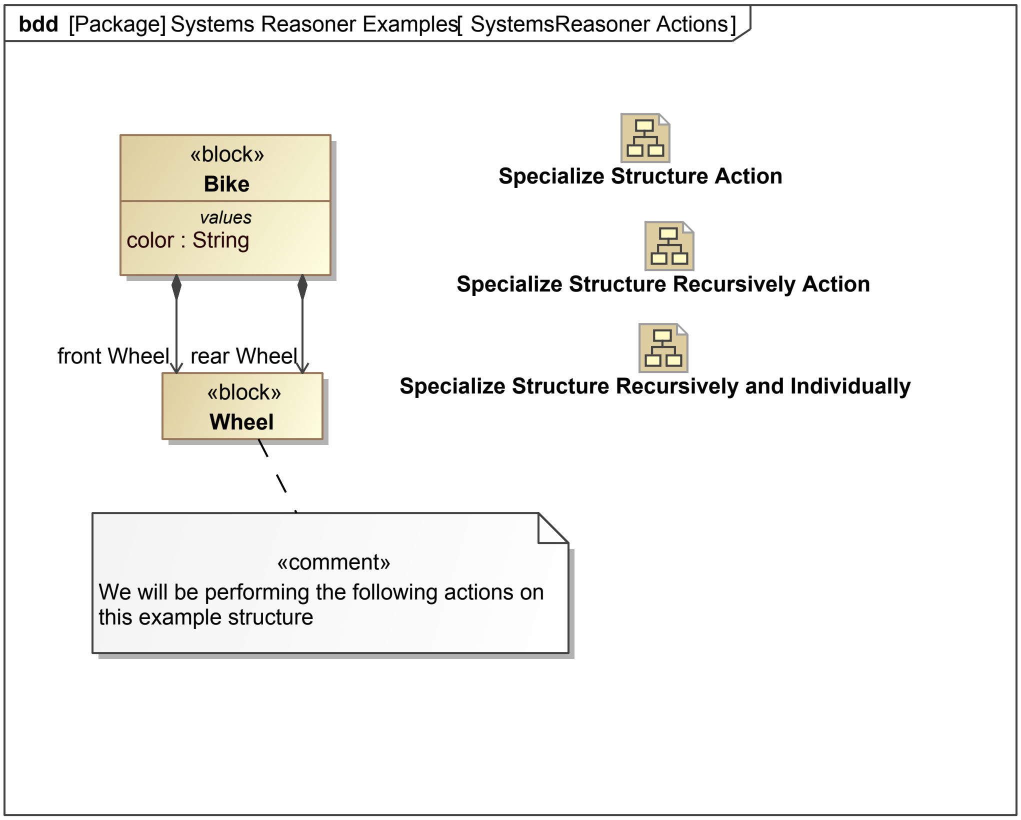
13 Guidelines for Modeling Non-Functional Aspects

13.1 Systems Reasoner Documentation

13.1.1 Requirements

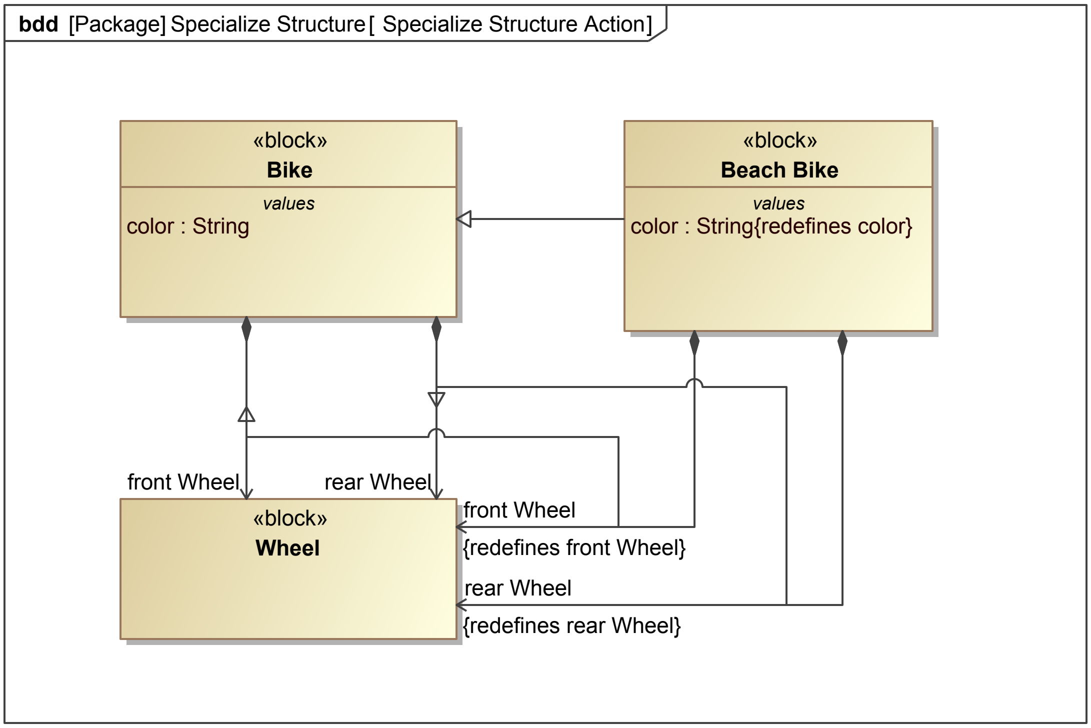
The BST pattern: - Every redefinable element of a Class shall be redefined in its specialized Classifier. - The type of a redefined element shall be the same as of the redefining element. - Associations between specialized Classifiers shall inherit from the corresponding Associations between the Generals. -Every aspect defined by a dependency with the < > Stereotype shall be realized within the same Classifier. -

