

Diagram Definition: Implementation in Papyrus 1.1

Maged Elaasar, Ph.D., P.Eng.
Crossplatform Software Inc.
<http://magedelaasar.com>

1st Papyrus Workshop on DSML Technologies
June 22-23, 2015, Toulouse, France

ABOUT THE SPEAKER



Maged Elaasar, Ph.D., P.Eng.
Consulting Software Engineer
Crossplatform Software Inc.
<http://magedelaasar.com/contact/>

Dr. Elaasar has 18+ years of experience as a software engineer and computer scientist. His domain of expertise is application development and model driven engineering. He has consulted many international clients in these areas over the years.

Additionally, Dr. Elaasar is a senior software architect at the Jet Propulsion Laboratory, California Institute of Technology, where he leads the area of Model Based Systems Engineering. He is also a NASA representative to the OMG, where he co-leads several modeling standards. Prior to that, he was a senior software architect at IBM for 13 years, leading modeling technology on Eclipse and in the context of the Rational family of modeling tools. He holds 12+ US. Patents in modeling technologies.

Furthermore, Dr. Elaasar has a Ph.D. in Electrical and Computer Engineering from Carleton University in Canada (2012). He is also an adjunct professor at the department of Systems and Computer Engineering in the same university. He frequently presents in reputable venues. He also regularly publishes in refereed conferences and journals.

ACKNOWLEDGEMENTS

- **The work presented here was funded by CEA LIST and carried by Crossplatform Software Inc.**

WHY DIAGRAM DEFINITION?

- **Graphical modeling languages at OMG have been defined by their**
 - Abstract syntax: **formally using MOF**
 - Concrete syntax: **informally using text and example diagrams**
 - Leads to ambiguity and inconsistency in specifications
 - Increases cost of developing and learning modeling tools
 - Hinders tool interoperability and promotes vendor lock-in
- **Formal diagram definition is needed to**
 - Enable the interchange of modeling diagrams among tools
 - Enable the consistent rendering of diagrams by tools
 - Enable the consistent interpretation of diagrams by users

WHAT IS DIAGRAM DEFINITION?

- **Diagram Definition (DD)** is an OMG specification that enables the formal specification of concrete graphical syntax of MOF-based languages
 - Version 1.0 has release July 2012
 - Version 1.1 is in progress
- **DD provides two standard metamodels**
 - Diagram Interchange (DI): enables the definition of diagram interchange syntax
 - Diagram Graphics (DG): enables the definition of diagram graphical syntax
- **DD provides an architecture that allows for the definition and mapping of a language's concrete graphical syntax to its abstract syntax**

DIAGRAM INTERCHANGE (DI)

- Defines graphical syntax that users have control over
 - Examples: elements to visualize, diagram layout, notational choices, stylistic choices
- Provides a core **abstract** DI pattern that is realized by each modeling language

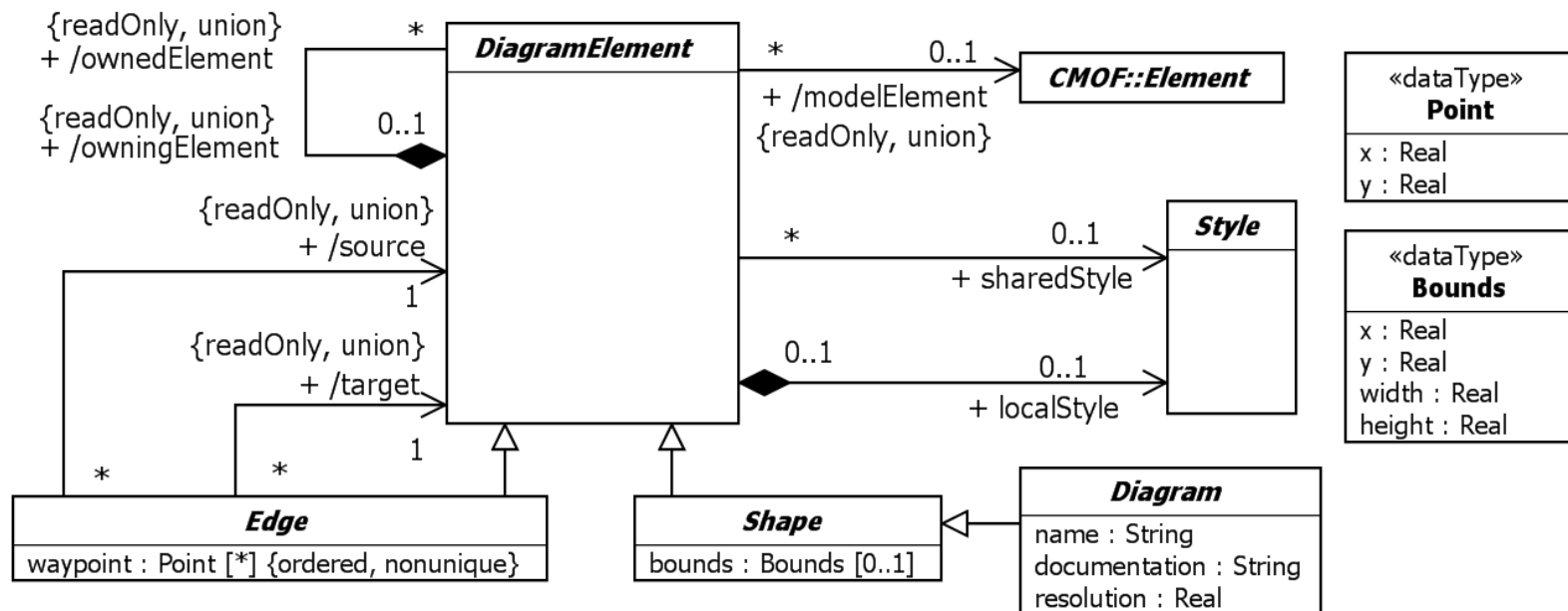


DIAGRAM GRAPHICS (DG)

- Defines graphical syntax that specifications have control over
 - Examples: shape and line notations for each abstract syntax element
- Provides extensive 2D graphics primitives (similar to SVG)

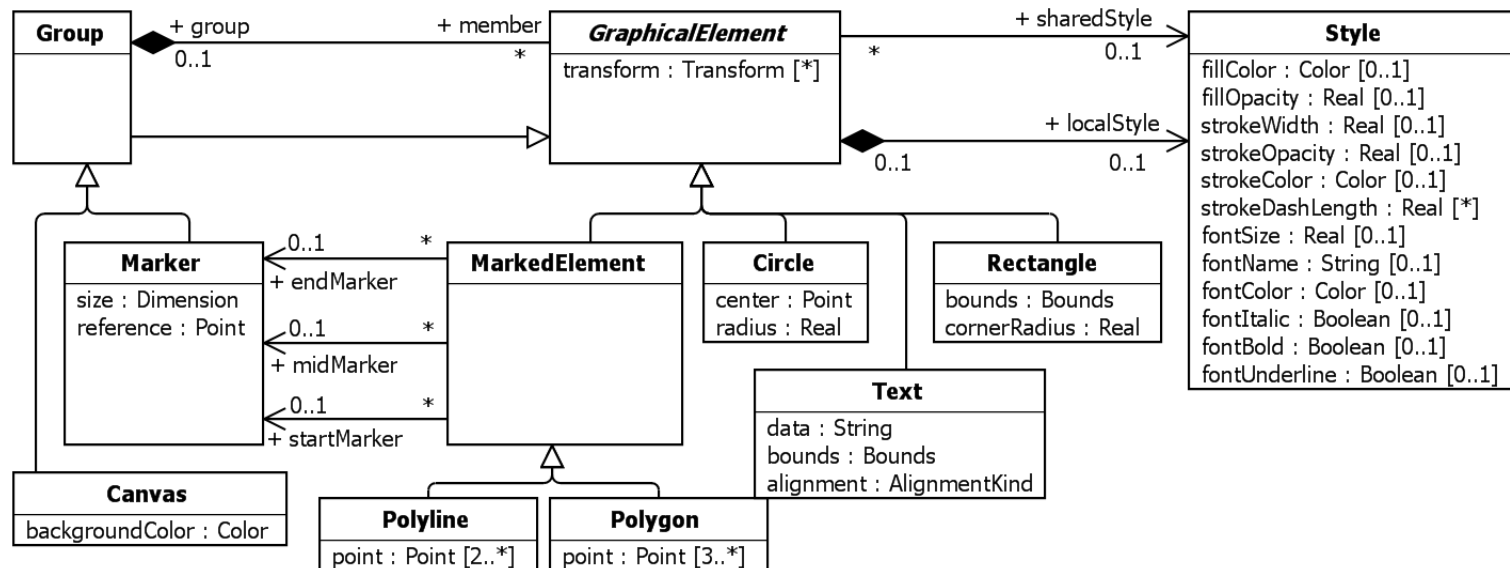
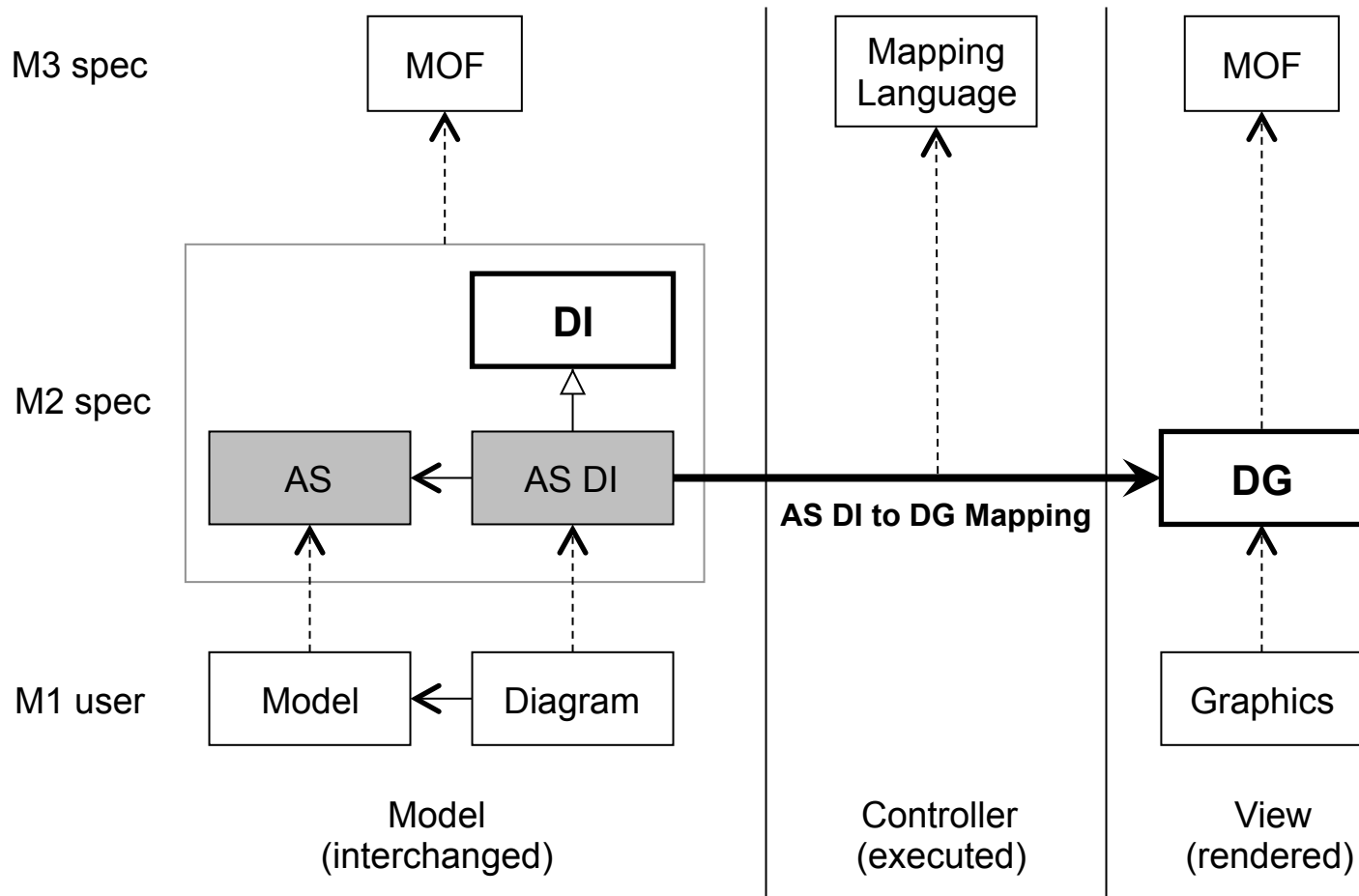


DIAGRAM DEFINITION ARCHITECTURE



-->	Instantiates	□	DD Spec	DI : Diagram Interchange	AS: Abstract Syntax
—▷	Specializes	▒	Language Spec	DG: Diagram Graphics	CS : Concrete Syntax
—→	References	→	Transforms		

