10 Miscelleanous

Table of Contents

List of Figures

1 Introduction

2 Systems Reasoner Documentation

2.1 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. -

2.2 Actions

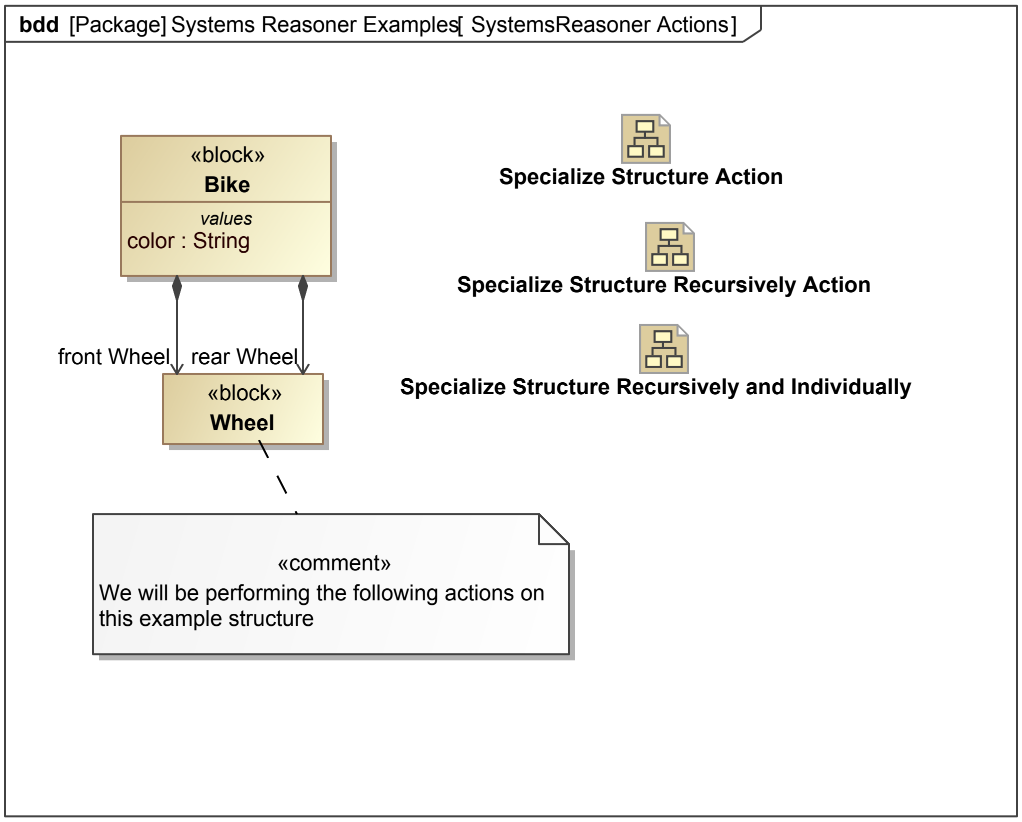

bdd [Package] Systems Reasoner Examples[SystemsReasoner Actions]
«block»
Bike
values
color : String
front Wheel
rear Wheel
«block»
Wheel
Specialize Structure Action
Specialize Structure Recursively Action
Specialize Structure Recursively and Individually
«comment»
We will be performing the following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

