

# **Practical UML Modeling with Rational Software Architect**

# Basic Facts

- IBM Rational Software Architect is the latest generation of Rational Rose UML modeling tools
  - ROSE = Rational Object-oriented Software Engineering
  - First developed by James Rumbaugh, Ivar Jacobson, and Grady Booch in Rational Software as part of efforts to support model-based design for object-oriented languages.
    - These three guys proposed UML.
    - Rational Rose was introduced in OOPSLA'92 as the first UML-based modeling tool.
- Rational was acquired by IBM in December 2002 for 2.1 Billions.
- Rational Rose and its successor IBM Rational Software Architect are still the leading UML-based modeling tools by far.

# Basic Facts

- Rational Software Architect is an integrated visual modeling tool for development of object-oriented software.
- Rose uses UML to provide graphical methods for non-programmers wanting to model business processes as well as programmers modeling application logic.
- Developed on the Eclipse platform.
  - Eclipse is a platform for building, deploying, and managing software across the lifecycle.
  - The Eclipse platform encourages rapid development of integrated features based on a **plug-in** model.
  - A wider range of plug-in is available, including the support for SVN.

# Homework/project use

- Rational Software Architect (RSA) has been installed on all CS lab machine and ELEC.
  - To launch it, use “/cslab/bin/rational”
- You can obtain a copy of RSA for use in senior design.
  - Full featured standard edition for Linux/Windows. Free of charge. To obtain it,
    - Download the IBM academic license agreement form by following the link in the course website;
    - Send me an email containing the following sentence: “I have read and accepted IBM agreement for IBM academic initiative.”
    - Upon receiving your email, I will send you instruction to download your copy.