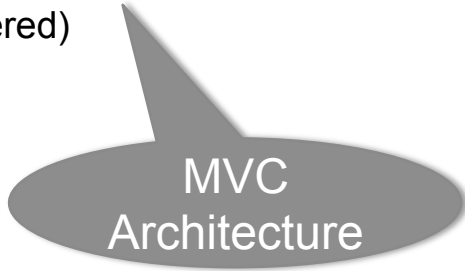


DIAGRAM DEFINITION IMPLEMENTATION

- **An implementation for DD has been added in Papyrus 1.1 consisting of:**

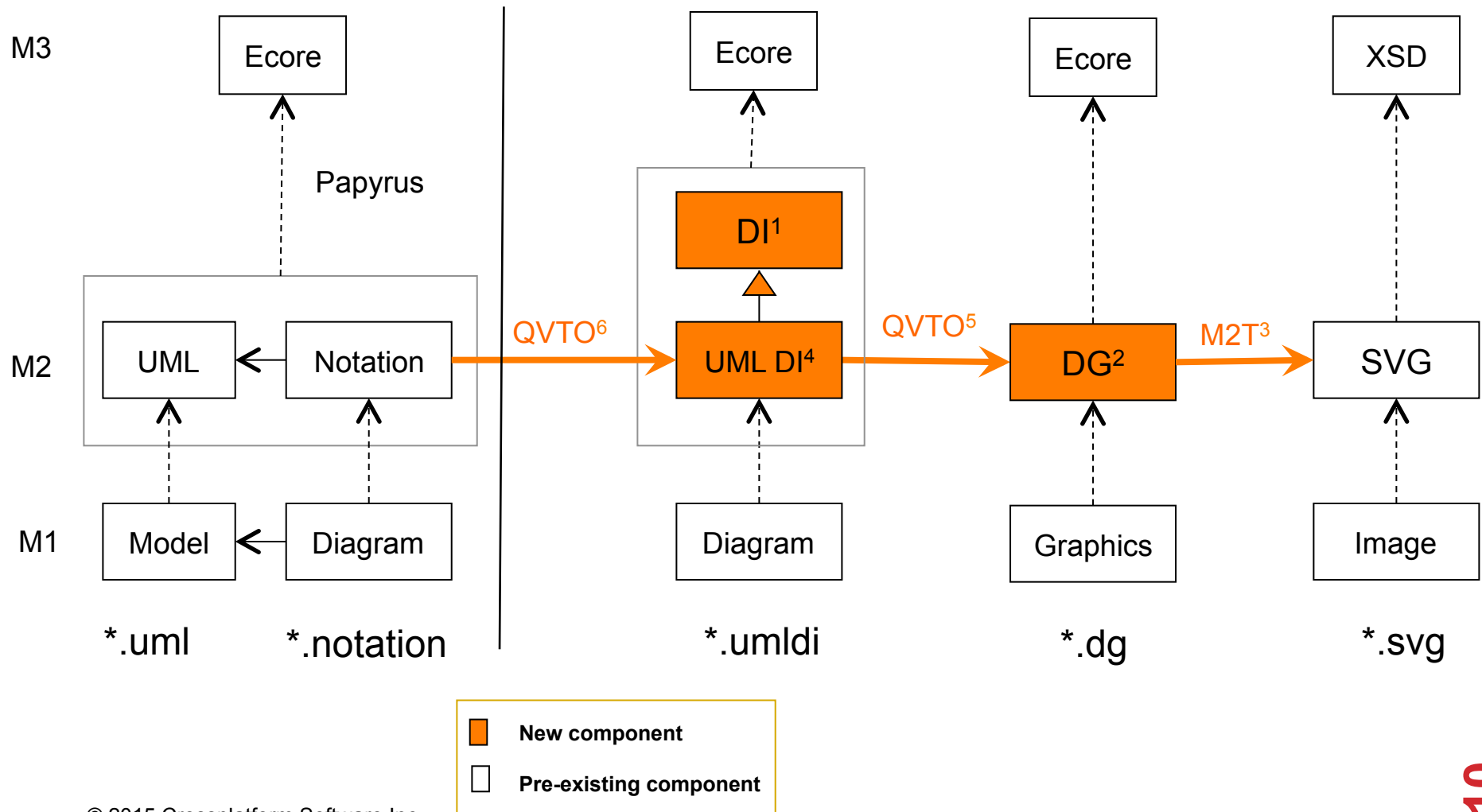
- An implementation of DD 1.0
 1. Ecore-based API for **DI** metamodel (*with changes to spec*)
 2. Ecore-based API and editor for **DG** metamodel (*with changes to spec*)
 3. Model to text mapping from **DG to SVG** (*potential contribution to spec*)

- An implementation of UML DD 2.5
 4. Ecore-based API and editor for **UML DI** metamodel (*with changes to spec*)
 5. QVTO-based mapping from **UML DI to DG** metamodels (*potential contribution to spec*)

- An implementation of a diagram exporter from Papyrus
 6. QVTO-based mapping from **Papyrus DI to UML DI** metamodels

DIAGRAM DEFINITION IMPLEMENTATION ARCHITECTURE IN PAPYRUS

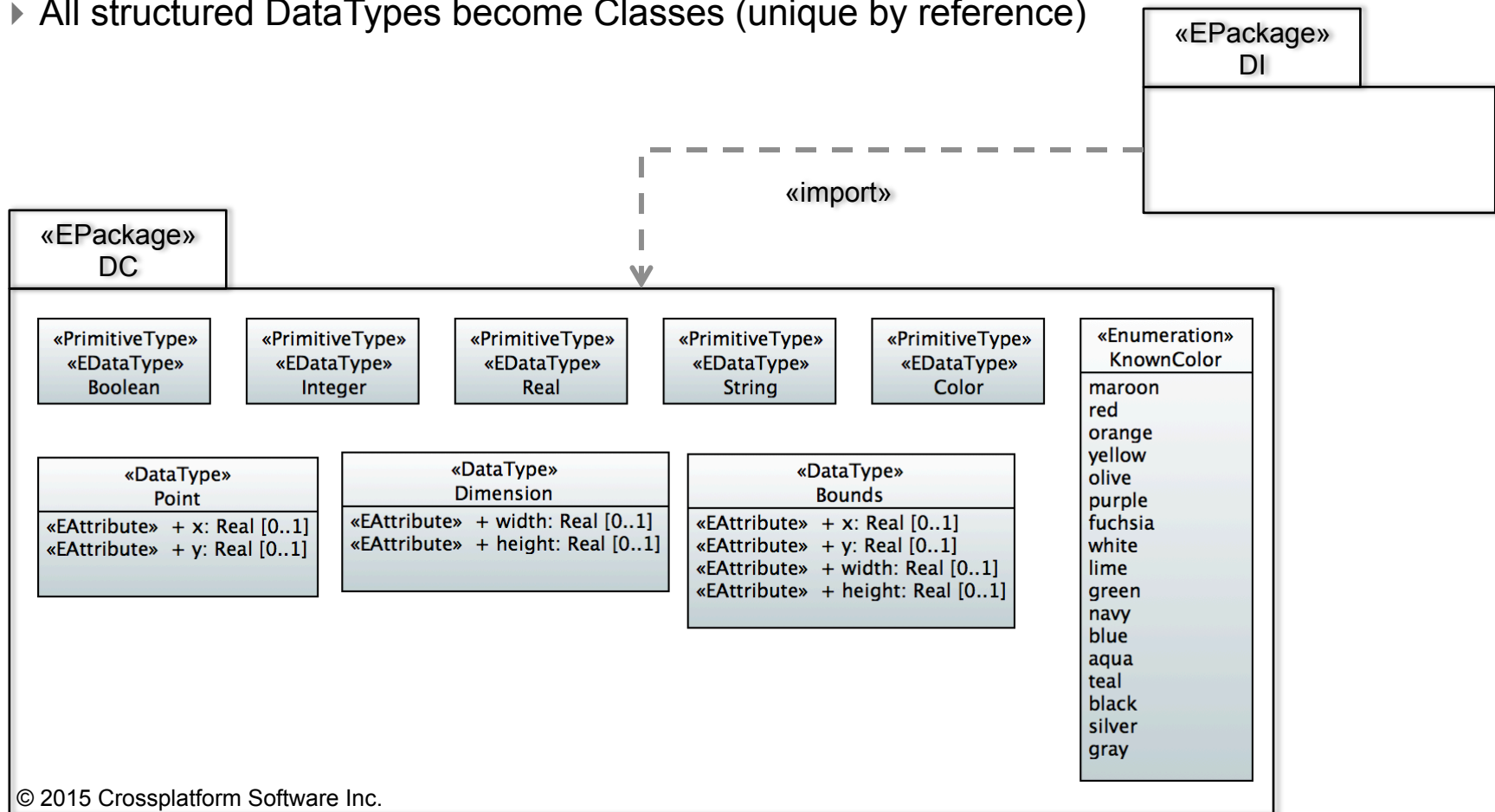
- What has been implemented:



1. DI METAMODEL

■ Changes to DC metamodel

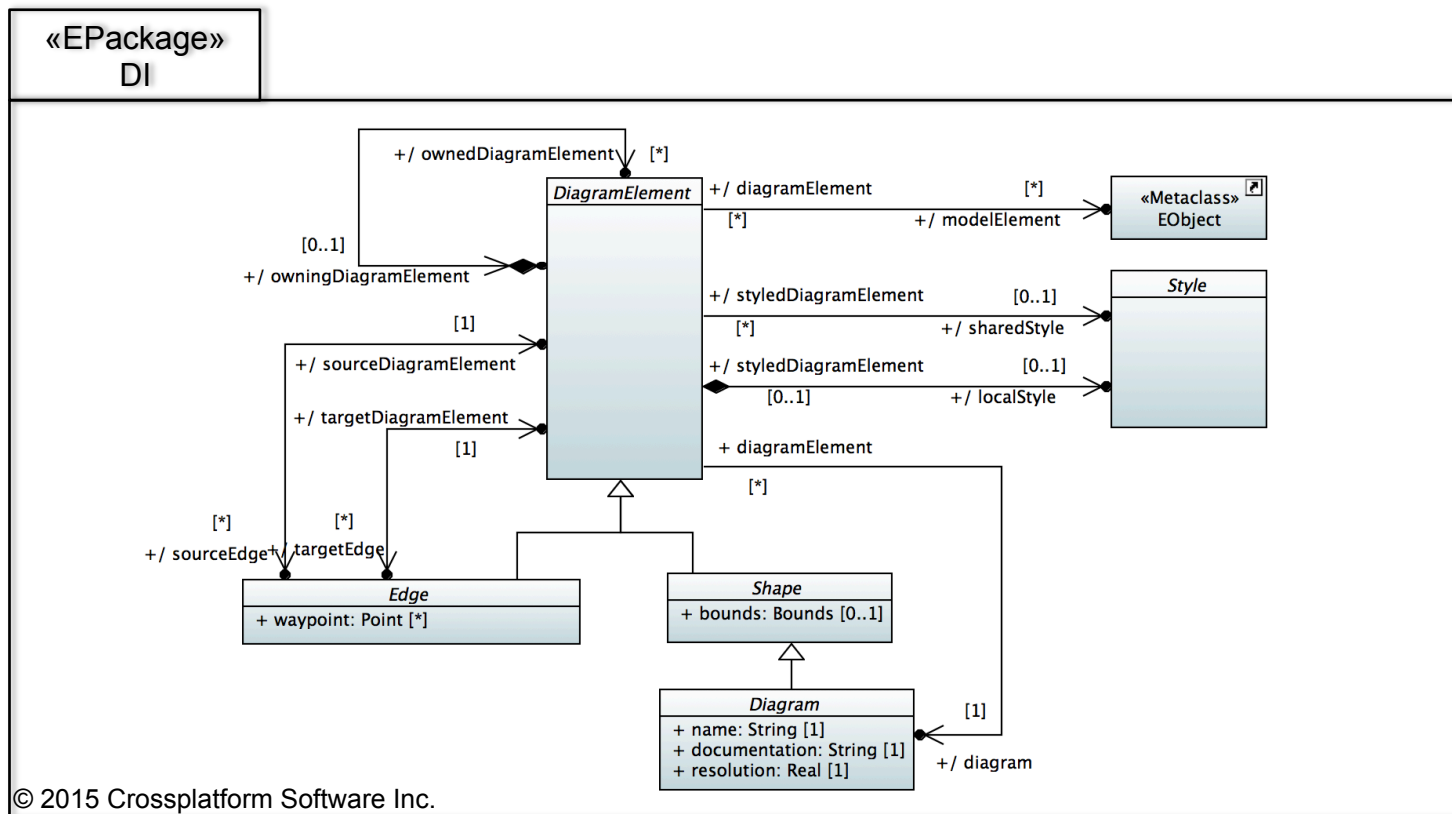
- ▶ DC::Color is made a PrimitiveType (with literals in the form #RRGGBB)
- ▶ DC::KnownColors enumeration literals have associated color literal values
- ▶ All structured DataTypes become Classes (unique by reference)



1. DI METAMODEL

■ Changes to DI metamodel

- ▶ Replaced Element by DiagramElement in names of properties to avoid conflicts with UML
- ▶ Made source/targetDiagramElement associations bi-directional
- ▶ Made Edge::waypoint unique (Point is a Class now)
- ▶ Added the property /DiagramElement::diagram



2. DG METAMODEL

- **Changes to DG metamodel (mostly alignment to SVG)**
 - ▶ Convert structured Datatypes into Classes
 - ▶ Refactored style support to be more CSS like (i.e., rule-based cascading style sheets)
 - ▶ Defined reusable concepts by specializing class Definition (with id property)
 - ▶ Added a Use graphical element to define reusable graphical templates
 - ▶ Added RootCanvas as a specialization of Canvas
 - ▶ Added the concept of Paint which can be color or a PaintServer (Pattern or Gradient)
- **Implemented DG multi-tab editor**
 - ▶ **A tab to manipulate the DG model tree**
 - ▶ **A tab to see the XMI serialization of the model**
 - ▶ **Supports multiple roots of type RootCanvas**

2. DG METAMODEL

Basic Shapes.dg

Model

- platform:/resource/org.eclipse.papyrus.dd.
 - Root Canvas
 - Circle
 - Ellipse
 - Rectangle
 - Rectangle
 - Line
 - Polyline
 - Polygon
 - Path
 - Path
 - Path
 - Path
 - Path
 - Text
 - Image
 - Definitions
 - Style Sheet
 - Style Rule
 - Style Selector canvas
 - Style
 - Style Rule
 - Style Rule

