

44. Domain-Specific Languages - Modular Metamodels in Reuseware, based on Invasive Composition

1

Prof. Dr. Uwe Aßmann

Dr. Jendrik Johannes

Technische Universität Dresden

Institut für Software- und
Multimediatechnik

<http://st.inf.tu-dresden.de>

Version 13-1.0, 01.01.14

- 1) The DSL Taipan
- 2) Reuseware
- 3) Extending the metamodel of Taipan for modularity
- 4) Reuseware tool

DevBoost

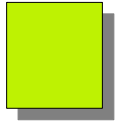
 reuseware
composition framework

Obligatory Literature

2

- ▶ [1] Jakob Henriksson, Jendrik Johannes, Steffen Zschaler, and Uwe Aßmann. Reuseware - adding modularity to your language of choice. Journal of Object Technology, 6(9):127-146, 2007. On Language-Independent Model Modularisation, Transactions on Aspect-Oriented Development, 2008
- ▶ [2] <http://reuseware.org>
- ▶ [3] [http://wiki.eclipse.org/index.php/GMF Tutorial#Quick Start](http://wiki.eclipse.org/index.php/GMF_Tutorial#Quick_Start)

44.1 Reuse Languages and Metamodel Modularity



3

44.1 Building Modularisation into Taipan DSL

4

A reuse (sub-)language is a sublanguage providing modularity

- ▶ Languages need modularization concepts to improve reusability and reduce complexity of applications and tools
- ▶ Challenges of modularization (on M1):
 - Modularization needs reuse concepts in syntax and semantics
- ▶ Requirement for the reuse language on M2:
 - The reuse language itself should be modular, to be composable with other languages
 - The metamodel of a reuse language should be an M2-module
 - Reuse languages requires additional tooling support
- ▶ We have already discussed role-based metamodel composition
 - Here we show how to use invasive composition for metamodel components on M2 and their composition
- ▶ A metamodel composition system is a composition system for

44.1 Building Modularisation into Taipan DSL

5

A reuse (sub-)language is a sublanguage providing modularity

- ▶ Languages need modularization concepts to improve reusability and reduce complexity of applications and tools
- ▶ Challenges of modularization (on M1):
 - Modularization needs reuse concepts in syntax and semantics
- ▶ Requirement for the reuse language on M2:
 - The reuse language itself should be modular, to be composable with other languages
 - The metamodel of a reuse language should be an M2-module
 - Reuse languages requires additional tooling support
- ▶ We have already discussed role-based metamodel composition
 - Here we show how to use invasive composition for metamodel components on M2 and their composition
- ▶ A metamodel composition system is a composition system for

Metamodel Composition

6

- ▶ This chapter presents a toolkit to build reuse languages
 - based on invasive metamodel composition, implemented in the Reuseware toolkit [1][2]
 - Does not influence design of DSL syntax or semantics
 - DSL syntax can be extended at the end
 - Composes modularized models to monolithic models
 - DSL semantics do not require extension
 - Generic tooling can be used with arbitrary DSLs

Building Modularisation into a DSL

7

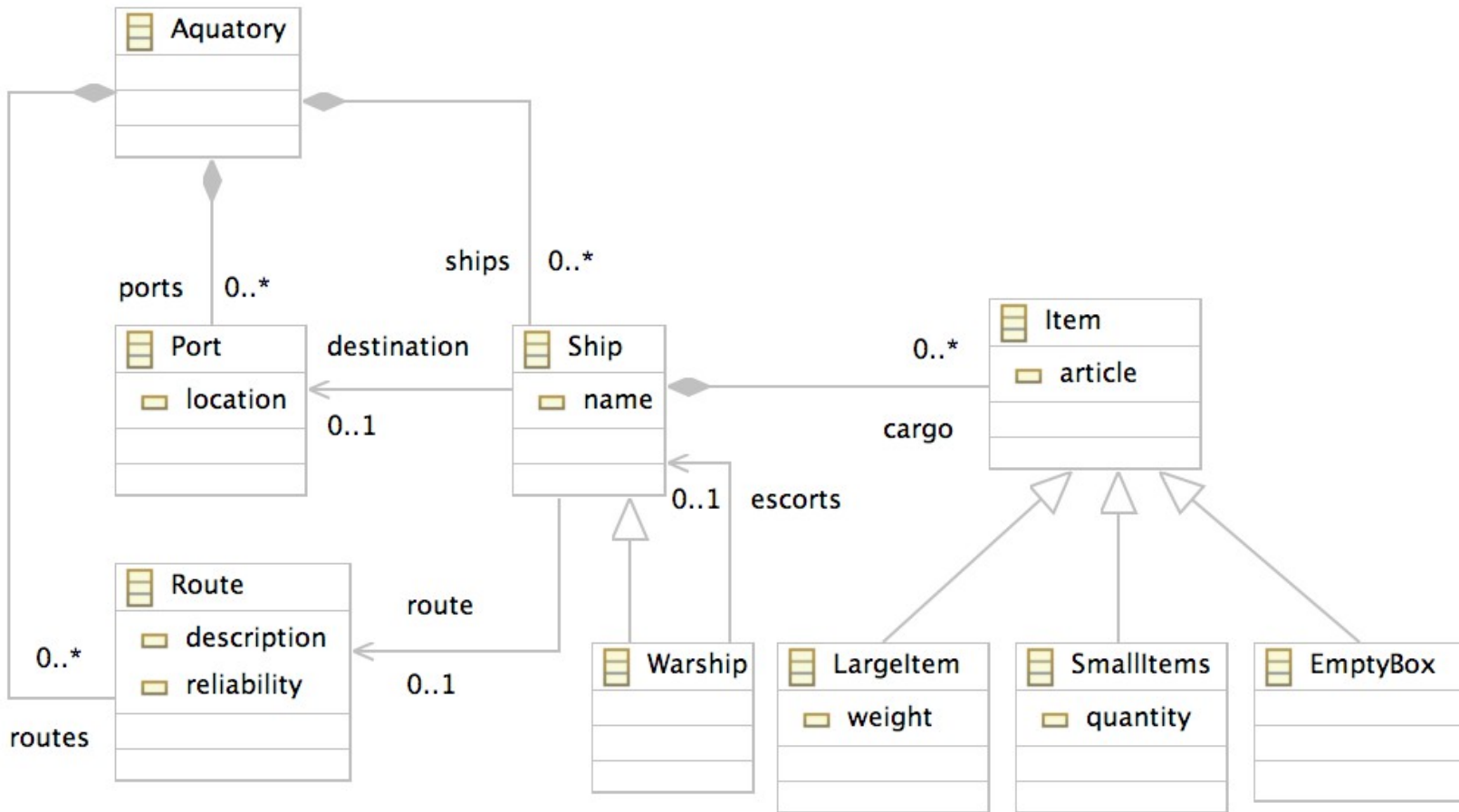
► Reuseware approach

- Define a ***composition system*** with modularisation concepts (see CBSE course)
- Composition systems define **component model**
 - E.g., Modules, Packages, Aspects, etc.
- **Composition techniques**
 - E.g., parameterization, extension, weavings
- And **composition languages**
 - For the structure in the large
- Optional: Extend DSL syntax with concepts for variation points
 - Variation points allow definition of templates
- Define a reuse extension for your DSL
 - Binds the composition system to your DSL
 - E.g., what are the specifics of a module in your DSL, what identifies an aspect, etc.
- Reuseware can handle modularization in your DSL

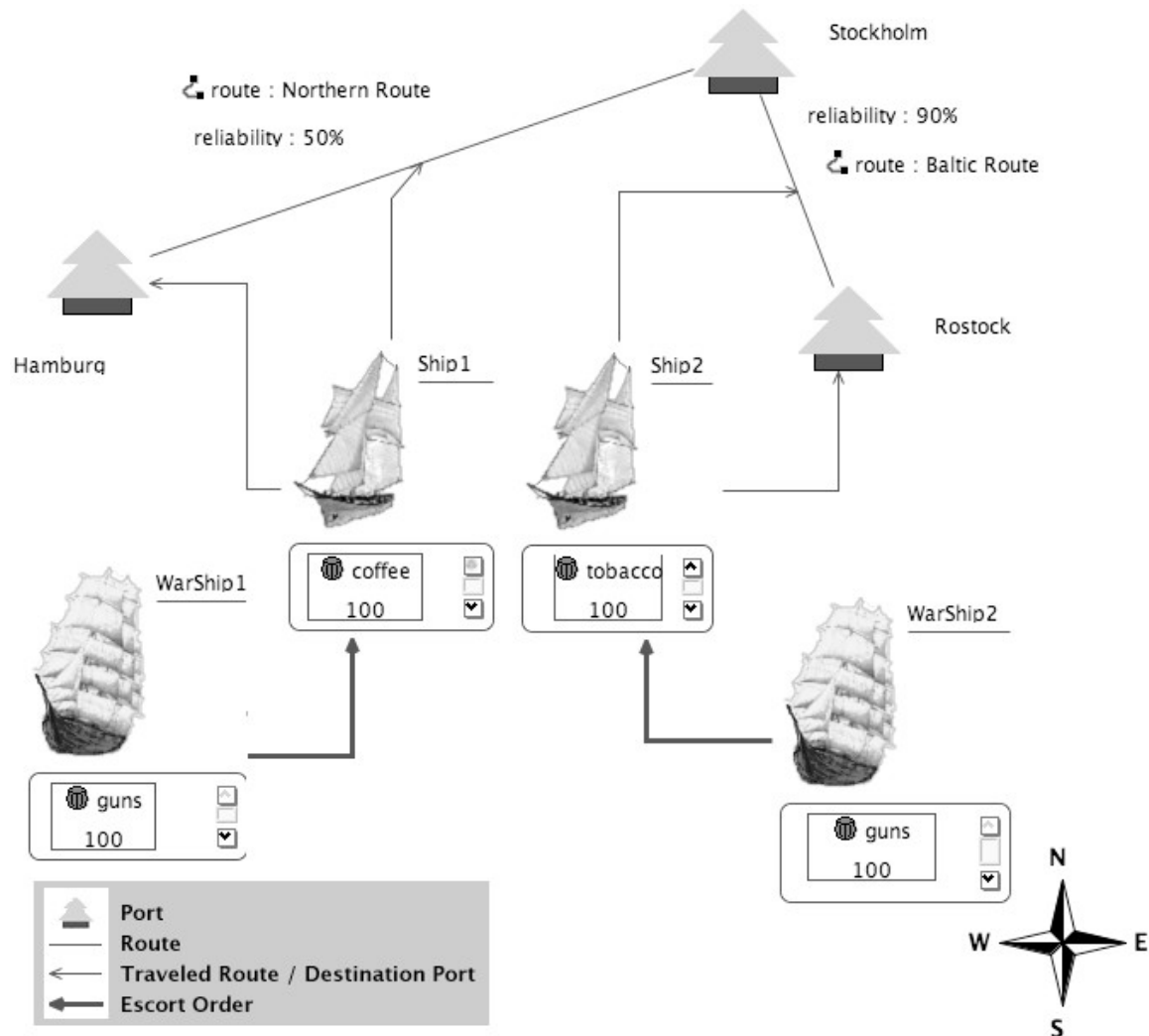
Building a DSL: Modularisation – Example

