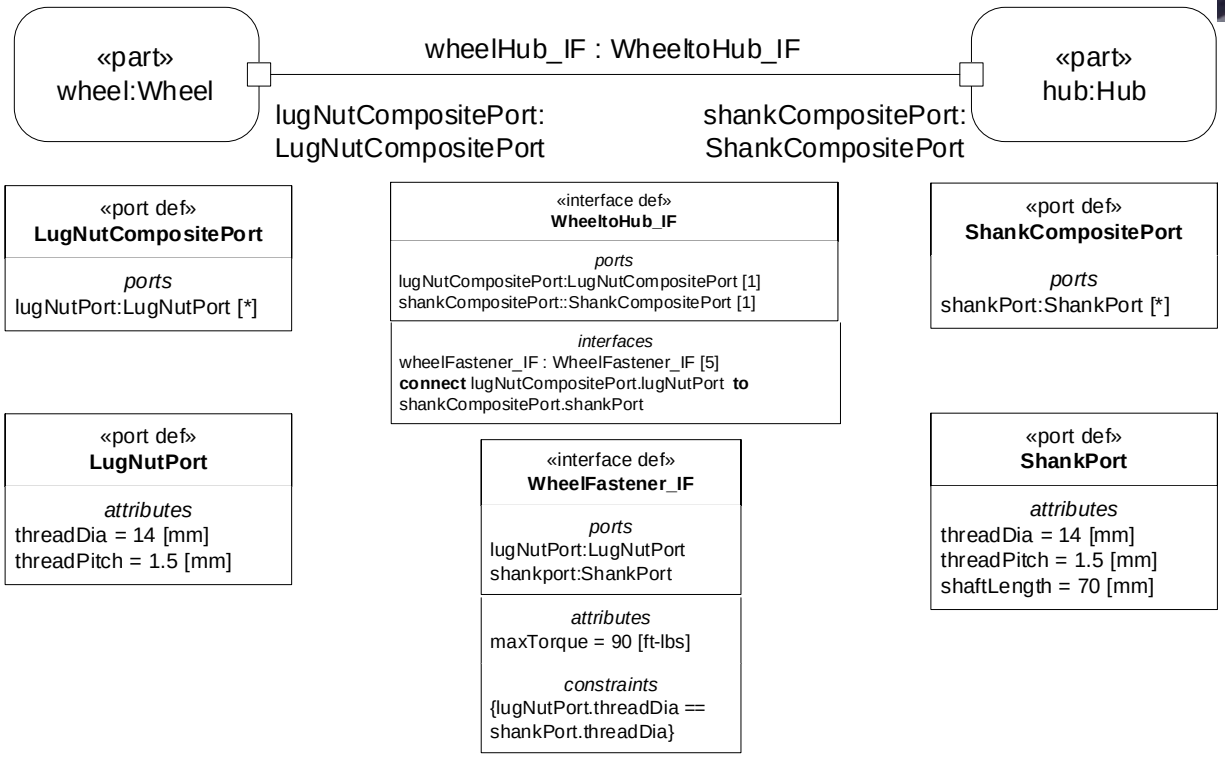
- A connection definition connects two usage elements (e.g., two parts)
- An interface definition is a connection definition whose ends are compatible ports
  - Can contain flows
  - Can contain other kinds of features
- An interface defined by an interface definition is used to connect ports
  - Ports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



Interface
Connecting a Wheel to a Hub

SST

- Usage of WheelToHub interface definition specifies a compatible connection between a wheel and a hub
  - Connects composite ports on wheel and hub
  - Decomposes into 5 Wheel Fastener interfaces
- Wheel Fastener interface species a compatible connection between a lugnut and shank
  - Contains features needed for a compatible connection



Wheel Hub Interface

SST

- Usage of WheelHubInterface to specify interface between rear axle and wheels

```
part wheelHubAssy3{
    part wheel1:Wheel{
        port lugNutCompositePort :>> lugNutCompositePort {
            port lugNutPort [5] :>> lugNutPort {
                attribute :>> threadDia = 14 [mm];
                attribute :>> threadPitch = 1.5 [mm];
            }
            port lugNutPort1 [1] :> lugNutPort;
            port lugNutPort2 [1] :> lugNutPort;
            port lugNutPort3 [1] :> lugNutPort;
        }

    part hub1:Hub{
        port shankCompositePort :>> shankCompositePort {
            port shankPort [5] :>> shankPort {
                attribute :>> threadDia = 14 [mm];
                attribute :>> threadPitch = 1.5 [mm];
                attribute :>> shaftLength = 70 [mm];
            }
            port shankPort1 [1] :> shankPort;
            port shankPort2 [1] :> shankPort;
            port shankPort3 [1] :> shankPort;
        }

    interface wheelHubInterface:WheelHubInterface
        connect lugNutCompositePort ::> wheel1.lugNutCompositePort [1] to shankCompositePort ::> hub1.shankCompositePort [1] {
            interface wheelFastenerInterface1 :> wheelFastenerInterface
                connect lugNutPort ::> lugNutCompositePort.lugNutPort1 to shankPort ::> shankCompositePort.shankPort1 {
                    attribute :>> maxTorque = 90 * 1.356 [N*m];
                }
            interface wheelFastenerInterface2 :> wheelFastenerInterface
                connect lugNutPort ::> lugNutCompositePort.lugNutPort2 to shankPort ::> shankCompositePort.shankPort2 {
                    attribute :>> maxTorque = 90 * 1.356 [N*m];
                }
            interface wheelFastenerInterface3 :> wheelFastenerInterface
                connect lugNutPort ::> lugNutCompositePort.lugNutPort3 to shankPort ::> shankCompositePort.shankPort3 {
                    attribute :>> maxTorque = 90 * 1.356 [N*m];
                }
    }
}
```

```
interface def WheelFastenerInterface{
    end lugNutPort:LugNutPort;
    end shankPort:ShankPort;
    attribute maxTorque : Torque;
    constraint {lugNutPort.threadDia == shankPort.threadDia}
}
interface def WheelHubInterface{
    end lugNutCompositePort:LugNutCompositePort;
    end shankCompositePort:ShankCompositePort;
    interface wheelFastenerInterface:WheelFastenerInterface [5]
        connect lugNutCompositePort.lugNutPort to shankCompositePort.shankPort;
}
```

«interface def»
WheeltoHub_IF

ports
lugNutCompositePort:LugNutCompositePort [1]
shankCompositePort::ShankCompositePort [1]

interfaces
wheelFastener_IF : WheelFastener_IF [5]
connect lugNutCompositePort.lugNutPort **to** shankCompositePort.shankPort

«interface def»
WheelFastener_IF

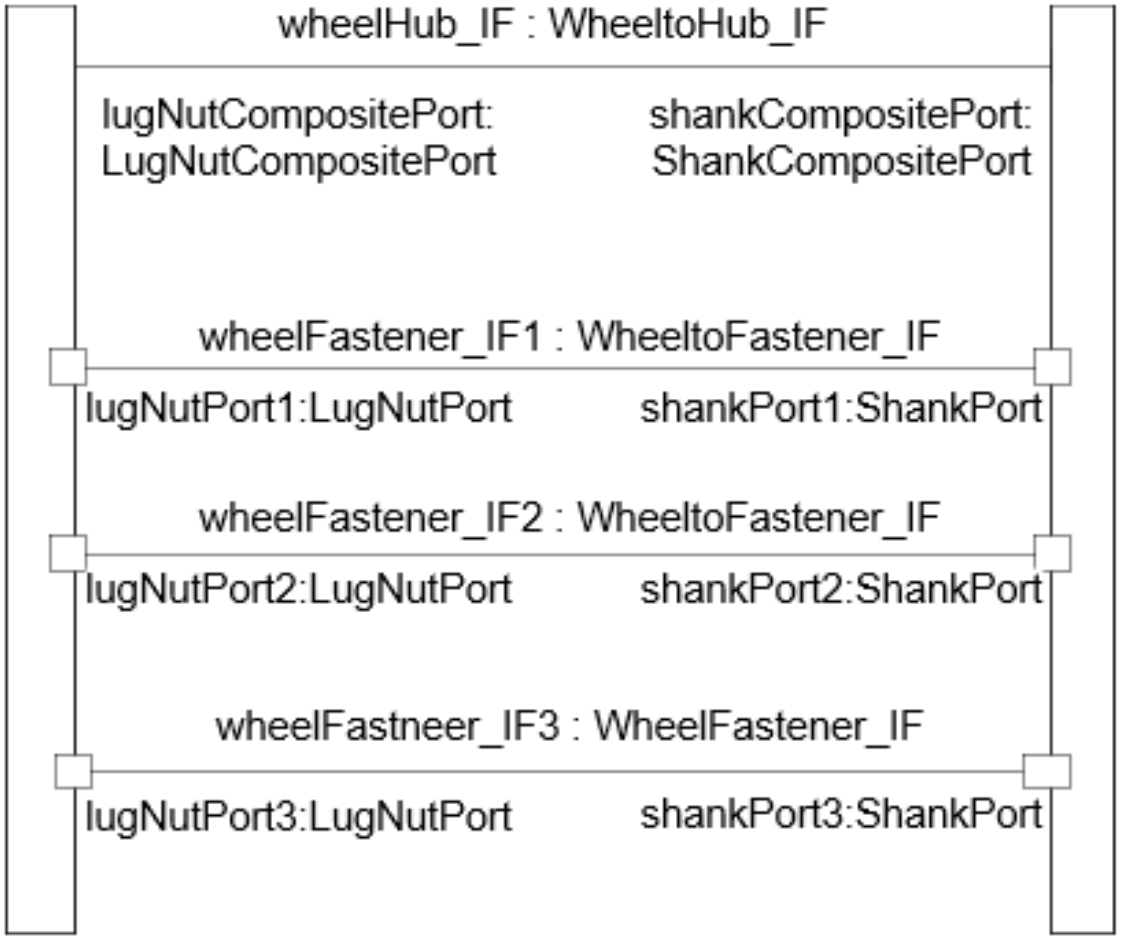
ports
lugNutPort:LugNutPort
shankport:ShankPort

attributes
maxTorque = 90 [ft-lbs]

constraints
{lugNutPort.threadDia == shankPort.threadDia}

Nested Wheel Hub Interface

SST

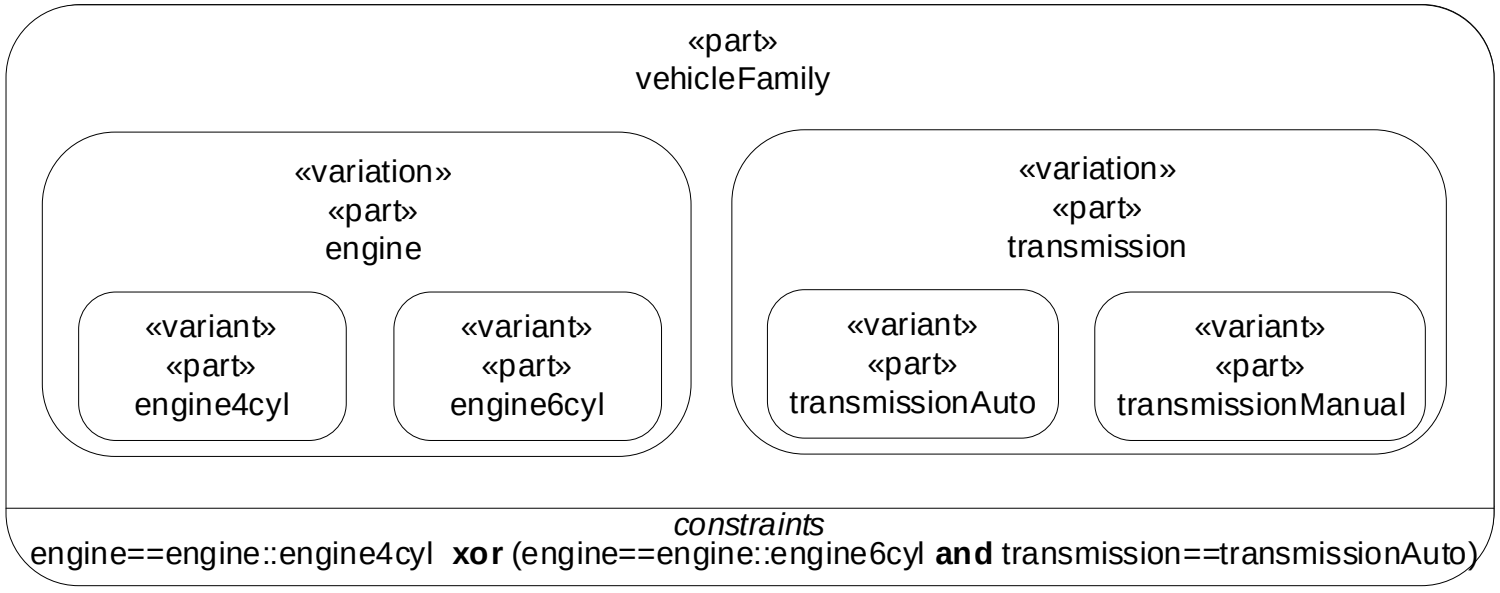


Module 6
Variability

Variability v2

SST

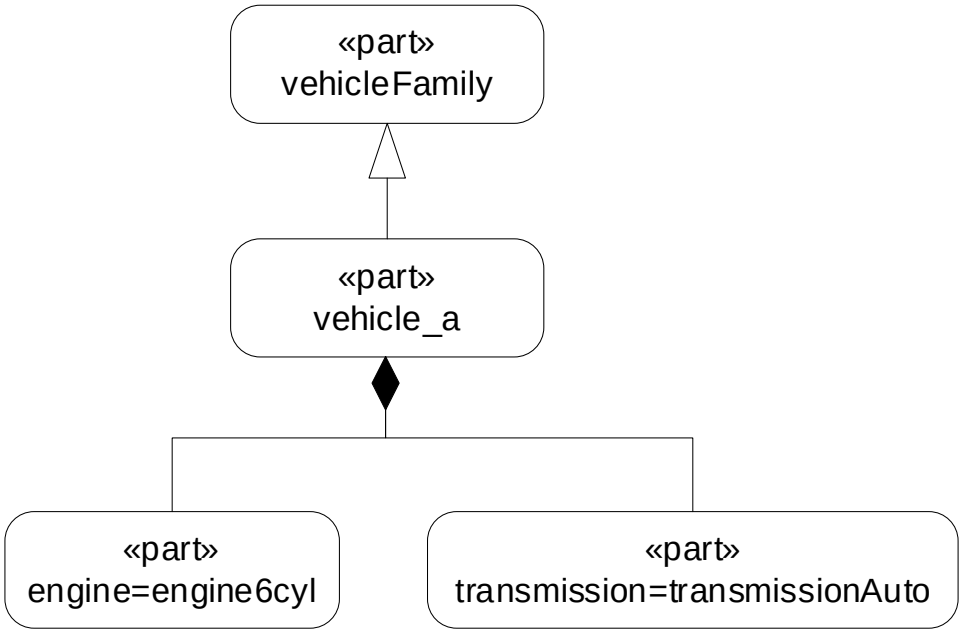
- Vehicle family is the superset model (150%)
- *Variation* points represent elements that can vary
  - *Can be* applied to all definition and usage elements
- A variant represents a particular choice at a variation point
- A choice at one variation point can constrain choices at other variation points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



Selected Vehicle Configuration

SST

- Vehicle_a subsets vehicleFamily to represent a particular design configuration
- Selected parts must satisfy variability constraints
  - Model is inconsistent if constraints are not satisfied
  - Variability modeling applications can automate the selection of valid configurations