Model Source SVG Canvas

Basic Shapes.dg

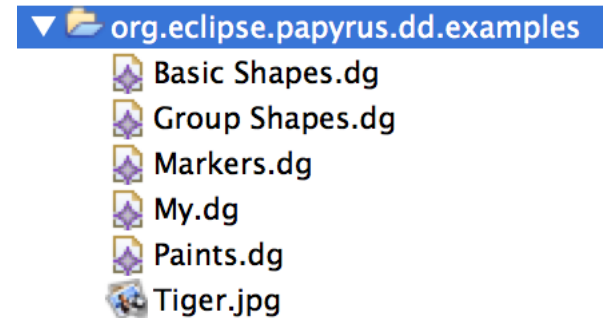
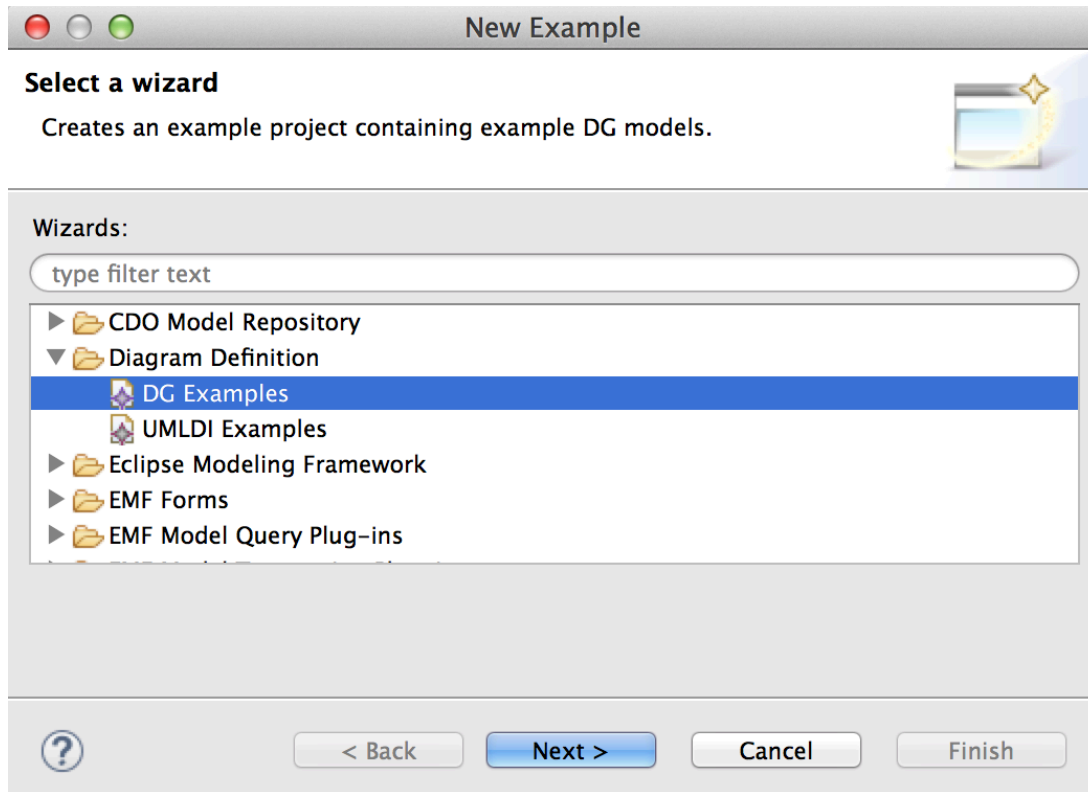
Source

```
<?xml version="1.0" encoding="UTF-8"?>
<dg:RootCanvas xmi:version="2.0" xmlns:xmi="http://www.omg.org/"
  <member xsi:type="dg:Circle" radius="40.0">
    <center x="50.0" y="50.0"/>
  </member>
  <member xsi:type="dg:Ellipse">
    <center x="150.0" y="50.0"/>
    <radii width="50.0" height="30.0"/>
  </member>
  <member xsi:type="dg:Rectangle">
    <bounds x="210.0" y="25.0" width="80.0" height="50.0"/>
  </member>
  <member xsi:type="dg:Rectangle" cornerRadius="10.0">
    <bounds x="300.0" y="25.0" width="80.0" height="50.0"/>
  </member>
  <member xsi:type="dg:Line">
    <style>
      <stroke color="#000000"/>
    </style>
    <start x="390.0" y="80.0"/>
    <end x="440.0" y="20.0"/>
  </member>
  <member xsi:type="dg:Polyline">
    <style>
      <fill/>
      <stroke color="#000000"/>
    </style>
    <point x="450.0" y="80.0"/>
    <point x="450.0" y="20.0"/>
    <point x="480.0" y="50.0"/>
    <point x="510.0" y="20.0"/>
    <point x="510.0" y="80.0"/>
  </member>
</dg:RootCanvas>
```

Model Source SVG Canvas

2. DG METAMODEL

- Added an example project with DG models



3. DG TO SVG MAPPING

- **Implemented a model (DG) to text (SVG) transformation**
 - ▶ **Used the EMF-generated visitor pattern: `DGSwitch<Object>` to read the model**
 - ▶ **Used the Batik 1.7 API to create a corresponding SVG DOM**
 - ▶ **Added a JS script to implement text wrapping and alignment**

- **Implemented DG multi-tab editor**
 - ▶ **A tab for displaying the corresponding SVG DOM**
 - ▶ **A tab for rendering the corresponding SVG Image**

3. DG TO SVG MAPPING

Basic Shapes.dg


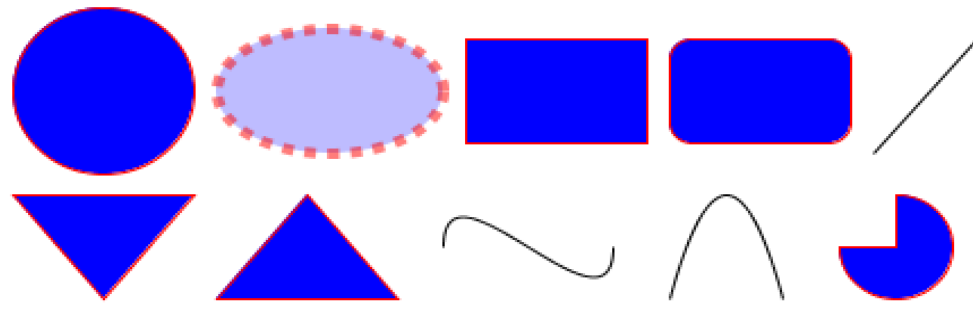
SVG

```
<svg overflow="visible" xmlns:xlink="http://www.w3.org/1999
<script xlink:href="file:/Applications/eclipse-modeling-mar
<defs>
  <style type="text/css"><![CDATA[
    * { fill: #0000FF; stroke: #FF0000; }
    text { font-family: "Times New Roman"; font-size: 40.0;
    ellipse { fill-opacity: 0.25; stroke-opacity: 0.5; stroke-w
  </style>
</defs>
<circle r="40.0" cx="50.0" cy="50.0"/>
<ellipse rx="50.0" ry="30.0" cx="150.0" cy="50.0"/>
<rect x="210.0" width="80.0" height="50.0" y="25.0"/>
<rect rx="10.0" ry="10.0" x="300.0" width="80.0" height="50.0"/>
<line y2="20.0" style="stroke: #000000;" x1="390.0" x2="440.0"/>
<polyline style="fill: none; stroke: #000000;" points="450.0,
<polygon points="520.0,40.0 550.0,20.0 580.0,40.0 580.0,
<path d="M10.0,100.0 L90.0,100.0 L50.0,150.0 Z"/>
<path d="m140.0,100.0 l40.0,50.0 l-80.0,0.0 z"/>
<path style="fill: none; stroke: #000000;" d="M200.0,125.0
<path style="fill: none; stroke: #000000;" d="M300.0,150.0
<path d="M400.0,125.0 l-25.0,0.0 a25.0,25.0 0.0 1,0 25.0,
<text text-anchor="start" x="450.0" y="100.0">
  Hello World!
</text>
<image preserveAspectRatio="xMidYMid meet" x="10.0" wid
</svg>
```

Model Source SVG Canvas

Basic Shapes.dg

Canvas

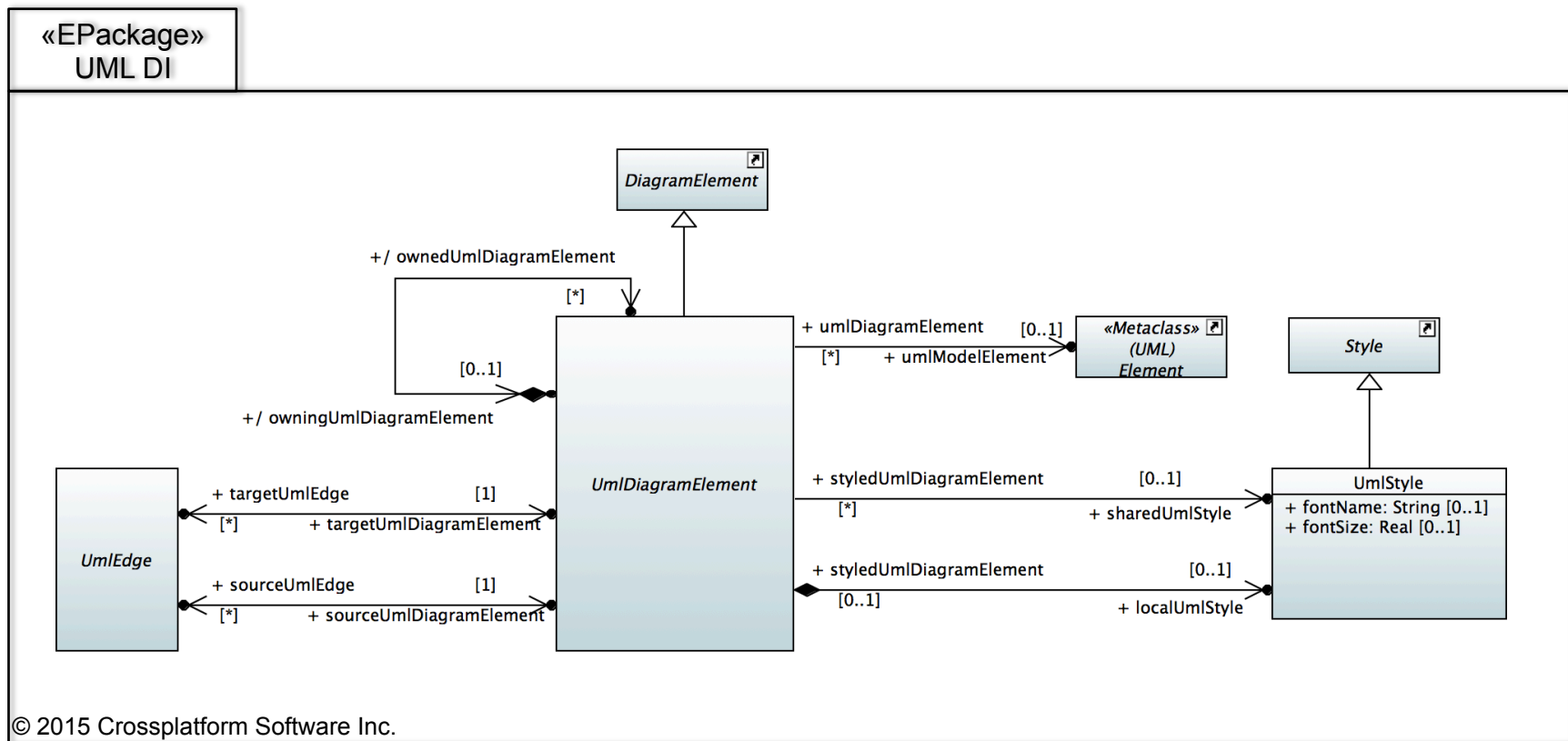


Model Source SVG Canvas

4. UML DI METAMODEL

Changes to UML DI metamodel

- ▶ Replaced Element with umlElement in the name of the properties
- ▶ Replaced redefinitions with subsetting across the board
- ▶ UMLDI::Diagram does not inherit UML::PackageableElement



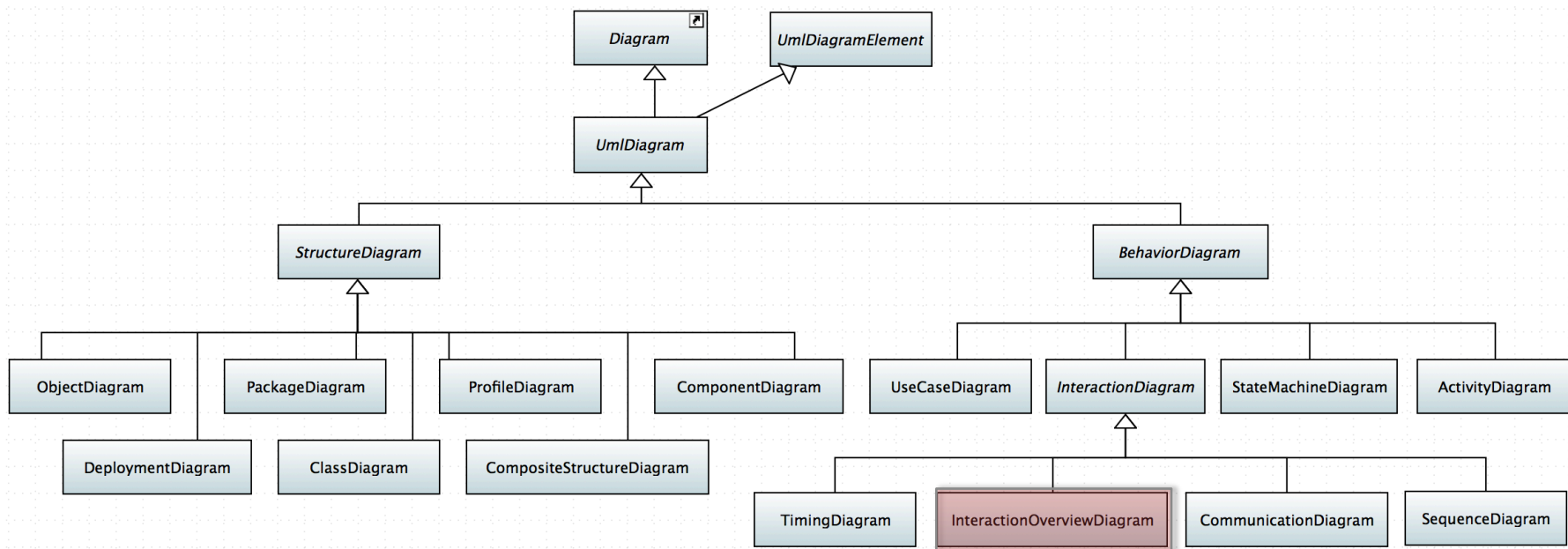
4. UML DI METAMODEL

▪ **Changes to UML DI metamodel**

- ▶ Defined the basic building blocks:
 - Diagram (composes shapes and edges)
 - Shape (composes labels and compartments)
 - Edge (composes labels)
 - Label
 - Compartment
- ▶ Adopted the approach of deeply specializing the building blocks as needed
 - Added building block compositions in the proper context only
 - Added normative options in the proper context only
 - Disadvantage: the metamodel is big
 - 14 labels
 - 30 compartments
 - 30+ edges
 - 80+ shapes
 - 14 diagrams