Figure 1. SystemsReasoner Actions

2.3 Specialize Structure Action

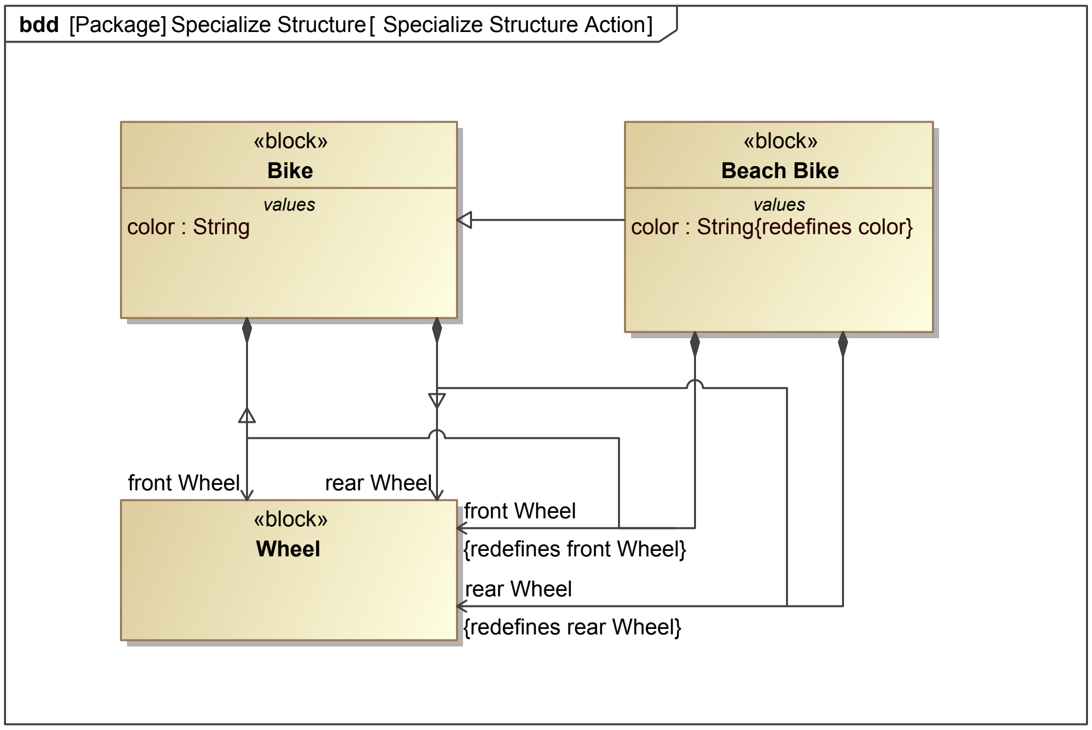


Figure 2. Specialize Structure Action

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.

2.4 Specialize Structure Recursively Action

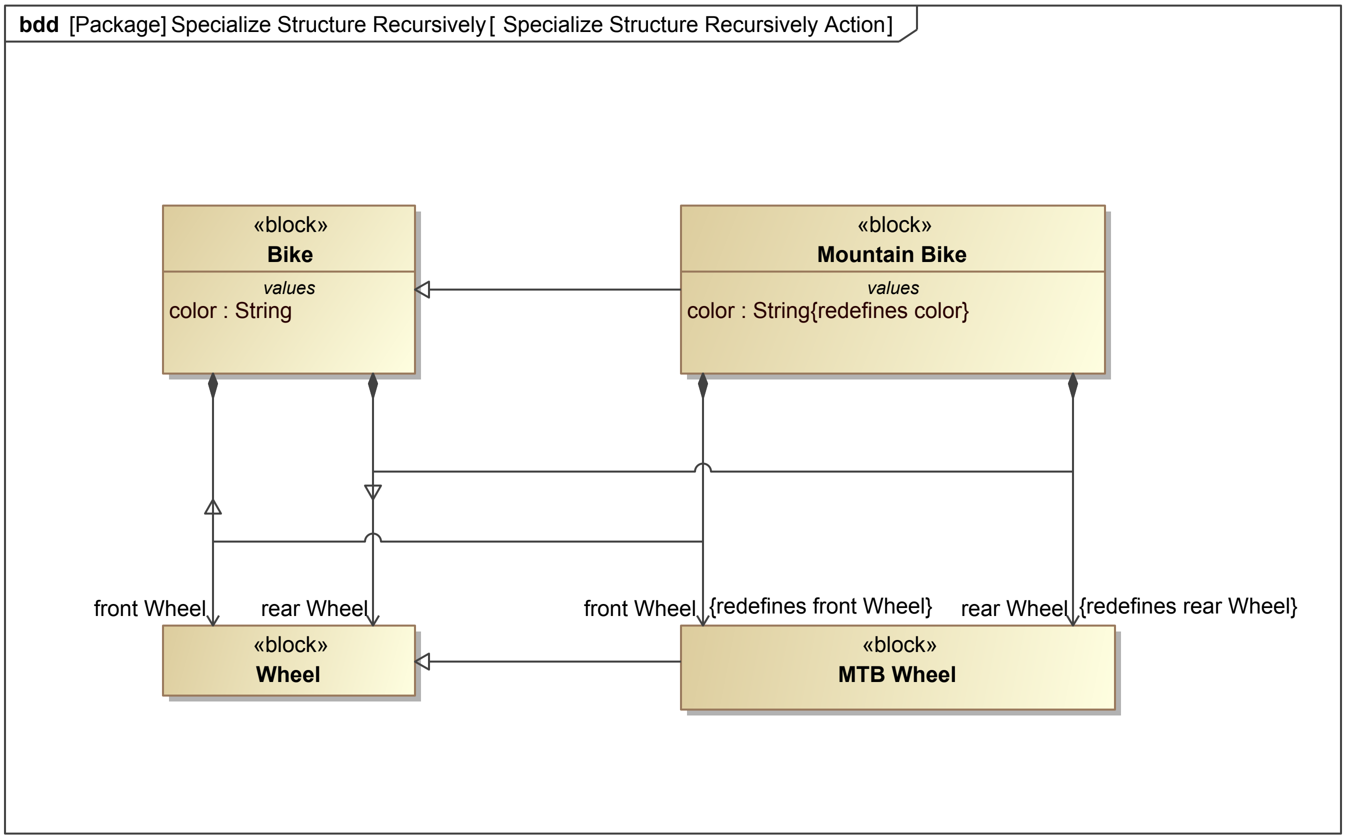


Figure 3. Specialize Structure Recursively Action

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.