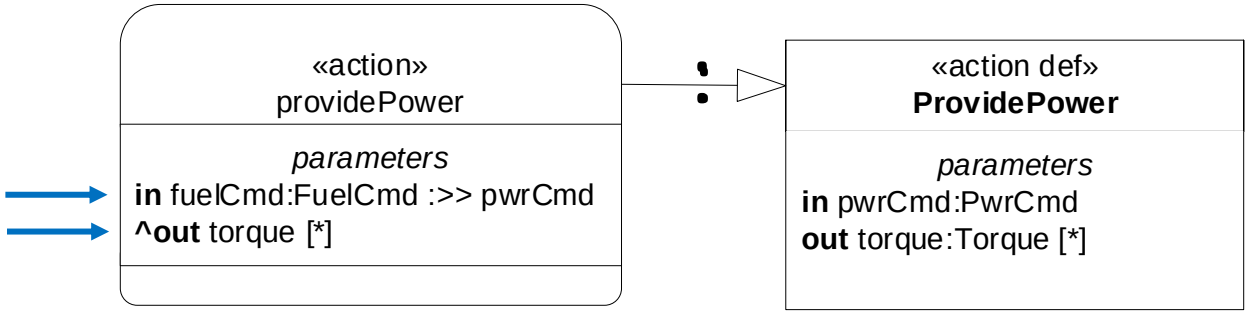
Behavior

Module 7
Actions

Actions

SST

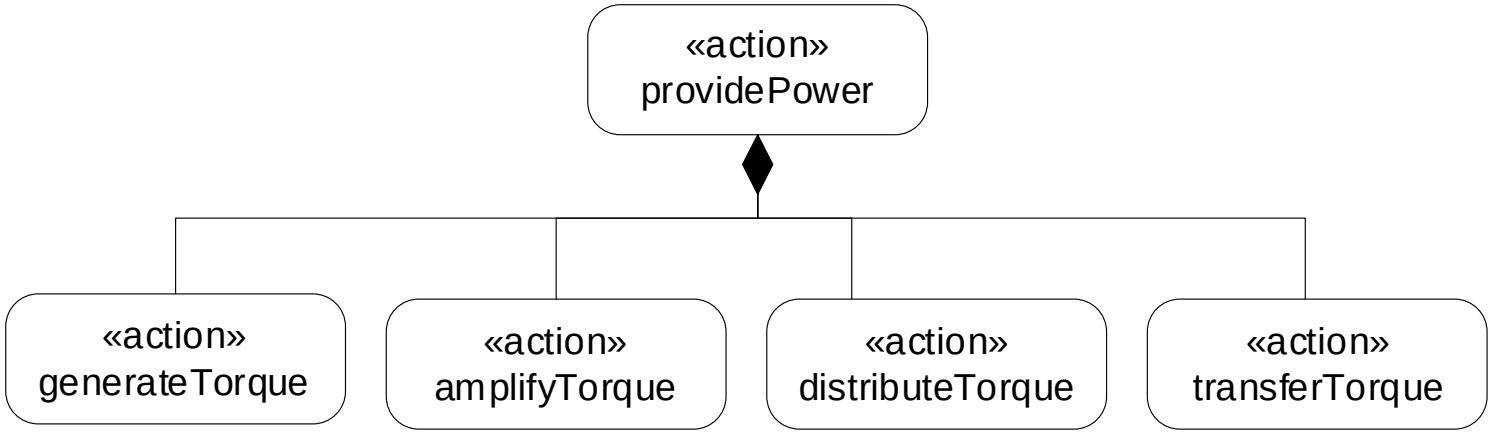
- Actions are defined by action definitions v2
  - Inherit features (e.g., input and output parameters)
  - Can redefine or subset inherited features or add new features



Action Decomposition

SST

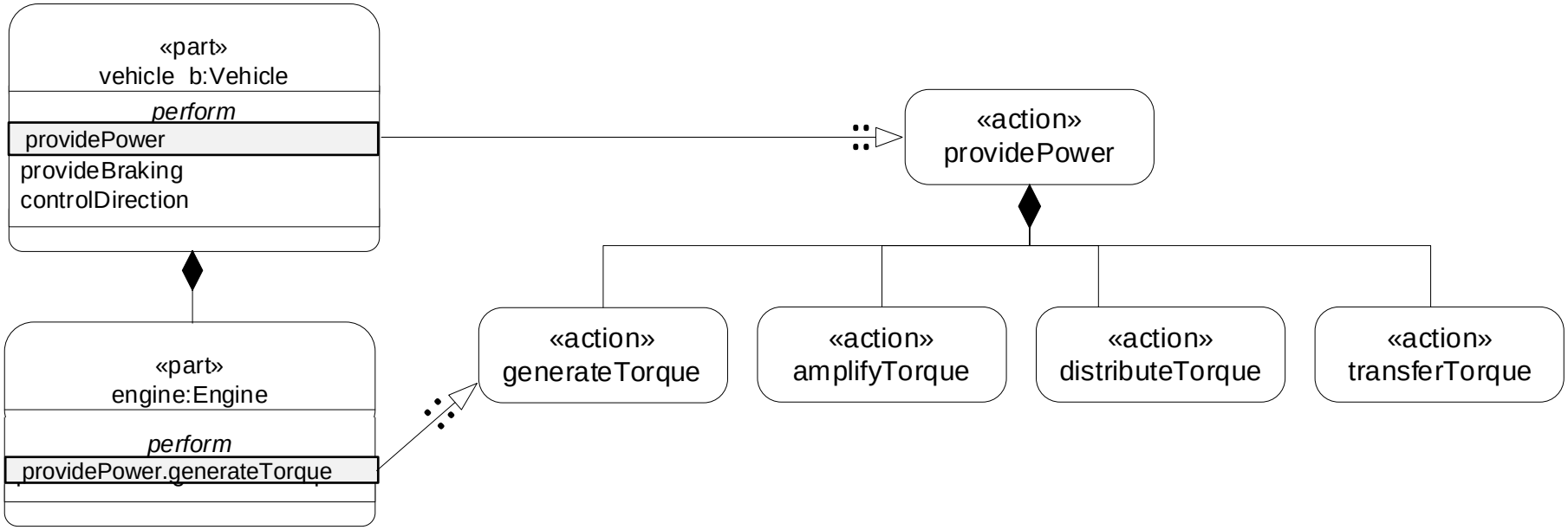
- Actions can be decomposed in a similar way as parts v2



Part Performs Actions

SST

- A part that performs an action can reference an action in an action tree
- This is done through reference subsetting



Perform Action

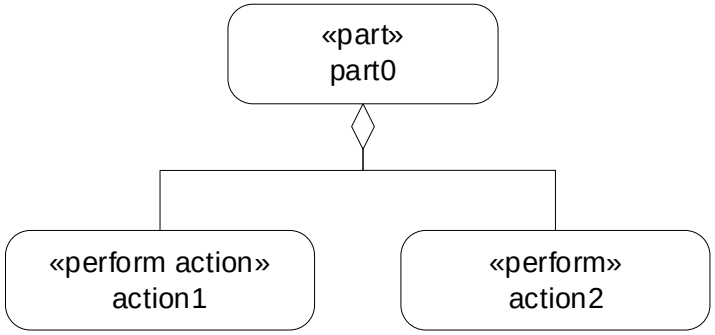
SST

- Show two kinds
  - **perform** action1
  - **perform action** action1
- The difference between these two is as follows
  - Line with arrow
    part part1 {**perform** action1} (*example from previous slide*)
    this creates an anonymous reference feature that subsets action1 which is often in a separate action tree. Name is properly qualified such as package1::action1). This is like a call action.
  - Line with white diamond
    part part1 {**perform action** action1}
    This creates an explicit reference feature called action1 that can either subset or bind to another action
  - This pattern is repeated for many different kinds of features.

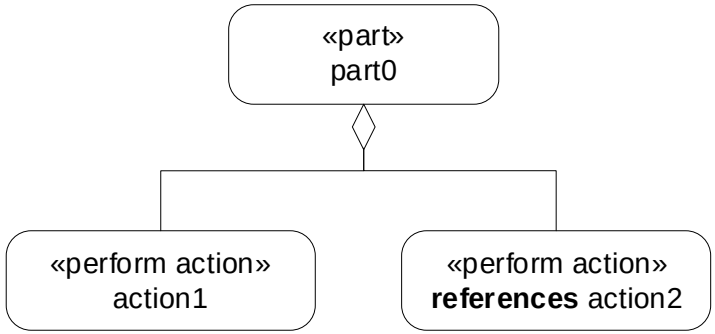
Perform Action
Simple Example

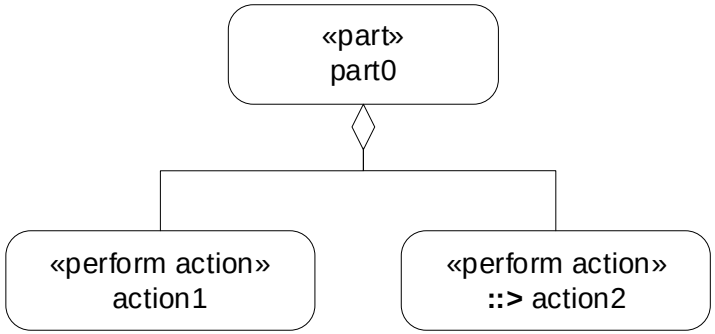
SST

```
package PerformActions{
   part part0{
      perform action action1;
      perform action2;
   }
   action action2;
}
```



Alternatives



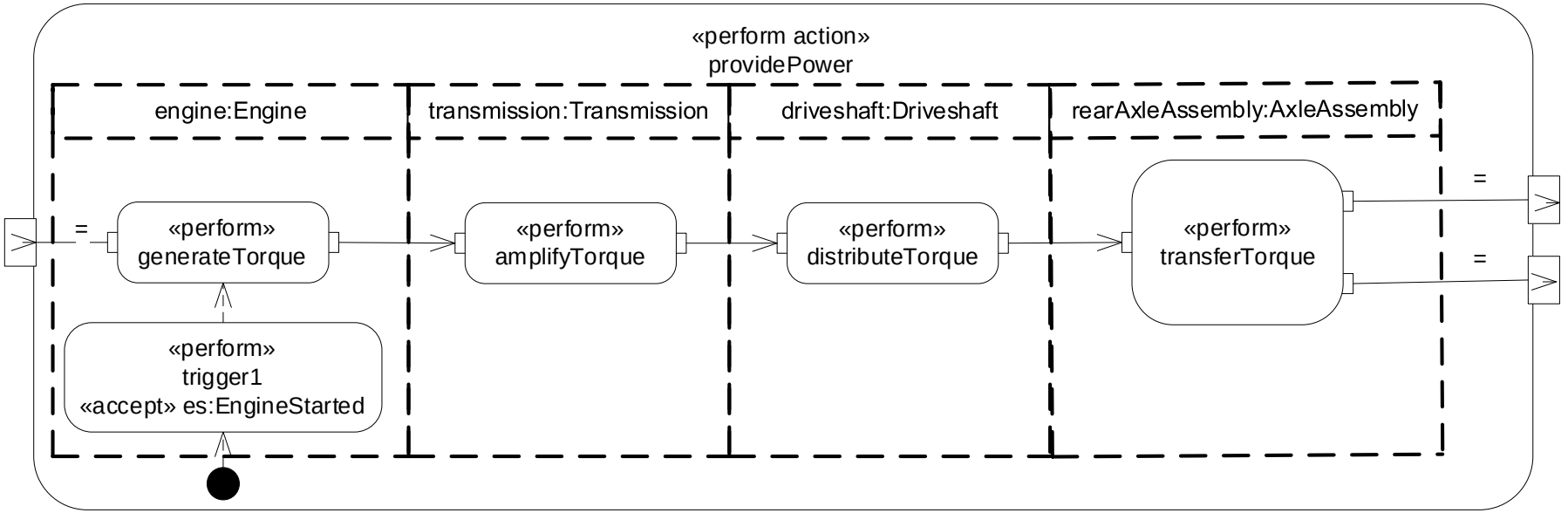


Module 8
Action Flow

Action Flow

SST

- Actions can include successions 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

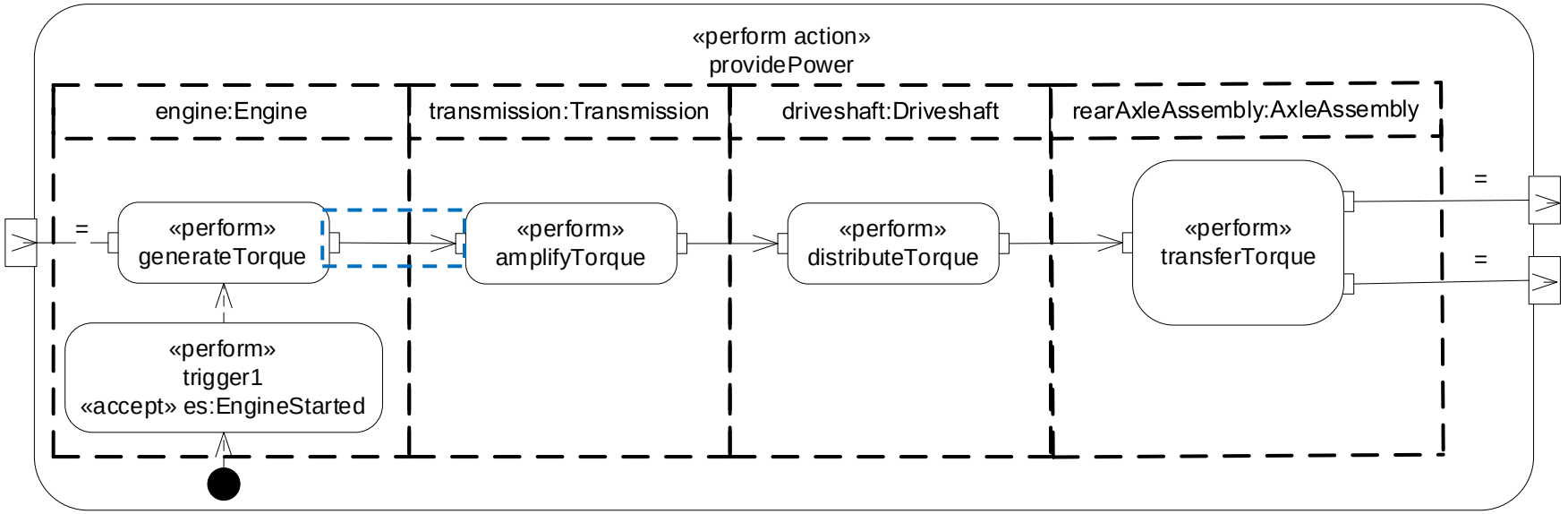


Note: Parameter notation subject to change

Input/Output Flows

SST

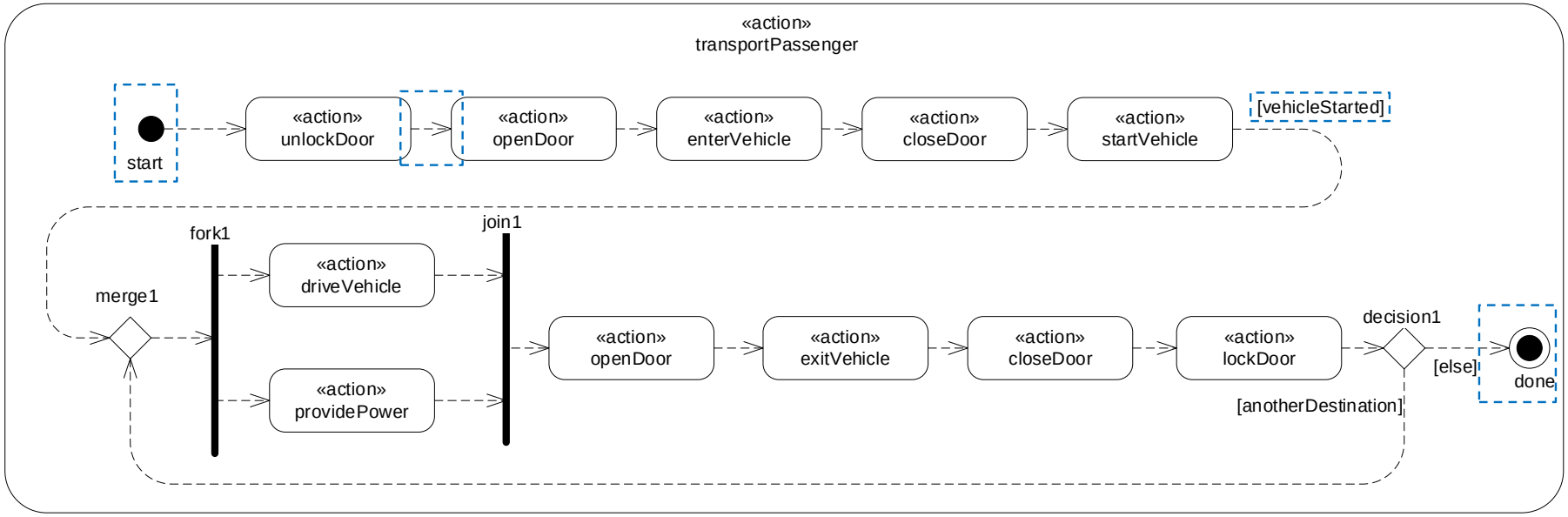
- A flow is a kind of connection that transfers items from a source output to a target input
  - A flow is a transfer that continues while the source and target actions execute (this is the default) v2
  - A succession flow is a transfer that only occurs after the source action ends and before the target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