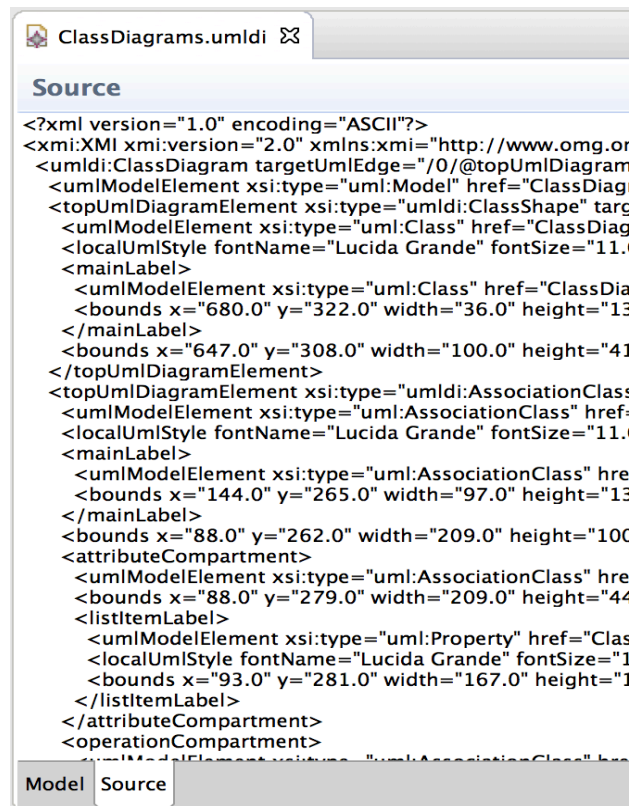
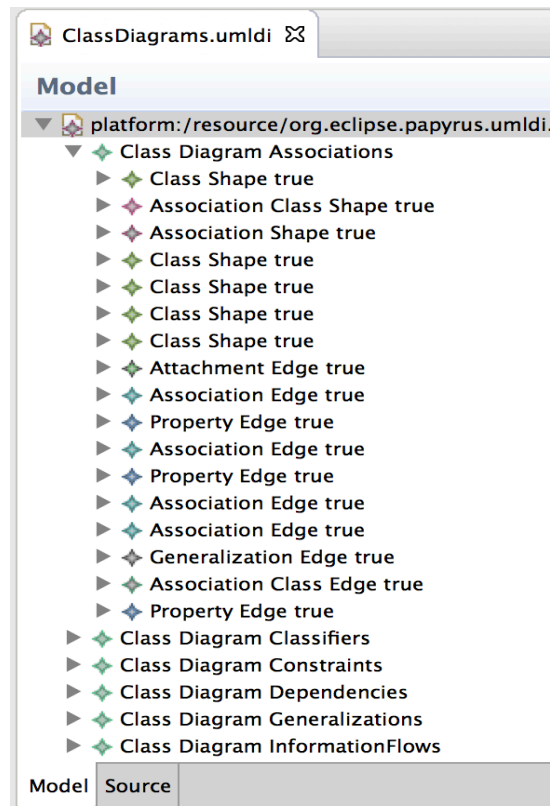
4. UML DI METAMODEL

- All UML diagrams are supported (except Interaction Overview)
 - ▶ Issues with the support for Interaction Overview diagrams in Papyrus



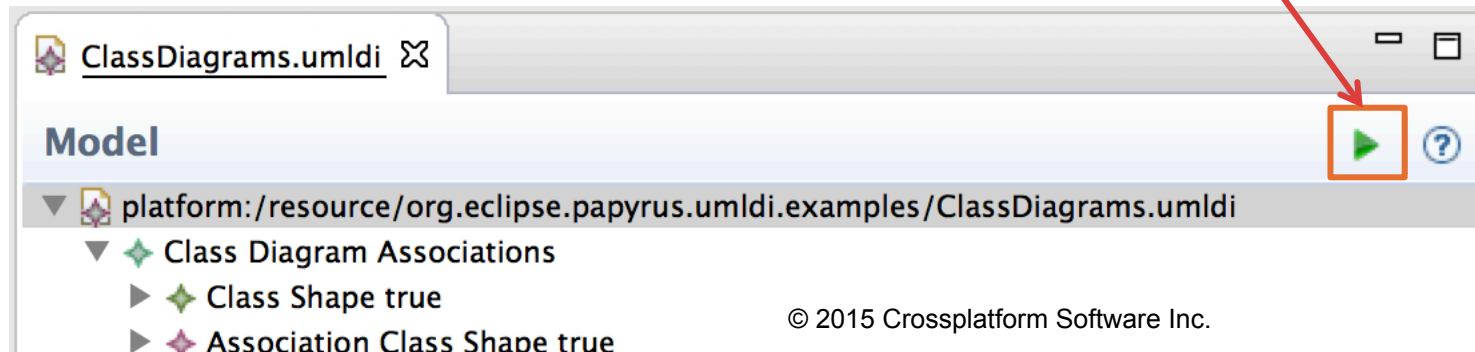
4. UML DI METAMODEL

- Implemented UML DI multi-tab editor
 - ▶ A tab to manipulate the UML DI model tree
 - ▶ A tab to see the XMI serialization of the model
 - ▶ Supports multiple roots of type UMLDiagram



5. UML DI TO DG MAPPING

- **Implemented a model (UMLDI) to model (DG) transformation**
 - ▶ Transformation is specified in QVTO
 - Highly modular design: a module for every kind of building block
 - Leverage of rule composition, inheritance, and overriding for conciseness
 - Added a black box library to define some math and color functions
 - ▶ Non-normative styles used are: font name and font size (from UML DI)
 - ▶ Layout constraints (position/size) is copied from UML DI (no automatic layout in DG)
 - ▶ Text for all labels are derived in DG
 - ▶ Most important (but not all yet) normative options are captured
- **Transformation is invoked from a UML DI editor by clicking this button**

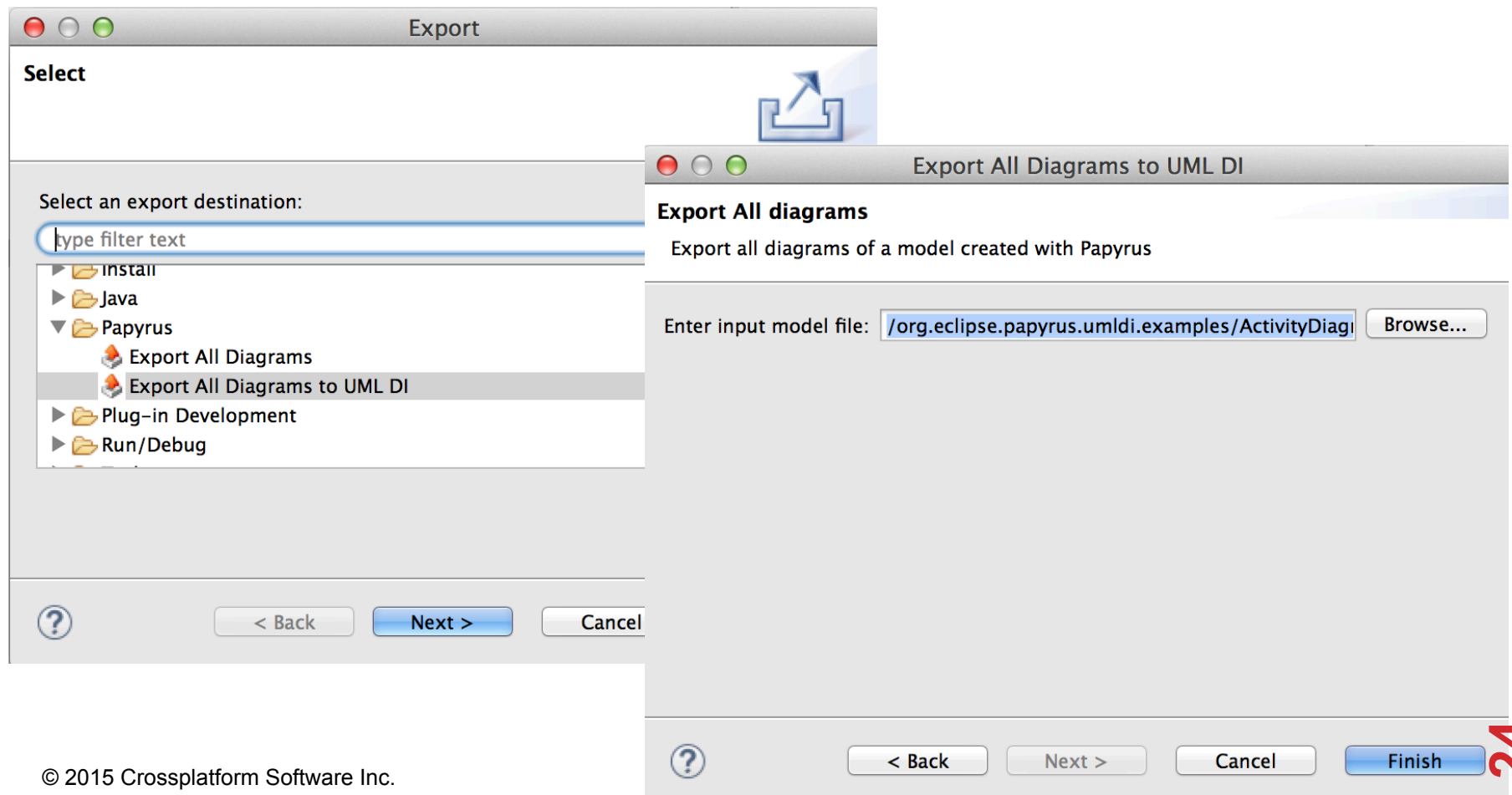


6. PAPYRUS DI TO UML DI MAPPING

- **Implemented a model (Papyrus DI) to model (UML DI) transformation**
 - ▶ Papyrus DI uses the GMF Notation metamodel
 - Highly abstract metamodel defining only the building blocks
 - The UML details are added with factories generated from other GMF models
 - ▶ Transformation is specified in QVTO
 - Highly modular transformation design
 - Leverage of rule composition, inheritance, and overriding for conciseness
 - Slight adjustments of Papyrus notations to improve export output
 - Added a black box library to perform the following:
 - Render a diagram at the beginning and dispose of it at the end
 - Get the exact rendered bounds of nodes and waypoints of edges
 - Map Papyrus view types (numbers) to corresponding UML DI ones
 - Papyrus assigns different view types to the same view in different diagrams
 - Retrieved this information from extension points augmented by other maps

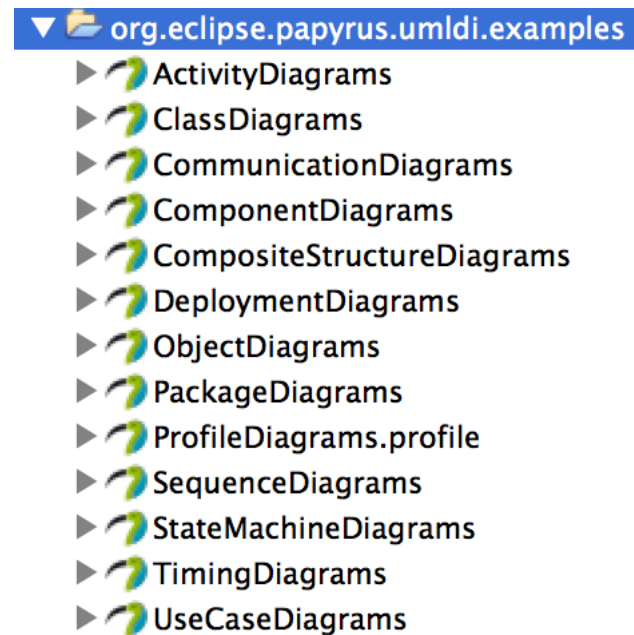
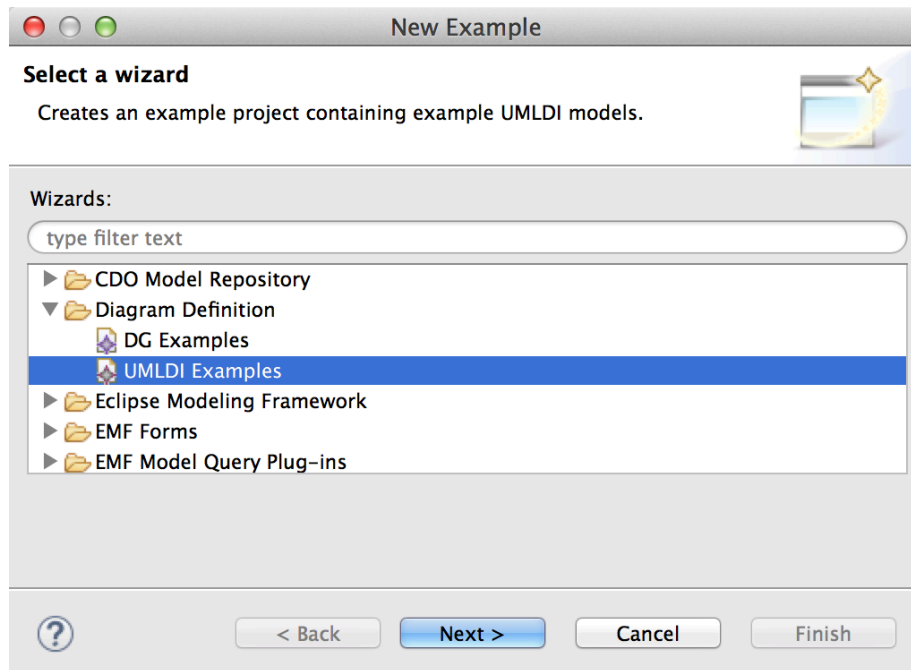
6. PYPYRUS DI TO UML DI MAPPING