8

- ▶ Taipan DSL^[3] for modeling ship fleets (Metamodel excerpt)

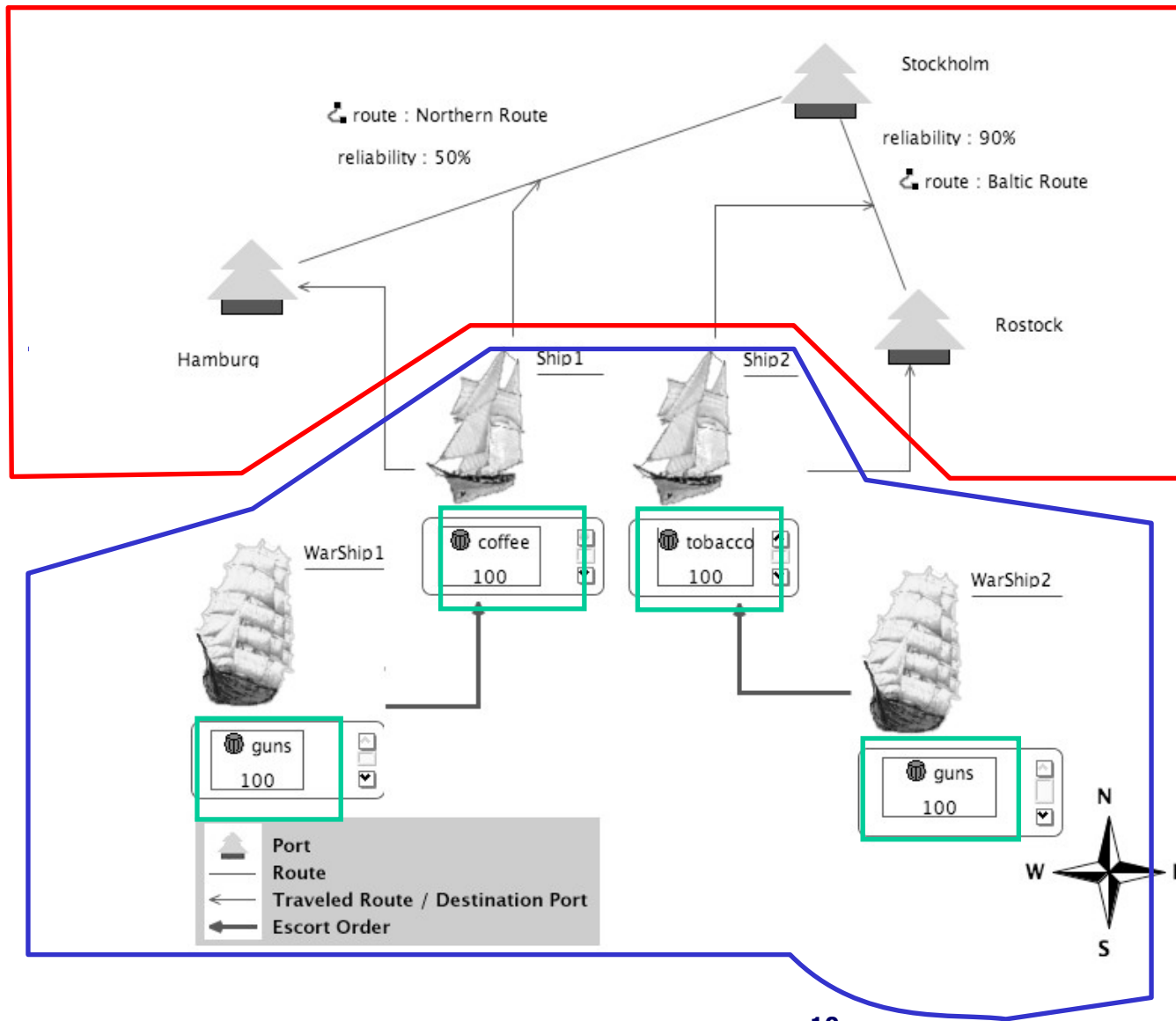


A Specification in the Taipan DSL: A Model with Ships



Building a DSL: Modularisation of Metamodel

10



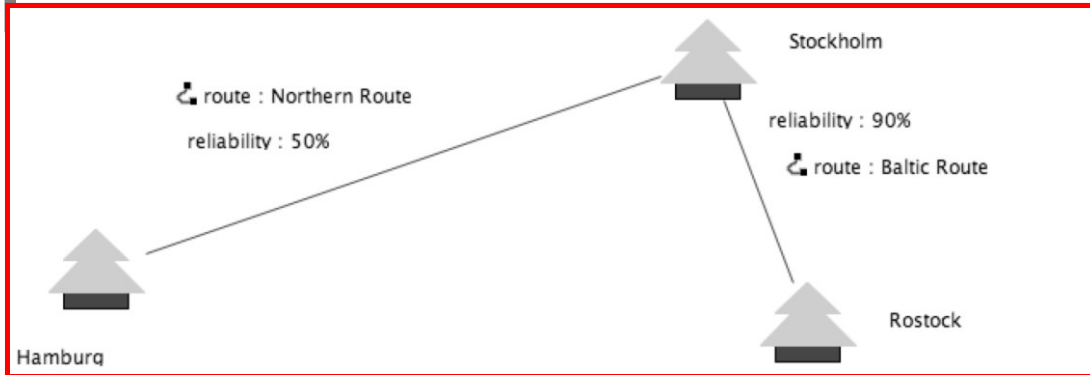
Different concerns should be separated into model fragments

- Port model (configuration of ports and routes)
- Flotilla model (ships and their relations)
- Cargo model (Cargo and its properties)



Building a DSL: Modularisation of Metamodel

11

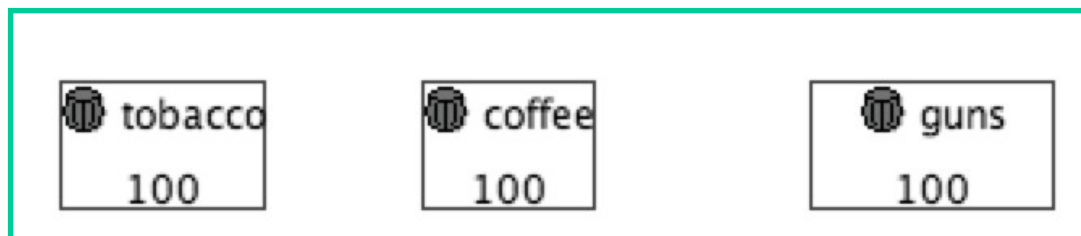
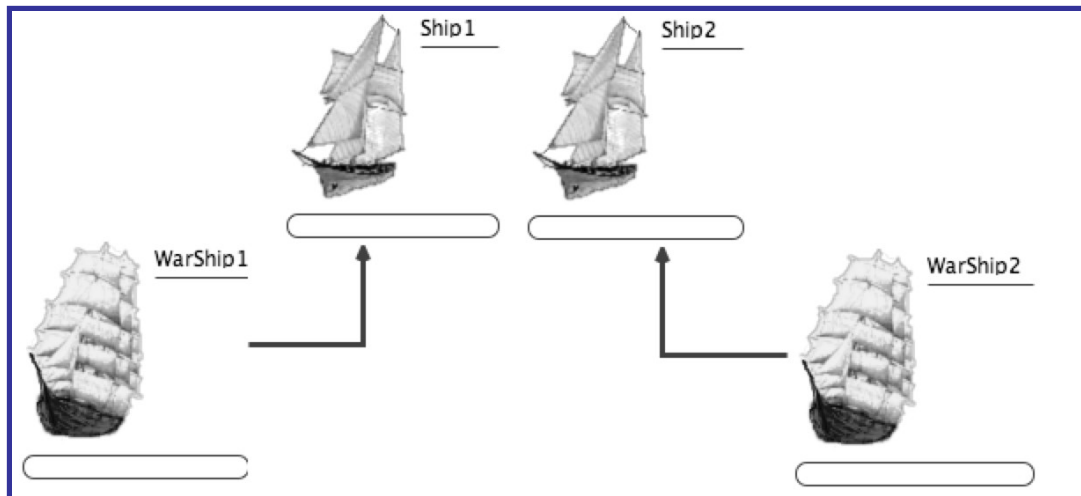


Different concerns should be separated into model fragments

- Port model (configuration of ports and routes)

- Flotilla model (ships and their relations)

- Cargo model (Cargo and its properties)



44.2 The Reuseware MetaCASE Tool - Overview

12

- ▶ **Model fragments** (model snippets) are partial models that may contain variation points
 - Offer a **Composition Interface**
 - **Composition Interface** consists of **Ports**
 - **Ports** point at elements of the model fragment that can be accessed for composition
- ▶ **Composition Programs**
 - Define **composition links** between Ports
 - Can be executed to produce a composed model where model fragments are merged at the elements pointed out by the linked Ports

Building a DSL: Reuseware - Overview

13

- ▶ Composition Systems
 - Define modularisation concepts (e.g., Modules, Packages, Aspects)
 - Define relations between modularisation concepts (e.g, an aspect relates to a core)
- ▶ Reuse extensions (for DSLs)
 - Define how modularization concepts defined in a composition system are realized in a concrete DSL
 - Define which ports are related to which model elements of a model fragment