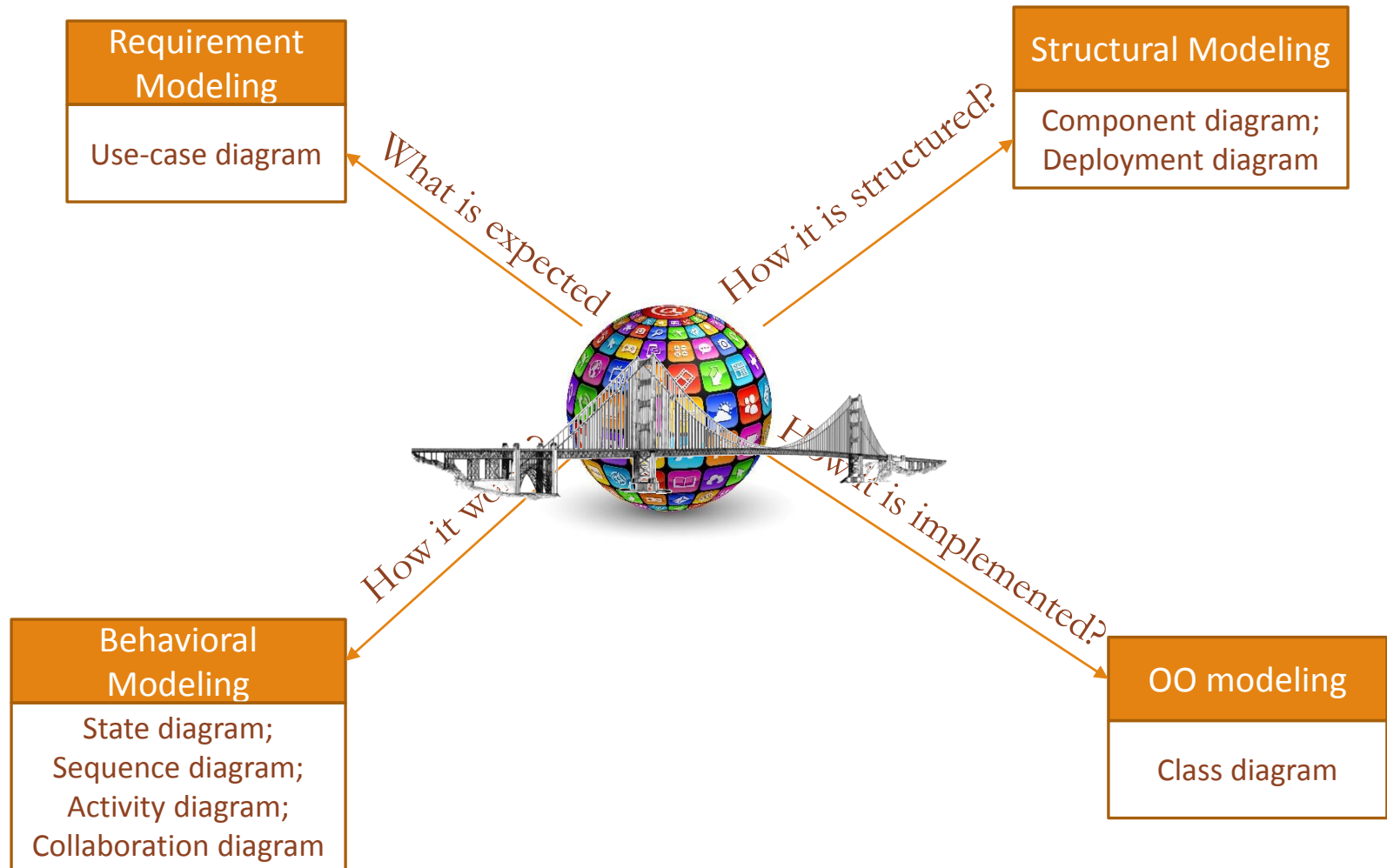
# **UML: an overview**

# Why design matters

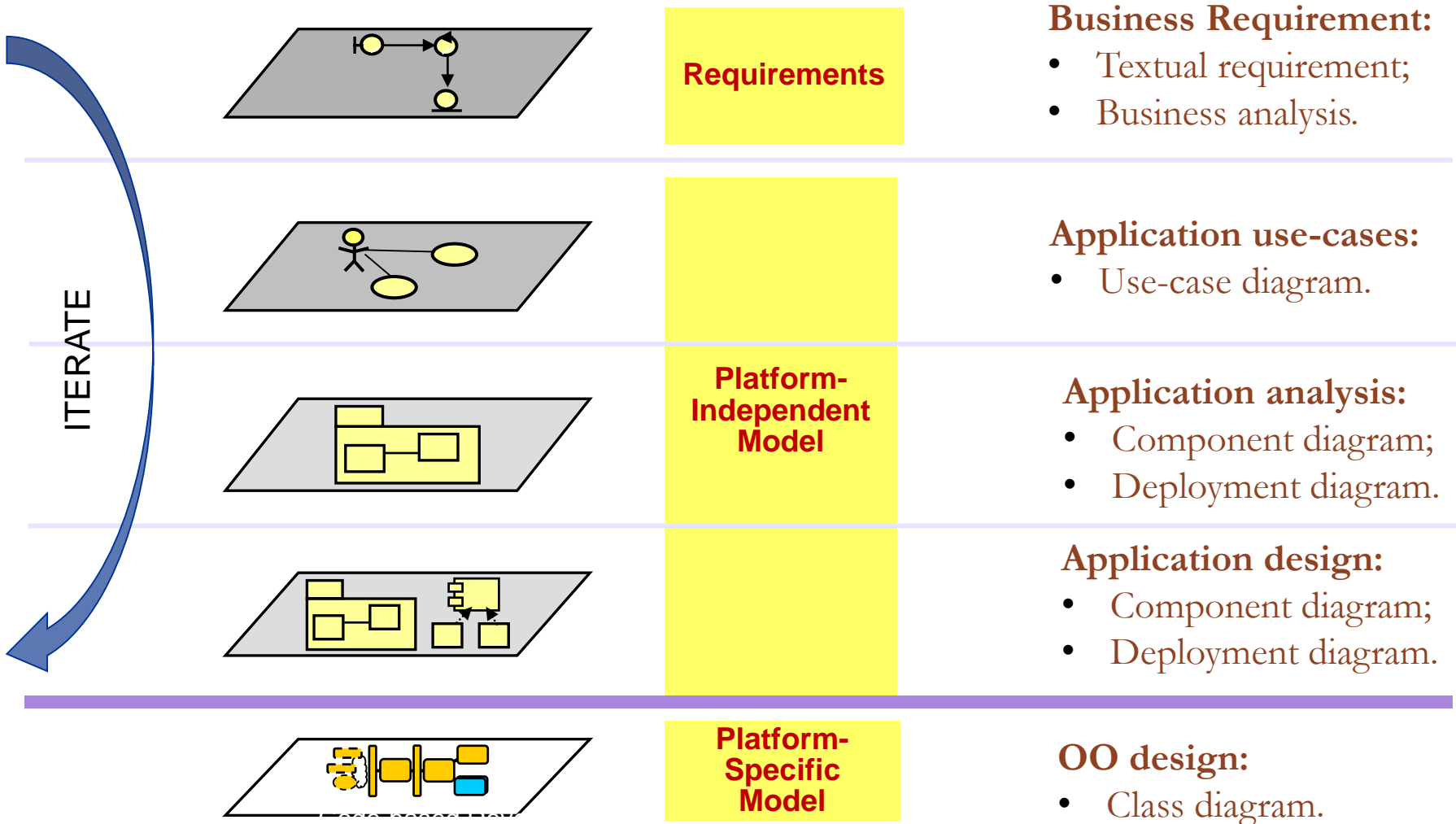


How the customer explained it

# UML: 360° view of design



# UML: design workflow

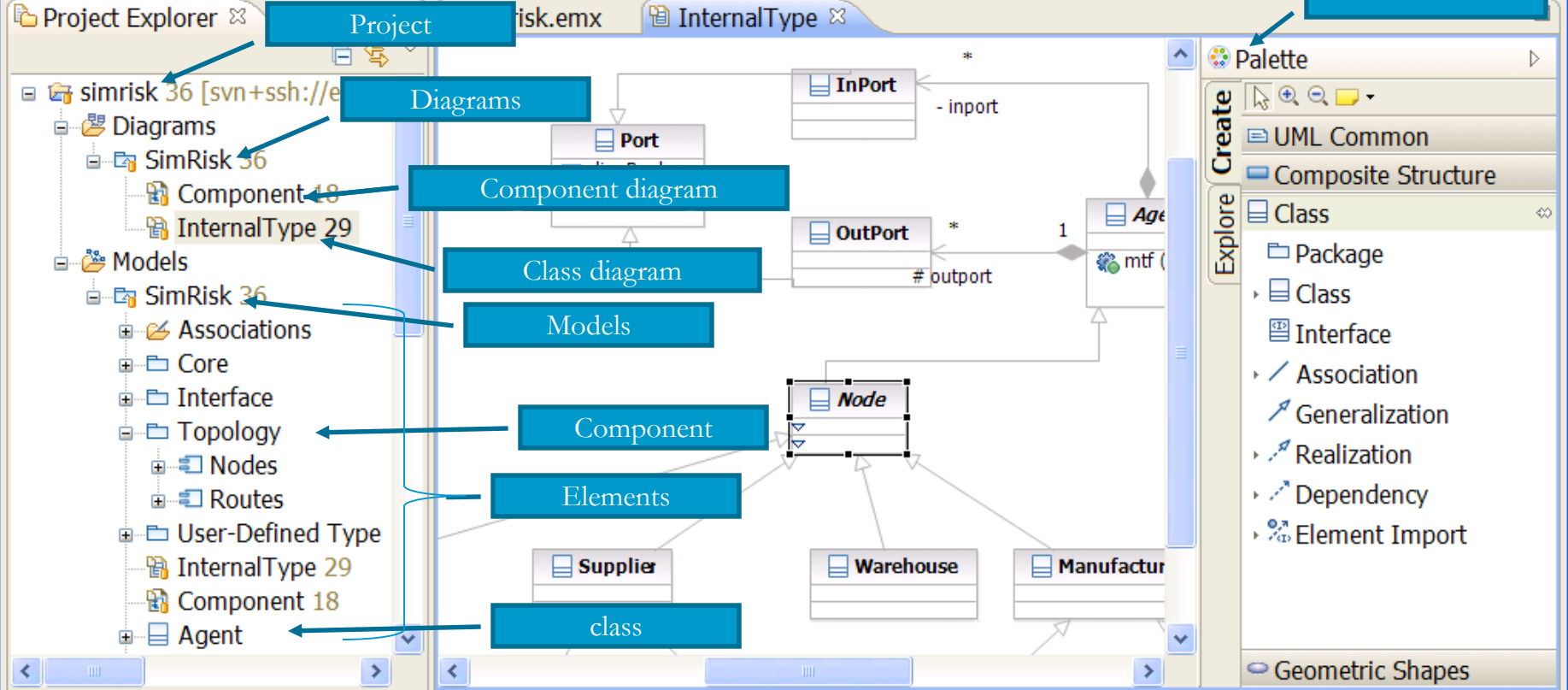




# UML Modeling with Software Architect

Rational Architect supports UML 2.0, including the following diagrams,

- Use-case diagram
- Class diagram
- State Diagram (StateChart)
- Interaction Diagrams, including,
  - Sequence diagram
  - Collaboration diagram
- Component Diagram
- Deployment Diagram
- ...



History Tasks Problems Properties

<Class> SimRisk::Node

<b>General</b>	Name:	Node
<b>Attributes</b>	Visibility:	<input checked="" type="radio"/> Public <input type="radio"/> Private <input type="radio"/> Protected <input type="radio"/> Package

Not connected

<No Current Work>