- **Implemented an export wizard for a Papyrus DI model**
 - ▶ Runs the transformation on all diagrams in the model
 - ▶ Produces a UML DI model with multiple root UMLDiagrams

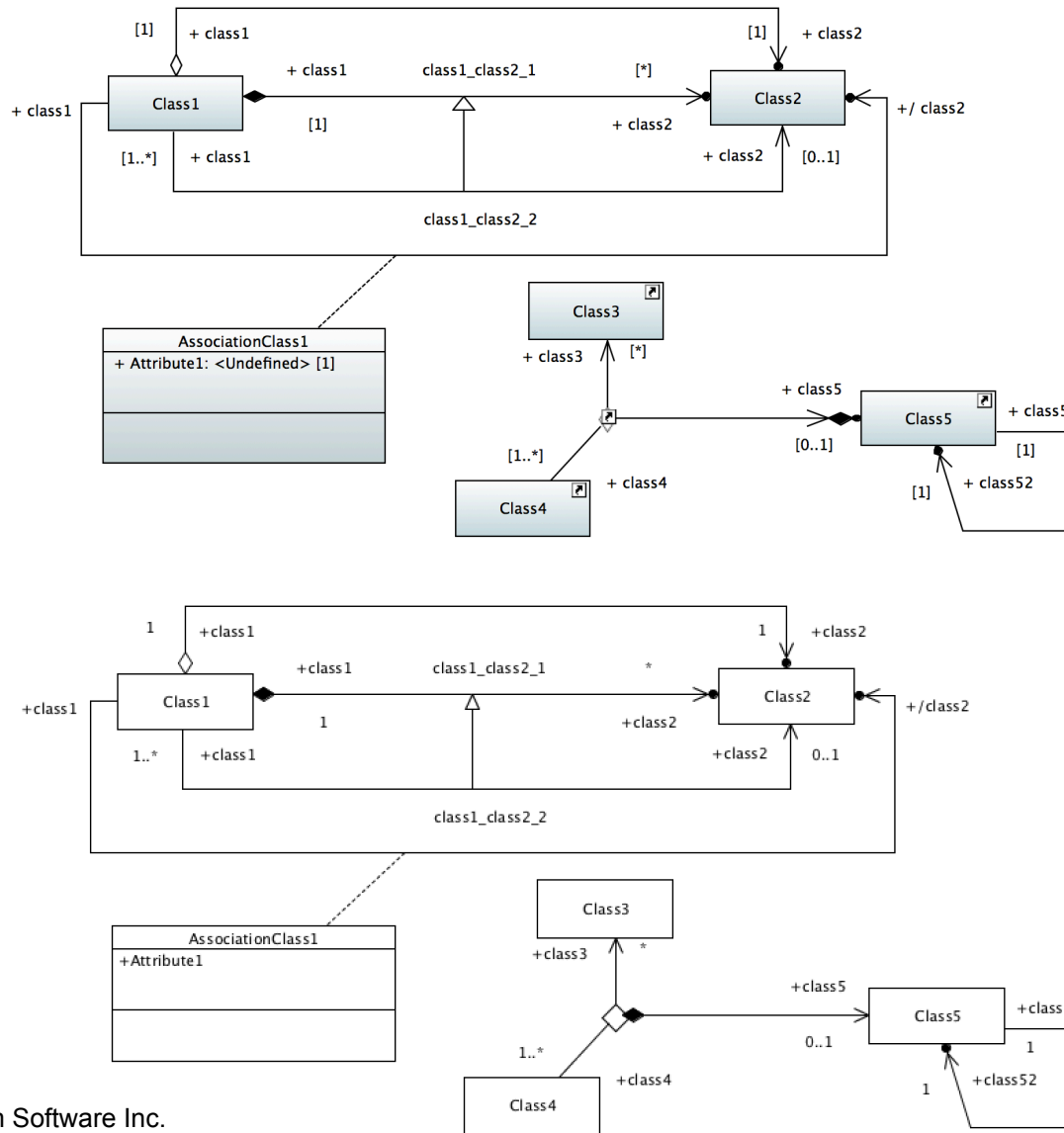


6. PAPYRUS DI TO UML DI MAPPING

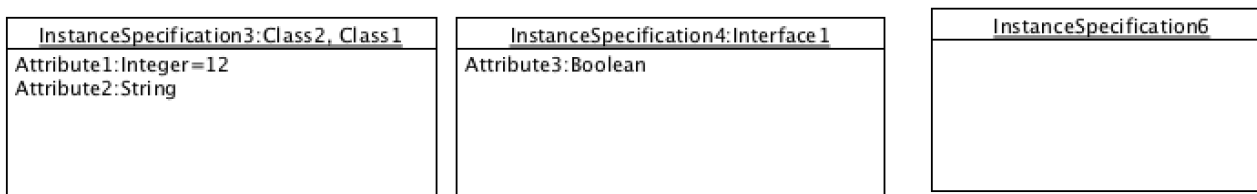
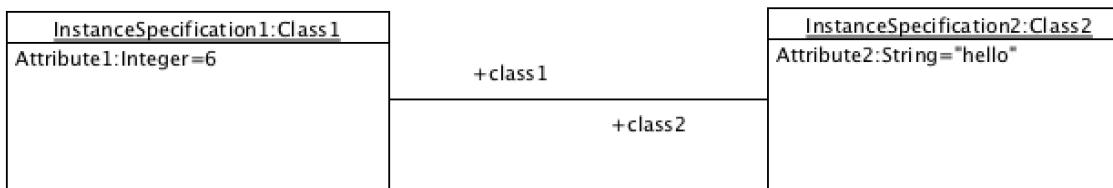
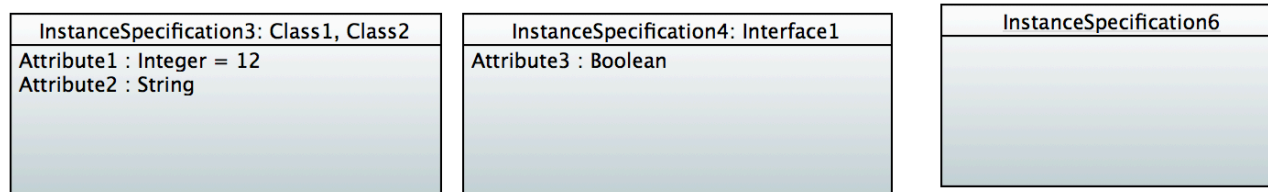
- **Added an example project consisting of**
 1. Papyrus DI models for all supported UML diagram kinds
 2. Corresponding UML DI models exported from 1
 3. Corresponding DG models transformed from 2