Defining Composition Systems with Reuseware

14

- ▶ A composition system defines fragment components with
 - **Fragment roles**
 - Role a model fragment plays in the modularisation (e.g., aspect or core)
 - Fragment roles collaborate through associations between ports
 - **Static ports of a fragment component**
 - Defined for one fragment role
 - Each fragment playing the role has to offer the port
 - **Dynamic ports**
 - Defined for one fragment role
 - Each fragment playing the role can offer several of these ports
 - **Contribution Associations**
 - Defines that two ports are related
 - Executing a composition link between the two ports will trigger the copying of model elements
 - **Configuration Associations**
 - Defines that two ports are related
 - Executing a composition link between the two ports will NOT trigger the copying of model elements

ReuseTaipan - a Composition System for the Taipan Metamodel, Specified in Reuseware-FraCL

15

```
compositionsystem reuseTaipan {  
  
  fragment role TravelSpace {  
    static port VehicleContainer;  
    dynamic port Routes;  
    dynamic port Places;  
  }  
  
  fragment role Flotilla {  
    static port Vehicles;  
    dynamic port RouteSlots;  
    dynamic port PlaceSlots;  
  }  
  
  contribution Flotilla.Vehicles --> TravelSpace.VehicleContainer;  
  configuration Flotilla.RouteSlots --> TravelSpace.Routes;  
  configuration Flotilla.PlaceSlots --> TravelSpace.Places;  
  
  fragment role ItemHolder {  
    dynamic port ItemSpaces;  
  }  
  
  fragment role ItemContainer {  
    dynamic port Items;  
  }  
  
  contribution ItemContainer.Items --> ItemHolder.ItemSpaces;  
}
```

Building a DSL: ReuseTaipan - a Composition System

16

```
compositionsystem reuseTaipan {  
  fragment role TravelSpace {  
    static port VehicleContainer;  
    dynamic port Routes;  
    dynamic port Places;  
  }  
  
  fragment role Flotilla {  
    static port Vehicles;  
    dynamic port RouteSlots;  
    dynamic port PlaceSlots;  
  }  
  
  contribution Flotilla.Vehicles --> TravelSpace.VehicleContainer;  
  configuration Flotilla.RouteSlots --> TravelSpace.Routes;  
  configuration Flotilla.PlaceSlots --> TravelSpace.Places;  
  
  fragment role ItemHolder {  
    dynamic port ItemSpaces;  
  }  
  
  fragment role ItemContainer {  
    dynamic port Items;  
  }  
  
  contribution ItemContainer.Items --> ItemHolder.ItemSpaces;  
}
```

A **TravelSpace** offers a place where vehicles can be placed (**VehicleContainer**) and a number of **Routes** and **Places**

Building a DSL: ReuseTaipan - a Composition System

17

```
compositionsystem reuseTaipan {  
  
  fragment role TravelSpace {  
    static port VehicleContainer;  
    dynamic port Routes;  
    dynamic port Places;  
  }  
  
  fragment role Flotilla {  
    static port Vehicles;  
    dynamic port RouteSlots;  
    dynamic port PlaceSlots;  
  }  
  
  contribution Flotilla.Vehicles --> TravelSpace.VehicleContainer;  
  configuration Flotilla.RouteSlots --> TravelSpace.Routes;  
  configuration Flotilla.PlaceSlots --> TravelSpace.Places;  
  
  fragment role ItemHolder {  
    dynamic port ItemSpaces;  
  }  
  
  fragment role ItemContainer {  
    dynamic port Items;  
  }  
  
  contribution ItemContainer.Items --> ItemHolder.ItemSpaces;  
}
```

A Flotilla offers a set of **Vehicles** and has a number of placeholders for routes (**RouteSlots**) and places (**PlaceSlots**)

Building a DSL: ReuseTaipan - a Composition System

18

```
compositionsystem reuseTaipan {  
  
  fragment role TravelSpace {  
    static port VehicleContainer;  
    dynamic port Routes;  
    dynamic port Places;  
  }  
  
  fragment role Flotilla {  
    static port Vehicles;  
    dynamic port RouteSlots;  
    dynamic port PlaceSlots;  
  }  
  
  contribution Flotilla.Vehicles --> TravelSpace.VehicleContainer;  
  configuration Flotilla.RouteSlots --> TravelSpace.Routes;  
  configuration Flotilla.PlaceSlots --> TravelSpace.Places;  
  
  fragment role ItemHolder {  
    dynamic port ItemSpaces;  
  }  
  
  fragment role ItemContainer {  
    dynamic port Items;  
  }  
  
  contribution ItemContainer.Items --> ItemHolder.ItemSpaces;  
}
```

A Flotilla contributes **Vehicles** to a **TravelSpace's VehicleContainer**; a **RouteSlots** can be configured with a **Route**; a **PlaceSlots** can be configured with a **Place**

Building a DSL: ReuseTaipan - a Composition System

19

```
compositionsyntax reuseTaipan {  
  
  fragment role TravelSpace {  
    static port VehicleContainer;  
    dynamic port Routes;  
    dynamic port Places;  
  }  
  
  fragment role Flotilla {  
    static port Vehicles;  
    dynamic port RouteSlots;  
    dynamic port PlaceSlots;  
  }  
  
  contribution Flotilla.Vehicles --> TravelSpace.VehicleContainer;  
  configuration Flotilla.RouteSlots --> TravelSpace.Routes;  
  configuration Flotilla.PlaceSlots --> TravelSpace.Places;  
  
  fragment role ItemHolder {  
    dynamic port ItemSpaces;  
  }  
  
  fragment role ItemContainer {  
    dynamic port Items;  
  }  
  
  contribution ItemContainer.Items --> ItemHolder.ItemSpaces;  
}
```

An ItemHolder offers different ItemSpaces

Building a DSL: ReuseTaipan - a Composition System

20

```
compositionsystem reuseTaipan {  
  
  fragment role TravelSpace {  
    static port VehicleContainer;  
    dynamic port Routes;  
    dynamic port Places;  
  }  
  
  fragment role Flotilla {  
    static port Vehicles;  
    dynamic port RouteSlots;  
    dynamic port PlaceSlots;  
  }  
  
  contribution Flotilla.Vehicles --> TravelSpace.VehicleContainer;  
  configuration Flotilla.RouteSlots --> TravelSpace.Routes;  
  configuration Flotilla.PlaceSlots --> TravelSpace.Places;  
  
  fragment role ItemHolder {  
    dynamic port ItemSpaces;  
  }  
  
  fragment role ItemContainer {  
    dynamic port Items;  
  }  
  
  contribution ItemContainer.Items --> ItemHolder.ItemSpaces;  
}
```

An ItemContainer contains and offers Items

Building a DSL: ReuseTaipan - a Composition System

21

```
compositionsystem reuseTaipan {  
  
  fragment role TravelSpace {  
    static port VehicleContainer;  
    dynamic port Routes;  
    dynamic port Places;  
  }  
  
  fragment role Flotilla {  
    static port Vehicles;  
    dynamic port RouteSlots;  
    dynamic port PlaceSlots;  
  }  
  
  contribution Flotilla.Vehicles --> TravelSpace.VehicleContainer;  
  configuration Flotilla.RouteSlots --> TravelSpace.Routes;  
  configuration Flotilla.PlaceSlots --> TravelSpace.Places;  
  
  fragment role ItemHolder {  
    dynamic port ItemSpaces;  
  }  
  
  fragment role ItemContainer {  
    dynamic port Items;  
  }  
  
  contribution ItemContainer.Items --> ItemHolder.ItemSpaces;  
}
```

Items can be individually assigned to ItemSpaces

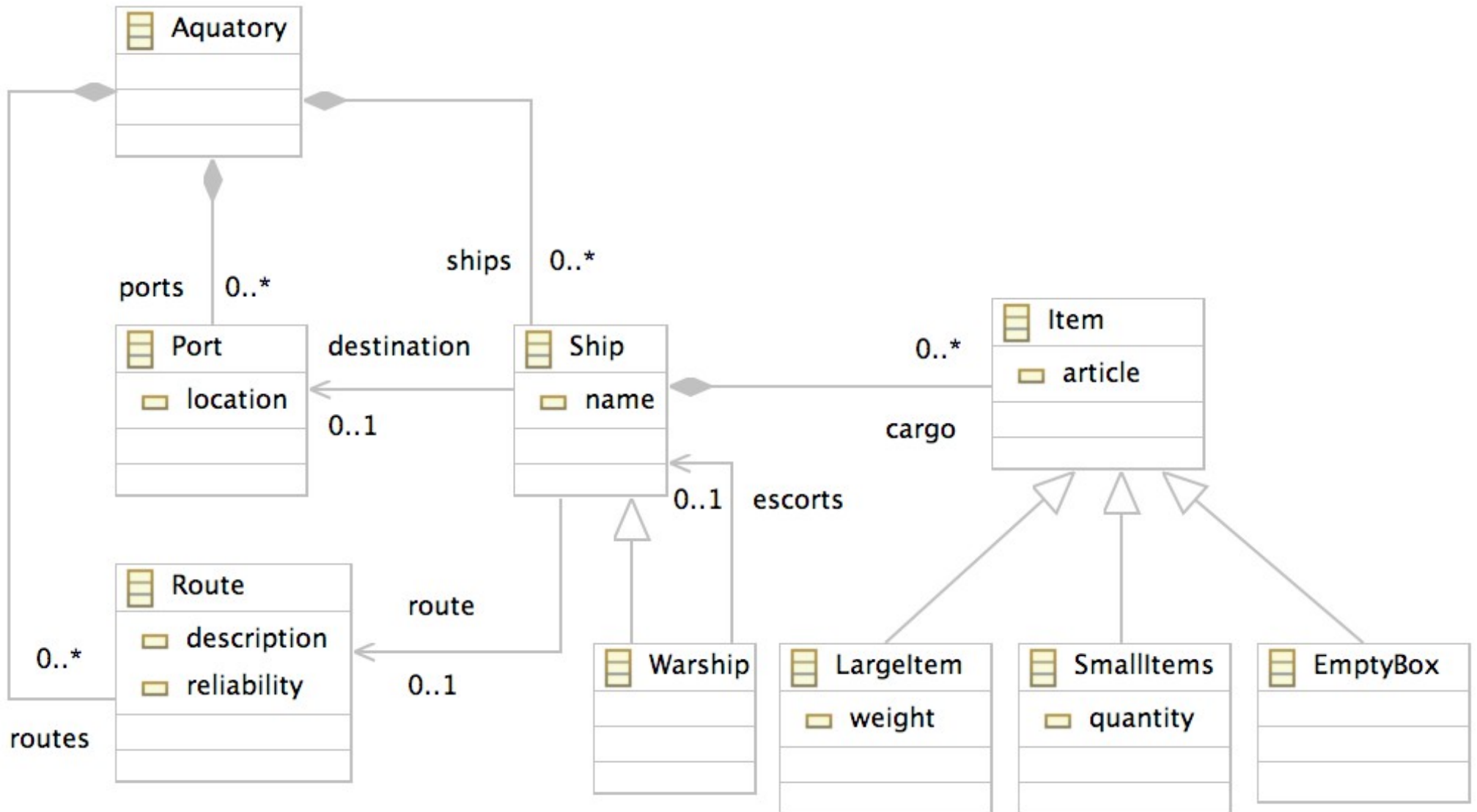
44.3 Building a DSL: Extending a Metamodel for Variation