2.5 Specialize Structure Recursively and Individually Action

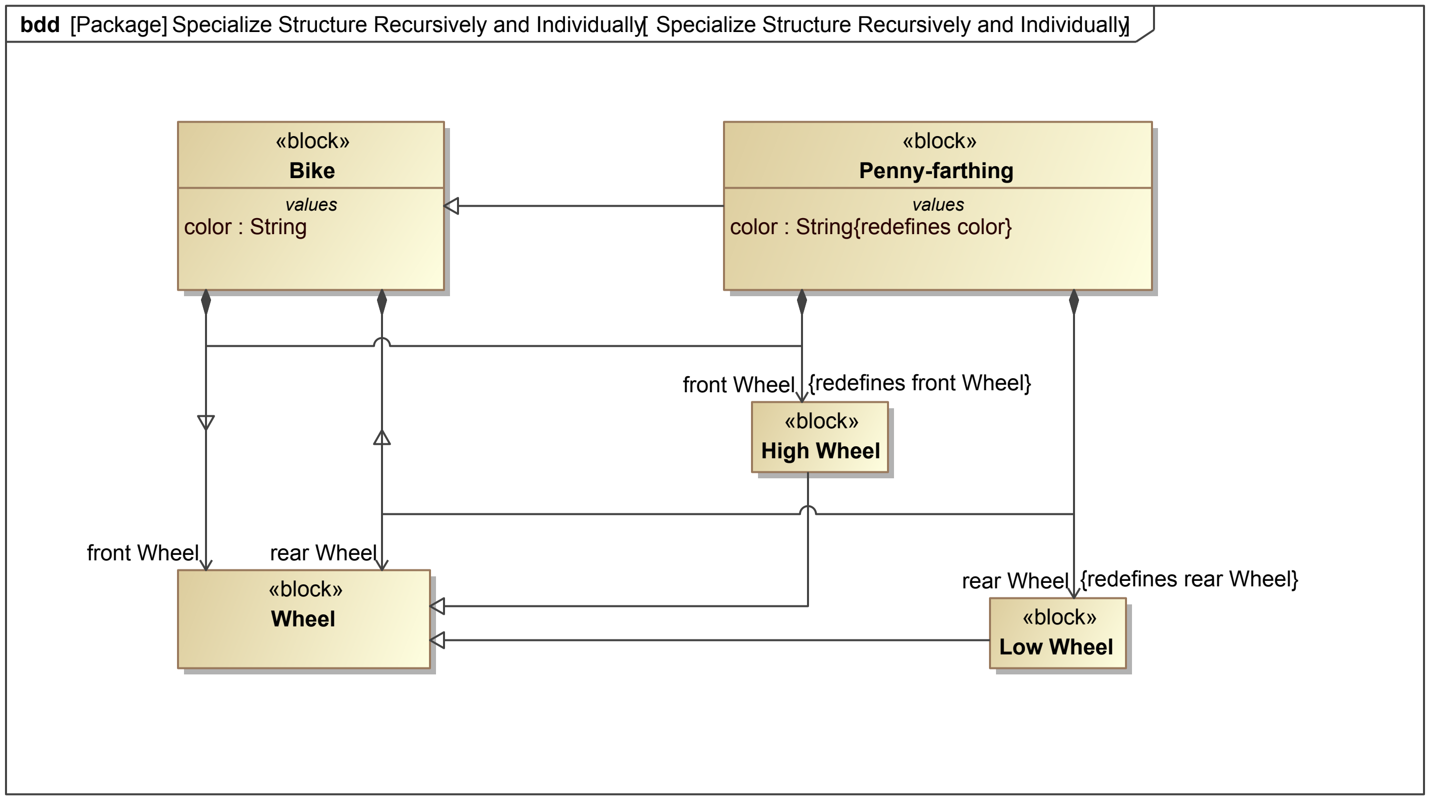


Figure 4. Specialize Structure Recursively and Individually

2.6 Import CSV Action

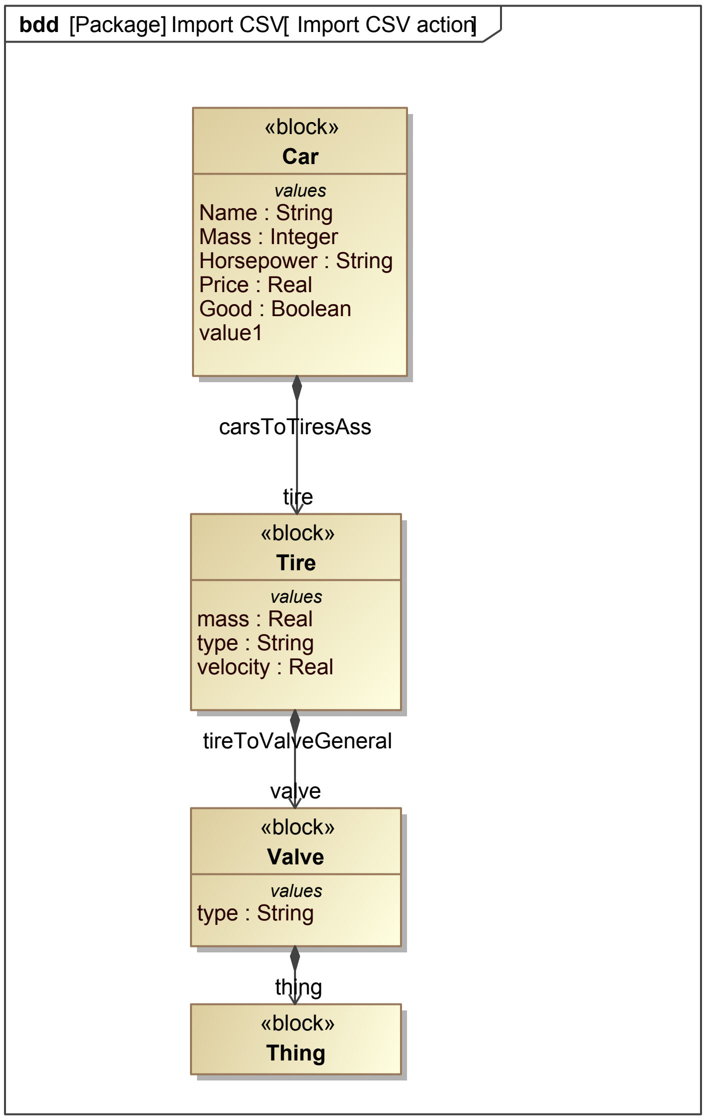