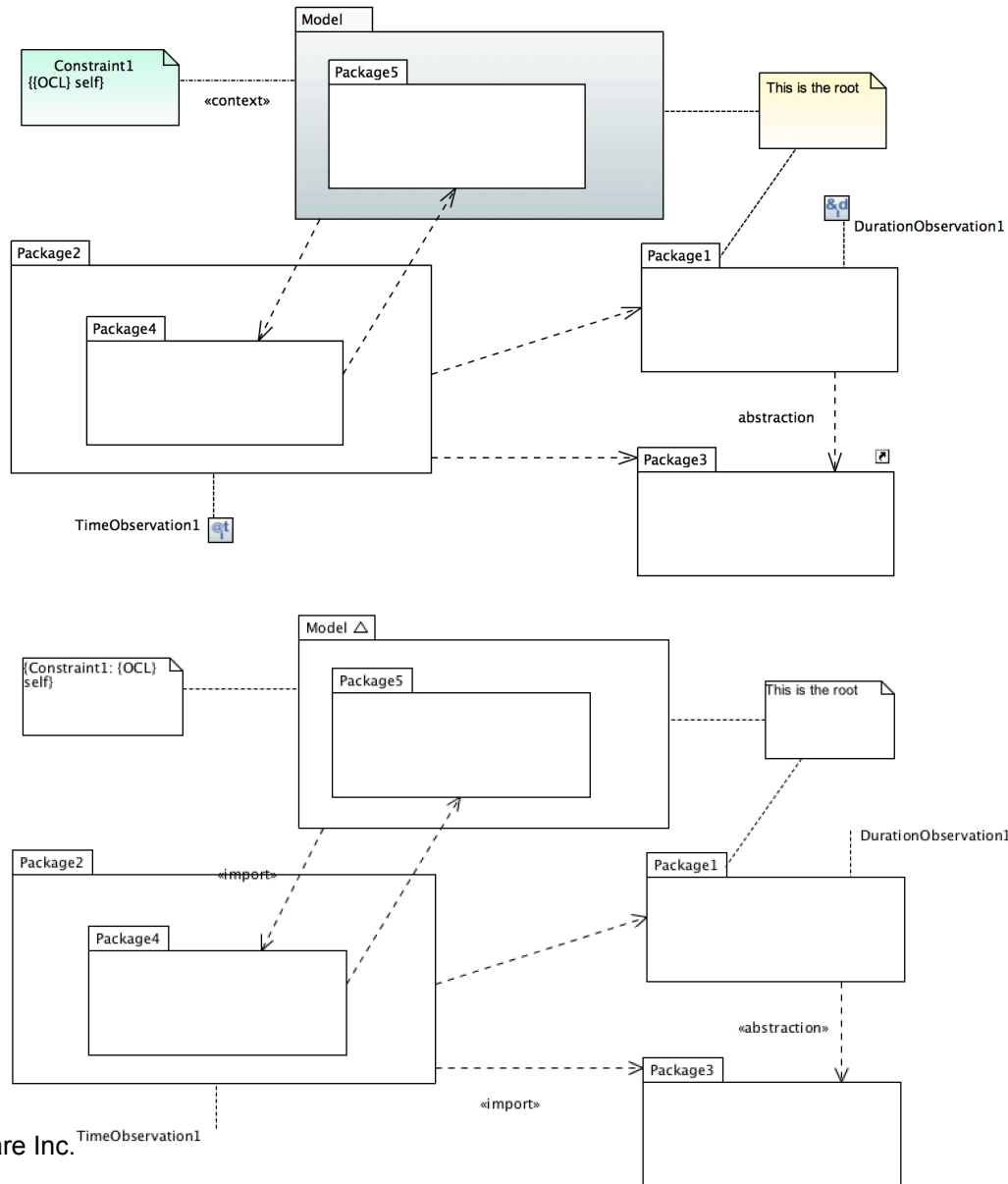
EXAMPLE: CLASS DIAGRAM



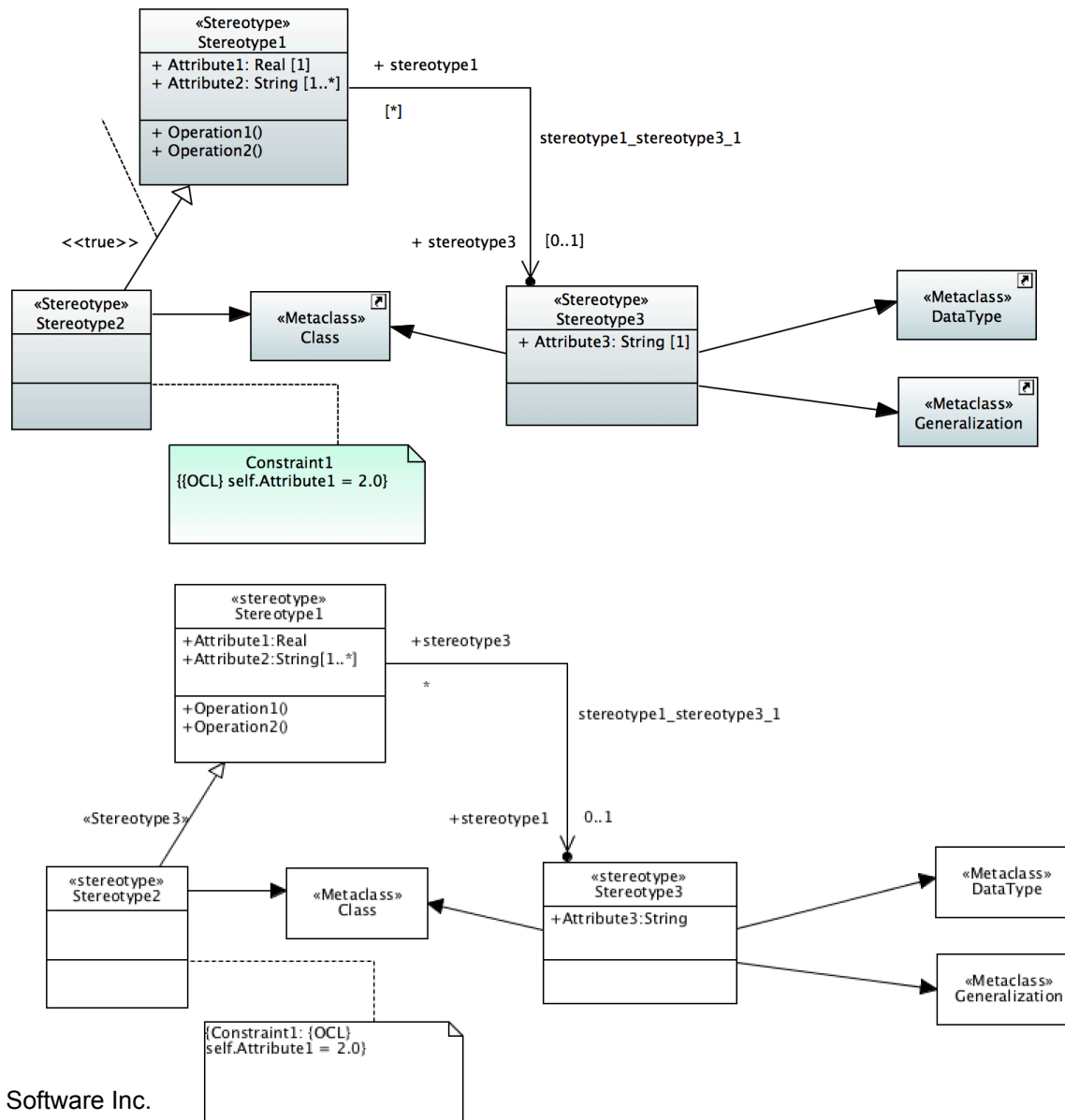
EXAMPLE: OBJECT DIAGRAM



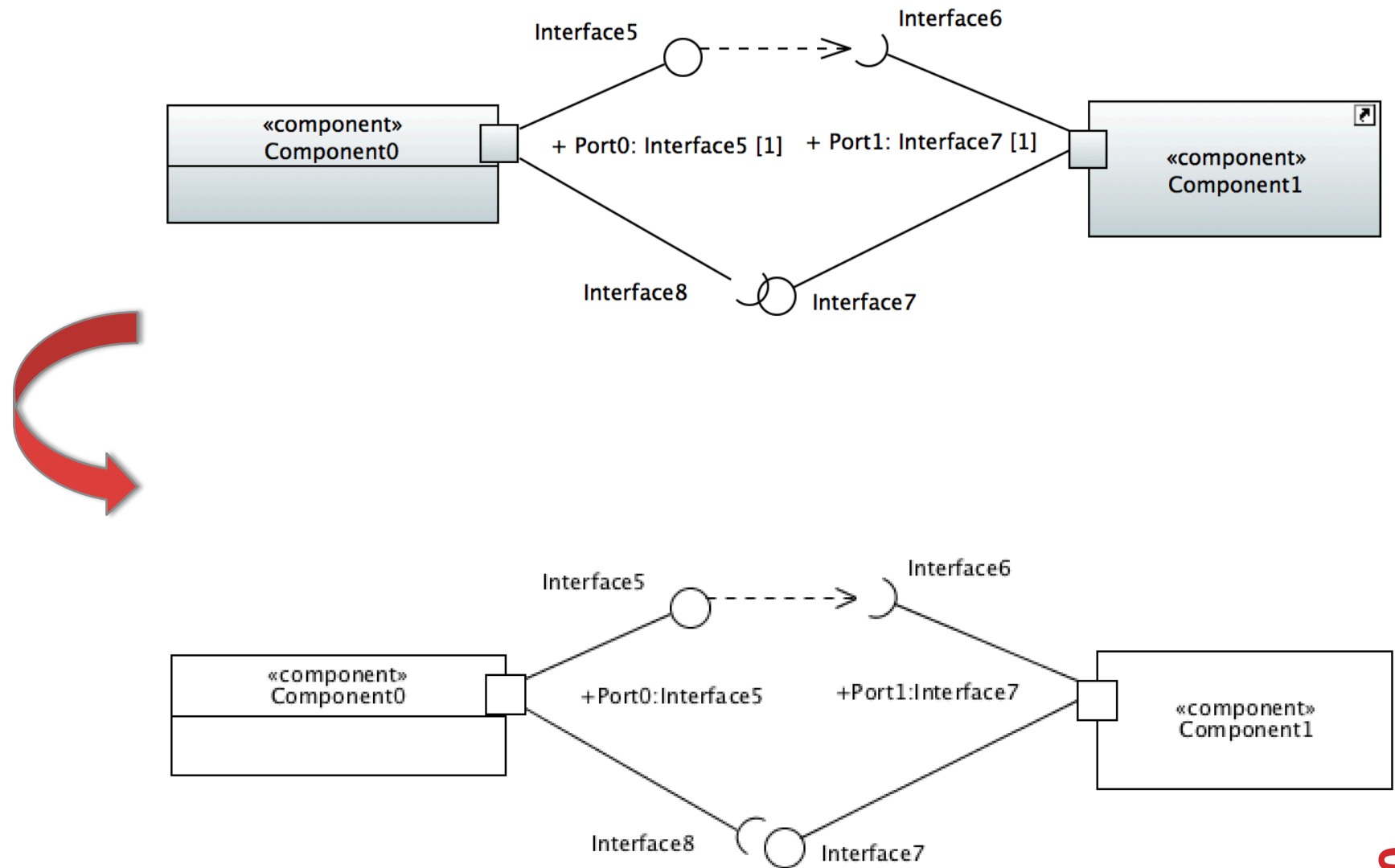
EXAMPLE: PACKAGE DIAGRAM



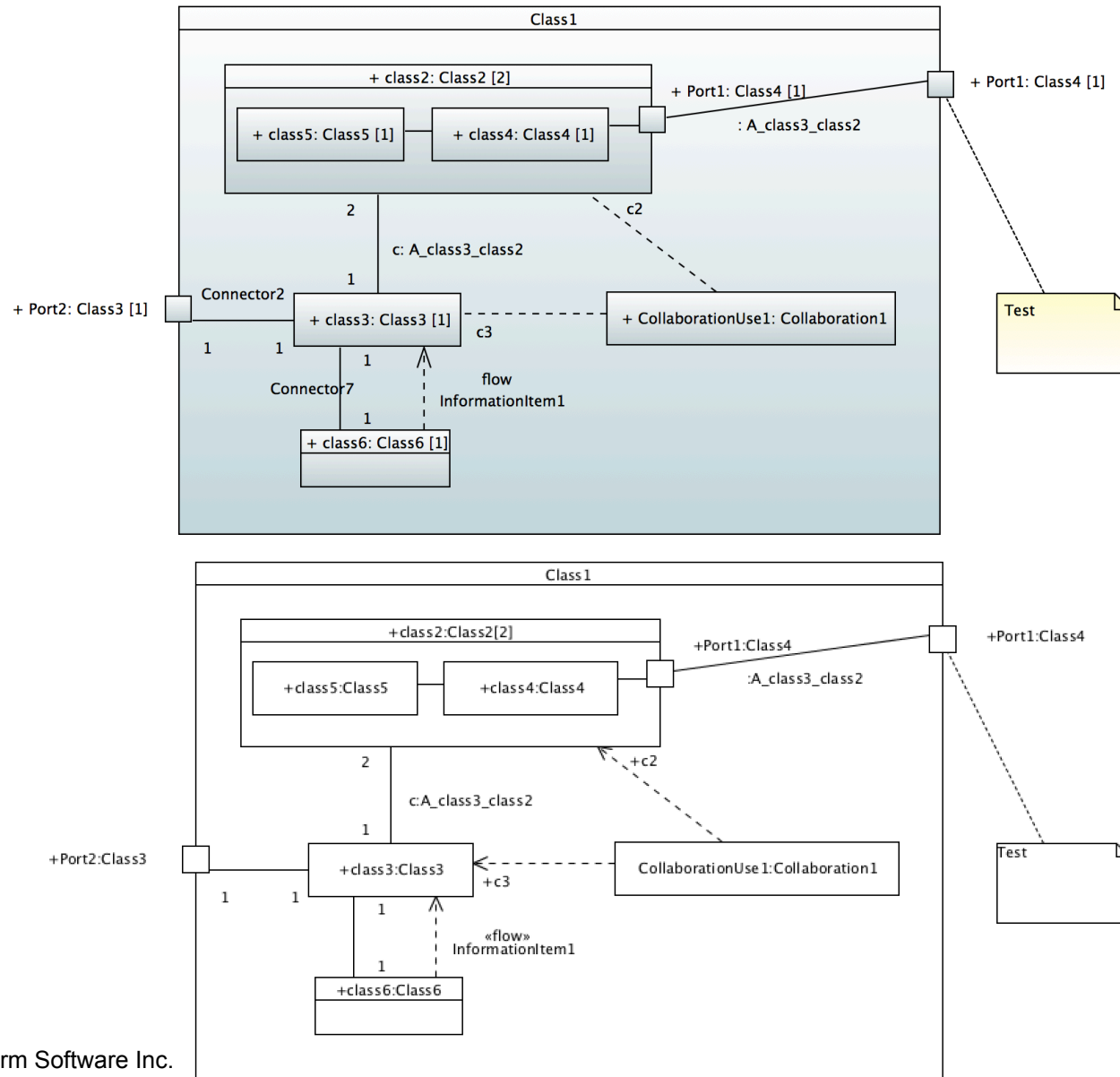
EXAMPLE: PROFILE DIAGRAM



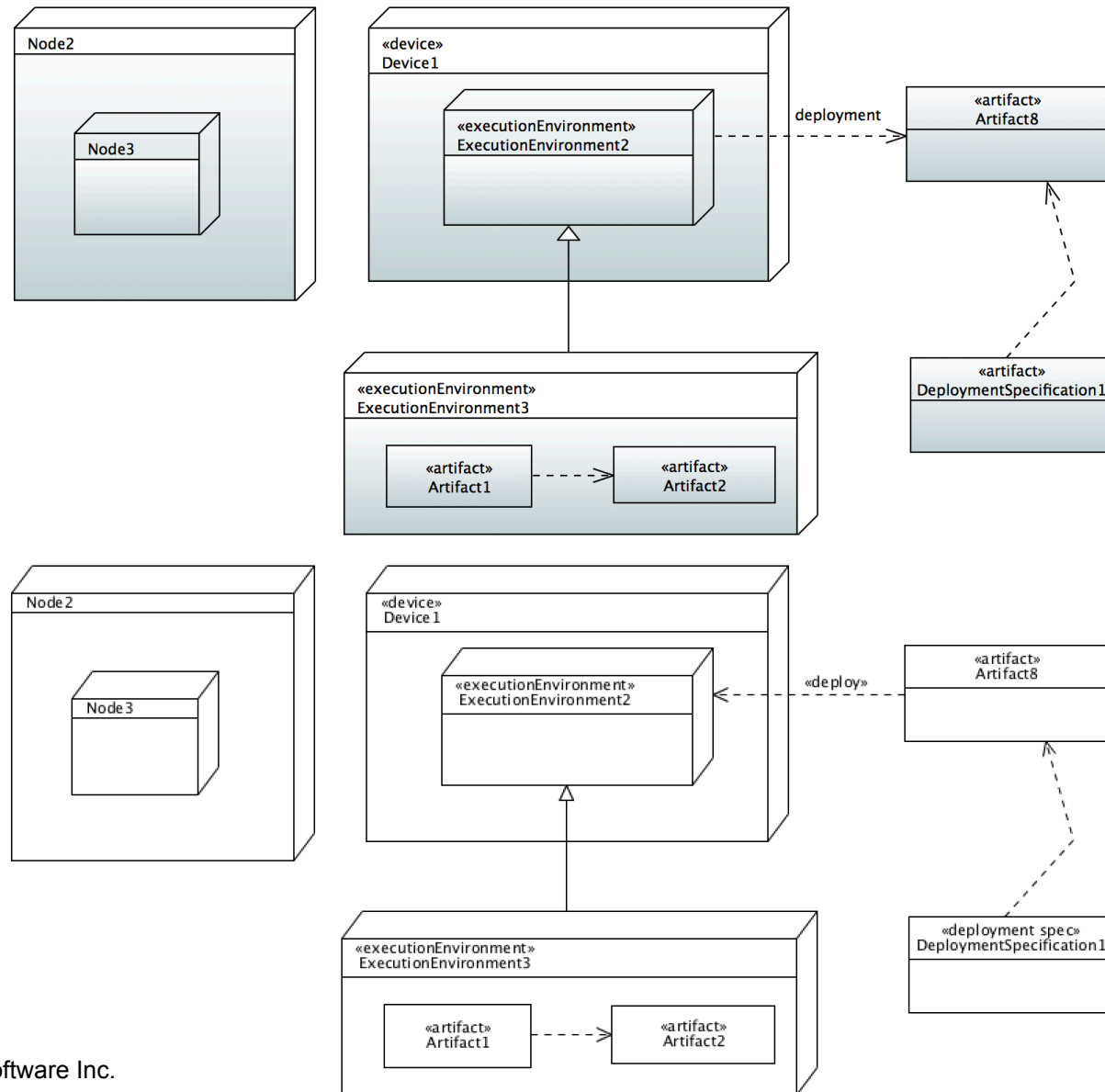
EXAMPLE: COMPONENT DIAGRAM