Controlling Actions [1]

SST

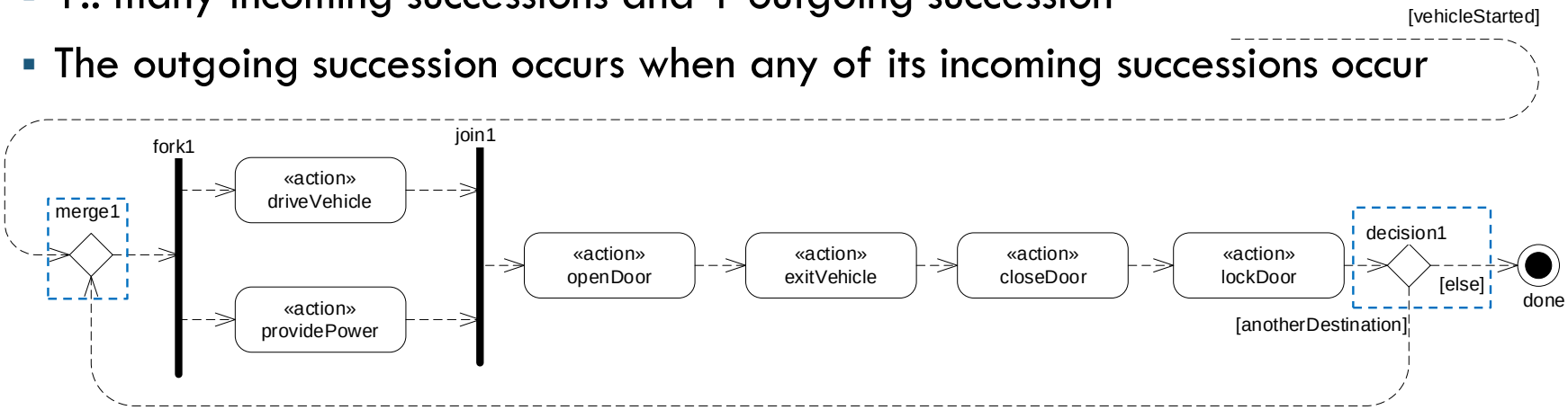
- A start action and done action represent the start and end of an action sequence
- A succession asserts that the target action can start execution only after the source action ends execution v2
- A conditional succession asserts that the target action can start only if the guard condition is true (if…then).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



Controlling Actions [2]

SST

- 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

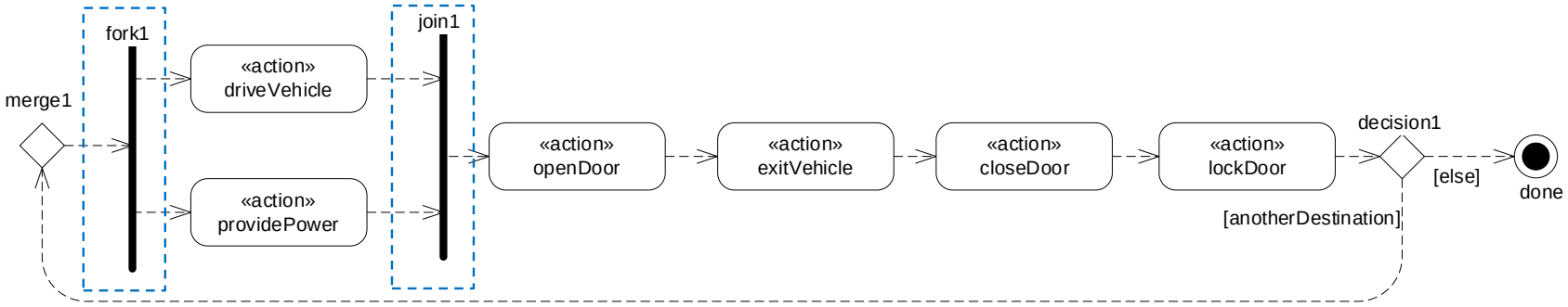


Controlling Actions [3]

SST

- 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

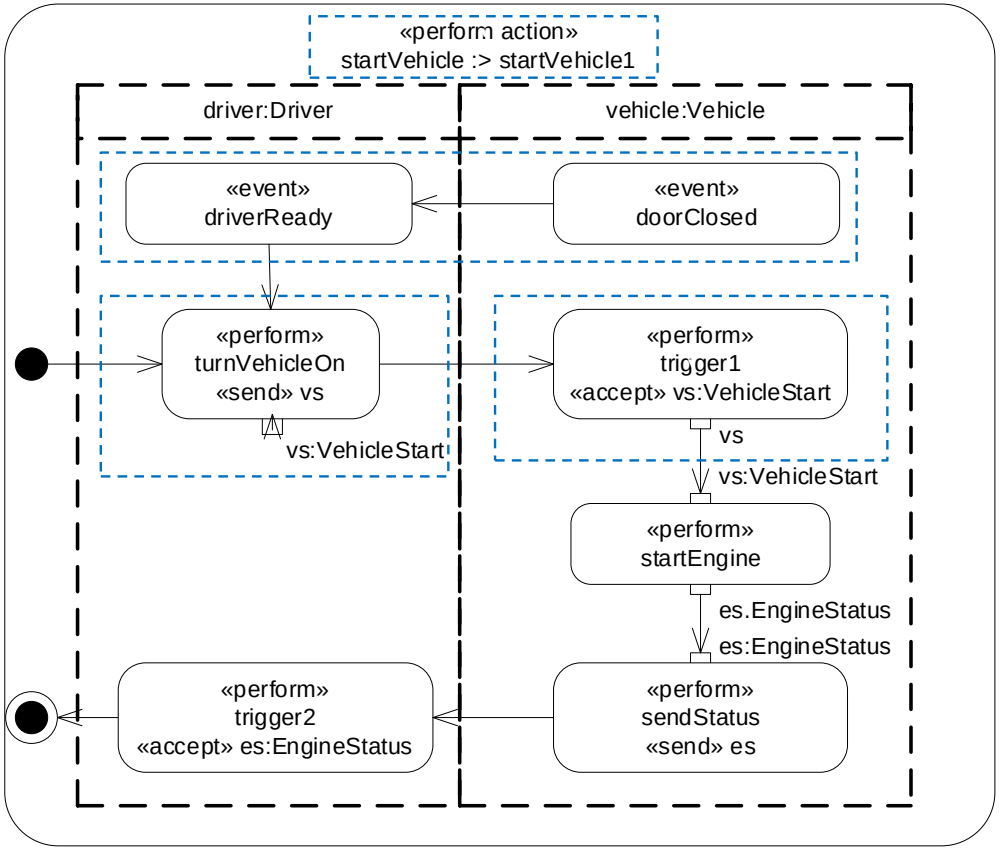
Note: Control nodes cannot currently be applied to input/output flows



Controlling Actions [4]

SST

- *startVehicle* subsets *startVehicle1* behavior which is independent of structure
- Event occurrences
- 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 (not shown)*



Scenario_10
Textual Notation

SST

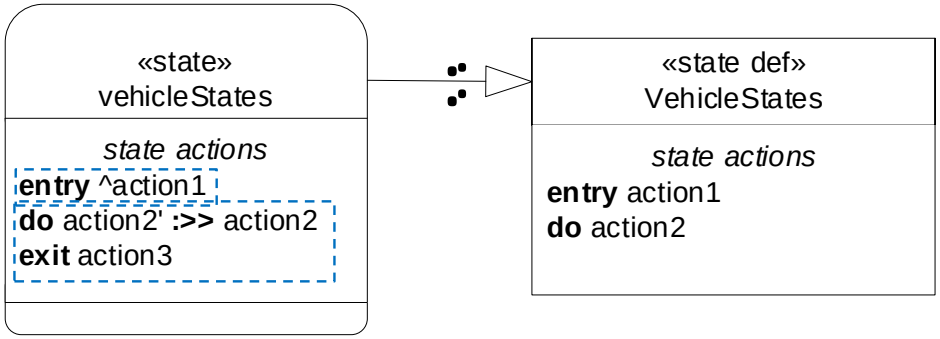
```
package Scenario_10{
        item def VehicleStart;
        item def EngineStatus;

        action startVehicle1{
          event occurrence doorClosed;
          event occurrence driverReady;
          first doorClosed then driverReady;
          first driverReady then turnVehicleOn;

          action turnVehicleOn send vs via source{
            in vs:VehicleStart;
            in source default self;
          }
          action trigger1 accept vs:VehicleStart;

          flow of VehicleStart from trigger1.vs to startEngine.vs;
          action startEngine{
            in item vs:VehicleStart;
            out item es:EngineStatus;
          }
          flow of EngineStatus from startEngine.es to sendStatus.es;

          action sendStatus send es via source{
            in es:EngineStatus;
            in source default self;
          }
          action trigger2 accept es:EngineStatus;

          message of VehicleStart from turnVehicleOn to trigger1;
          message of es:EngineStatus from sendStatus to trigger2;
        }
        part def Driver{
          port p1;
          port p2;
        }
        part def Vehicle{
          port p1;
        }

        part parto{
          perform action startVehicle:>startVehicle1{
            action :>> turnVehicleOn send vs via source{
              in :>> source = driver.p1;
            }
            action :>> trigger1 accept vs:VehicleStart via vehicle.p1;

            action :>> sendStatus send es via source{
              in es:EngineStatus;
              in :>> source = vehicle.p1;
            }
            action :>> trigger2 accept es:EngineStatus via vehicle.p1;

          }
          part driver : Driver {
            perform startVehicle.turnVehicleOn;
            perform startVehicle.trigger2;
            event startVehicle.driverReady;
          }
          part vehicle : Vehicle {
            perform startVehicle.trigger1;
            perform startVehicle.startEngine;
            perform startVehicle.sendStatus;
            event startVehicle.doorClosed;
          }
          interface driver.p1 to vehicle.p1;
          interface driver.p2 to vehicle.p1;
        }
}
```

Module 9
States

States

SST

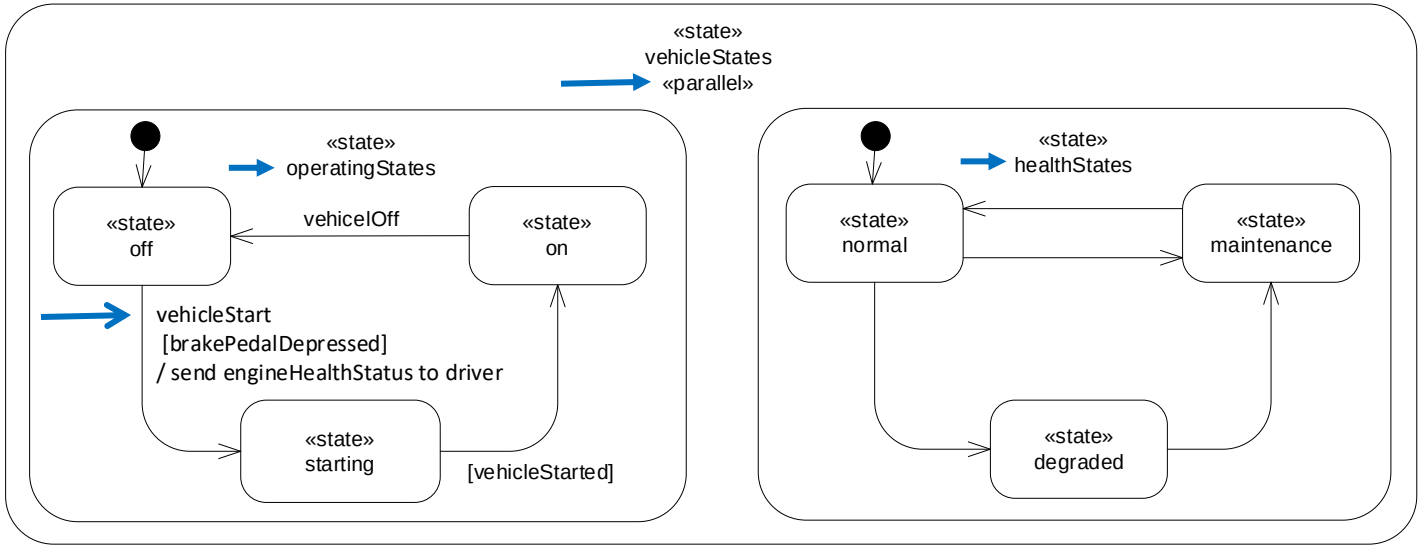
- States can be defined by state definitions v2
  - Inherit features (e.g., actions, nested states, transitions) from its state definition
  - Can redefine or subset inherited features or add new features



State-based Behavior [1]

SST

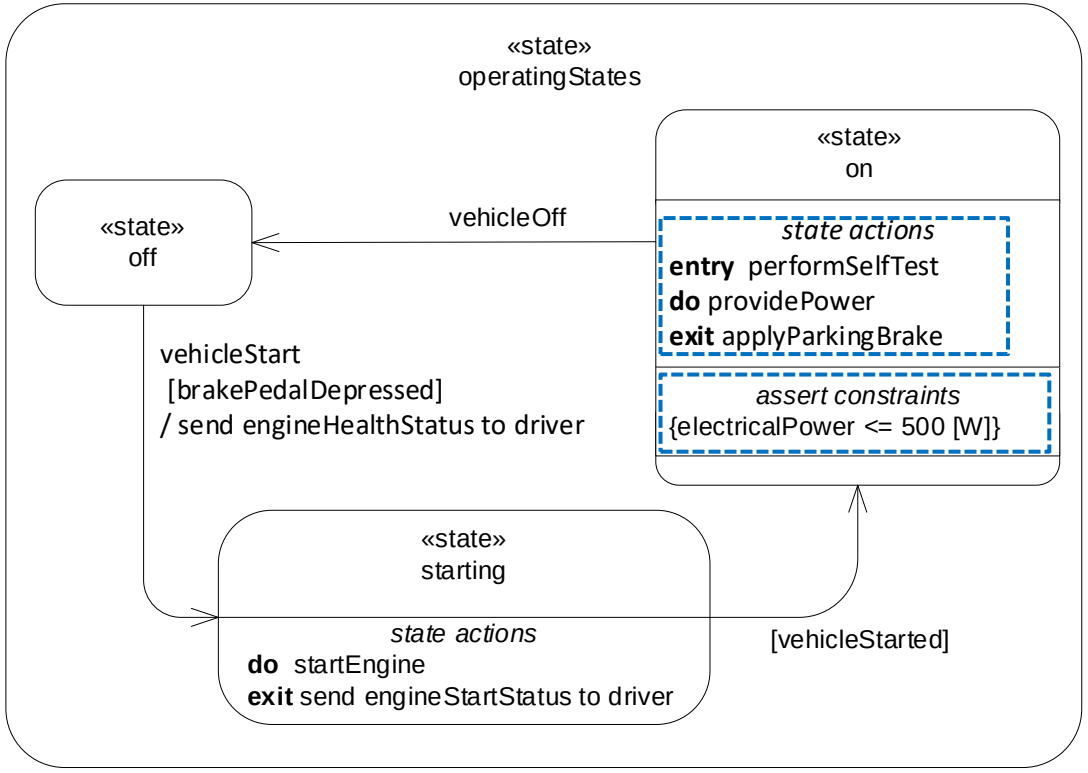
- States decompose into nested states (*without regions, which were in SysML v1*) v2
  - A parallel state decomposes into concurrent states
  - A non-parallel state (default) decomposes into sequential/exclusive states
- Transitions between sequential states
  - Triggered by events
  - Can include guard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



State-based Behavior [2]