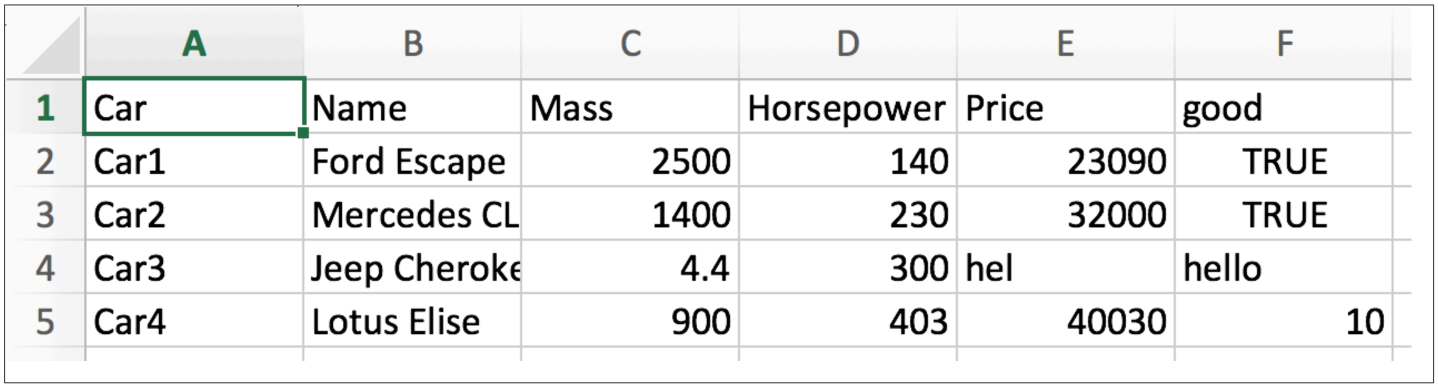


Figure 5. Import CSV action

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

Figure 6. Flat table in CSV.

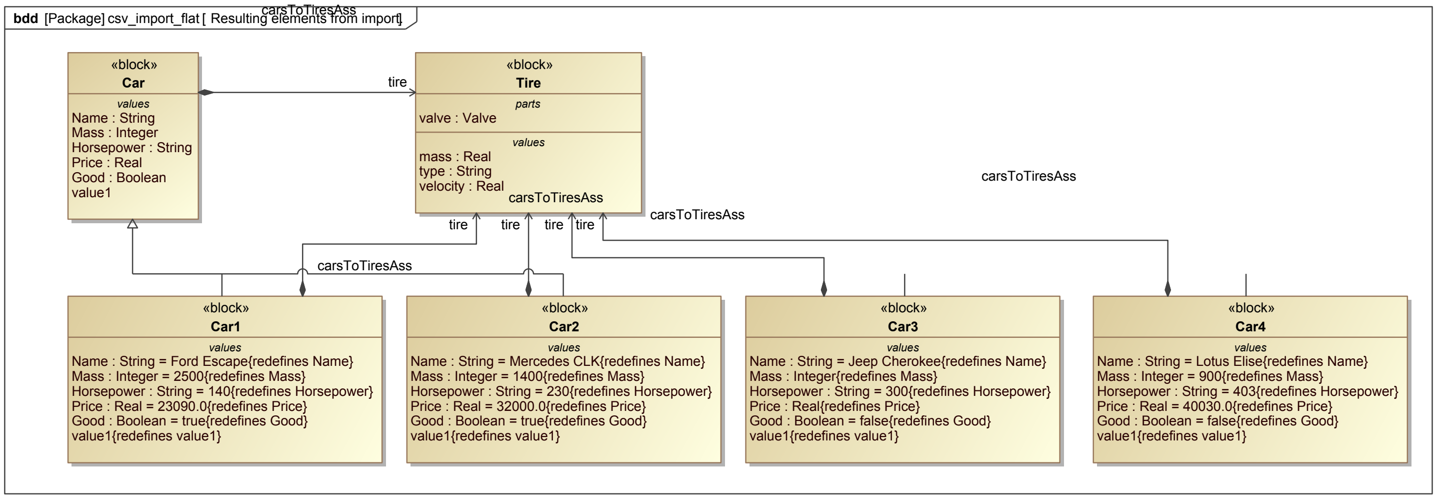
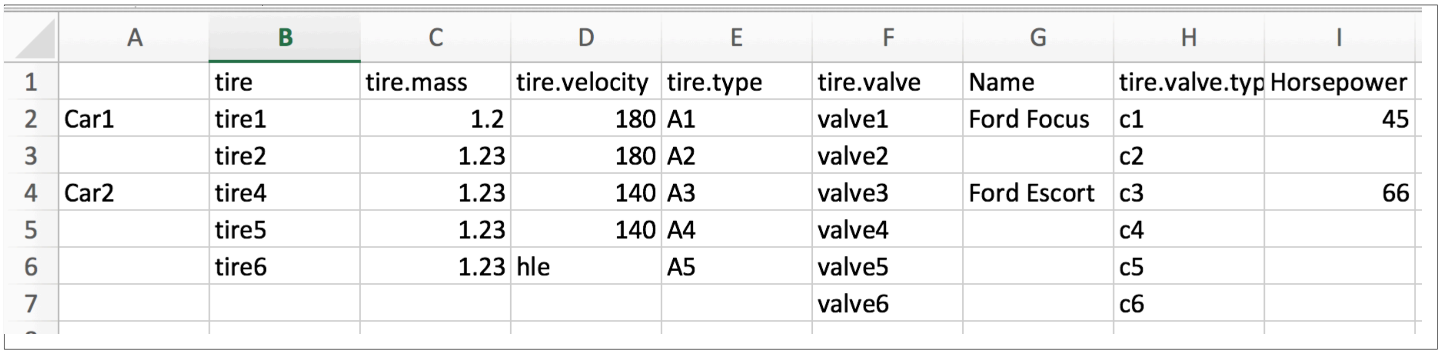