13.1.2 Actions


bdd [Package] Systems Reasoner Examples[SystemsReasoner Actions]
«block»
Bike
values
color : String
front Wheel
rear Wheel
«block»
Wheel
Specialize Structure Action
Specialize Structure Recursively Action
Specialize Structure Recursively and Individually
«comment»
We will be performing the following actions on this example structure

SystemsReasoner Actions

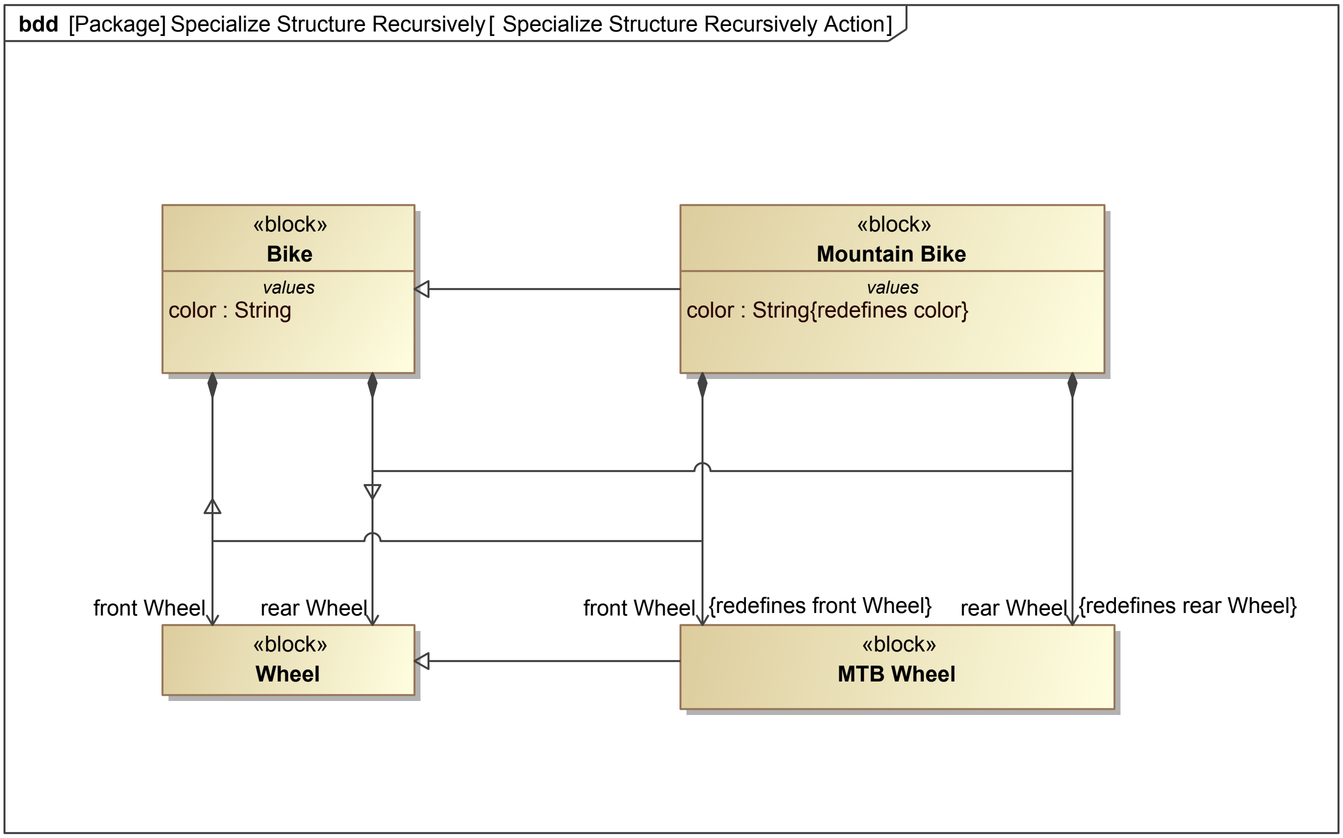
13.1.3 Specialize Structure Action



Specialize Structure Action

The specialize structure action will create a specialized block with all redefinable elements redefined on the specialized level. Part properties will point to the type of the generals part properties. The associations of the part properties have to inherit from their general counterparts.