SST

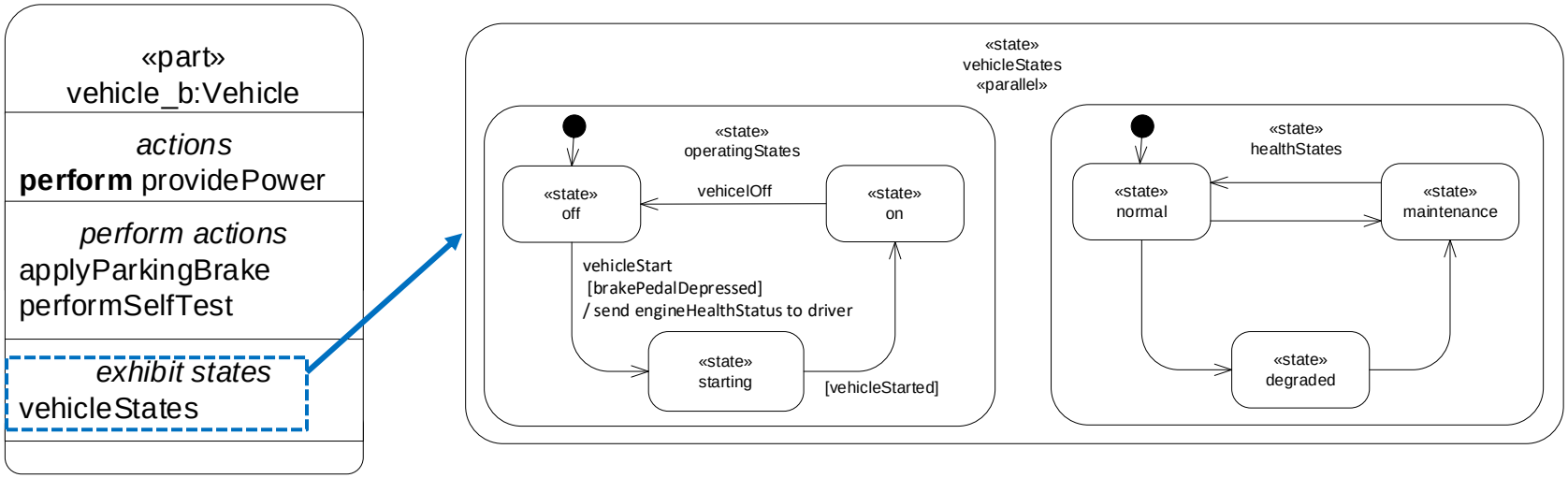
- States can have entry, exit, and do actions that can refer to actions in action trees or be defined locally
- States can include constraints



Part Exhibits States

SST

- A part exhibits states v2
- Owns its state-based behavior

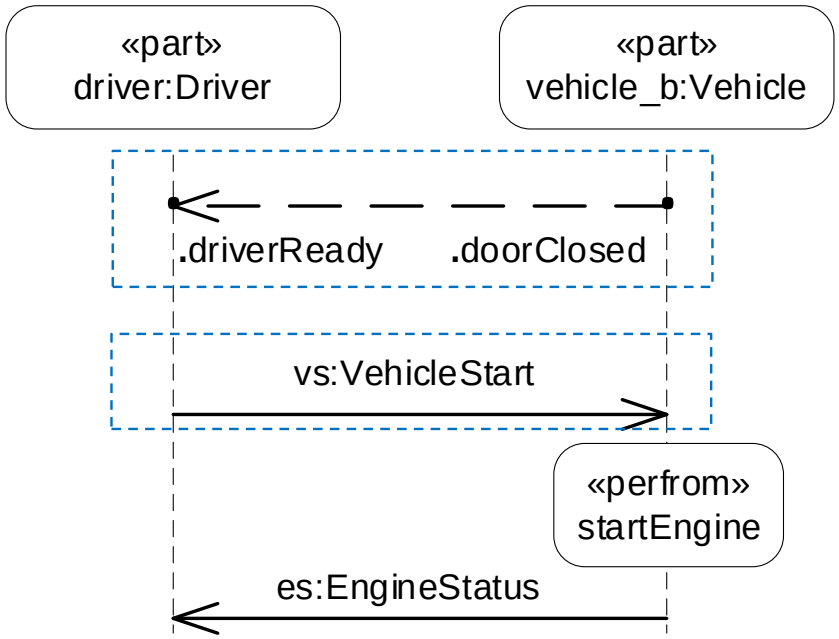


Module 10
Interactions

Sequence View

SST

- Represents an event sequence and messages (refer to action flow on slide 62)
  - Event sequence
  - Message
  - Driver and vehicle are in a shared context (not shown)

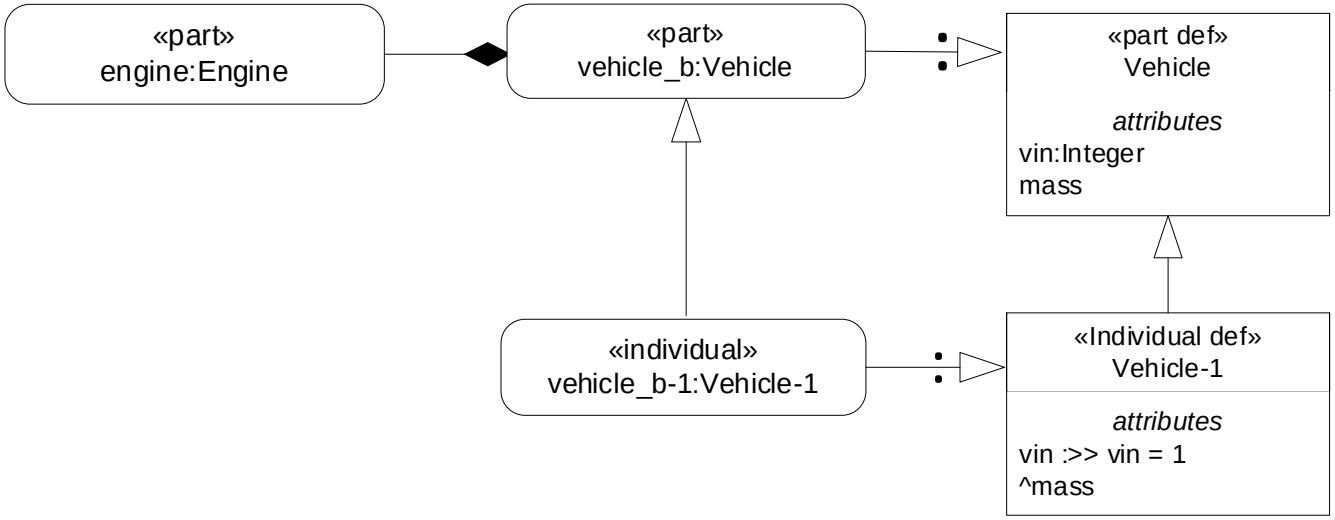


Module 11
Individuals

Individuals v2

SST

- 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 & Snaphots

4D Model & Occurrences

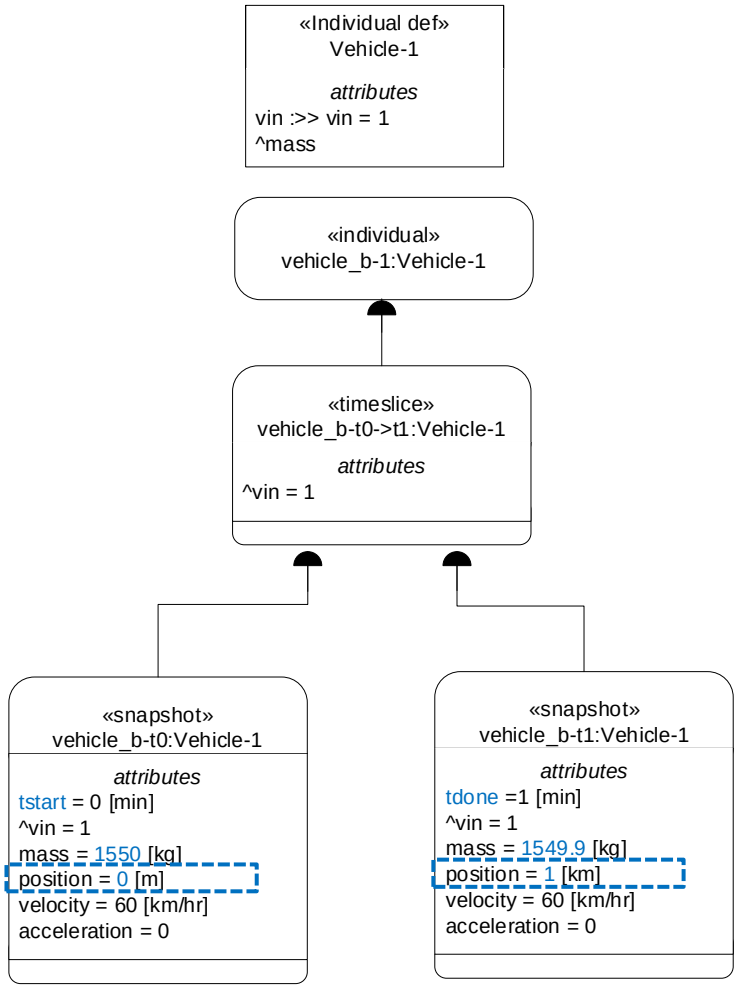
SST

- Each entity called an occurrence has a lifetime
  - Distinct from attributes which do not have a lifetime
  - Includes a reference clock that can quantify time (defaults to universal clock)
  - Can specify time slices and snapshots
- Spatial items are kinds of occurrences that have spatial extent that can change over their lifetime
  - Specified by shapes with position and orientation within coordinate frames
- Individuals
  - Unique occurrence with a lifetime

Lifetime with Snapshots and Time Slices v2

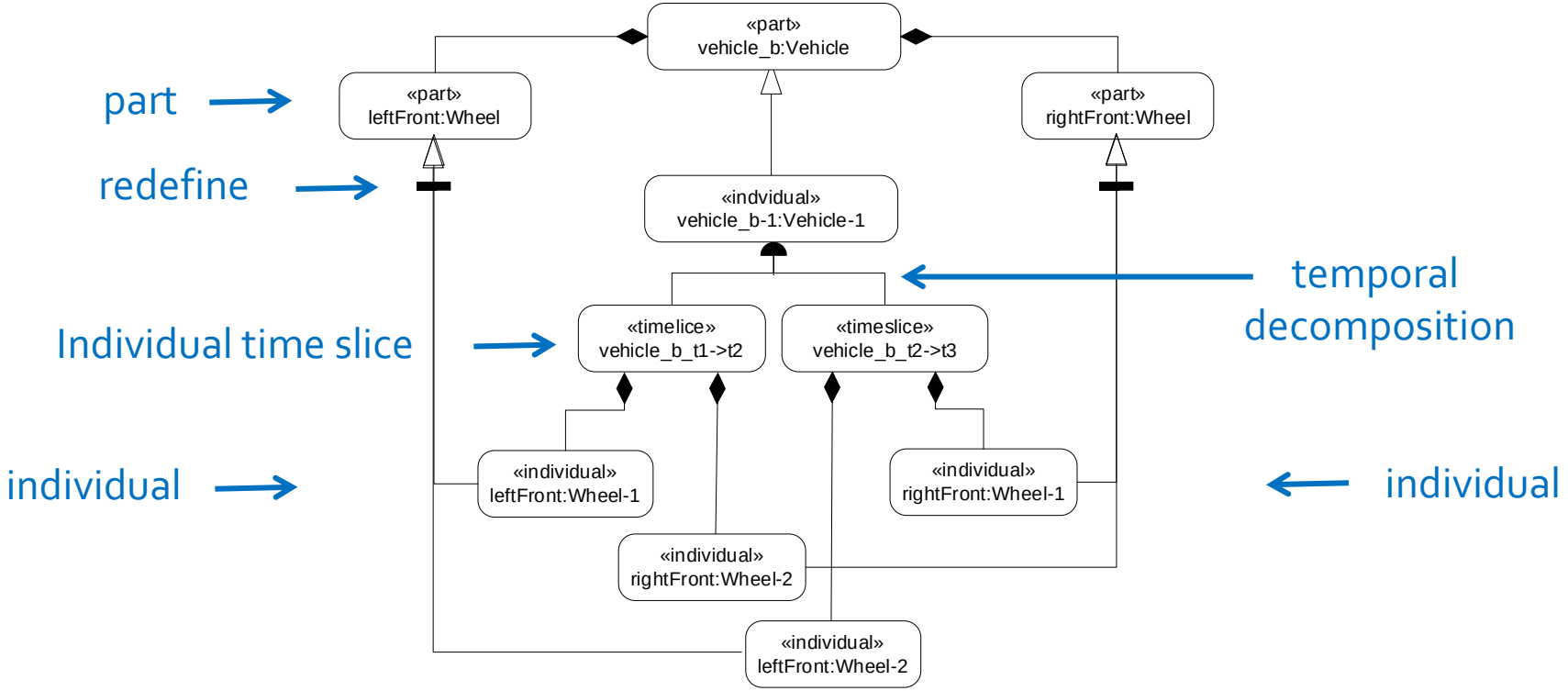
SST

- 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 time slices (i.e., durations of time)
- A time slice 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 time slice 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 time slices
  - Individual wheel (Wheel-1) is left front wheel during vehicle_b_t1-t2 time slice
  - Individual wheel (Wheel-1) is right front wheel during vehicle_b_t2-t3 time slice



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
- Default value means value can be over-ridden using redefinition

«part»
vehicle_b:Vehicle

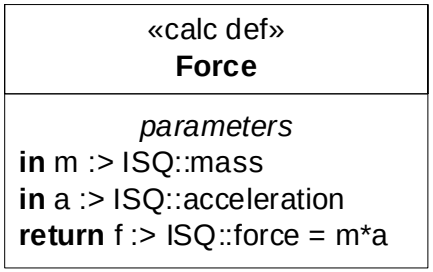
attributes

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)

Calculations v2

SST

- A kind of behavior generally used to represent reusable mathematical functions
  - Inputs
  - Expression
  - Returns a single result
    - Can have multiplicity



- Represent a calculation definition using the textual notation

```
calc def Force {
    in attribute m :>ISQ::mass;
    in attribute a :>ISQ::acceleration;
    return f  :>ISQ::force = m * a;
}
```

Calculations v2

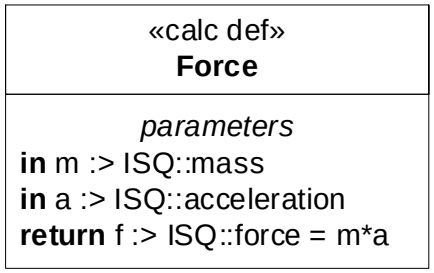
SST

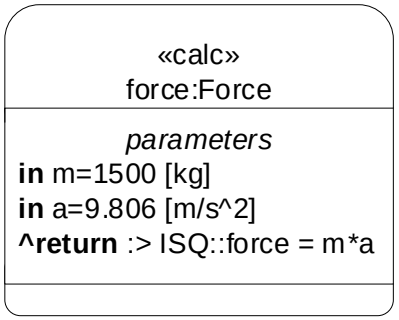
- Evaluate usage
  - Bind values to input parameters. Then evaluate expression and return a result
  - **calc** force : Force {

    in m=1500 [kg];

    in a=9.806 [m/s^2];

    }
- Can bind a calc return value to an attribute
  - **attribute** f1 = Force (m=1500 [kg] , a = 9.806 [m/s ^2]);





Binding Connection

SST

- 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 inconsistent (e.g., 3=4) is inconsistent
  - 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 can be used to establish equality of any kind of feature
  - A part can be bound (e.g., engine = 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 (e.g., kilograms conform to MassValue)
- The values are associated with the units v2
- A change in a unit can apply the unit conversion factor if a tool supports this
  - **attribute** m:MassValue = 25 [kg] (an equivalent value 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
  - distancePerVolume :> scalarQuantities = distance / volume v2
- Libraries
  - International System of Quantities (ISQ)
  - International System of Units (SI)
  - US Customary Units (USCustomaryUnits) v2

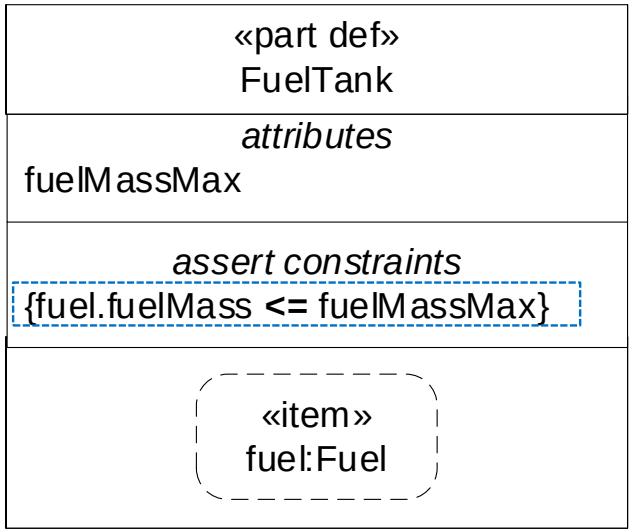
Module 15
Constraints

Constraint Expression [1]