bdd [Package] csv_import_flat [Resulting elements from import]
carsToTiresAss
«block»
Car
values
Name : String
Mass : Integer
Horsepower : String
Price : Real
Good : Boolean
value1
tire
«block»
Tire
parts
valve : Valve
values
mass : Real
type : String
velocity : Real
carsToTiresAss
carsToTiresAss
carsToTiresAss
carsToTiresAss
tire
tire
tire
tire
«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}

Figure 7. 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	

Figure 8. 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.

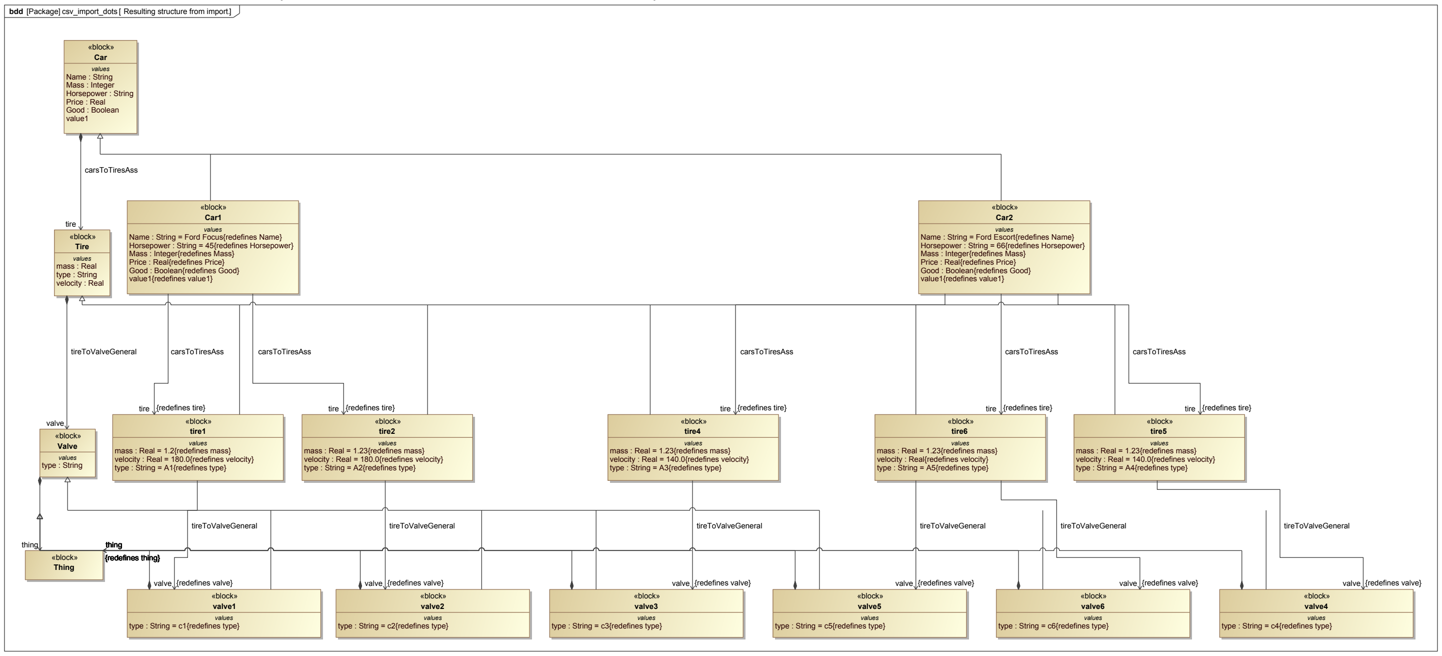
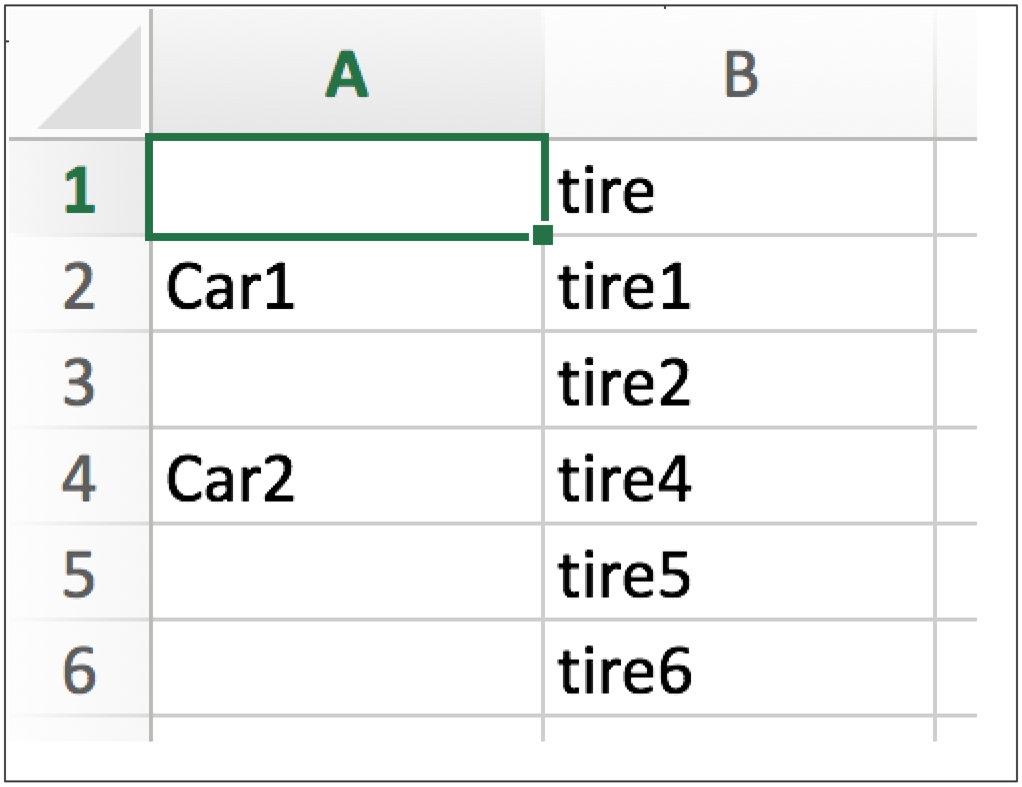