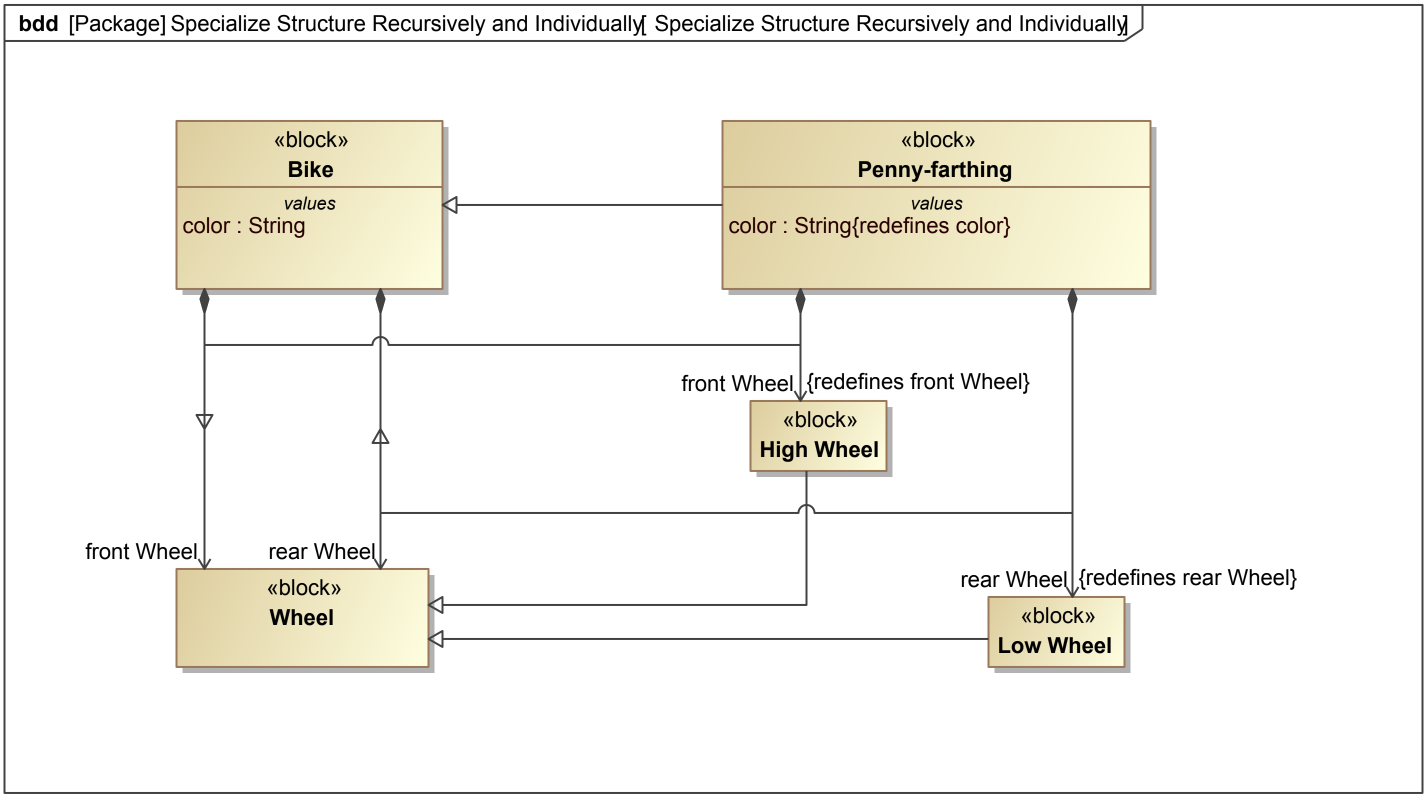
13.1.4 Specialize Structure Recursively Action



Specialize Structure Recursively Action

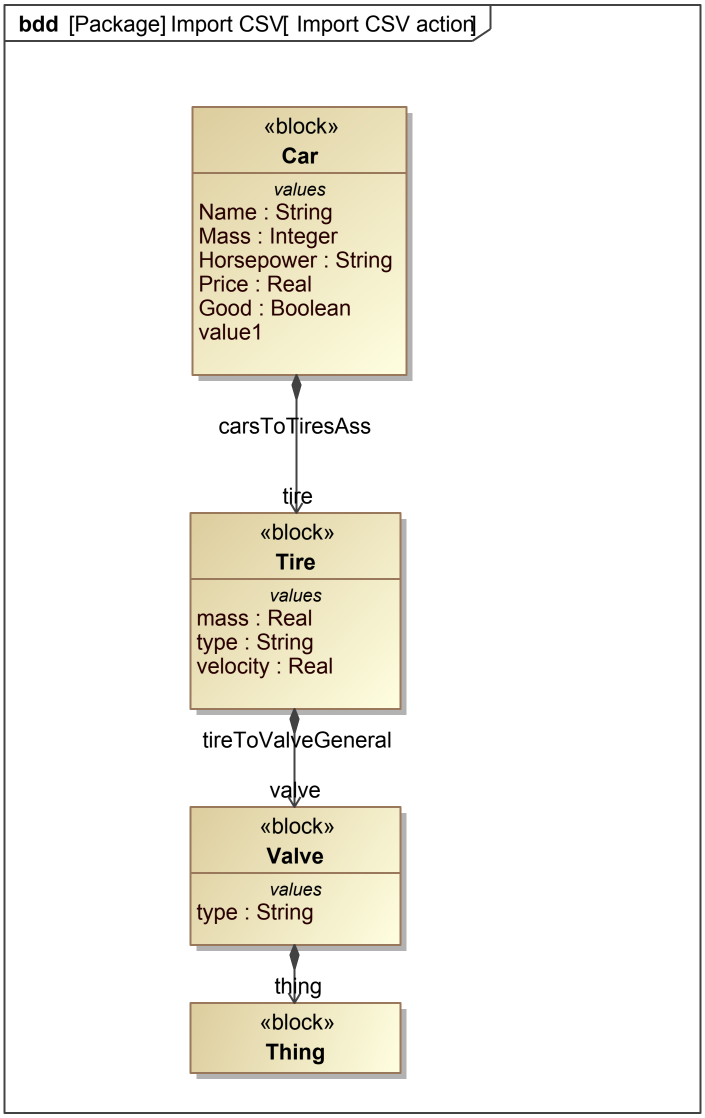
The specialize structure recursively action will create a specialized Block with all redefinable Elements redefined on the specialized level. Part properties will point to a new type and part properties of that element will also be pointing to new types recursively. If a Block has two part properties as in the example to the left, they will both point to the same type.