# UML modeling with Software Architect: examples

# Software Architect: IDE

Software Architect uses a hierarchy to specify models.

- Each session contains a workspace.
  - At the launch, software architecture will ask your workspace directory.
  - Among other things, it store your personal configuration for underlying projects.
- Each workspace contains multiple projects.
  - A project is the collection of models, or a case, you are working on.
  - It can be located in a directory different from workspace directory.
  - It can be shared by different workspace.
  - There is a variety way to incorporate a project into a workspace, e.g., using existing project files, using CVS or SVN etc.

# Import

Select

Choose import source

To activate this dialog, right click in  
Project explorer, choose "import project"



Select an import source:

type filter text

- General
  - Archive File
  - Existing Projects into Workspace
  - File System
  - Preferences
- C/C++
- Code Coverage
- CVS
- Jazz Source Control
- Plug-in Development
- Profiling and Logging
- RAS
- Run/Debug
- SVN
  - Project from SVN

Add existing project to a workspace

Import project directly from a SVN repository.



< Back

Next >

Finish

Cancel

# Software Architect: IDE

- Each model may contain several UML diagrams to express the different “views” of a model, for example,
  - A class diagram provides inheritance hierarchy and associations among classes.
  - A component diagram provides the architect design.
  - Software architect supports diagrams defined in UML 2.0.
- Each model may contain several elements, whose relations are defined in diagrams.
  - RSA lists all the elements with their hierarchy in project explorer for easy navigation of these elements.