bdd [Package] csv_import_dots [Resulting structure from import]
«block»
Car
values
Name : String
Mass : Integer
Horsepower : String
Price : Real
Good : Boolean
value1
carsToTiresAss
tire
«block»
Tire
values
mass : Real
type : String
velocity : Real
«block»
Car1
values
Name : String = Ford Focus{redefines 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}
«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}
valve
«block»
Valve
values
type : String
tireToValveGeneral
thing
thing
{redefines thing}
«block»
Thing
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}

Figure 9. 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

Figure 10. Table as template in CSV.

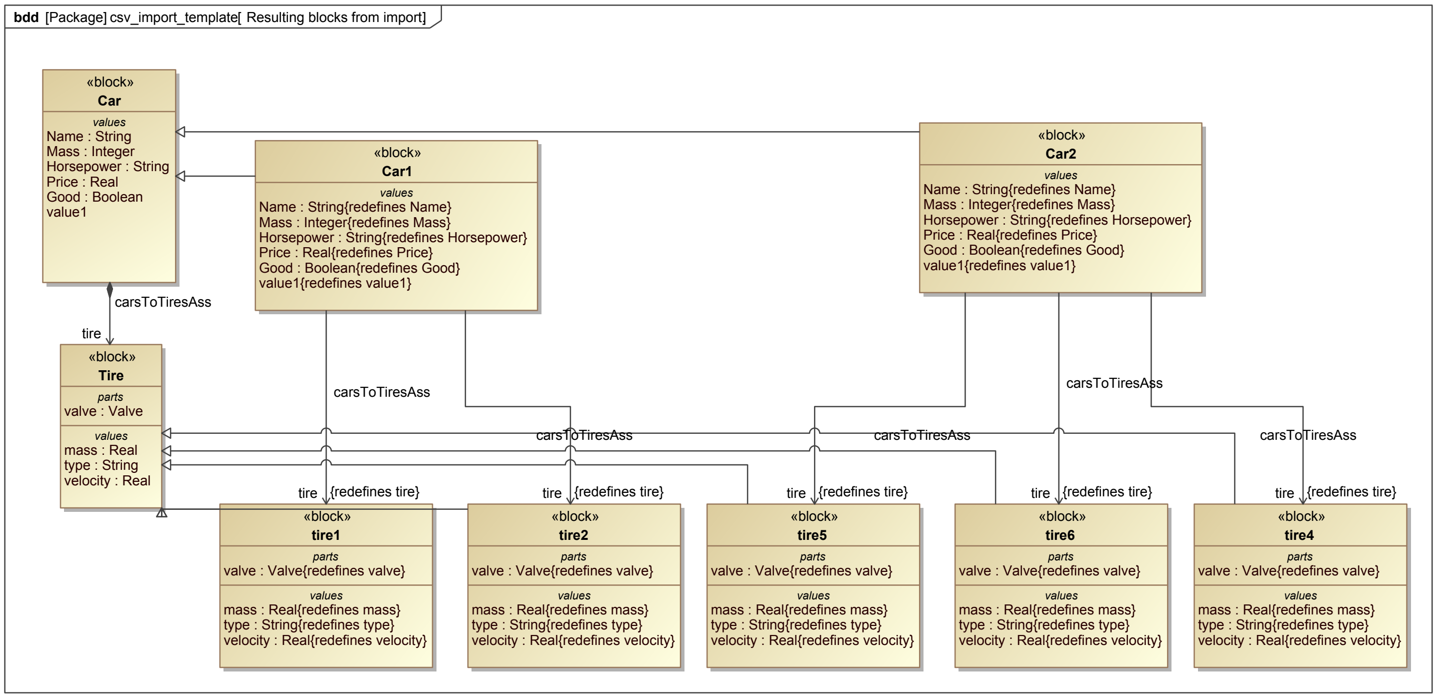

bdd [Package] csv_import_template[Resulting blocks from import]
«block»
Car
values
Name : String
Mass : Integer
Horsepower : String