13.1.5 Specialize Structure Recursively and Individually Action



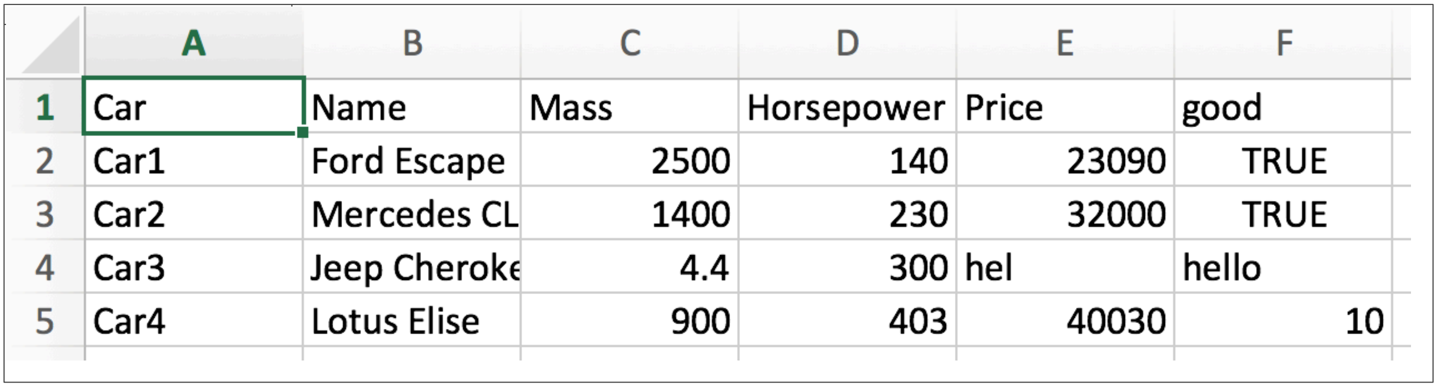
Specialize Structure Recursively and Individually

13.1.6 Import CSV Action



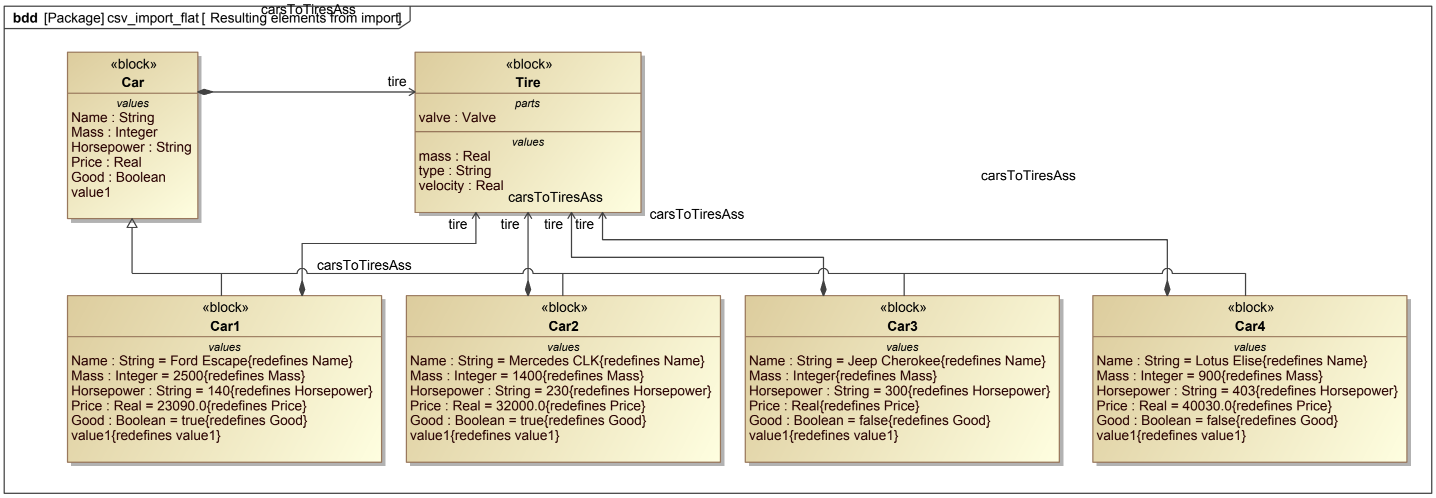
Import CSV action

The import CSV action allows for the creation of block specific types from a CSV file.
Each line is a new element, each row is a property of an element. Nested elements can be expressed by using dot notation (mass of a tire is tire.mass).



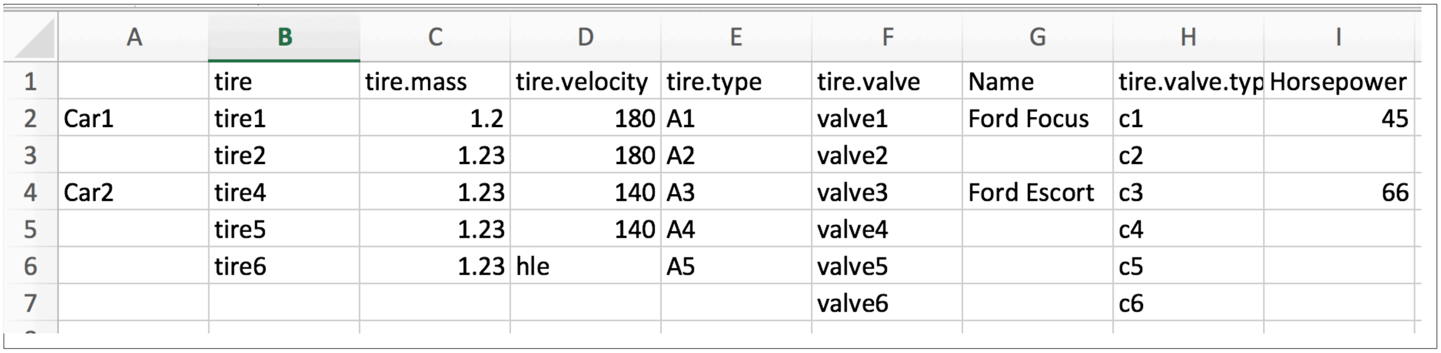
	A	B	C	D	E	F
1	Car	Name	Mass	Horsepower	Price	good
2	Car1	Ford Escape	2500	140	23090	TRUE
3	Car2	Mercedes CL	1400	230	32000	TRUE
4	Car3	Jeep Cheroke	4.4	300	hel	hello
5	Car4	Lotus Elise	900	403	40030	10