SST

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

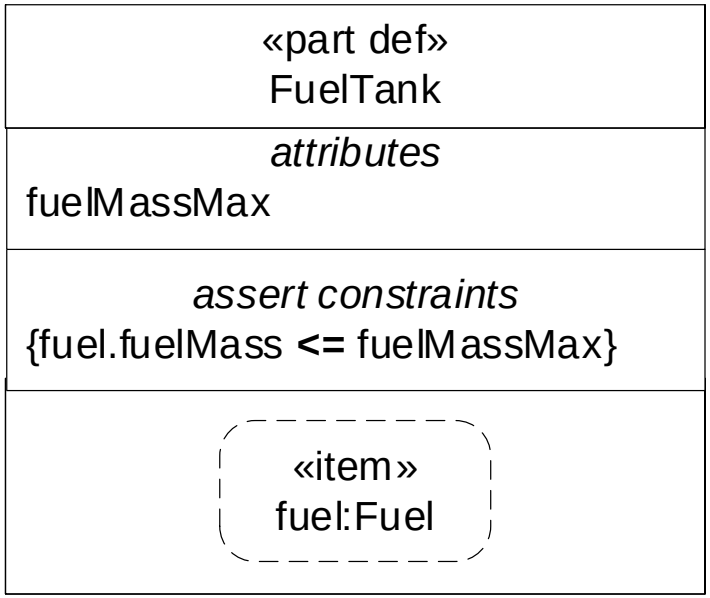
```
part def FuelTank {
        attribute mass :> ISQ::mass;
        ref item fuel:Fuel{
          attribute redefines fuelMass;
        }
        attribute fuelMassMax:>ISQ::mass;
        assert constraint {fuel.fuelMass<=fuelMassMax}
}
```

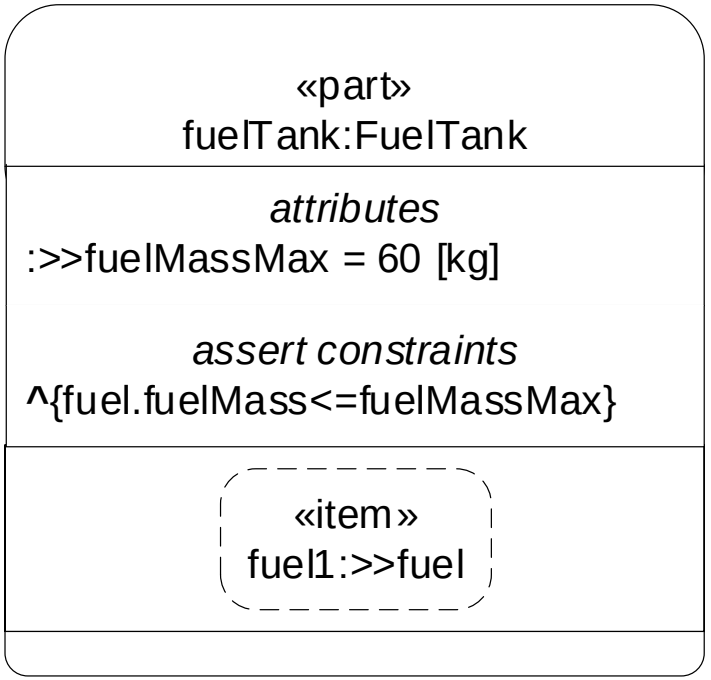


Constraint Expression [2]

SST

- Usage inherits constraint expression and redefines constraint parameters

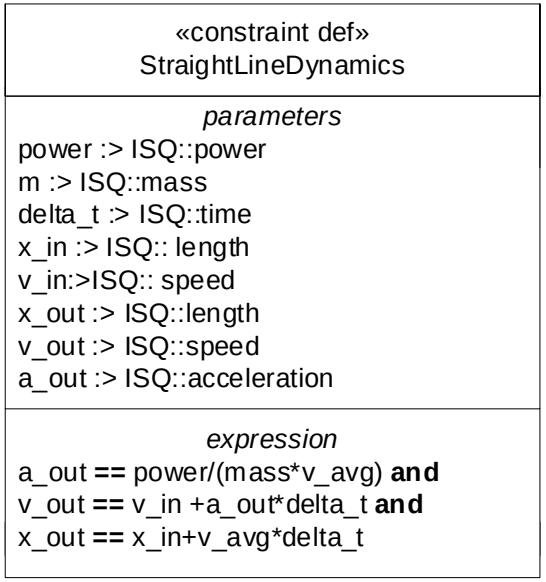


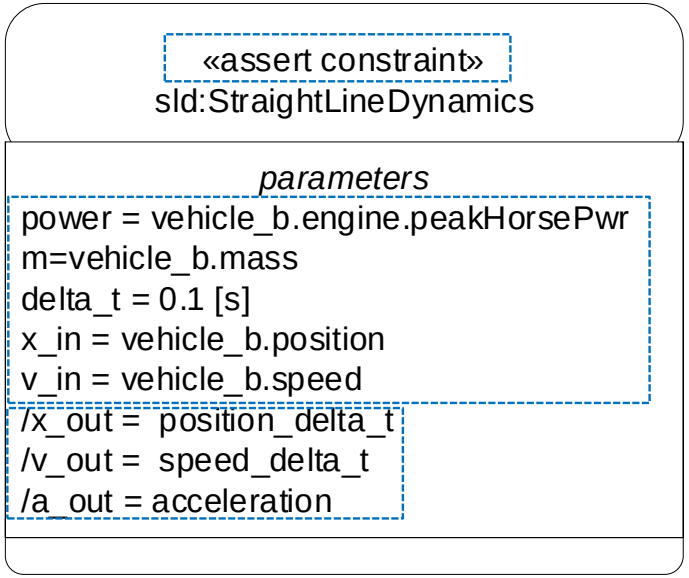


Constraint Definition

SST

- Enables reuse of a constraint expression
- Default parameter direction is 'in'
- Can assert a constraint on the usage
- Bind parameters on the usage
- Can mark dependent parameters as derived with slash

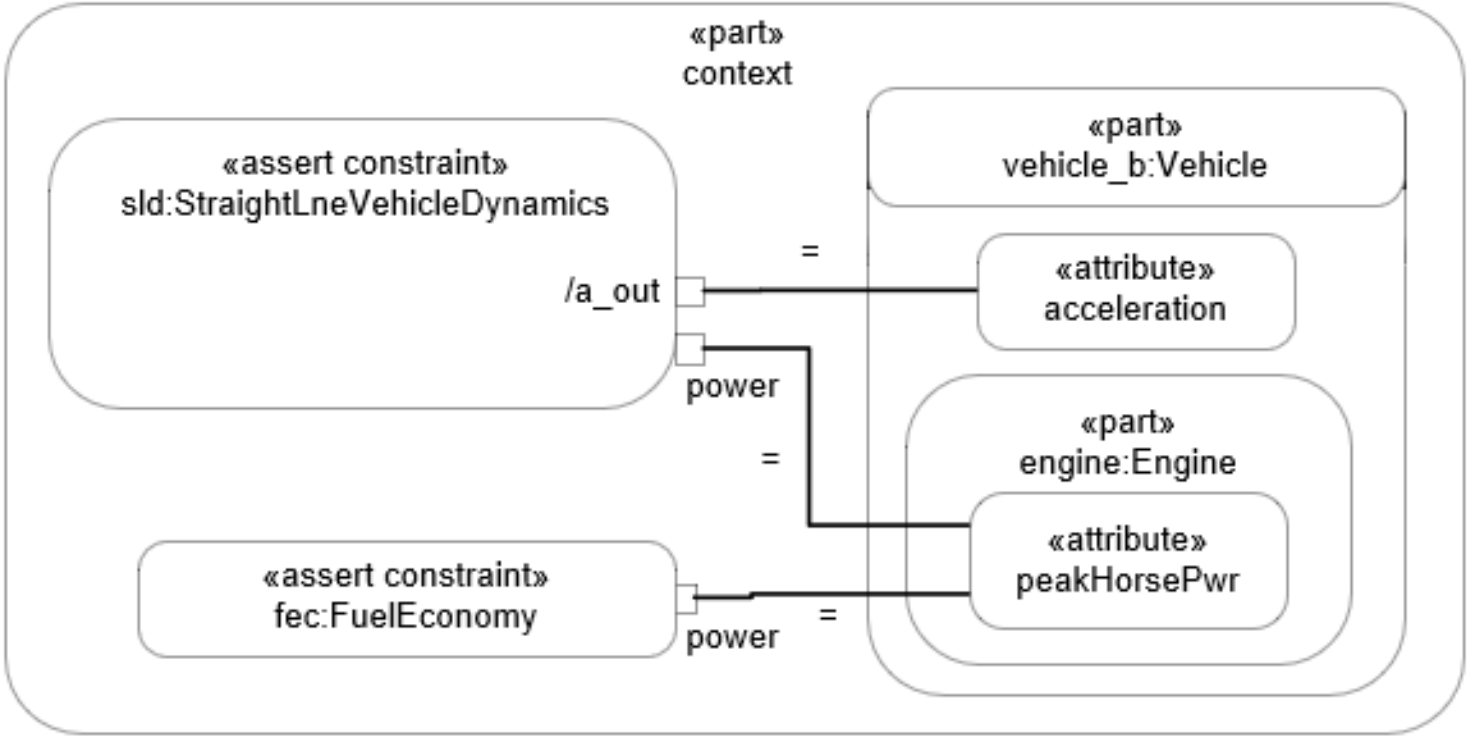




Binding Constraints

SST

- Equivalent to SysML v1 parametric diagram

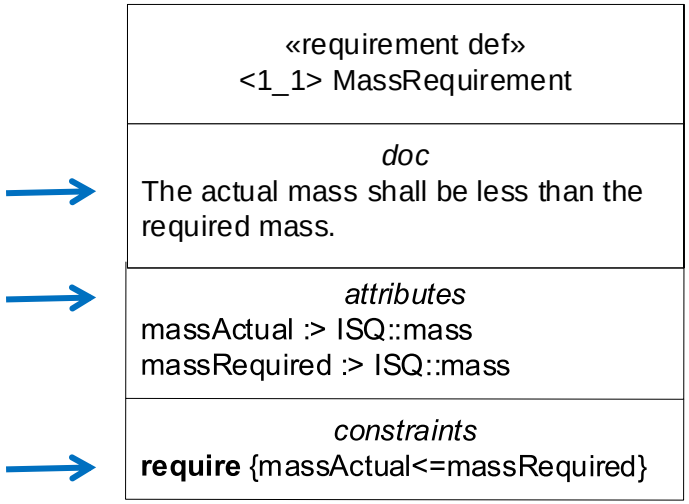


Module 16
Requirements

Requirement Definition

SST

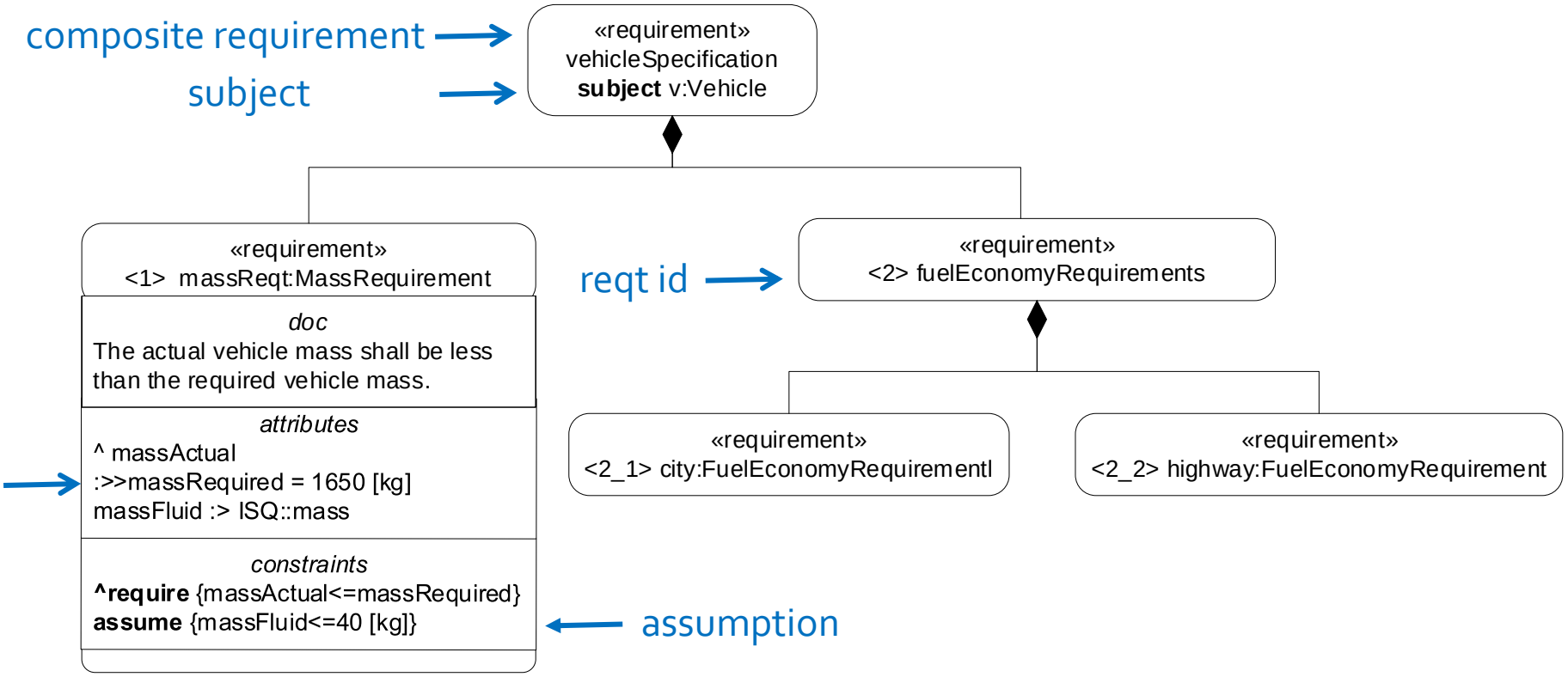
- A constraint definition that a valid design solution must satisfy that can include: v2
  - Identifier
  - Shall statement
  - Constraint expression that can be evaluated to true or false
    - can apply to performance, functional, interface and other kinds of requirements if desired
  - Assumed constraint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