Price : Real
Good : Boolean
value1
«block»
Car1
values
Name : String{redefines Name}
Mass : Integer{redefines Mass}
Horsepower : String{redefines Horsepower}
Price : Real{redefines 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}

Figure 11. 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.

2.7 Add Aspects Action

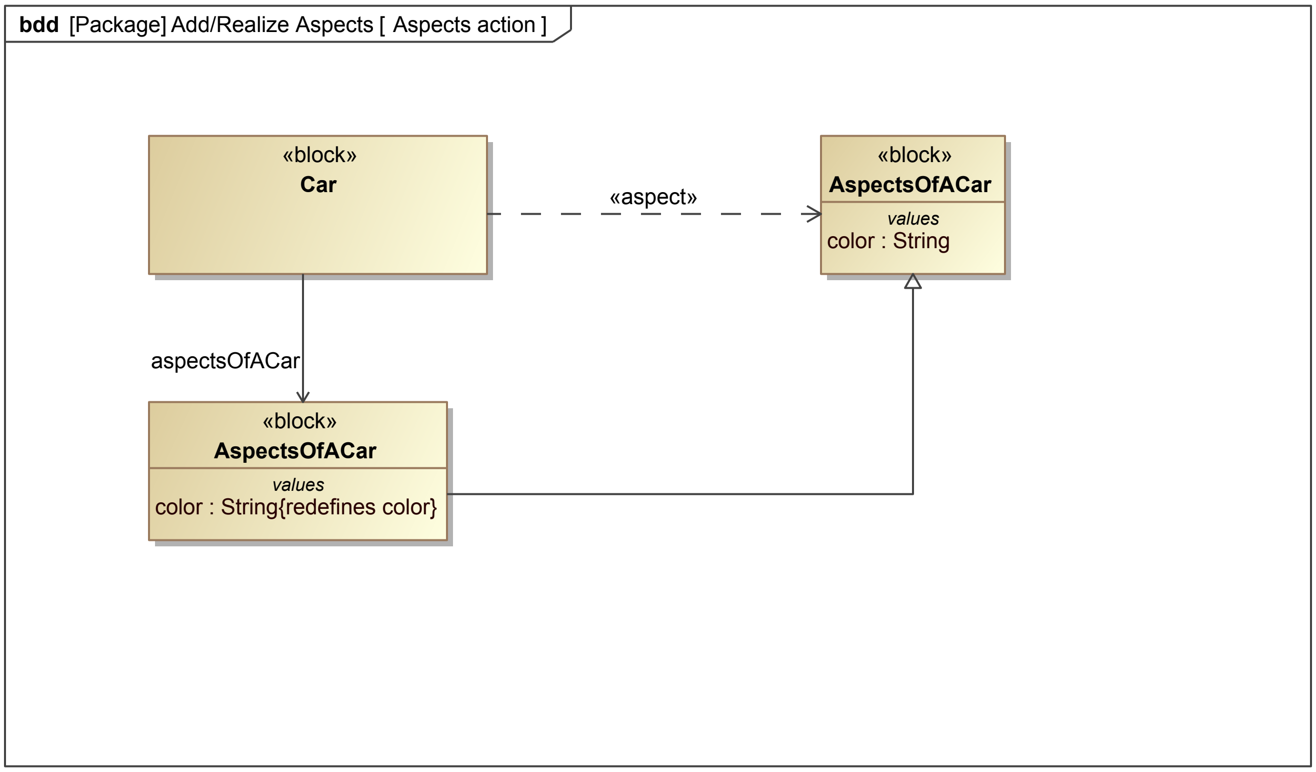


Figure 12. Aspects action

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.

3 WBS/Orgchart Pattern

3.1 Intent

The WBS/Orgchart Pattern provides a reusable modeling construct to specify the workflow between an organizational domain's projects, missions, and persons. The term "Orgchart" refers to the encompassing view of an organizational domain. While Work Breakdown Structure (WBS) refers to the roles of each person in the organizational domain and their deliverables.