Flat table in CSV.


bdd [Package] csv_import_flat [Resulting elements from import]
carsToTiresAss
«block»
Car
values
Name : String
Mass : Integer
Horsepower : String
Price : Real
Good : Boolean
value1
tire
«block»
Tire
parts
valve : Valve
values
mass : Real
type : String
velocity : Real
carsToTiresAss
carsToTiresAss
carsToTiresAss
tire
tire
tire
tire
carsToTiresAss
«block»
Car1
values
Name : String = Ford Escape{redefines Name}
Mass : Integer = 2500{redefines Mass}
Horsepower : String = 140{redefines Horsepower}
Price : Real = 23090.0{redefines Price}
Good : Boolean = true{redefines Good}
value1{redefines value1}
«block»
Car2
values
Name : String = Mercedes CLK{redefines Name}
Mass : Integer = 1400{redefines Mass}
Horsepower : String = 230{redefines Horsepower}
Price : Real = 32000.0{redefines Price}
Good : Boolean = true{redefines Good}
value1{redefines value1}
«block»
Car3
values
Name : String = Jeep Cherokee{redefines Name}
Mass : Integer{redefines Mass}
Horsepower : String = 300{redefines Horsepower}
Price : Real{redefines Price}
Good : Boolean = false{redefines Good}
value1{redefines value1}
«block»
Car4
values
Name : String = Lotus Elise{redefines Name}
Mass : Integer = 900{redefines Mass}
Horsepower : String = 403{redefines Horsepower}
Price : Real = 40030.0{redefines Price}
Good : Boolean = false{redefines Good}
value1{redefines value1}

Resulting elements from import.

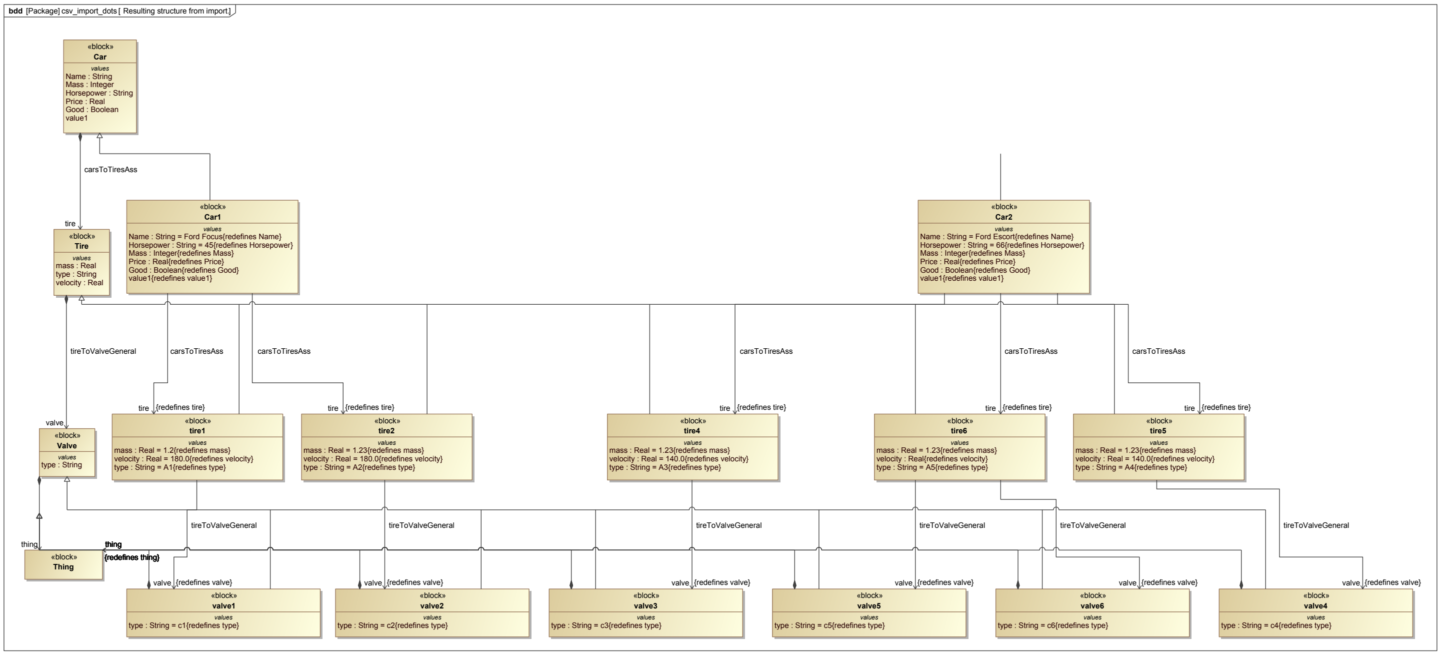
The diagram above shows the result from importing this excel table in CSV format. For every line a new special class is created. The columns are either class names (first column) or property names.