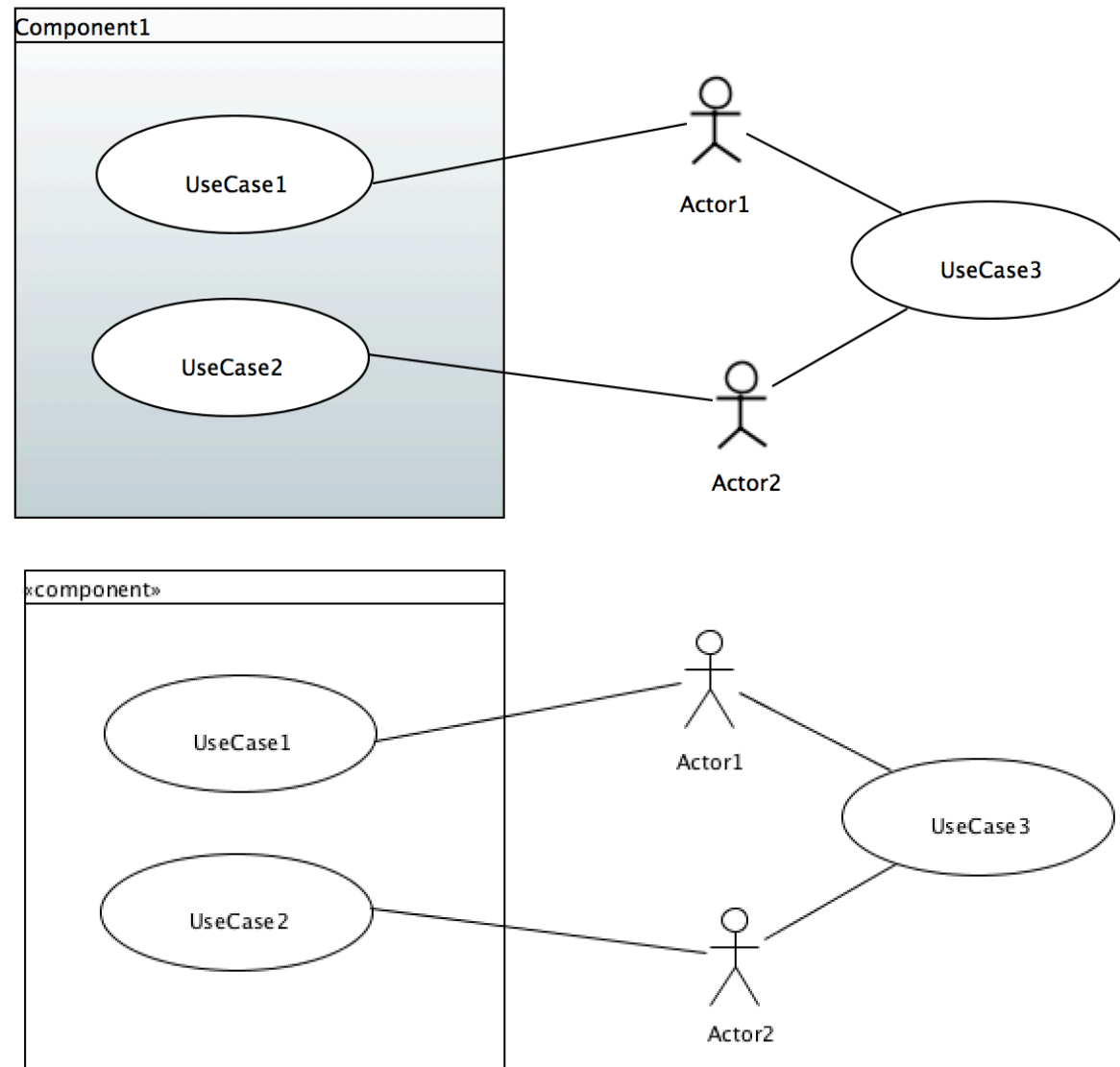
EXAMPLE: COMPOSITE STRUCTURE DIAGRAM



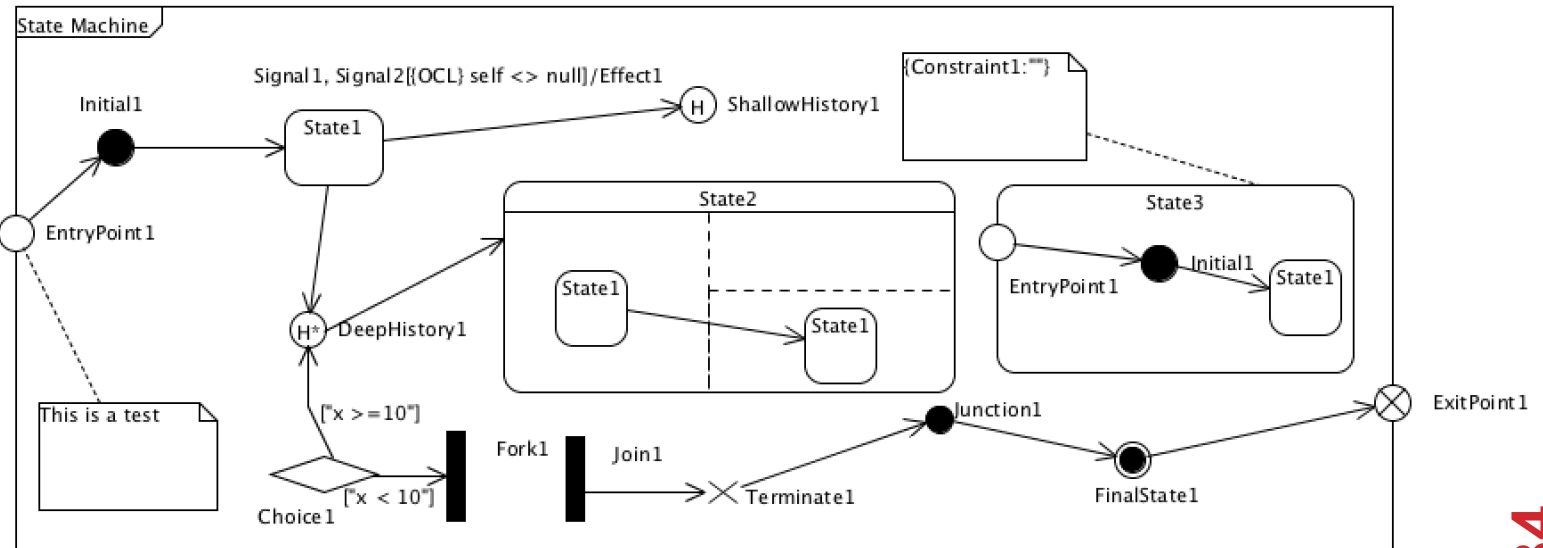
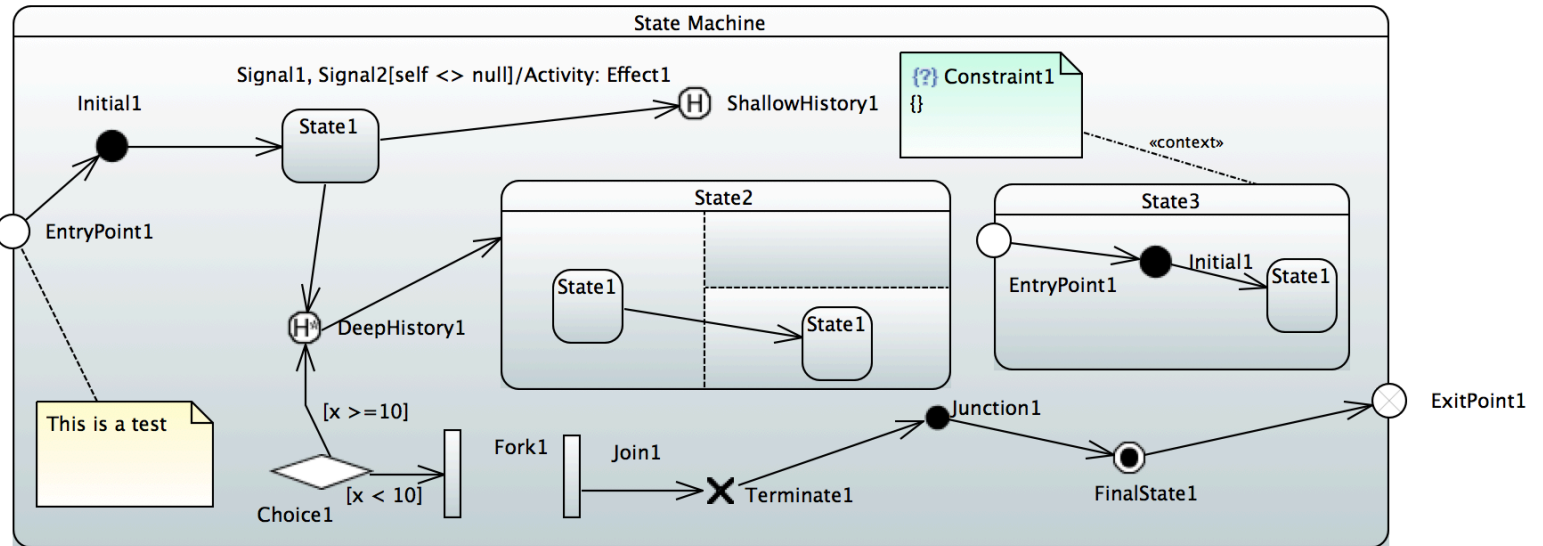
EXAMPLE: DEPLOYMENT DIAGRAM



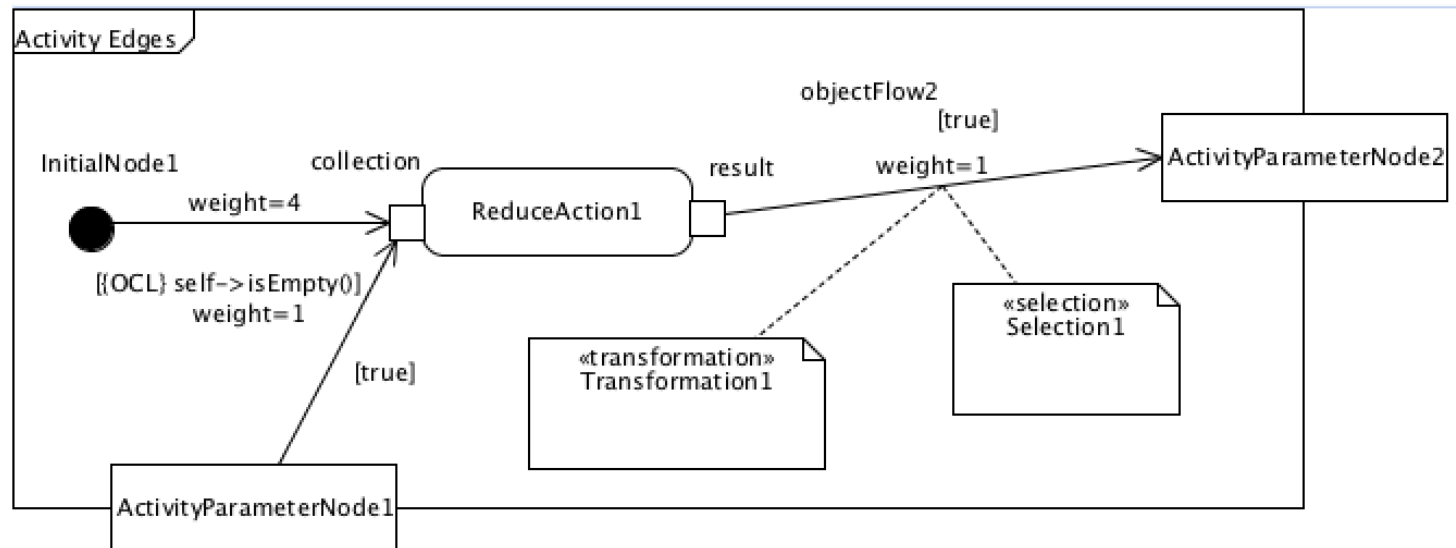
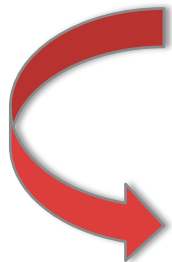
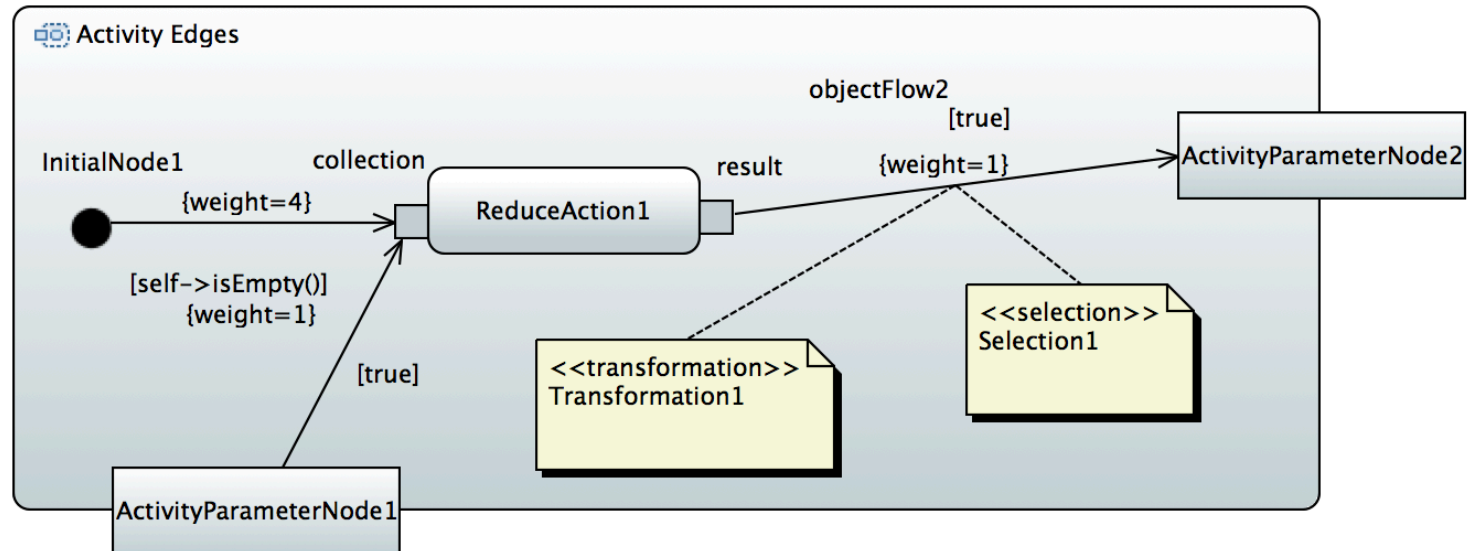
EXAMPLE: USE CASE DIAGRAM