22

- ▶ Three kinds of variation points required in the metamodels
 - RouteSlot
 - PortSlot
 - ItemSpace
- ▶ For each kind of variation point we...
 - Introduce a superclass for the metaclass that defines the elements which may replace the variation point
 - e.g., we introduce **RouteType** as a superclass of **Route** in the case of RouteSlot
 - We redirect all references to the metaclass to the new superclass
 - e.g., all references to **Route** are redirected to **RouteType**
 - We introduce a new subclass for the just introduced superclass that represents the variation point. This class needs properties from which a name can be derived.
 - e.g., we introduce **RouteSlot** as a subclass of **RouteType**

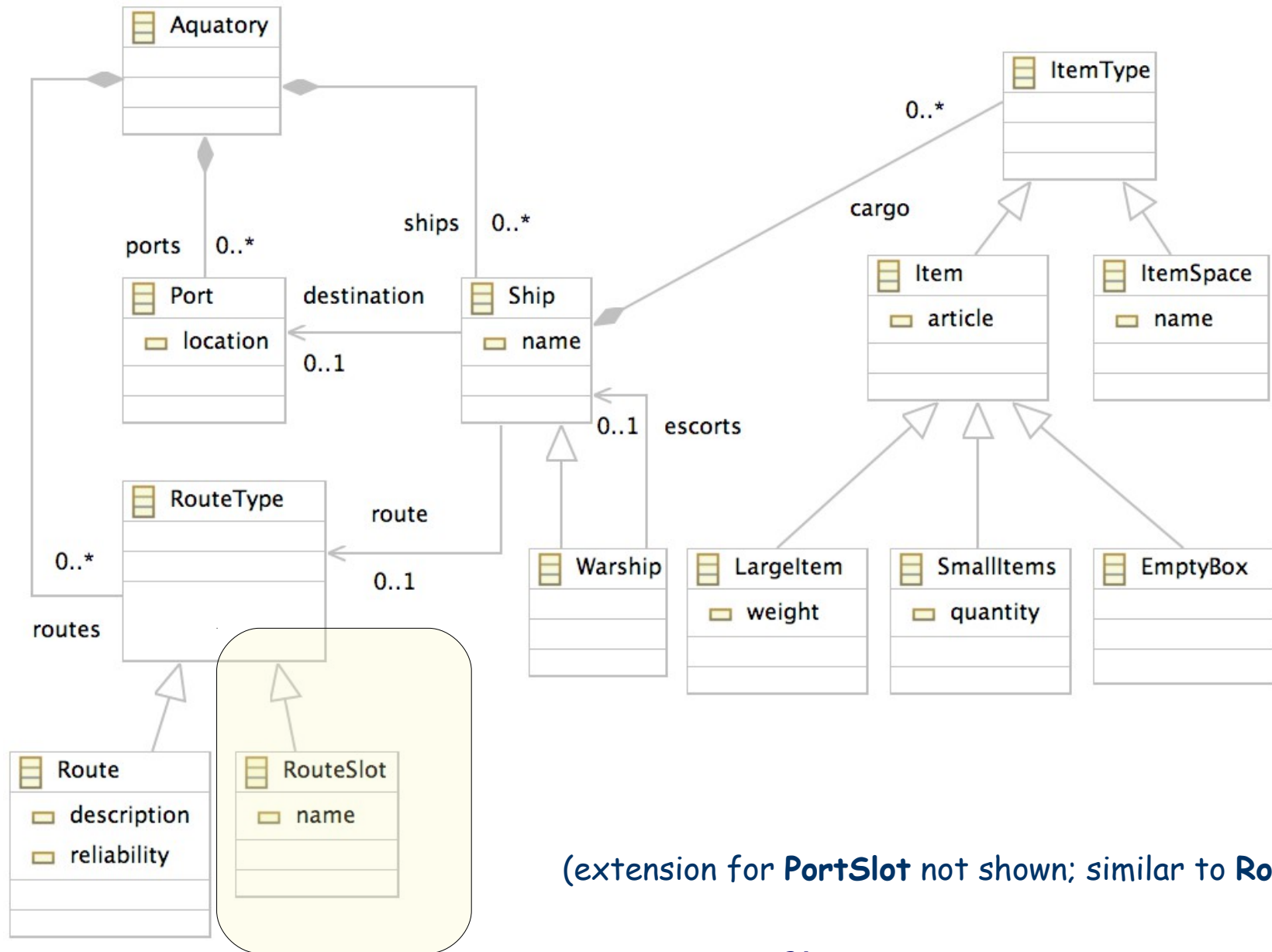
The Taipan Metamodel (Rpt.)

23



Extending the Taipan Metamodel for Variation

24



(extension for **PortSlot** not shown; similar to **RouteSlot**)



Building a DSL: Reuseware - Reuse Extensions

25

- ▶ A **reuse extension of a metamodel** is an extended metamodel defining
 - How a composition interface defined by a fragment role (which is defined in a composition system) is linked to the content of a model fragment
 - Each port links to a set of model elements treated as:
 - Prototype: Element that can be copied with its contained elements
 - Anchor: Element that can be referenced by other elements
 -
 - Hook: Variation point where Prototypes can be put
 - Slot: Variation point where Anchors can be put
- ▶ Reuseware-CL is a language to define reuse extensions of metamodels
 - to make a metamodel composable
 -

Building a DSL: Binding ReuseTaipan to Taipan DSL

```
reuseextension reuseTaipan implements reuseTaipan
epackages <http://www.eclipse.org/examples/gmf/taipan>
RootClass TravelSpace {
  fragment role TravelSpace {
    port VehicleContainer {
      Aquatory.ships is hook {}
      Aquatory.ports is hook {}
      Aquatory.routes is hook {}
    }
    port Routes {
      Route is anchor {
        port expr = $self.description$
      }
    }
    port Places {
      Port is anchor {
        port expr = $self.location.concat('Port')$
      }
    }
  }
}

fragment role Flotilla {
  port Vehicles {
    Aquatory.ships is prototype {}
    Aquatory.ports is prototype {}
    Aquatory.routes is prototype {}
  }
  port RouteSlots {
    RouteSlot is slot {
      port expr = $self.name$
    }
  }
  port PlaceSlots {
    PortSlot is slot {
      port expr = $self.name$
    }
  }
}
...

```

The ReuseTaipan composition system is bound to the Taipan DSL (referred to by the URI of its metamodel)

Building a DSL: Binding ReuseTaipan to Taipan DSL

```
reuseextension reuseTaipan implements reuseTaipan
epackages <http://www.eclipse.org/examples/gmf/taipan>
Rootclass TravelSpace {
  fragment role TravelSpace {
    port VehicleContainer {
      Aquatory.ships is hook {}
      Aquatory.ports is hook {}
      Aquatory.routes is hook {}
    }
    port Routes {
      Route is anchor {
        port expr = $self.description$
      }
    }
    port Places {
      Port is anchor {
        port expr = $self.location.concat('Port')$
      }
    }
  }
}

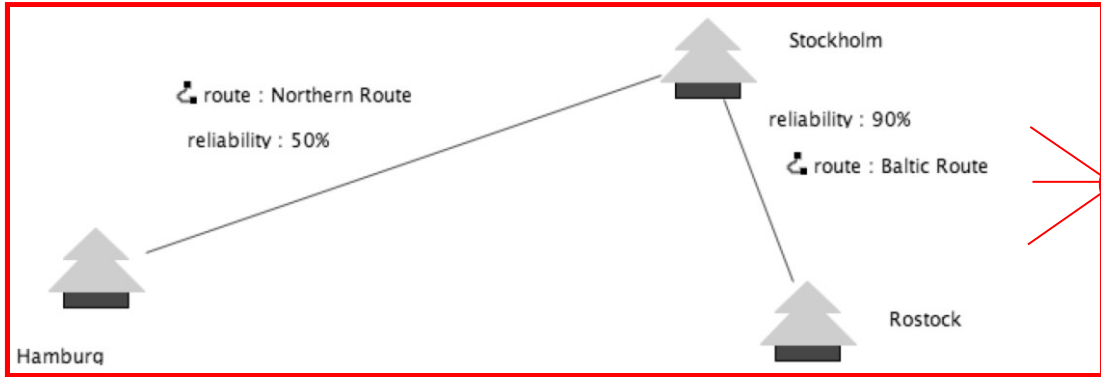
fragment role Flotilla {
  port Vehicles {
    Aquatory.ships is prototype {}
    Aquatory.ports is prototype {}
    Aquatory.routes is prototype {}
  }
  port RouteSlots {
    RouteSlot is slot {
      port expr = $self.name$
    }
  }
  port PlaceSlots {
    PortSlot is slot {
      port expr = $self.name$
    }
  }
}
...