	A	B	C	D	E	F	G	H	I
1		tire	tire.mass	tire.velocity	tire.type	tire.valve	Name	tire.valve.typ	Horsepower
2	Car1	tire1	1.2	180	A1	valve1	Ford Focus	c1	45
3		tire2	1.23	180	A2	valve2		c2	
4	Car2	tire4	1.23	140	A3	valve3	Ford Escort	c3	66
5		tire5	1.23	140	A4	valve4		c4	
6		tire6	1.23	hle	A5	valve5		c5	
7						valve6		c6	

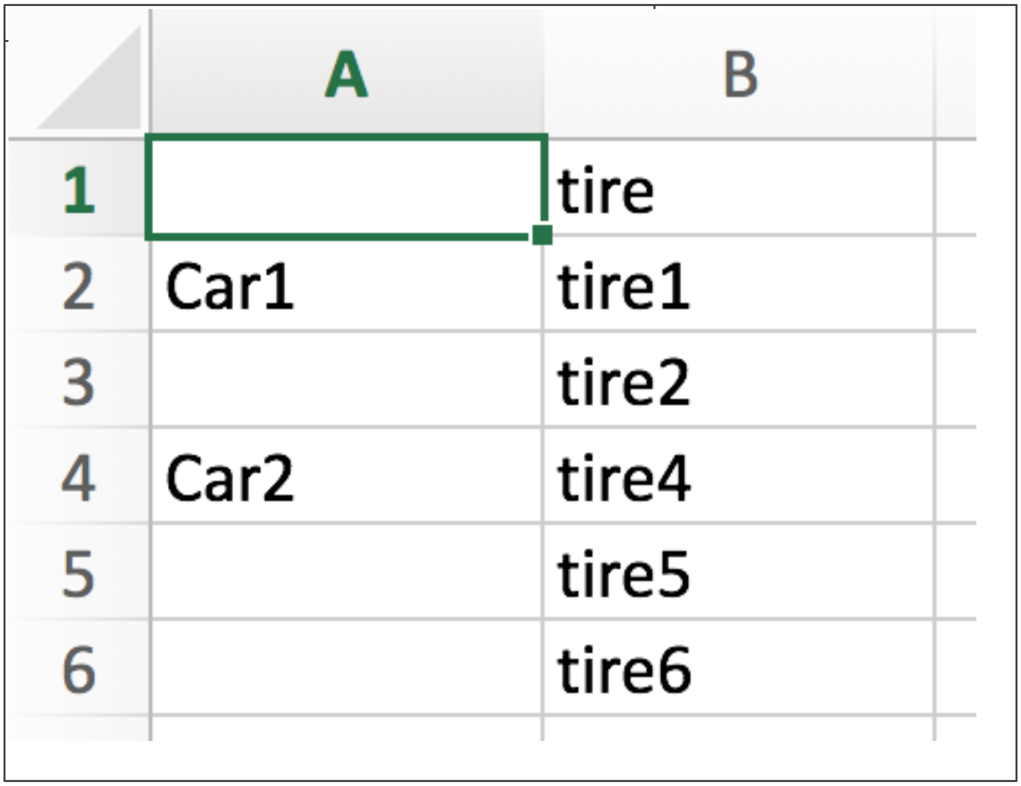
Table with dot notation in CSV.

To import part properties, the dot notation is used. The car has tires and tires can have mass so tire.mass describes the mass of the tire of the car. It is best to leave the element values of the parents empty after the first line when more children are created. In the example above, the car name is only listed twice because there are only two cars created.


bdd [Package] csv_import_dots [Resulting structure from import.]
«block»
Car
values
Name : String
Mass : Integer
Horsepower : String
Price : Real
Good : Boolean
value1
carsToTiresAss
tire
«block»
Tire
values
mass : Real
type : String
velocity : Real
«block»
Car1
values
Name : String = Ford Focus{redefines Name}
Horsepower : String = 45{redefines Horsepower}
Mass : Integer{redefines Mass}
Price : Real{redefines Price}
Good : Boolean{redefines Good}
value1{redefines value1}
«block»
Car2
values
Name : String = Ford Escort{redefines Name}
Horsepower : String = 66{redefines Horsepower}
Mass : Integer{redefines Mass}
Price : Real{redefines Price}
Good : Boolean{redefines Good}
value1{redefines value1}
tireToValveGeneral
carsToTiresAss
carsToTiresAss
carsToTiresAss
carsToTiresAss
carsToTiresAss
tire {redefines tire}
tire {redefines tire}
tire {redefines tire}
tire {redefines tire}
tire {redefines tire}
valve
«block»
Valve
values
type : String
«block»
tire1
values
mass : Real = 1.2{redefines mass}
velocity : Real = 180.0{redefines velocity}
type : String = A1{redefines type}
«block»
tire2
values
mass : Real = 1.23{redefines mass}
velocity : Real = 180.0{redefines velocity}
type : String = A2{redefines type}
«block»
tire4
values
mass : Real = 1.23{redefines mass}
velocity : Real = 140.0{redefines velocity}
type : String = A3{redefines type}
«block»
tire6
values
mass : Real = 1.23{redefines mass}
velocity : Real{redefines velocity}
type : String = A5{redefines type}
«block»
tire5
values
mass : Real = 1.23{redefines mass}
velocity : Real = 140.0{redefines velocity}
type : String = A4{redefines type}
tireToValveGeneral
tireToValveGeneral
tireToValveGeneral
tireToValveGeneral
tireToValveGeneral
tireToValveGeneral
thing
thing
{redefines thing}
«block»
Thing
valve {redefines valve}
valve {redefines valve}
valve {redefines valve}
valve {redefines valve}
valve {redefines valve}
valve {redefines valve}
«block»
valve1
values
type : String = c1{redefines type}
«block»
valve2
values
type : String = c2{redefines type}
«block»
valve3
values
type : String = c3{redefines type}
«block»
valve5
values
type : String = c5{redefines type}
«block»
valve6
values
type : String = c6{redefines type}
«block»
valve4
values
type : String = c4{redefines type}