Requirement Specification

SST

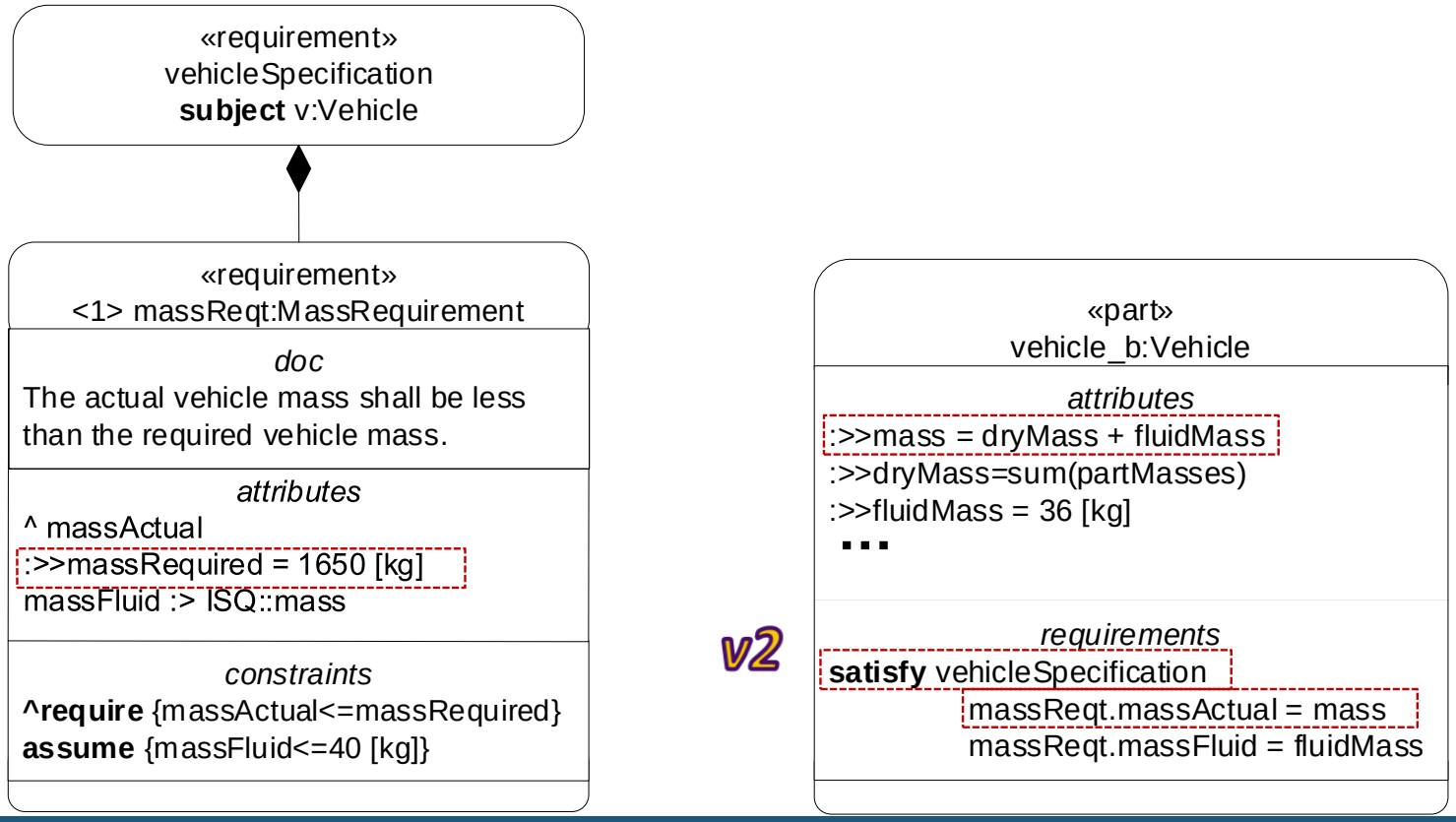
- A tree of requirements that contains nested requirement usages v2
  - Composite requirement can own or reference other requirements
  - Subject of nested requirements is the kind of entity being specified
  - Requirement features can redefine features of the requirement definition



Satisfying a Requirement

SST

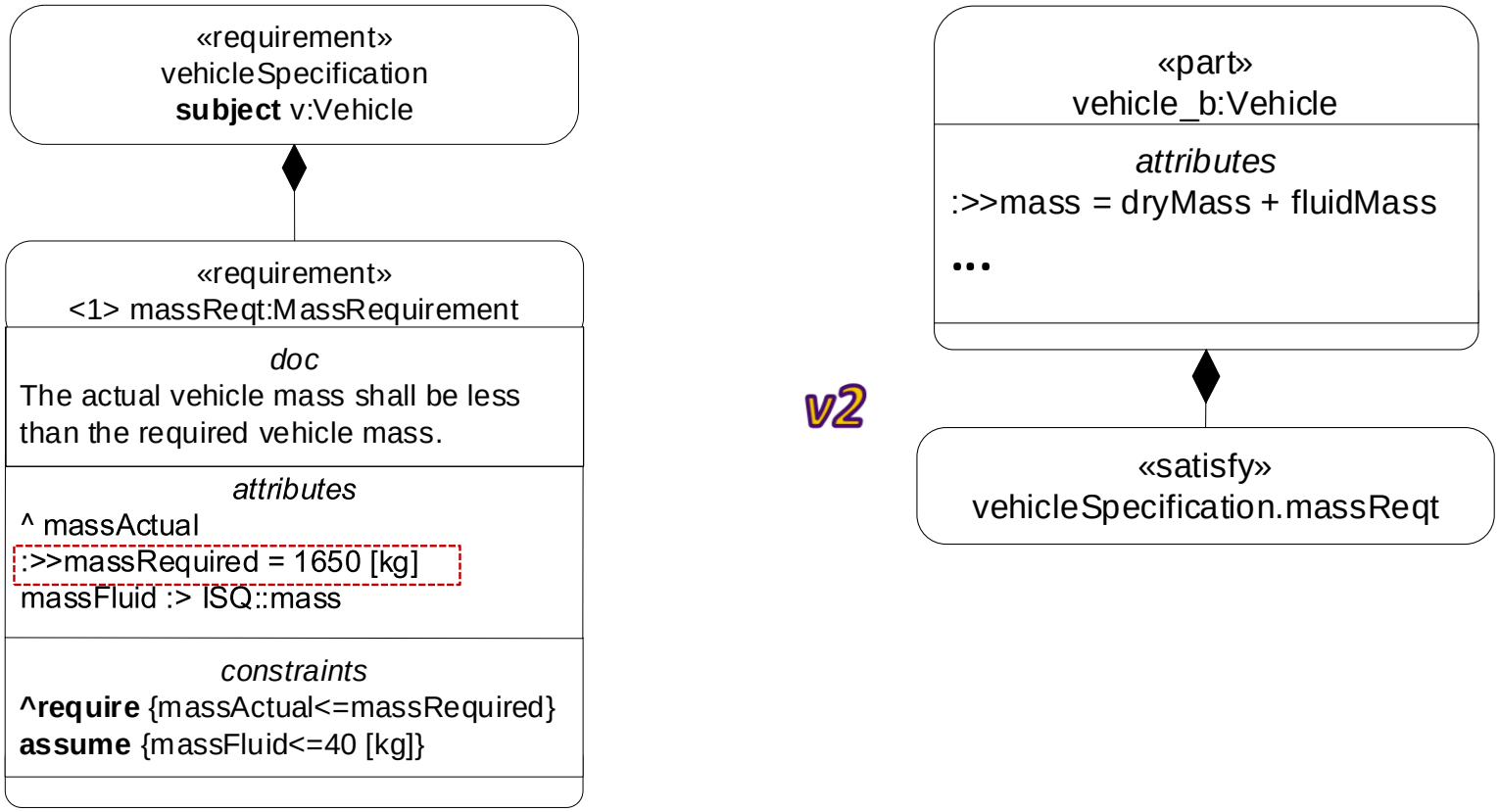
- The vehicleSpecification imposes constraints on the vehicle
  - The modeler can assert the vehicle mass satisfies the massRequired
    - Bind the vehicle mass to massActual of the requirement



Satisfying a Requirement Alternative Notation

SST

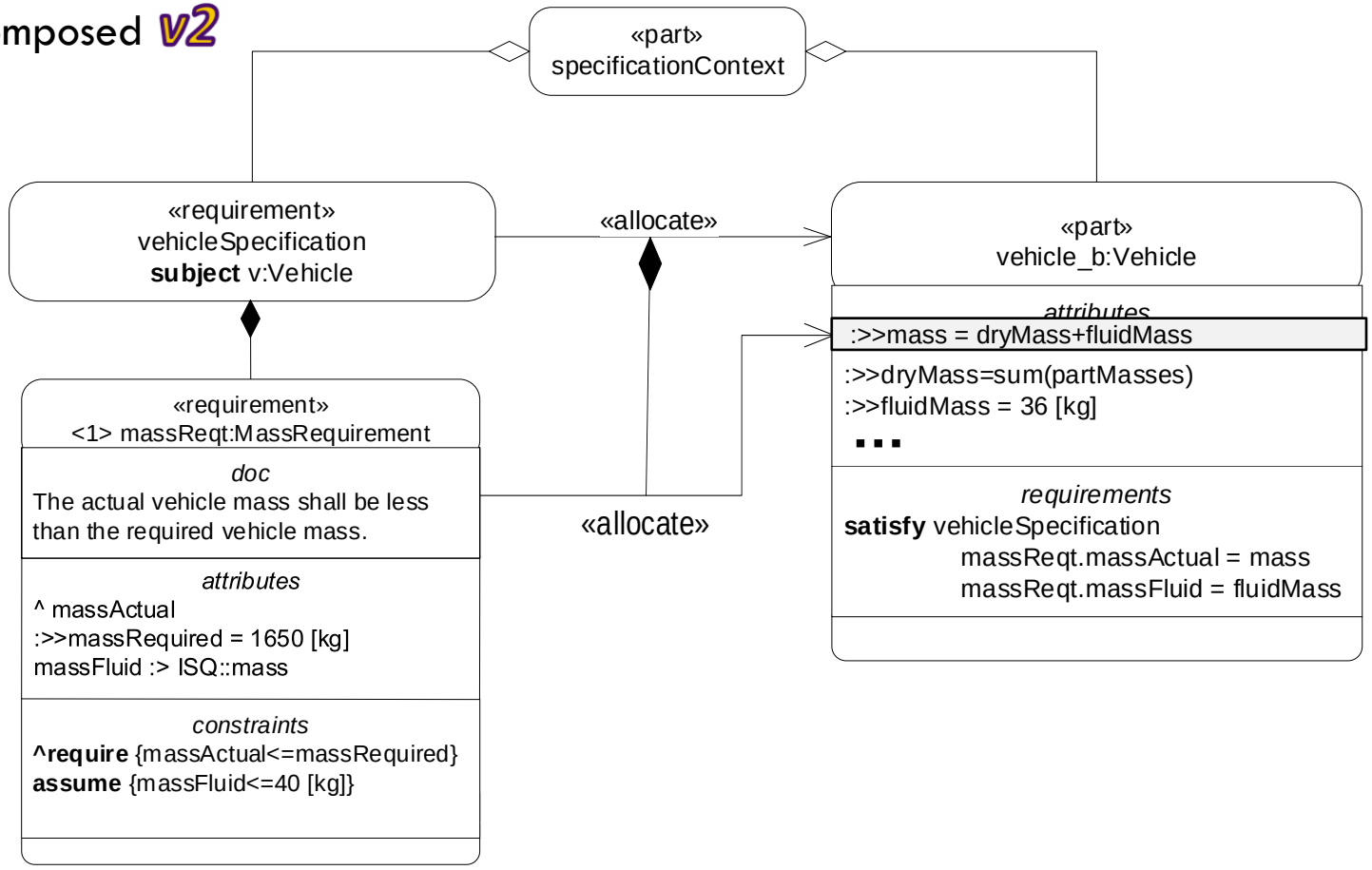
- Alternative notation



Requirement Allocation vs. Requirement Satisfaction

SST

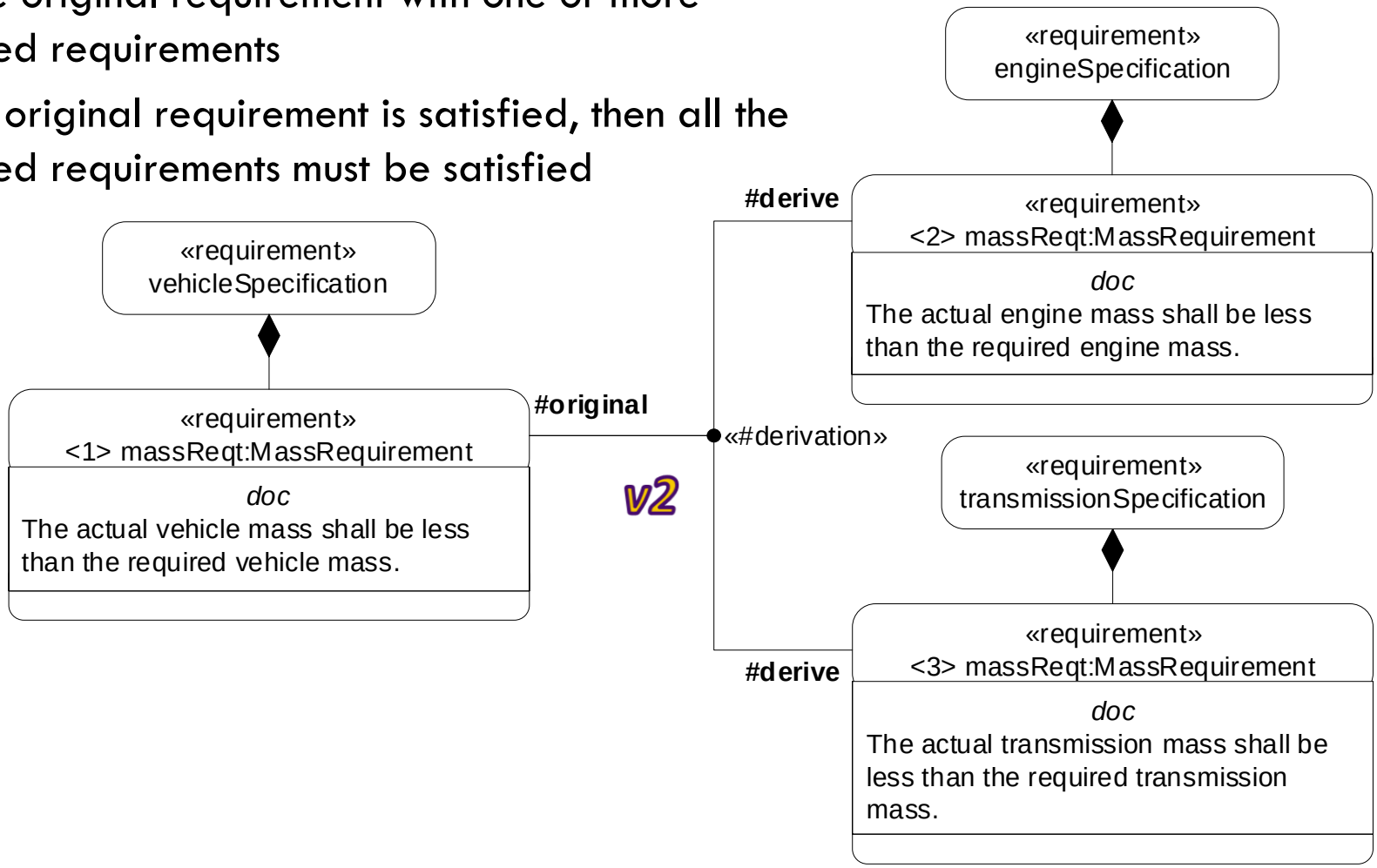
- Allocating a requirement assigns responsibility for realizing a requirement
- Less formal than requirement satisfaction (no explicit constraint assertion)
- Can be decomposed v2



Derived Requirement

SST

- Single original requirement with one or more derived requirements
- If the original requirement is satisfied, then all the derived requirements must be satisfied



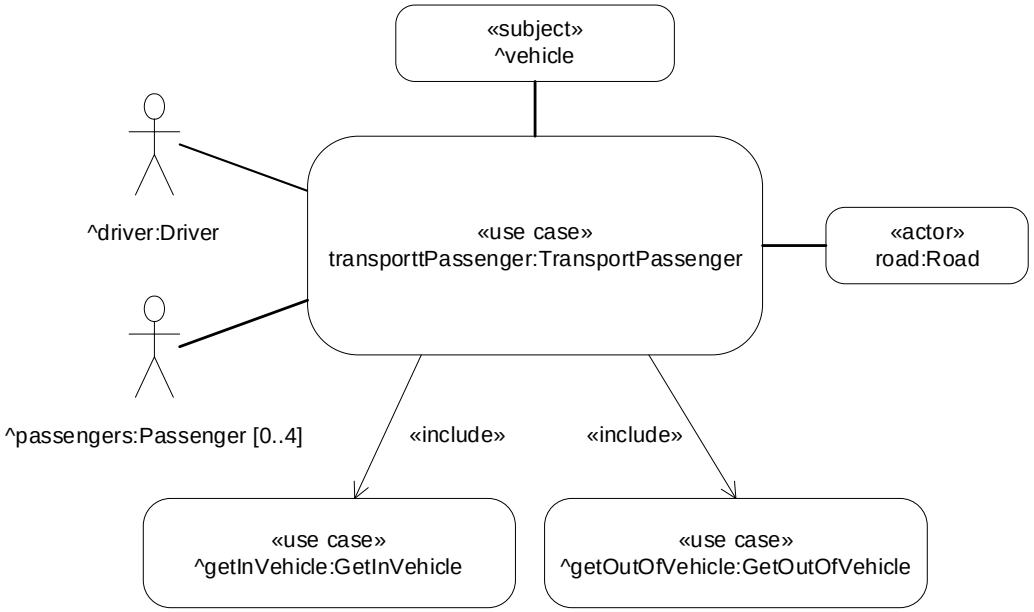
Module 17
Use Cases

Use Case

SST

- A kind of case
  - Subject of the use case (often the system of interest)
  - Objective is to provide value to one or more actor
    - Stakeholders are typically external to the system of interest referred to as actors
  - Required actions/steps performed by actors and subject to achieve the objective
  - Include use cases are always performed as part of the base use case
  - Constraints can specify pre and post conditions