```

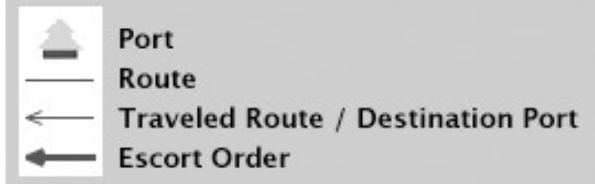
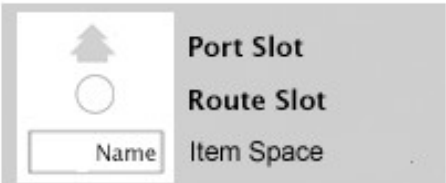
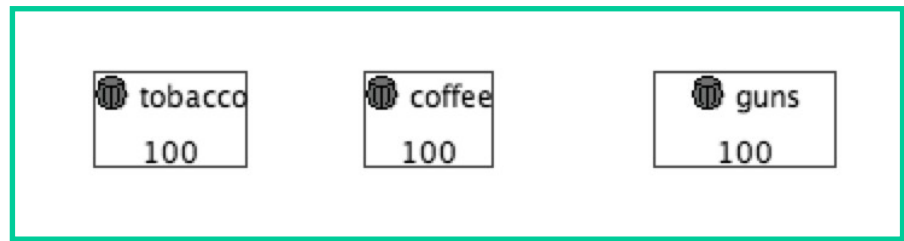
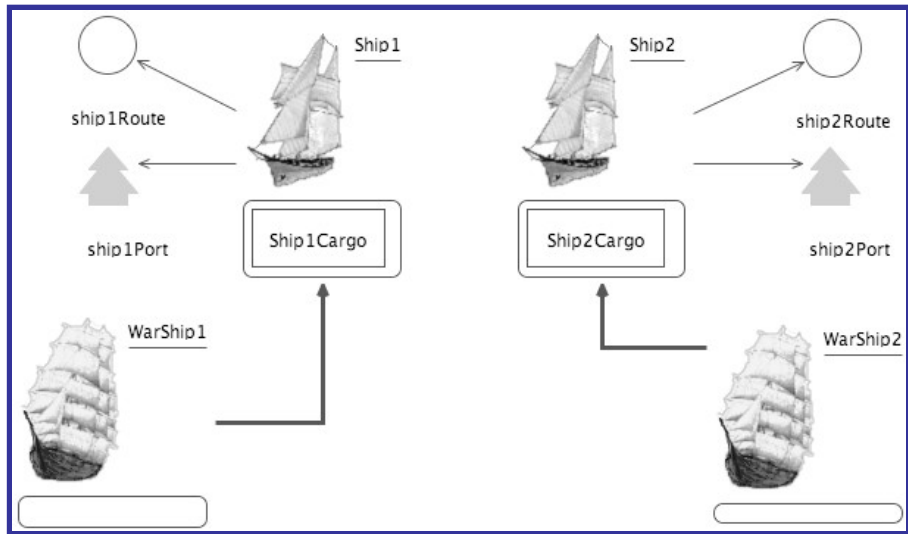
The references **ships**, **ports** and **routes** of the metaclass **Aquatory** all act as hooks accessible through the **VehicleContainer** port

Building a DSL: Binding ReuseTaipan to Taipan DSL

28



VehicleContainer



Building a DSL: Binding ReuseTaipan to Taipan DSL

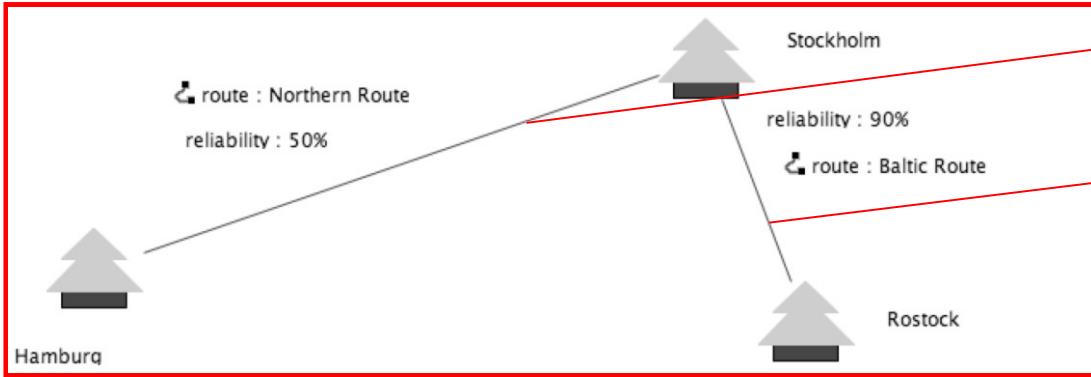
```
reuseextension reuseTaipan implements reuseTaipan
epackages <http://www.eclipse.org/examples/gmf/taipan>
RootClass TravelSpace {
  fragment role TravelSpace {
    port VehicleContainer {
      Aquatory.ships is hook {}
      Aquatory.ports is hook {}
      Aquatory.routes is hook {}
    }
    port Routes {
      Route is anchor {
        port expr = $self.description$
      }
    }
    port Places {
      Port is anchor {
        port expr = $self.location.concat('Port')$
      }
    }
  }

  fragment role Flotilla {
    port Vehicles {
      Aquatory.ships is prototype {}
      Aquatory.ports is prototype {}
      Aquatory.routes is prototype {}
    }
    port RouteSlots {
      RouteSlot is slot {
        port expr = $self.name$
      }
    }
    port PlaceSlots {
      PortSlot is slot {
        port expr = $self.name$
      }
    }
  }
  ...
}
```

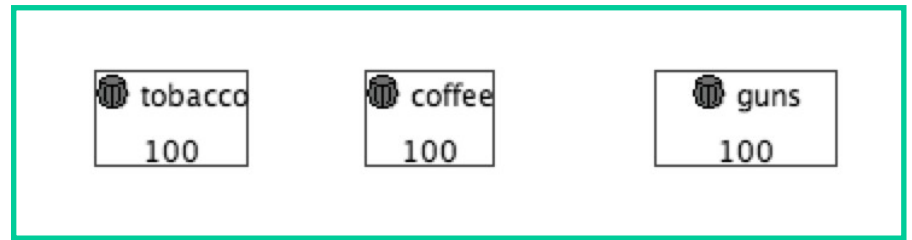
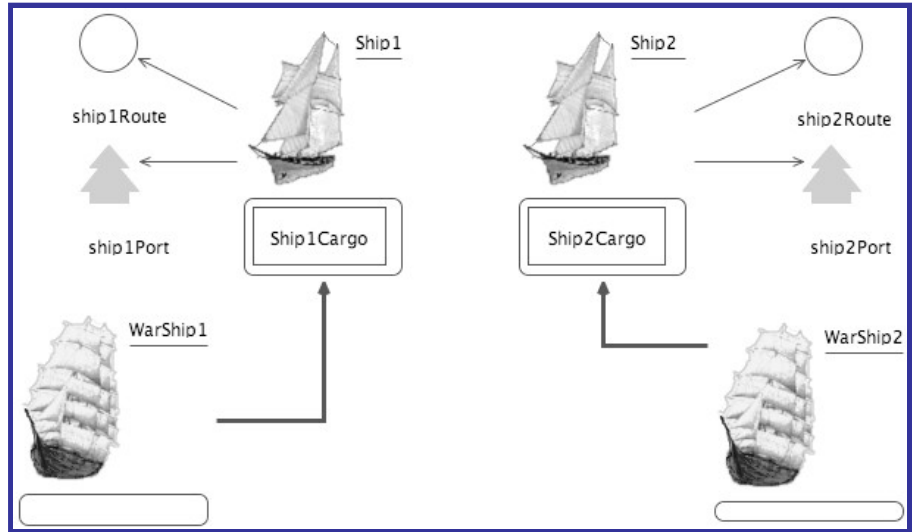
Each **Route** is an anchor accessible through individual ports; the ports are named using the **description** attribute of the **Route** metaclass
(*OCL Expression: self.description*)

Building a DSL: Binding ReuseTaipan to Taipan DSL Model Components

30



Northern Route
Baltic Route



	Port Slot
	Route Slot
<input type="text"/>	Name
<input type="text"/>	Item Space

	Port
	Route
	Traveled Route / Destination Port
	Escort Order



Building a DSL: Binding ReuseTaipan to Taipan DSL

```
reuseextension reuseTaipan implements reuseTaipan
epackages <http://www.eclipse.org/examples/gmf/taipan>
Rootclass TravelSpace {
  fragment role TravelSpace {
    port VehicleContainer {
      Aquatory.ships is hook {}
      Aquatory.ports is hook {}
      Aquatory.routes is hook {}
    }
    port Routes {
      Route is anchor {
        port expr = $self.description$
      }
    }
  }
  port Places {
    Port is anchor {
      port expr = $self.location.concat('Port')$
    }
  }
}

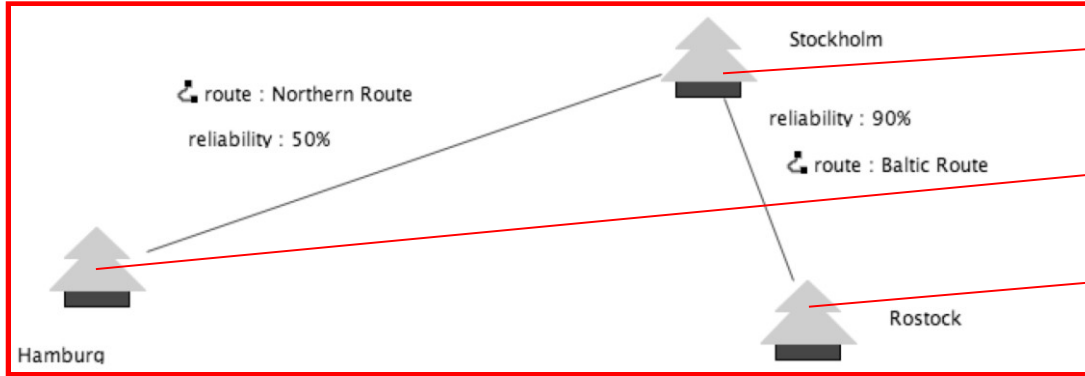
fragment role Flotilla {
  port Vehicles {
    Aquatory.ships is prototype {}
    Aquatory.ports is prototype {}
    Aquatory.routes is prototype {}
  }
  port RouteSlots {
    RouteSlot is slot {
      port expr = $self.name$
    }
  }
  port PlaceSlots {
    PortSlot is slot {
      port expr = $self.name$
    }
  }
}
...