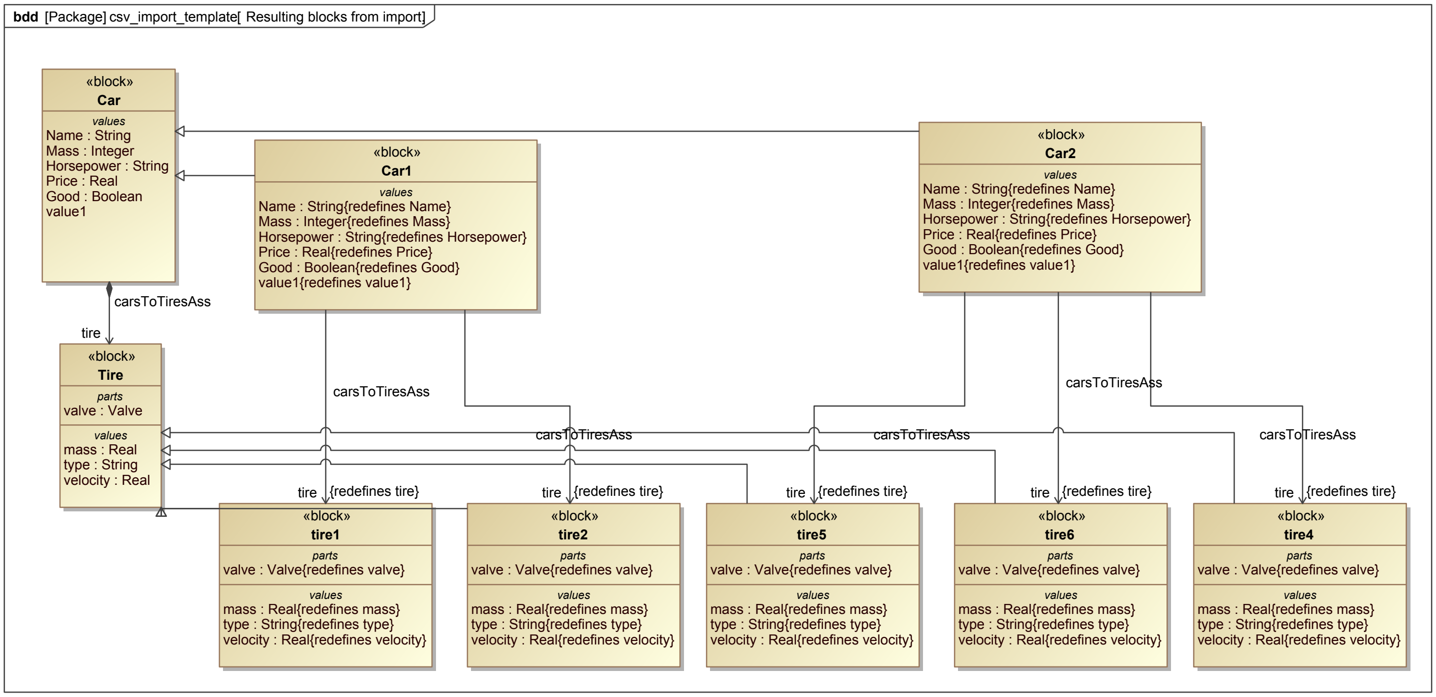
Resulting structure from import.

The figure above shows the structure that results from the import using the dot notation. Two cars are created with multiple tires and each tire has their own valve. Except for tire 6 that has two valves. On tire 6 it is also shown that if the value doesn't comply with the specified type (here a real) the value is left empty.



	A	B
1		tire
2	Car1	tire1
3		tire2
4	Car2	tire4
5		tire5
6		tire6

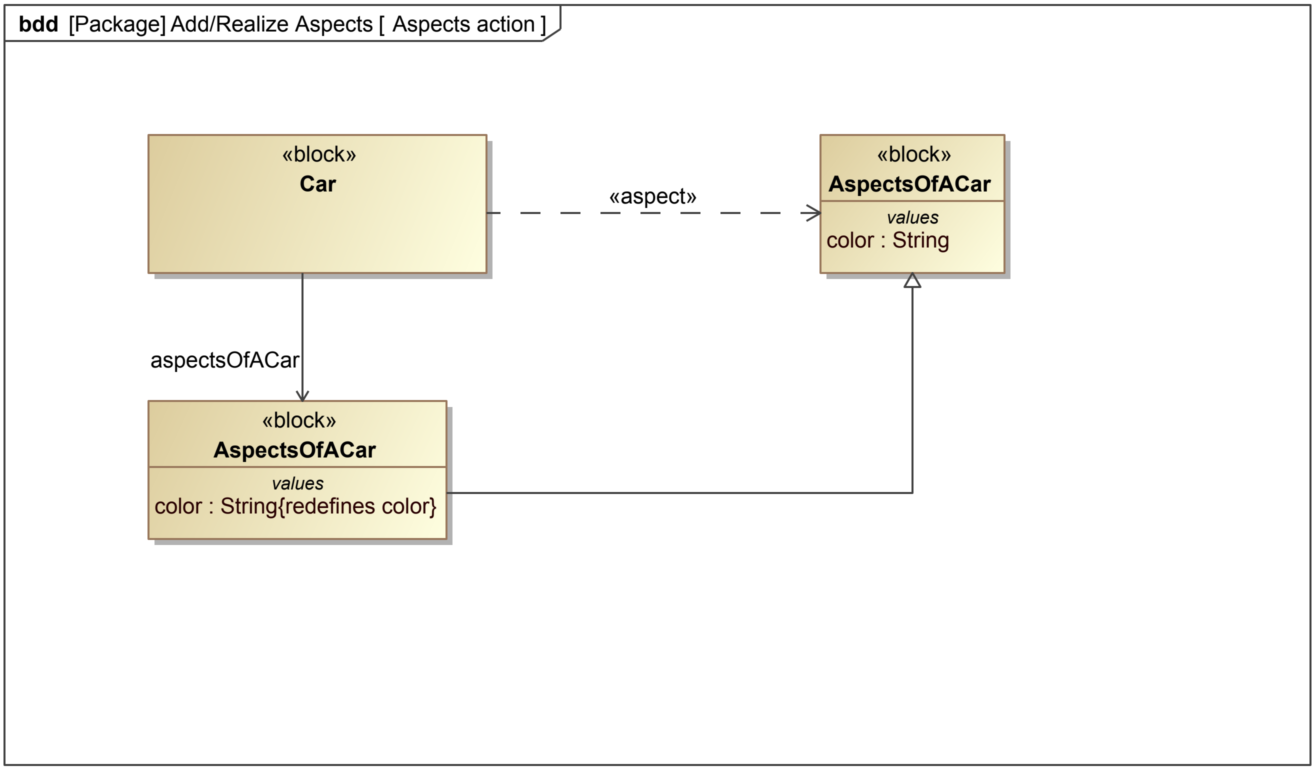
Table as template in CSV.