«use case def» TransportPassenger
objective **doc** Deiver passenger to destination safely, comfortably, and within accecptable time
subject vehicle:Vehicle
actors driver:Driver passengers:Passenger [0..4]
include use cases getInVehicle:GetInVehicle *getOutOfVehicle:GetOutOfVehicle*
constraints start.vehicle.position==startingLocation done.vehicle.position==endingLocation

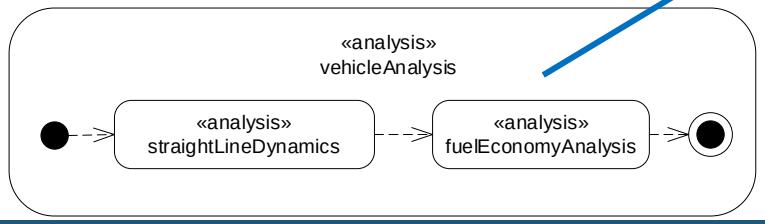


Module 18
Analysis Cases

Analysis Case v2

SST

- 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 parameters to the subject



«analysis»
fuelEconomyAnalysis

subject

v=vehicle_b

objective

doc Estimate the vehicle fuel economy

parameters

in scenario:Scenario
return calculatedFuelEconomy :> km/l

attributes

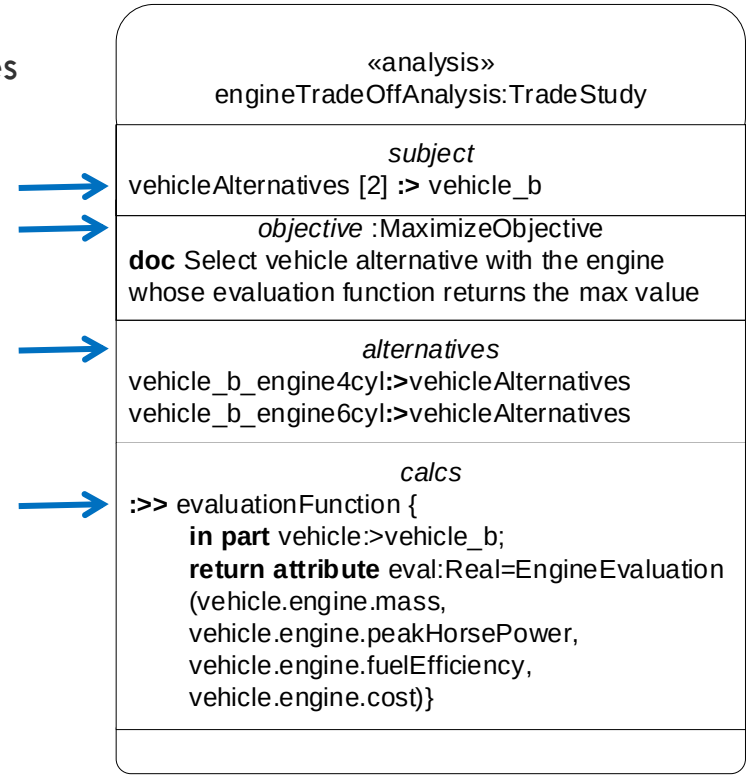
distance = TraveledDistance(scenario)
tpd_avg = AverageTravelTimePerDistance(scenario)
bsfc = ComputeBSFC(vehicle_b.engine)
f_a = BestFuelConsumptionPerDistance(vehicle_b.mass, bsfc, tpd_avg, distance)
f_b = IdlingFuelConsumptionPerTime(vehicle_b.engine)
calculatedFuelEconomy=FuelConsumption(f_a, f_b, tpd_avg)

convert to liters / 100 km

Trade-off Analysis v2

SST

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

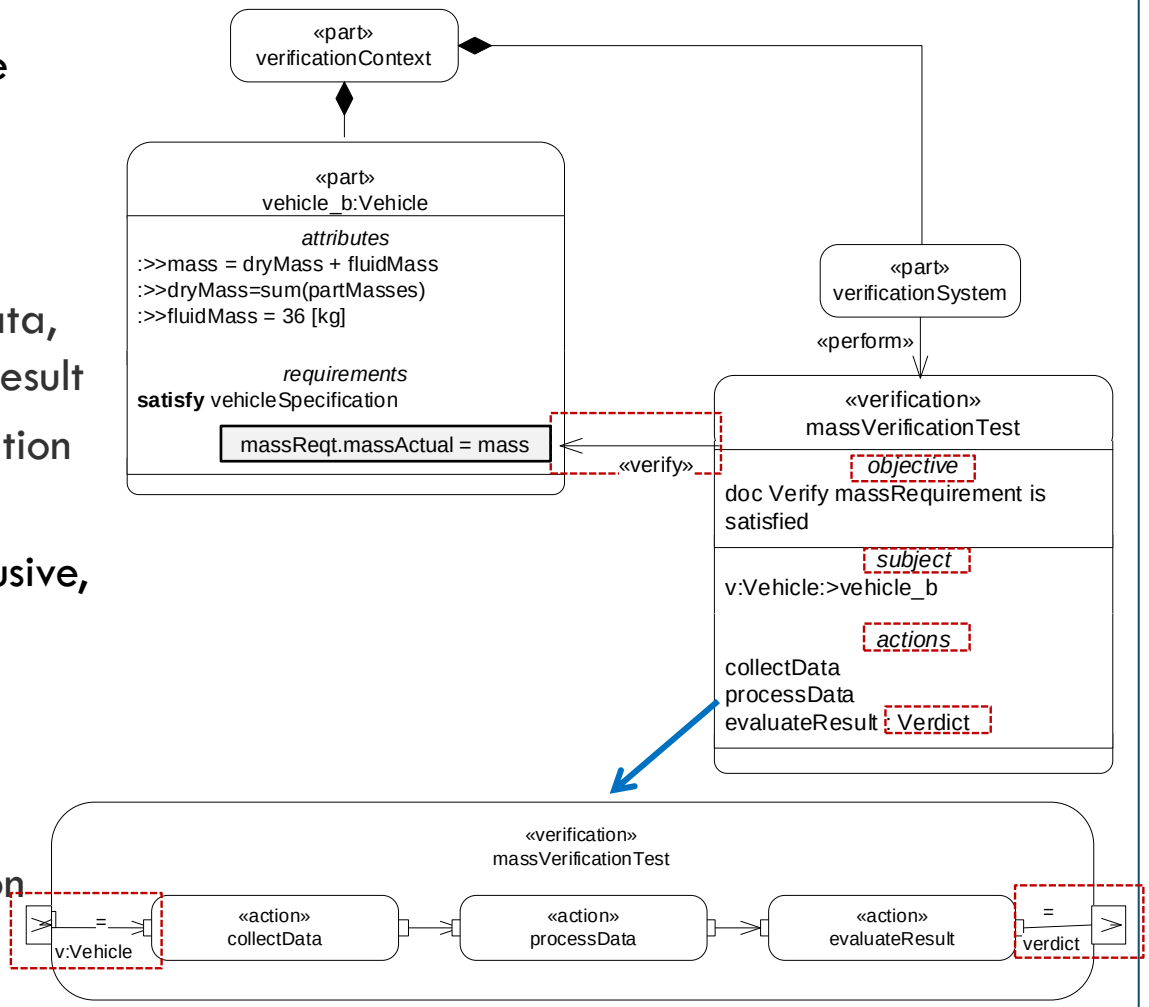


Module 19
Verification Cases

Verification Case

SST

- 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
  - Subject is an input to the verification
  - Returns a result called a verdict whose value is pass, fail, inconclusive, or error
  - Verify relationship relates verification case to requirement
  - Verification system interacts with subject to perform the verification case

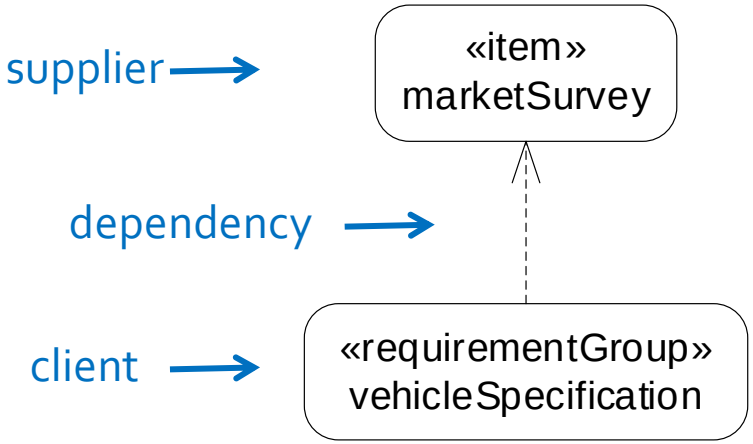


Module 20
Dependency, Allocation, and Cause-Effect Relationships

Dependency

SST

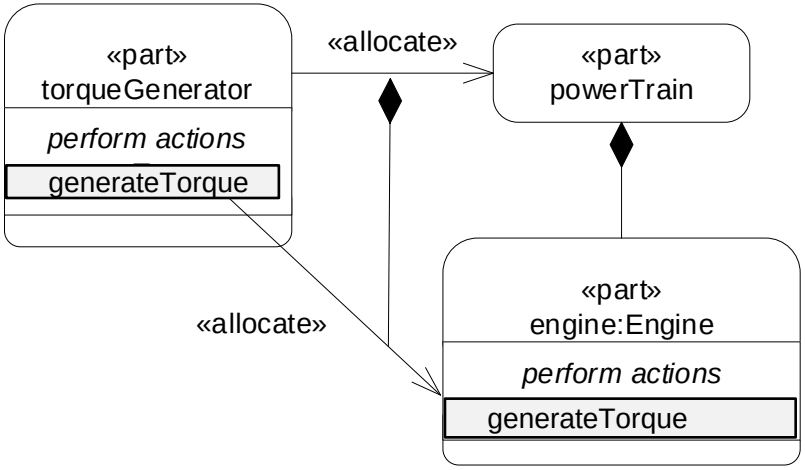
- A directed relationship where the element on the client end may be impacted when the element on the supplier end changes
  - Example: Use of dependency to represent traceability between specification and source document
  - Can have 1.. many clients and 1.. many suppliers



Allocation

SST

- 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 v2
  - Can include suballocations 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
  - …

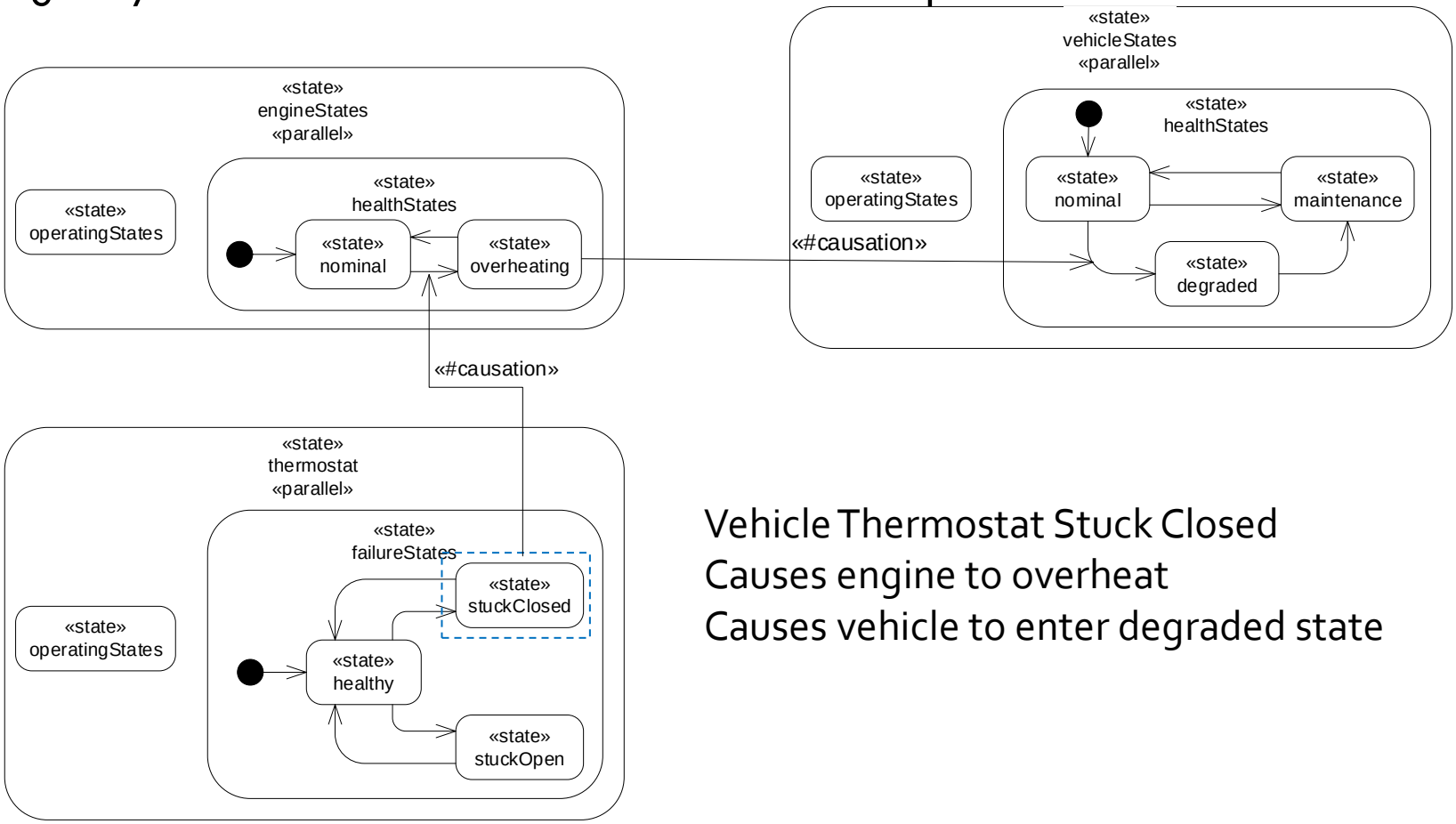


The torque generator
Is allocated to the powertrain,
and realized by the engine

Cause-Effect Relationship v2

SST

- Single relationship can have multiple causes of multiple effects
- Can logically combine different cause-effect relationships



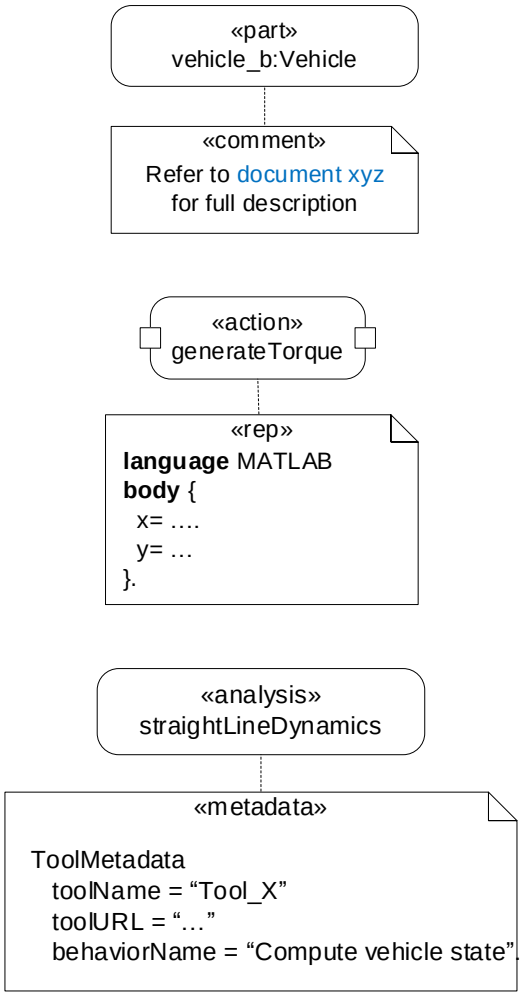
Vehicle Thermostat Stuck Closed
Causes engine to overheat
Causes vehicle to enter degraded state