```

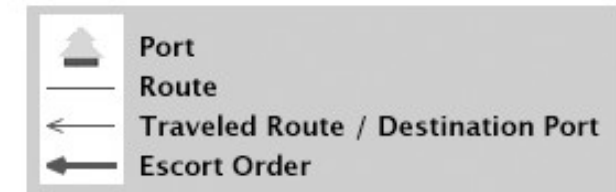
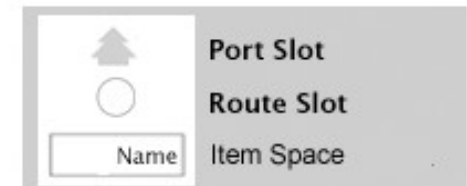
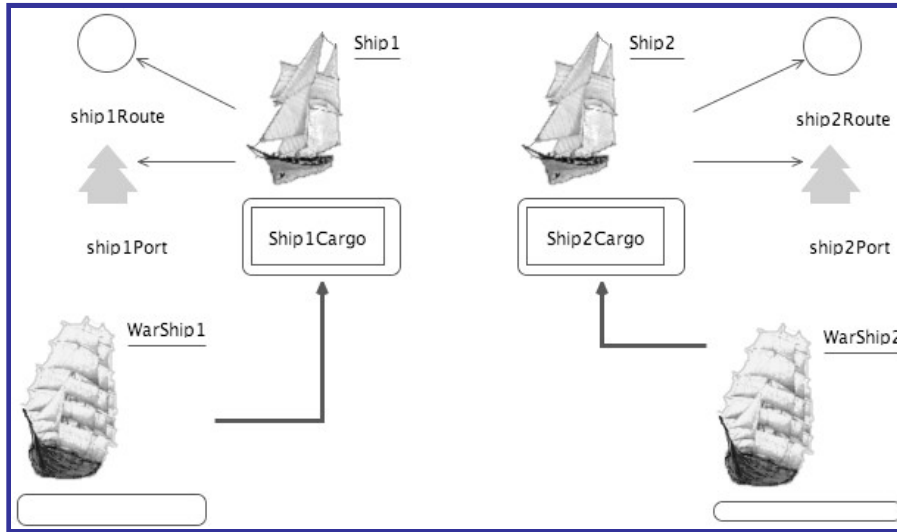
Each Port is an anchor accessible through individual ports; the ports are named using the **location** attribute of the Port metaclass

Building a DSL: Binding ReuseTaipan to Taipan DSL Model Components

32



-  **StockholmPort**
-  **HamburgPort**
-  **RostockPort**



Building a DSL: Binding ReuseTaipan to Taipan DSL

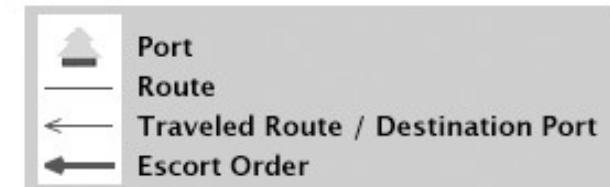
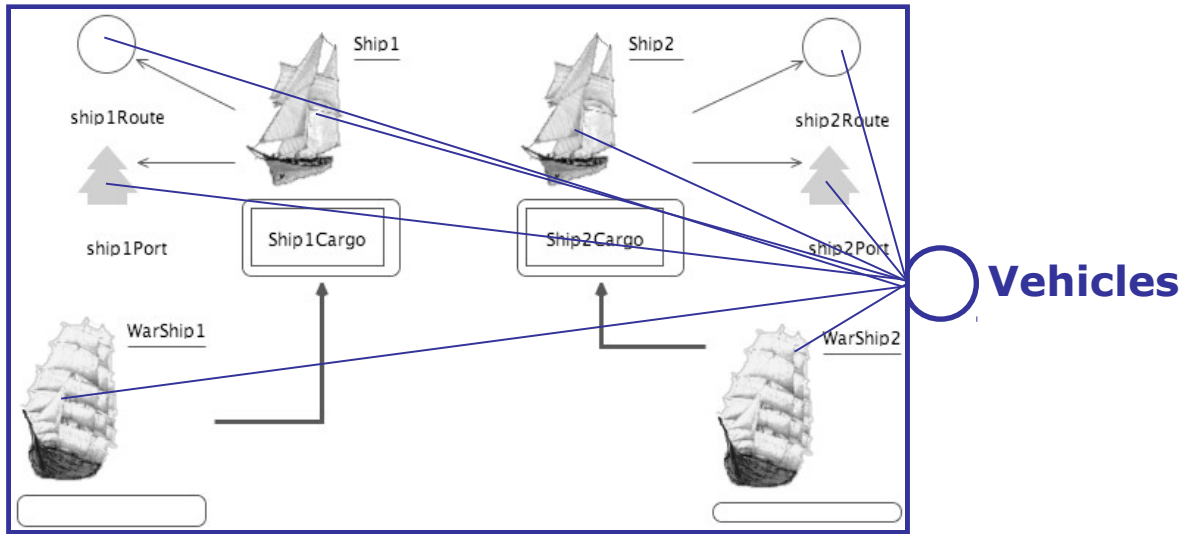
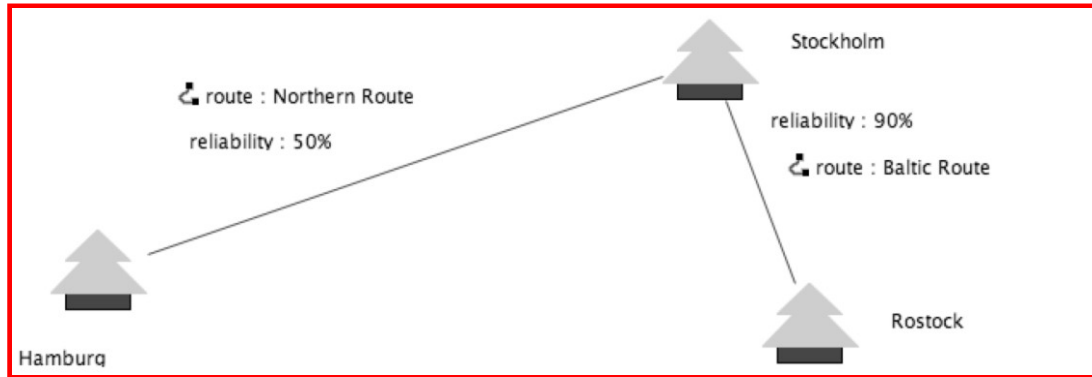
```
reuseextension reuseTaipan implements reuseTaipan
epackages <http://www.eclipse.org/examples/gmf/taipan>
Rootclass TravelSpace {
  fragment role TravelSpace {
    port VehicleContainer {
      Aquatory.ships is hook {}
      Aquatory.ports is hook {}
      Aquatory.routes is hook {}
    }
    port Routes {
      Route is anchor {
        port expr = $self.description$
      }
    }
    port Places {
      Port is anchor {
        port expr = $self.location.concat('Port')$
      }
    }
  }
}

fragment role Flotilla {
  port Vehicles {
    Aquatory.ships is prototype {}
    Aquatory.ports is prototype {}
    Aquatory.routes is prototype {}
  }
  port RouteSlots {
    RouteSlot is slot {
      port expr = $self.name$
    }
  }
  port PlaceSlots {
    PortSlot is slot {
      port expr = $self.name$
    }
  }
}
...
```

All elements of the references **ships**, **ports** and **routes** of the metaclass **Aquatory** act as prototypes accessible through the **Vehicles** port

Building a DSL: Binding ReuseTaipan to Taipan DSL Model Components

34



Building a DSL: Binding ReuseTaipan to Taipan DSL

```
reuseextension reuseTaipan implements reuseTaipan
epackages <http://www.eclipse.org/examples/gmf/taipan>
Rootclass TravelSpace {
  fragment role TravelSpace {
    port VehicleContainer {
      Aquatory.ships is hook {}
      Aquatory.ports is hook {}
      Aquatory.routes is hook {}
    }
    port Routes {
      Route is anchor {
        port expr = $self.description$
      }
    }
    port Places {
      Port is anchor {
        port expr = $self.location.concat('Port')$
      }
    }
  }
}

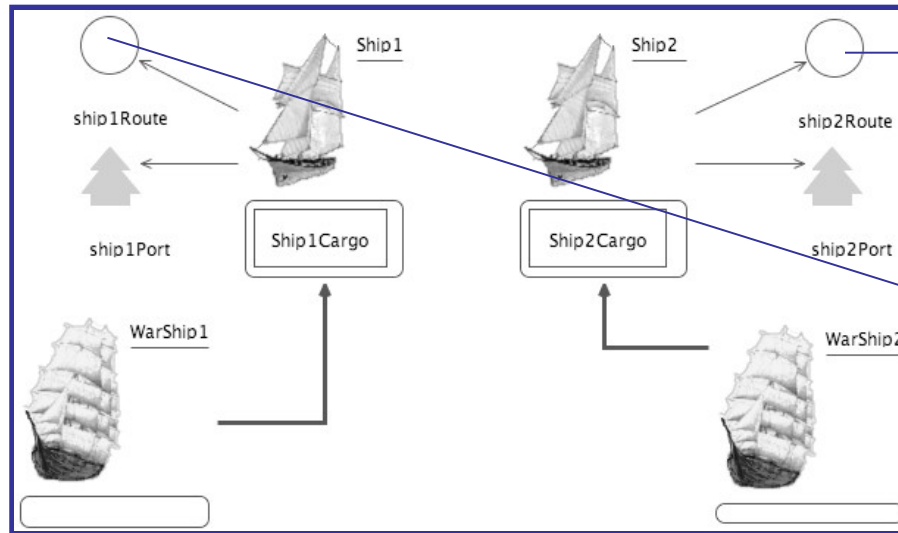
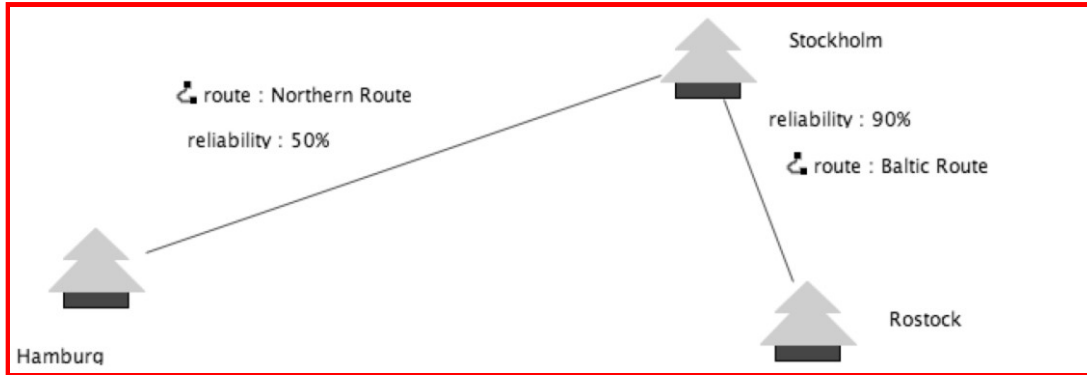
fragment role Flotilla {
  port Vehicles {
    Aquatory.ships is prototype {}
    Aquatory.ports is prototype {}
    Aquatory.routes is prototype {}
  }
  port RouteSlots {
    RouteSlot is slot {
      port expr = $self.name$
    }
  }
  port PlaceSlots {
    PortSlot is slot {
      port expr = $self.name$
    }
  }
}
...