bdd [Package] csv_import_template[Resulting blocks from import]
«block»
Car
values
Name : String
Mass : Integer
Horsepower : String
Price : Real
Good : Boolean
value1
«block»
Car1
values
Name : String{redefines Name}
Mass : Integer{redefines Mass}
Horsepower : String{redefines Horsepower}
Price : Real{redefines Price}
Good : Boolean{redefines Good}
value1{redefines value1}
«block»
Car2
values
Name : String{redefines Name}
Mass : Integer{redefines Mass}
Horsepower : String{redefines Horsepower}
Price : Real{redefines Price}
Good : Boolean{redefines Good}
value1{redefines value1}
carsToTiresAss
tire
«block»
Tire
parts
valve : Valve
values
mass : Real
type : String
velocity : Real
carsToTiresAss
carsToTiresAss
carsToTiresAss
carsToTiresAss
carsToTiresAss
tire
{redefines tire}
tire
{redefines tire}
tire
{redefines tire}
tire
{redefines tire}
tire
{redefines tire}
«block»
tire1
parts
valve : Valve{redefines valve}
values
mass : Real{redefines mass}
type : String{redefines type}
velocity : Real{redefines velocity}
«block»
tire2
parts
valve : Valve{redefines valve}
values
mass : Real{redefines mass}
type : String{redefines type}
velocity : Real{redefines velocity}
«block»
tire5
parts
valve : Valve{redefines valve}
values
mass : Real{redefines mass}
type : String{redefines type}
velocity : Real{redefines velocity}
«block»
tire6
parts
valve : Valve{redefines valve}
values
mass : Real{redefines mass}
type : String{redefines type}
velocity : Real{redefines velocity}
«block»
tire4
parts
valve : Valve{redefines valve}
values
mass : Real{redefines mass}
type : String{redefines type}
velocity : Real{redefines velocity}

Resulting blocks from import.

Two cars are created with their respective tires. The value properties are all created and redefined but no values are specified.

13.1.7 Add Aspects Action



Aspects action

The aspects pattern allows to add aspects to any existing block. The aspect relationship is defined by a dependency with an aspect stereotype. The definition of the aspect in the local context is realized by a block contained in the aspected block as a specialized of the aspect block.

13.2 Quality of Service

13.2.1 How do I define Quality of Service?

SysML activity diagrams offer only a rate to define details of a pin. Often more QoS are needed, like latency, jitter, clocked. The solution is to define a stereotype Qos with the properties clocked, jitter, latency, which can have different values for every Pin. If the QoS is valid for both ends of the edge, the edge itself is stereotyped, as suggested already by C.Bock. The main point of discussion is, if the pin of the action or the parameter of the activity shall be stereotyped. The correct approach seems to stereotype the parameter and the tool shall propagate it to the associated pin -. SysML status • SysML only provides only < > stereotype which extends Activity Edge and Parameter. • Allocation of Ports to Pins not addressed in SysML standard 1.1 • Synchronization of Parameter and Pin is tool-dependent. • UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms

13.2.2 How does it relate to Parts and Ports?

Activities are as usual allocated to parts or blocks. Each pin can be allocated to a flow port. In case pins are bundled on a port they are allocated to the same port. The allocation of pins to ports is optional. If there is a one to one mapping the data type of the flow port and the object node have to be the same. Allocation ObjectFlow to ItemFlow The ObjectFlow (Edge) describes that in the context of an Activity the output of one Action is bound to the input of another action. In the context of a block a item flow describes the flow of an object from one part or port to the connected part or port. The allocation of the ObjectFlow to an ItemFlow defines which ObjectFlow corresponds to which ItemFlow in a given context. Supplier and producer and context need always be defined. Allocation Pin to Port (not addressed in SysML standard 1.1) The pin defines which objects flows in/out of an action from a functional point of view. The port defines which object flows in/out of a block from a structural point of view. The allocation of pin to port defines the mapping of functional to structural view, independent of a context. The supplier and producer need to be known, e.g. when certain data flows over an Ethernet port but it is irrelevant who is connected to it. • Activities are allocated to blocks if the allocation is true for all parts of this block • Actions are allocated to parts if the activity is only relevant for a particular part. • Block operations to parts This is particularly the case when sequence diagrams are used to describe behavior rather than activity diagrams. Operations of a block (the whole) in a sequence diagram can be allocated to its parts, it is composed of.