Module 21
Annotations

Annotations

SST

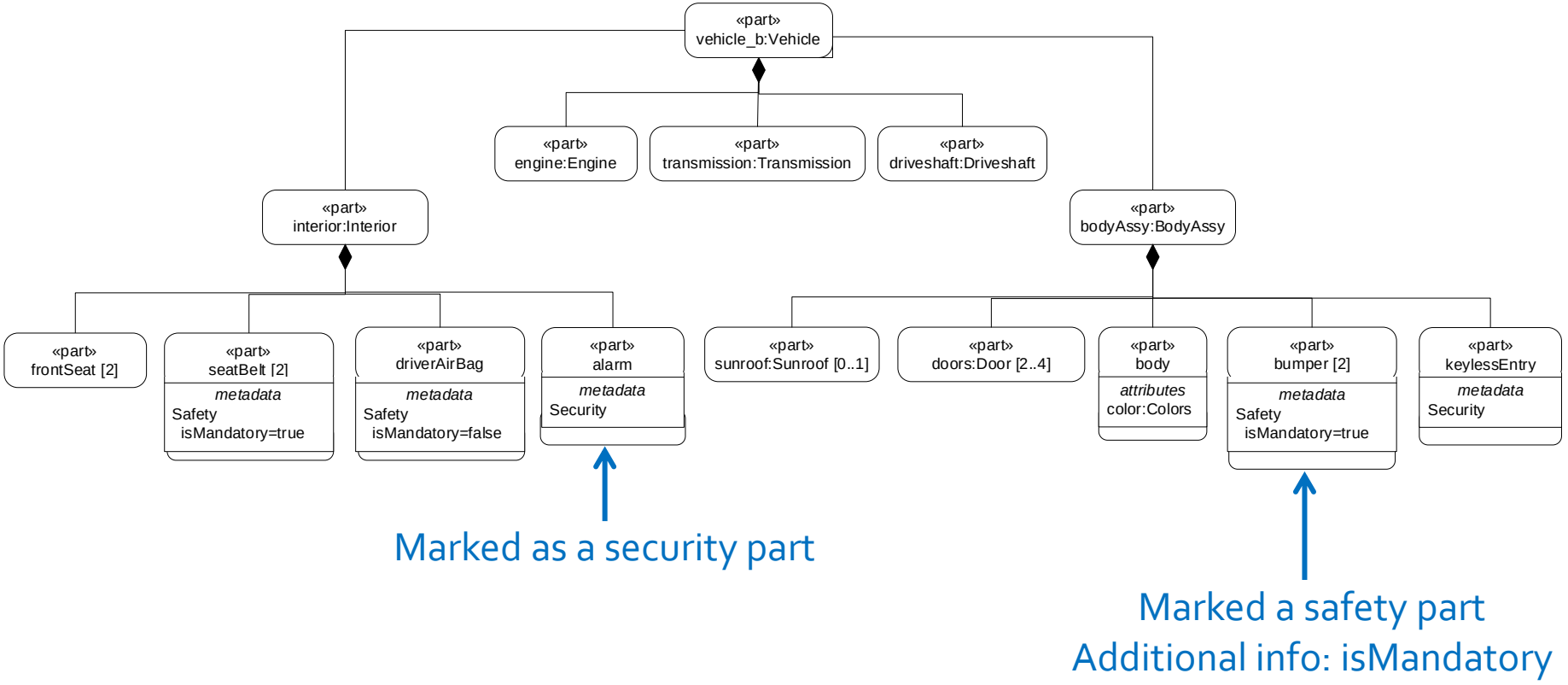
- An Annotating Element is an Element that is used to provide additional information about other elements
  - Can be named
  - Can apply to 0.. many elements
- An annotation is a kind of relationship that relates the annotating element to the element being annotated
- Kinds of annotating elements
  -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



Metadata

SST

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

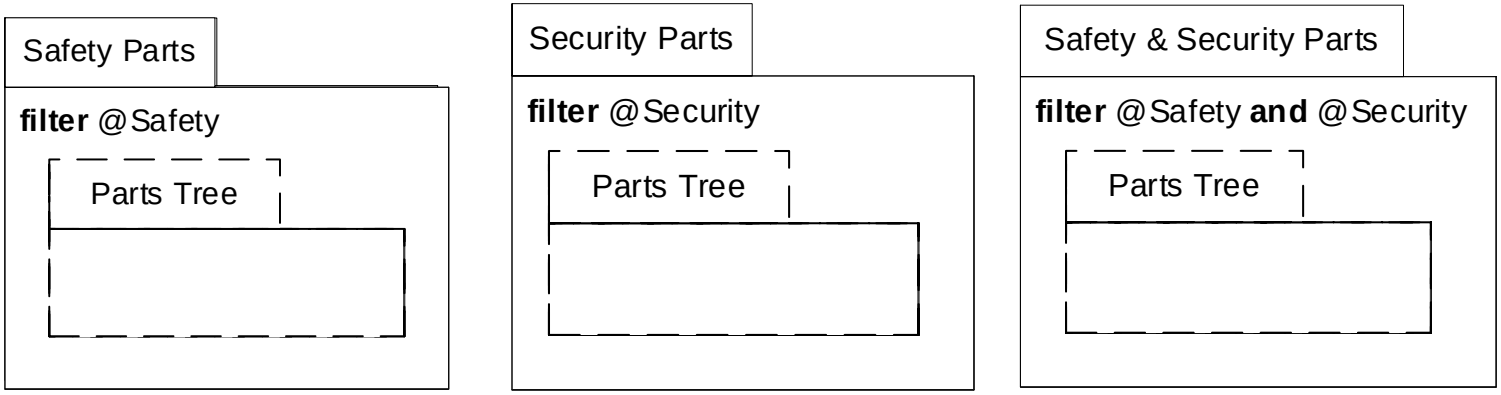


Module 22
Element Filters

Element Filters v2

SST

- 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

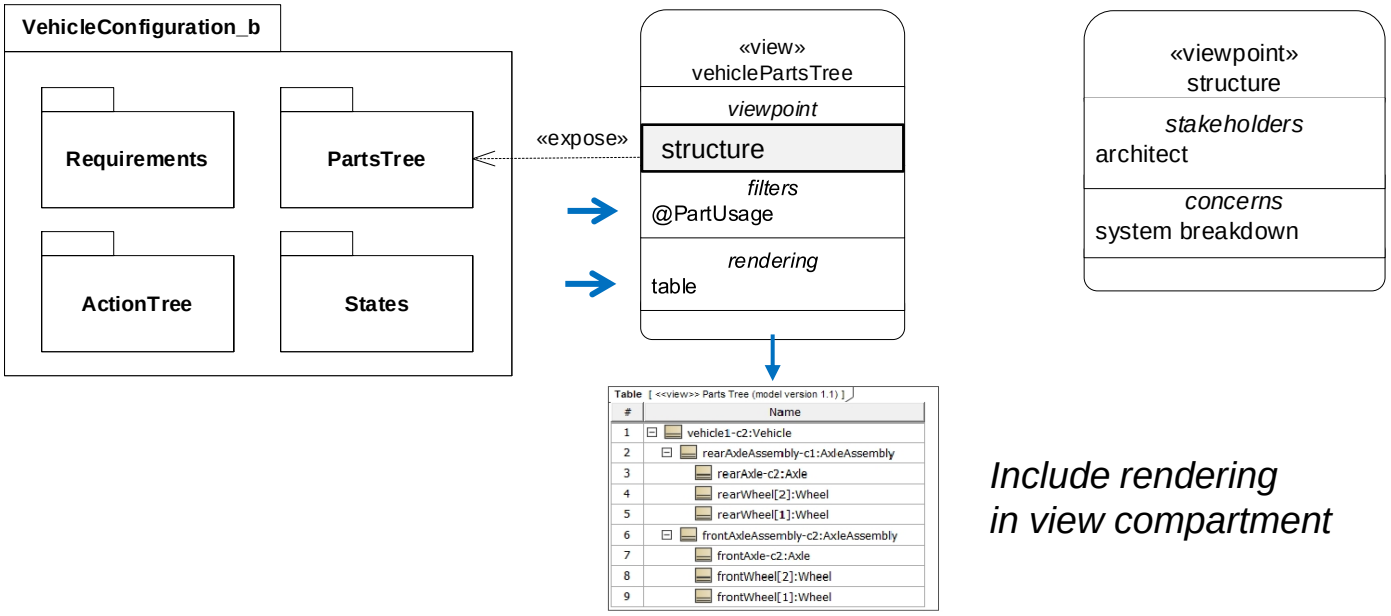


Filter selects parts marked with Safety metadata

Module 23
View & Viewpoint

View and 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)

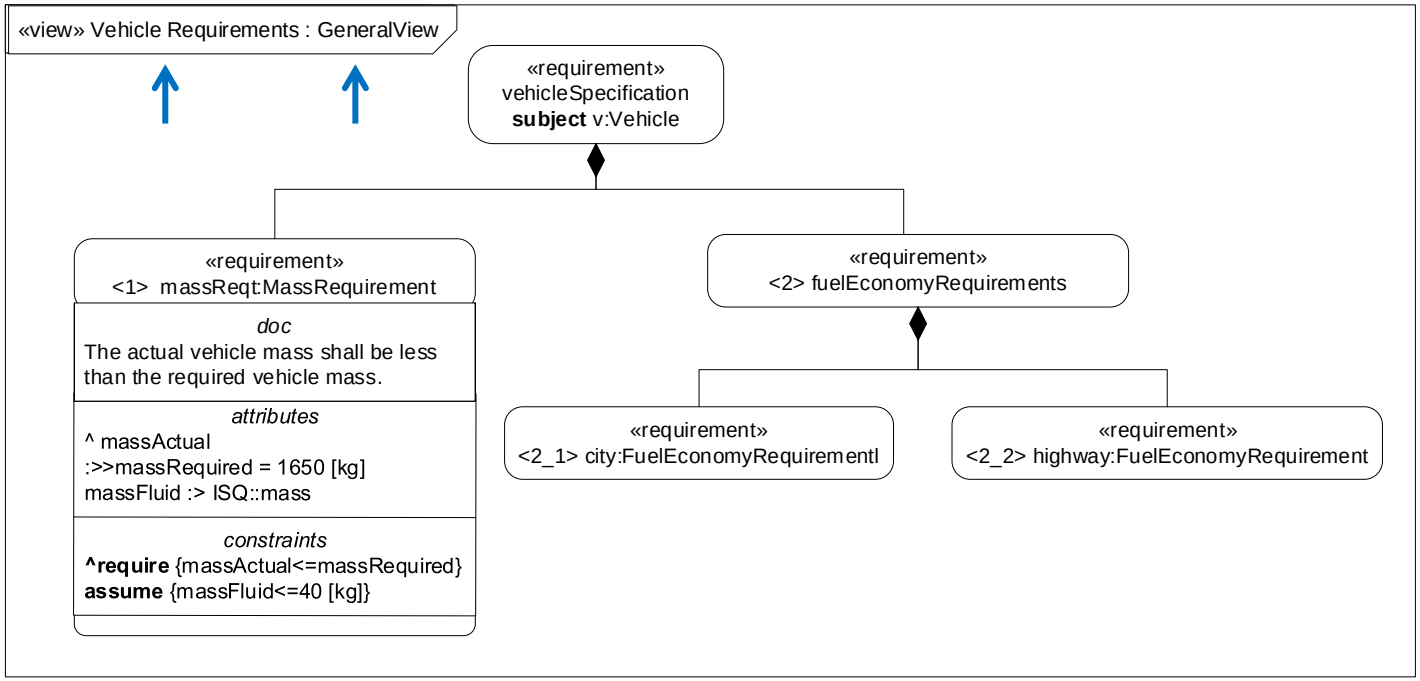


Include rendering in view compartment

SysML v2 Views and Diagrams

SST

- A view is a rendering of a selected model elements based on filter expression and exposed elements
  - Frame is the view element with view name v2
  - View is defined by a view definition (e.g., diagram kind)
  - View content



Standard View Definitions (SysML v2 vs SysML v1) Baseline as of January 04, 2023

SST

Legend

↗ Allocated To

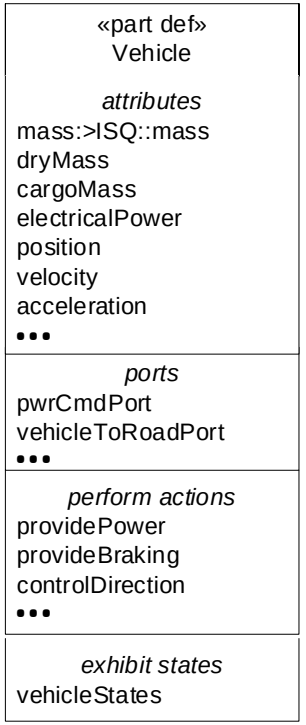
Standard views are normative, but user defined views are valid if they are consistent with the abstract syntax and graphical bnf.

SysML v2 Standard View Definitions		SysML v1 Diagrams: 1 Package Diagrat	2 Block Definition ·	3 Internal Block D·	4 Activity Diagrarr	5 State Machine D	6 Sequence Diagr	7 Use Case Diagra	8 Requirement Dia	9 Parametric Diag
		1	1	1	1	1	1	1	2	1
1 General View (gv)	3	↙	↙						↙	
2 Interconnection View (iv)	2			↙						↙
3 Action Flow View - (afv) w and w/o swimlane	1				↙					
4 State Transition View (stv)	1					↙				
5 Sequence View (sv)	1						↙			
6 Case View (cv)	1							↙		
7 Geometry View (gev)										
8 Grid View (grv)	1								↙	
9 Browser View (bv)										

SysML v2 Compartments

SST

- Compartments are views of the element represented by the node v2
- Compartment names represent view definitions
- Standard compartments / view definitions
  - Textual view of contained elements of a particular kind (e.g., attributes, actions, ports)
  - Textual view of elements at the other end of relationships (e.g., allocation, specialization, …)
- A compartment can also be rendered using graphical notation
- Ports and parameters are connectable nested nodes on the boundary of a structure or behavior



Module 24
Language Extension

Language Extension

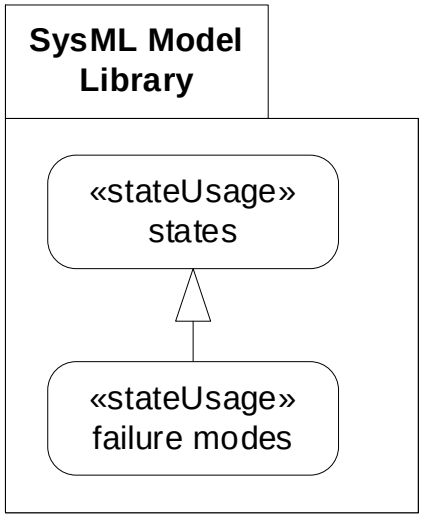
SST

- 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
- Define the keyword using metadata

v2

```
state 'failure modes'[*] nonunique;

metadata def <fm> 'failure mode' :> SemanticMetadata {
        :>> baseType = 'failure modes' meta SysML::StateUsage;
}
```

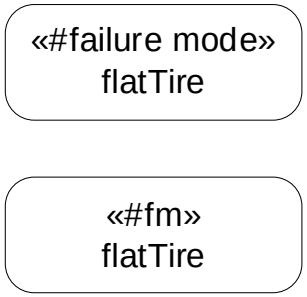
define the concept →



- Apply the concept to an element by annotating it with the metadata

```
#'failure mode' flatTire;

#fm flatTire;
```

apply the concept →



Contrasting SysML v2 with SysML v1

SysML v2 to v1 Terminology Mapping (partial)

SST

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
view / view def	view

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 flexible model organization (unowned members, package filters)…
- **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, ..)
  - Simple geometry
  - Query/filter expressions
  - Metadata
- **More extensible**
  - Simpler language extension capability
    - Based on model libraries
- **More interoperable**
  - Standardized API

Summary

Summary

SST

- SysML v2 is addressing SysML v1 limitations to improve MBSE adoption and effectiveness
  - Precision, expressiveness
  - Regularity, usability
  - Interoperability with other engineering models and tools
- Approach
  - Simplified SysML v2 metamodel with formal semantics overcomes fundamental UML limitations
  - Flexible graphical notations and textual notation
  - Standardized API for interoperability
- Specification submitted to OMG on February 20, 2023
  - Will request vote to adopt at March 2023 OMG meeting
  - Becomes a beta specification if approved

References

SST

- Monthly Release Repository (release 2023-02)
  - https://github.com/Systems-Modeling/SysML-v2-Release
- SysML v2 Specification (revised submission)
  - Note: refer to Annex B for Example Vehicle Model
- 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