```

Each **RouteSlot** is a slot accessible through individual ports; the ports are named using the **name** attribute of the **RouteSlot** metaclass

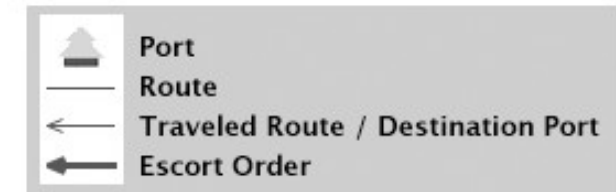
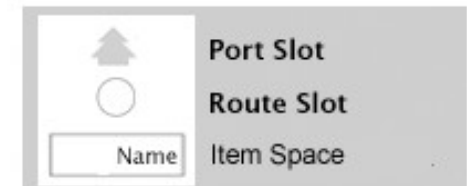
Building a DSL: Binding ReuseTaipan to Taipan DSL Model Components

36



ship2Route

ship1Route



Building a DSL: Binding ReuseTaipan to Taipan DSL

37

```
reuseextension reuseTaipan implements reuseTaipan
epackages <http://www.eclipse.org/examples/gmf/taipan>
Rootclass TravelSpace {
  fragment role TravelSpace {
    port VehicleContainer {
      Aquatory.ships is hook {}
      Aquatory.ports is hook {}
      Aquatory.routes is hook {}
    }
    port Routes {
      Route is anchor {
        port expr = $self.description$
      }
    }
    port Places {
      Port is anchor {
        port expr = $self.location.concat('Port')$
      }
    }
  }
}

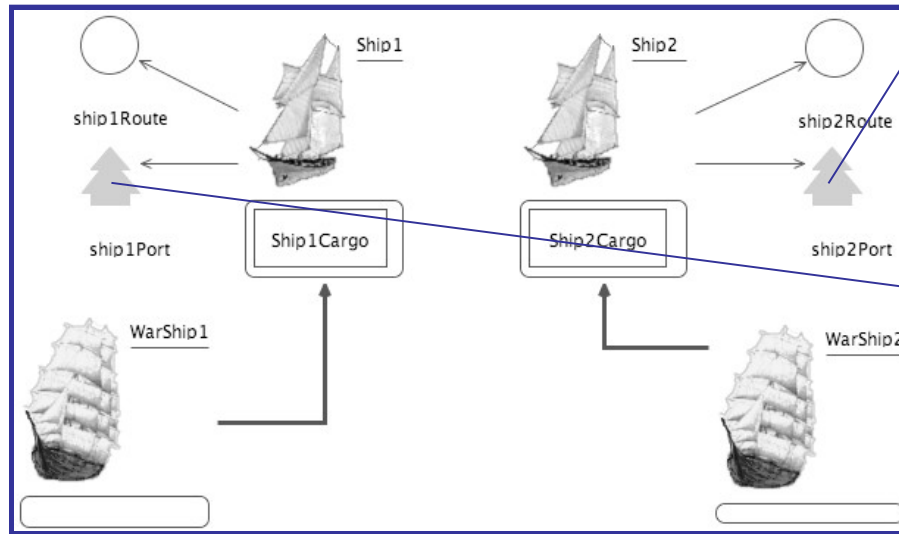
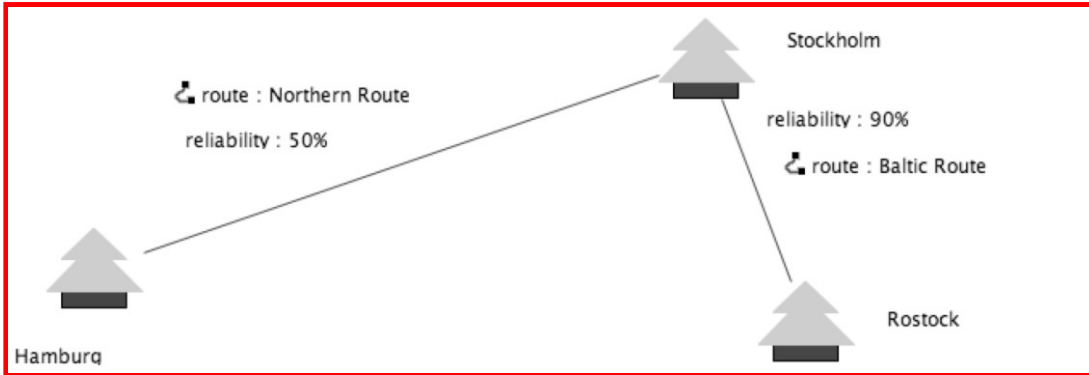
fragment role Flotilla {
  port Vehicles {
    Aquatory.ships is prototype {}
    Aquatory.ports is prototype {}
    Aquatory.routes is prototype {}
  }
  port RouteSlots {
    RouteSlot is slot {
      port expr = $self.name$
    }
  }
  port PlaceSlots {
    PortSlot is slot {
      port expr = $self.name$
    }
  }
}
...

```

Each **PortSlot** is a slot accessible through individual ports; the ports are named using the **name** attribute of the **RouteSlot** metaclass

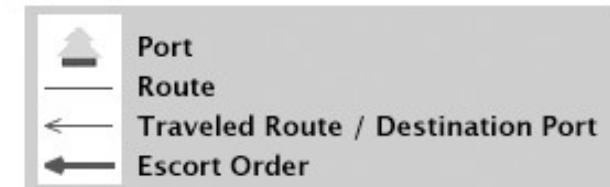
Building a DSL: Binding ReuseTaipan to Taipan DSL Model Components

38



ship2Port

ship1Port



Building a DSL: Binding ReuseTaipan to Taipan DSL

39

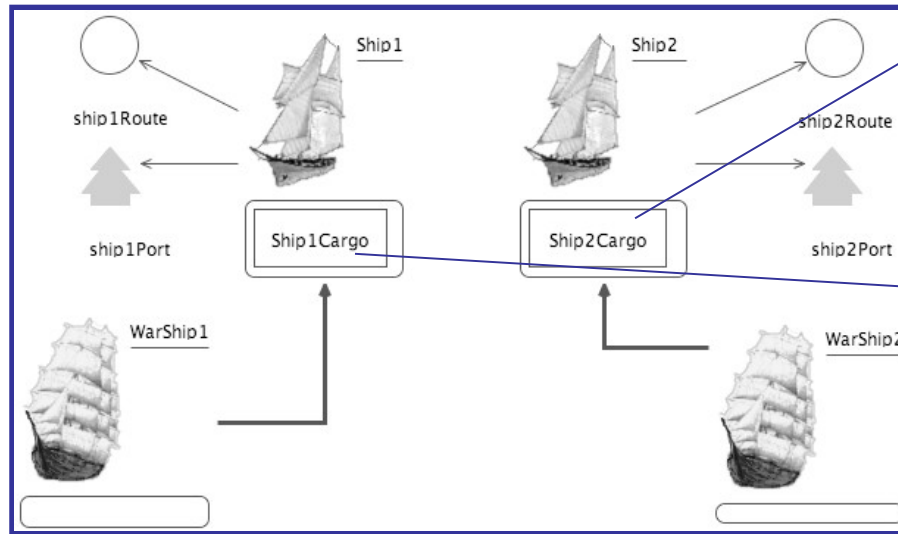
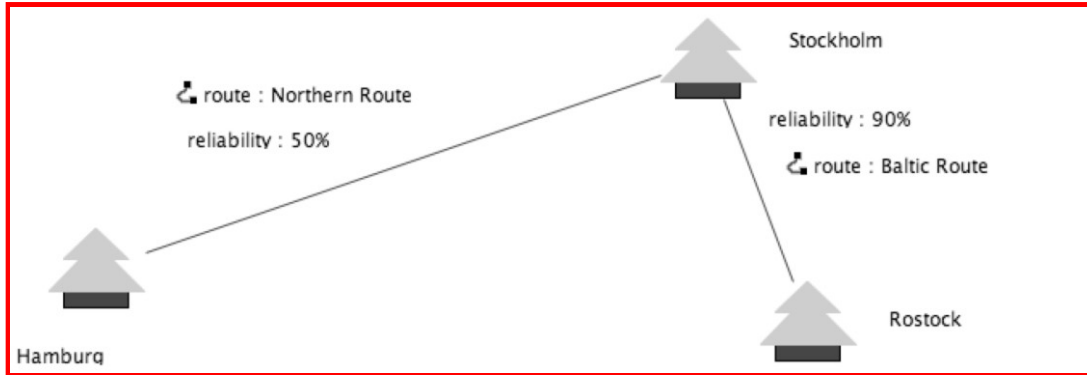
```
...
binding ItemHolder {
  binding ItemSpaces {
    ItemSpace is hook {
      port expr = $self.name$
    }
  }
}

binding ItemContainer {
  binding Items {
    Item is prototype {
      port expr = $self.article$
    }
  }
}
}
```

Each **ItemSpace** is a hook accessible through individual ports; the ports are named using the **name** attribute of the **ItemSpace** metaclass

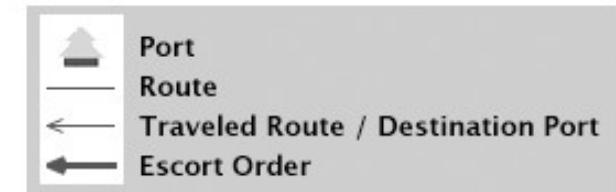
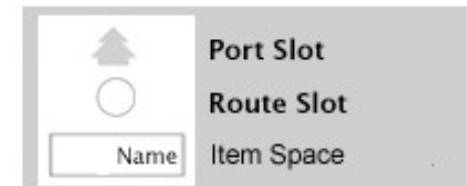
Building a DSL: Binding ReuseTaipan to Taipan DSL Model Components