3.2 Motivation

The WBS/Orgchart Pattern can be applied to any organizational domain that requires a method to define the structure of their organization, projects, missions, and persons, and to specify the connections between them. To apply the pattern the modelers domain, they will need to specialize from the pattern definition. See the Section 3.7.1 view of the Tooling section for more information.

3.3 Concept

Figure 13. Organization Chart Library

3.4 Consequences

3.5 Implementation

The sample model demonstrates how to apply the WBS/Orgchart Pattern to the modeler's own Organizational Domain. For this example it is assumed that the modeler had already specialized from the pattern's Organizational Domain block. For more information on how to specializing using the Systems Reasoner feature, view the Section 3.7.1 section.

Figure 14. Example Organization

Figure 15. My Organizational Domain

IBDs are used to provide an overview of an Organizational Domain. In this diagram 1

3.6 Known Uses

3.7 Tooling

3.7.1 Model Development Kit

The Systems Reasoner feature of the Model Development Kit plugin allows users to "Specialize Structure Recursively". The specialize structure recursively action can be used to create a specialized Block with all redefinable Elements redefined on the specialized level.

The Specialize Structure Recursively feature should be used to specialize from the WBS/Orgchart Pattern definition. While the pattern structure could be recreated manually, it is more efficient to use the feature.

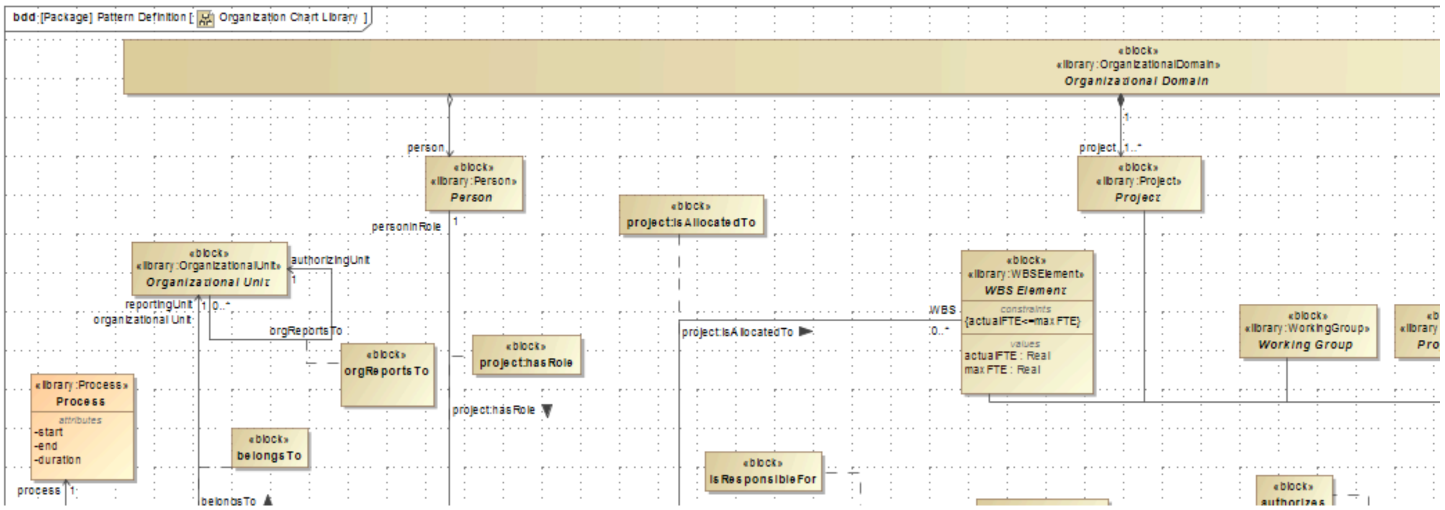

bdd [Package] Pattern Definition [Organization Chart Library]
«block» «library:OrganizationalDomain» Organizational Domain
person
«block» «library:Person» Person
personInRole
project 1..*
«block» «library:Project» Project
«block» project:isAllocatedTo
«block» «library:OrganizationalUnit» Organizational Unit
authorizingUnit
reportingUnit
organizational Unit
orgReportsTo
«block» orgReportsTo
«block» project:hasRole
project:hasRole
«block» «library:WBSElement» WBS Element
constraints {actualFTE<=maxFTE}
values actualFTE : Real maxFTE : Real
WBS
project:isAllocatedTo
«block» «library:WorkingGroup» Working Group
«library:Process» Process
attributes -start -end -duration
«block» belongsTo
belongsTo
process
«block» isResponsibleFor
«block» authorizes

Figure 16. Systems Reasoner Organizational Domain

By specializing the structure of the WBS/Orgchart Pattern, the modeler is able to reuse the pattern in their own context. The first step is to either navigate to the Organizational Domain either in the Containment Tree or in the Organization Chart Library diagram as seen above. After navigating to the Organizational Domain, right click the block, select "**Systems Reasoner**", and then select "**Specialize Structure Recursively**". In the next window you will be asked to select the container for the generated elements, and select "**OK**".

After performing the "Specialize Structure Recursively" action, a new Organizational Domain block will be generated from the pattern Organizational Domain block. All generated relations will redefine the relations from the pattern.

3.8 Related Patterns