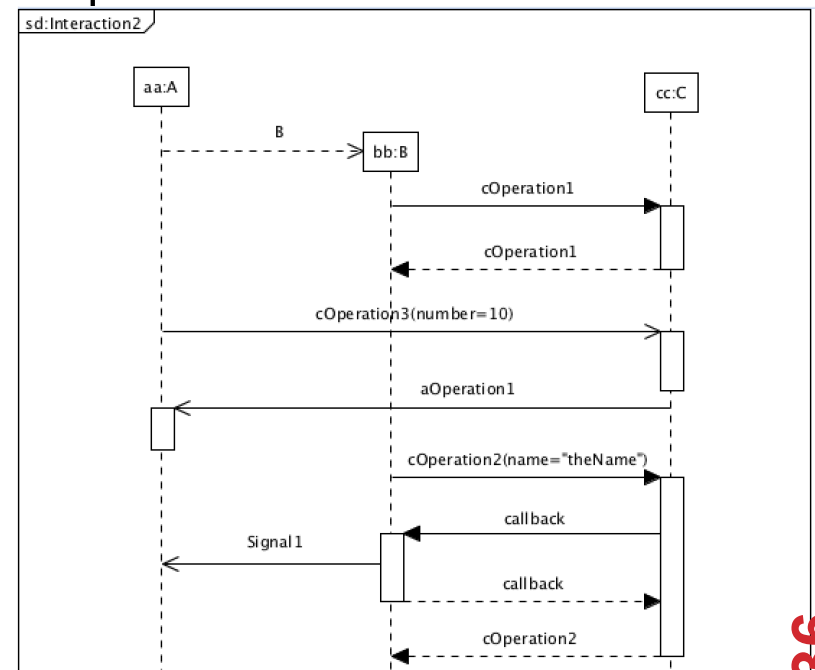
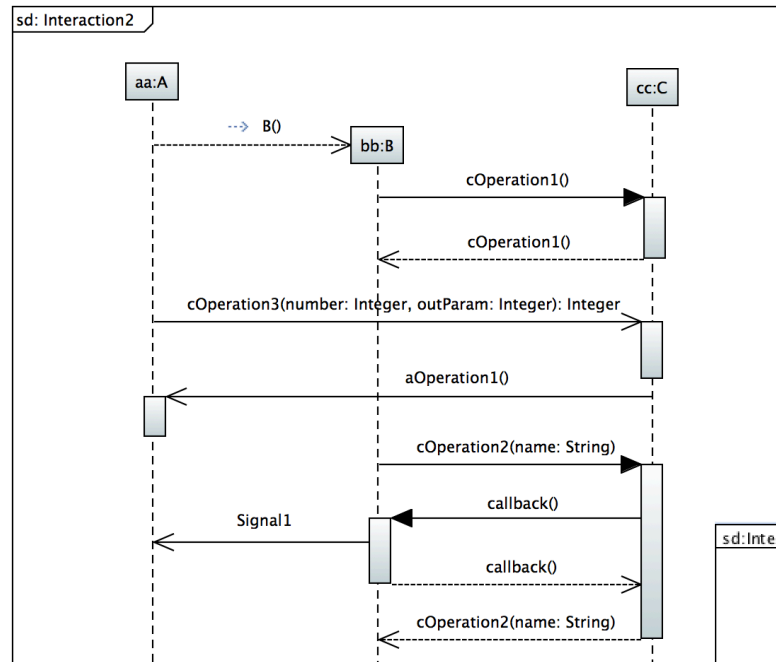
EXAMPLE: STATE MACHINE DIAGRAM



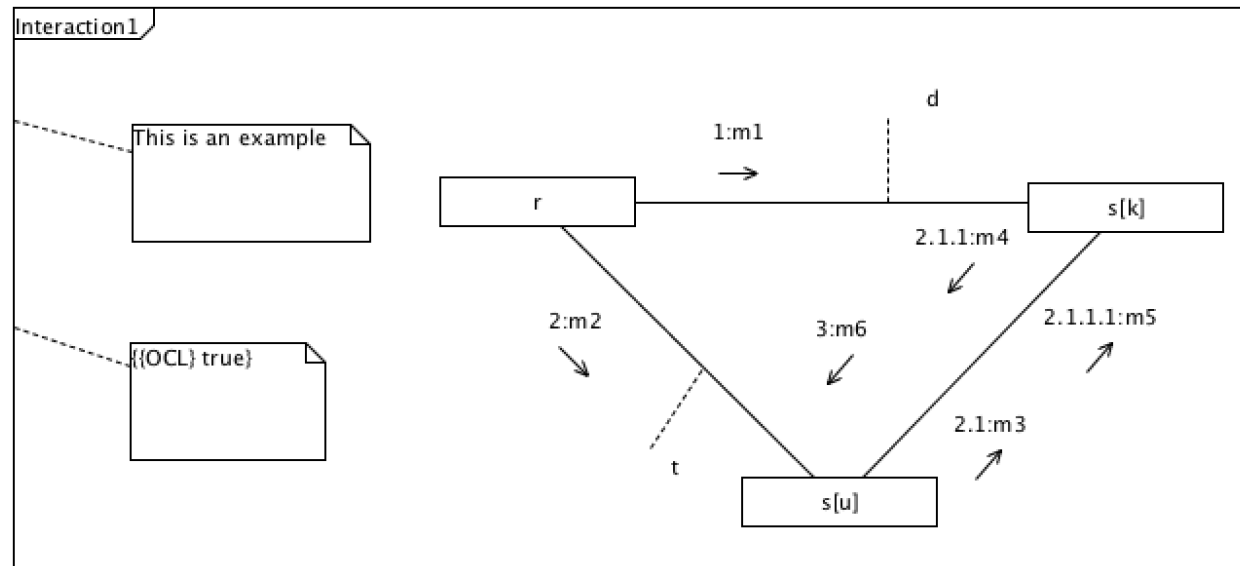
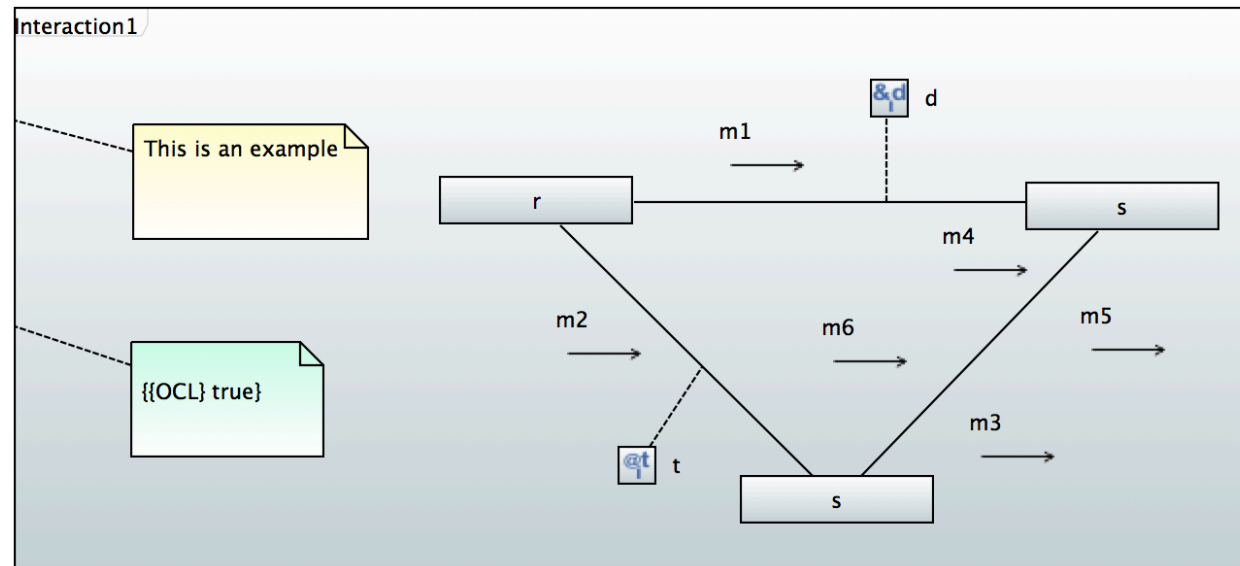
EXAMPLE: ACTIVITY DIAGRAM



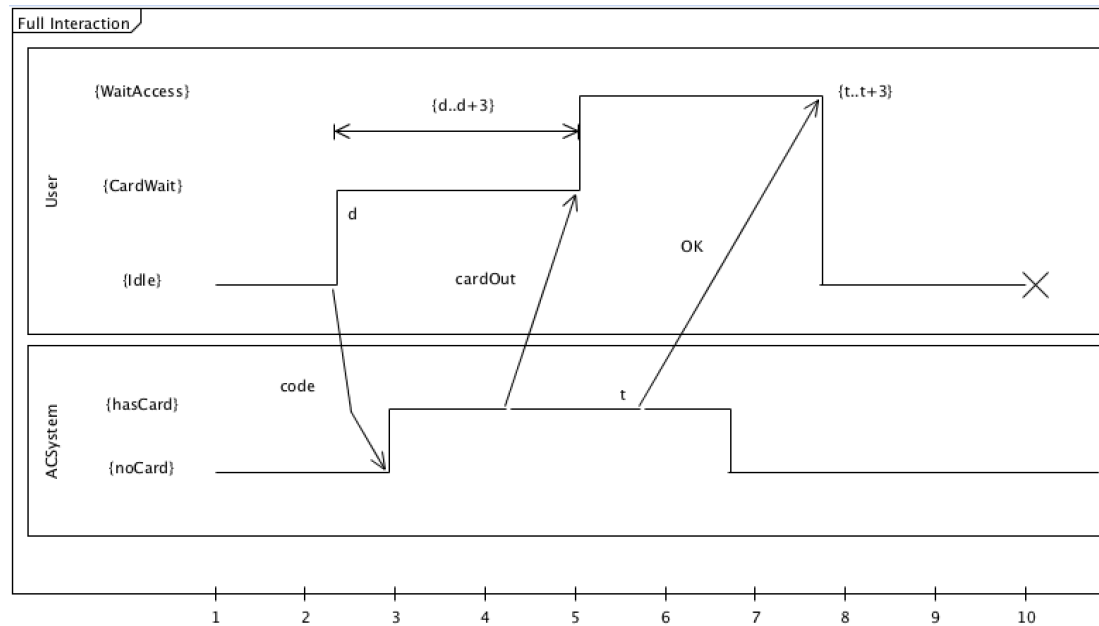
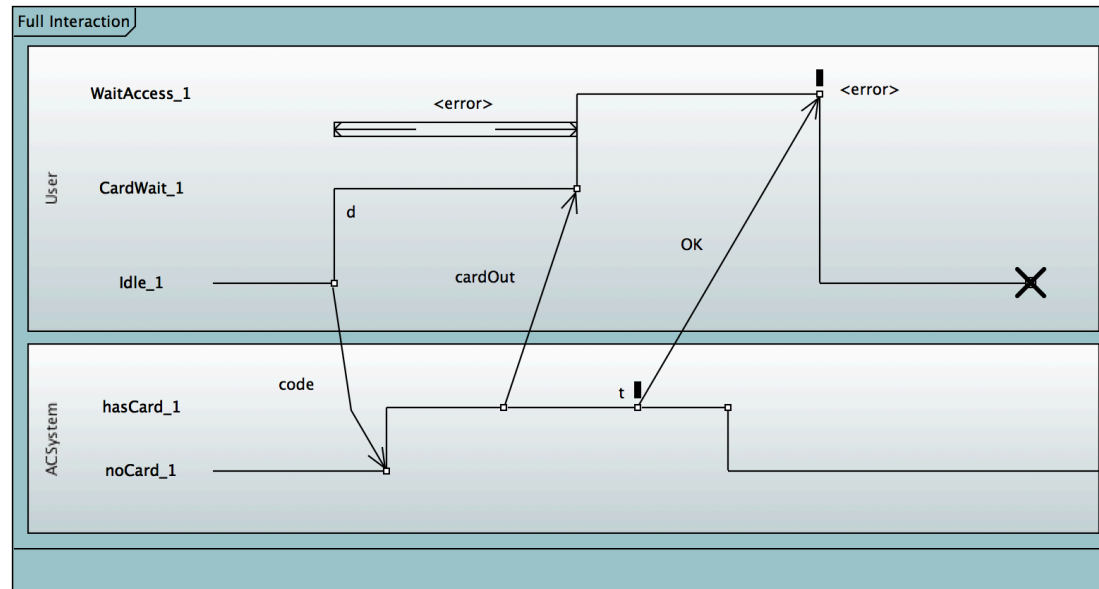
EXAMPLE: SEQUENCE DIAGRAM



EXAMPLE: COMMUNICATION DIAGRAM



EXAMPLE: TIMING DIAGRAM



NEXT STEPS

- Support the Eclipse DD project through its inoccupation phase
 - ▶ Fix issues reported against it
 - ▶ Redefine the DG to SVG mapping using Acceleo MTL
 - ▶ Remove dependency on local copy of Batik and depend instead on one from Orbit
- Improve the specifications
 - ▶ Push most of the changes made to the standard metamodels to the specifications
 - ▶ Contribute the DG to SVG mapping to the DD specification
 - ▶ Contribute the UMLDI to DG mapping to the UML specification
- Work on possible extensions:
 - ▶ Improve DG to incorporate declarative layout support
 - ▶ Design a DSL that consolidates DI and DG mapping specification
 - ▶ Experiment with bi-directional transformations to support diagram-based model editing
 - ▶ Specify DD specification for a Profile (e.g., SysML)
 - ▶ Generalize DD into a View/Viewpoint architecture specification
 - Ability to describe diagram as well as form and other document-based viewpoints