40



Ship2Cargo

Ship1Cargo



Building a DSL: Binding ReuseTaipan to Taipan DSL

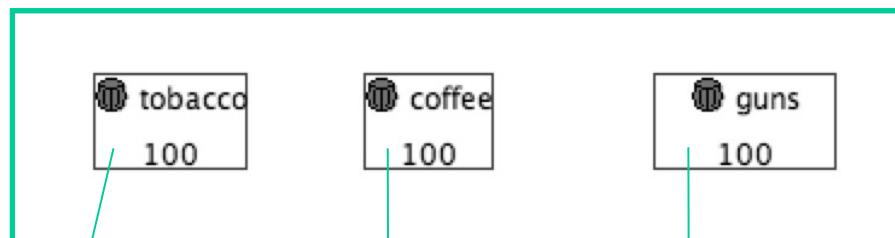
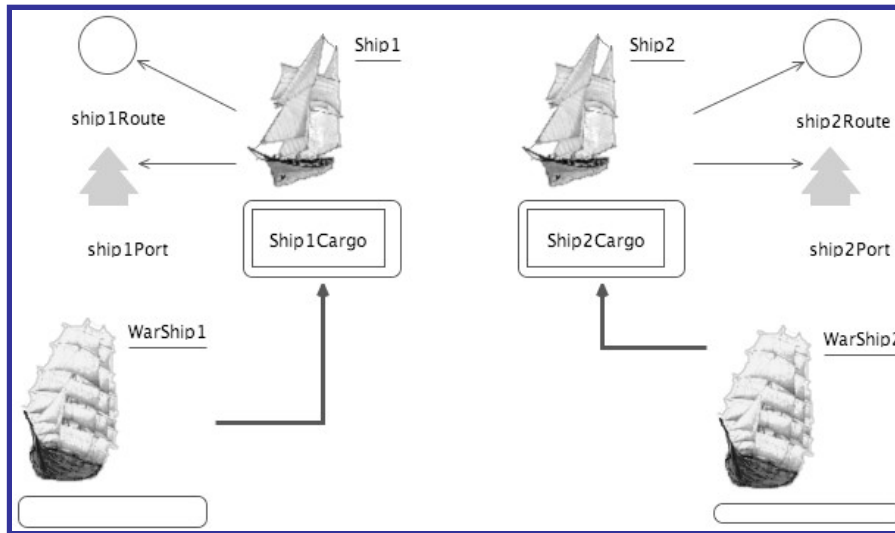
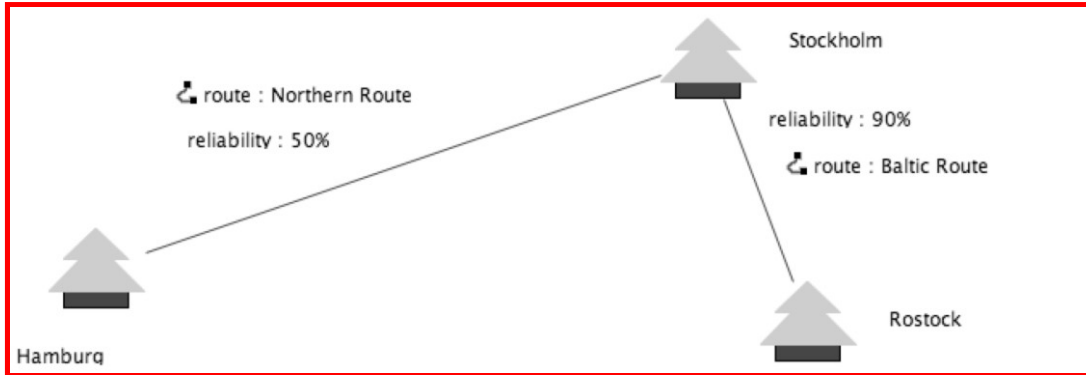
41

```
...  
  
fragment role ItemHolder {  
  port ItemSpaces {  
    ItemSpace is hook {  
      port expr = $self.name$  
    }  
  }  
}  
  
fragment role ItemContainer {  
  port Items {  
    Item is prototype {  
      port expr = $self.article$  
    }  
  }  
}  
}
```

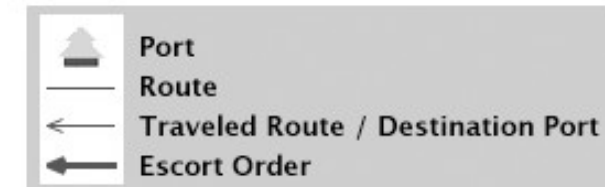
Each **Item** is a prototype accessible through individual ports; the ports are named using the **article** attribute of the **Items** metaclass

Building a DSL: Binding ReuseTaipan to Taipan DSL Model Components

42



tobacco coffee guns



44.4 Using Reuseware Tooling with a DSL

43

- ▶ Fragment Repository
 - Light-weight repository to manage and find reusable model fragments
 - Can instantly be used to build libraries of model fragments designed in a DSL

- ▶ Composition Program Editor
 - Independent of composition systems and reuse extensions
 - Can instantly be used to define compositions for the DSL
 - Layout can be customized if desired

Building a DSL: Using Reuseware Tooling with a DSL

44

Prof. U. Abmann, Softwareentwicklungswerkzeuge (SEW)

The screenshot displays the Eclipse IDE interface for a DSL project. The main editor shows a diagram with three fragments: **EuropeanSea.taipan**, **MyFlotilla.taipan**, and **MyCargo.taipan2**. The diagram illustrates connections between ports and roles. The **EuropeanSea.taipan** fragment has several ports on its right side, and the **MyFlotilla.taipan** fragment has ports on its left side. The **MyCargo.taipan2** fragment has ports on its left side. The connections are shown as lines with circular endpoints.

The Properties window at the bottom shows the **Fragment Instance MyFlotilla.taipan** with the following table:

Core	Property	Value
Appearance	Composition	reuseTaipan.Flotilla, reuseTaipan.ItemHolder
	Cs Fragment Roles	reuseTaipan.Flotilla, reuseTaipan.ItemHolder
	Name	MyFlotilla.taipan



Building a DSL: Using Reuseware Tooling with a DSL

45

Prof. U. Abmann, Softwareentwicklungswerkzeuge (SEW)

The screenshot shows the Eclipse IDE interface. On the left, the 'Lucida Grande' package explorer displays a tree structure of model fragments. The main editor shows a diagram with three boxes: 'EuropeanSea.taipan' (shaded grey), 'MyFlotilla.taipan', and 'MyCargo.taipan2'. Lines connect ports on these boxes, representing composition interfaces. A yellow callout box contains the text: 'The fragment repository shows model fragments, the fragment roles they can play and the details of the corresponding composition interfaces'. At the bottom, the 'Properties' view shows details for 'Fragment Instance MyFlotilla.taipan', including its composition roles and name.

Property	Value
Core	
Appearance	
Composition	
Cs Fragment Roles	reuseTaipan.Flotilla, reuseTaipan.ItemHolder
Name	MyFlotilla.taipan



Building a DSL: Using Reuseware Tooling with a DSL

46

Prof. U. Abmann, Softwareentwicklungswerkzeuge (SEW)

The screenshot shows the Eclipse IDE with a DSL composition diagram in the center. The diagram consists of three fragments: **EuropeanSea.taipan**, **MyFlotilla.taipan**, and **MyCargo.taipan2**. **EuropeanSea.taipan** is connected to **MyFlotilla.taipan** via four lines, each ending in a dashed circle. **MyFlotilla.taipan** is connected to **MyCargo.taipan2** via two lines, each ending in a solid circle. A red arrow points from the text box to the connection between **MyFlotilla.taipan** and **MyCargo.taipan2**.

A yellow text box contains the following text:

Fragments are added to a composition program; for each fragment one can define which fragment roles it should play in the composition program (e.g., myFlotilla is both *Flottila* and *ItemHolder*)

Below the diagram, the **Properties** view shows the **Fragment Instance MyFlotilla.taipan**. The **Core** tab is active, and the **Composition** section is expanded. The **Cs Fragment Roles** property is set to `reuseTaipan.Flottila, reuseTaipan.ItemHolder`. The **Name** property is set to `MyFlotilla.taipan`.

Property	Value
Core	
Appearance	
Composition	
Cs Fragment Roles	reuseTaipan.Flottila, reuseTaipan.ItemHolder
Name	MyFlotilla.taipan



Building a DSL: Using Reuseware Tooling with a DSL

47

Prof. U. Abmann, Softwareentwicklungswerkzeuge (SEW)

The screenshot shows the Eclipse IDE with a DSL model named 'TravelPlan.fcdi'. The model is composed of three fragments: 'EuropeanSea.taipan', 'MyFlotilla.taipan', and 'MyCargo.taipan2'. Composition links are shown as lines connecting ports on the fragments. Red arrows point to these links, indicating they are the focus of the explanation. The left sidebar shows a project tree with various reusable components like 'reuseTaipan.Flottilla', 'reuseTaipan.ItemContainer', and 'reuseTaipan.TravelSpace'. The bottom panel shows the 'Properties' view for the 'MyFlotilla.taipan' fragment, with a table of its composition properties.

Core	Property	Value
Appearance	Composition	
	Cs Fragment Roles	reuseTaipan.Flottilla, reuseTaipan.ItemHolder
	Name	MyFlotilla.taipan

Composition links define the composition; Reuseware can execute the composition program and produce an integrated taipan model



Building a DSL: Using Reuseware Tooling with a DSL

48

The screenshot displays the Eclipse IDE interface for a DSL project. The main editor shows a diagram titled "TravelPlan.taipan_diagram" with a tree structure. The root node is "Stockholm", which branches into "Hamburg" and "Rostock". "Hamburg" is connected to "Stockholm" via a route labeled "NorthernRoute" with a reliability of 80%. "Rostock" is connected to "Stockholm" via a route labeled "BalticRoute" with a reliability of 80%. Below the tree, two ships are shown: "Ship1" carrying "tobacco" and "Ship2" carrying "coffee". "WarShip1" and "WarShip2" are also present. The project explorer on the left shows a hierarchy of reusable components like "reuseTaipan.Flottilla", "reuseTaipan.ItemContainer", and "reuseTaipan.TravelSpace". The bottom panel shows the "Properties" window with a table for "Domain Model" and "Rulers & Grid".

Property	Value

The End

49

- ▶ Reuseware is open source, but also dual licensed, i.e., commercialized by the company www.devboost.de

DevBoost