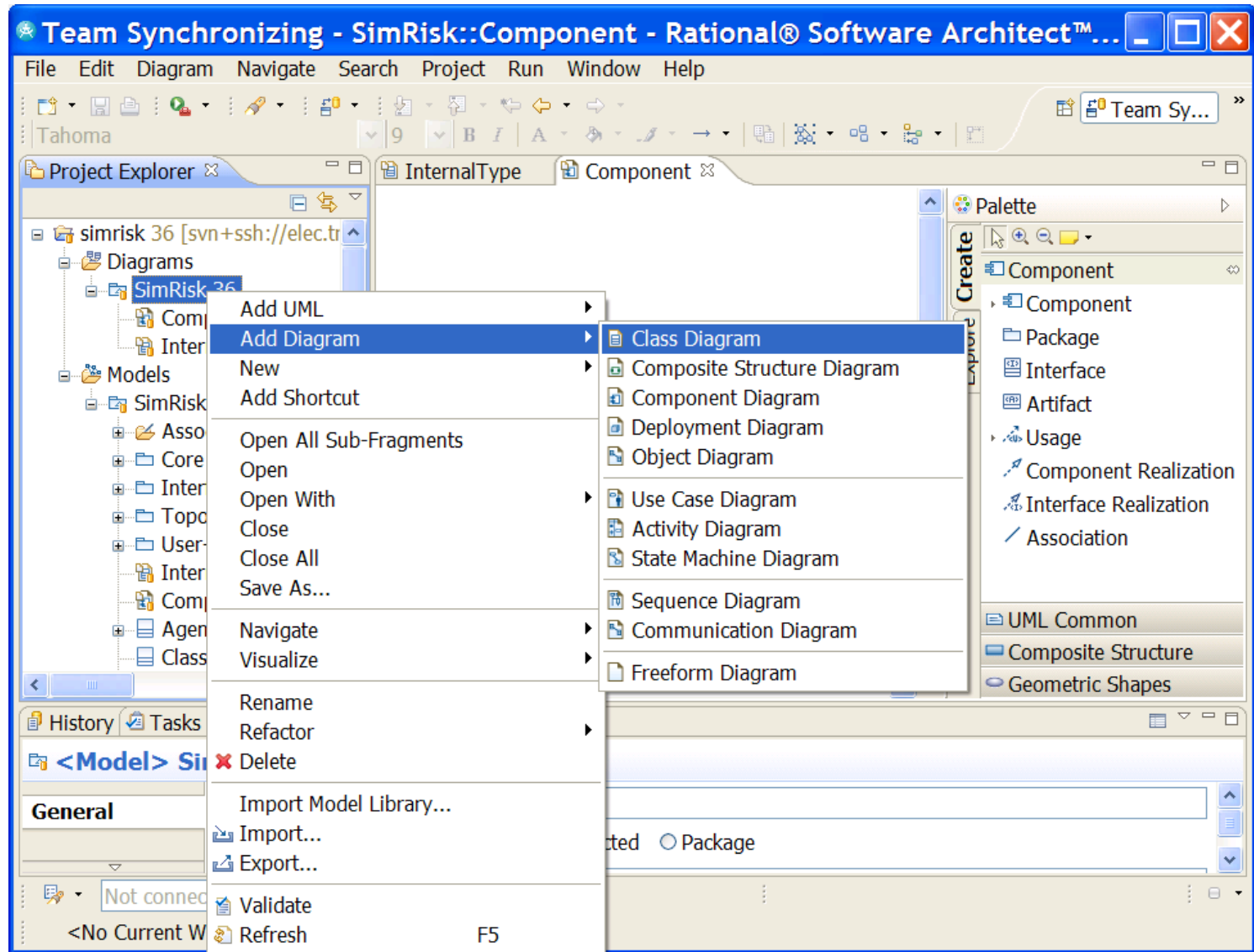
# Software Architect: IDE

- Elements are part of “unified” model and they are shared by different diagrams,
  - Sharing elements provide consistent among diagrams, for example, a class in a class diagram can also be used to,
    - define an object in a sequence diagram.
  - This is very useful in a team development environment, where different designers work on different diagrams (and aspects) for the same model.
    - To maintain the consistency, you need to merge models developed by different designers.

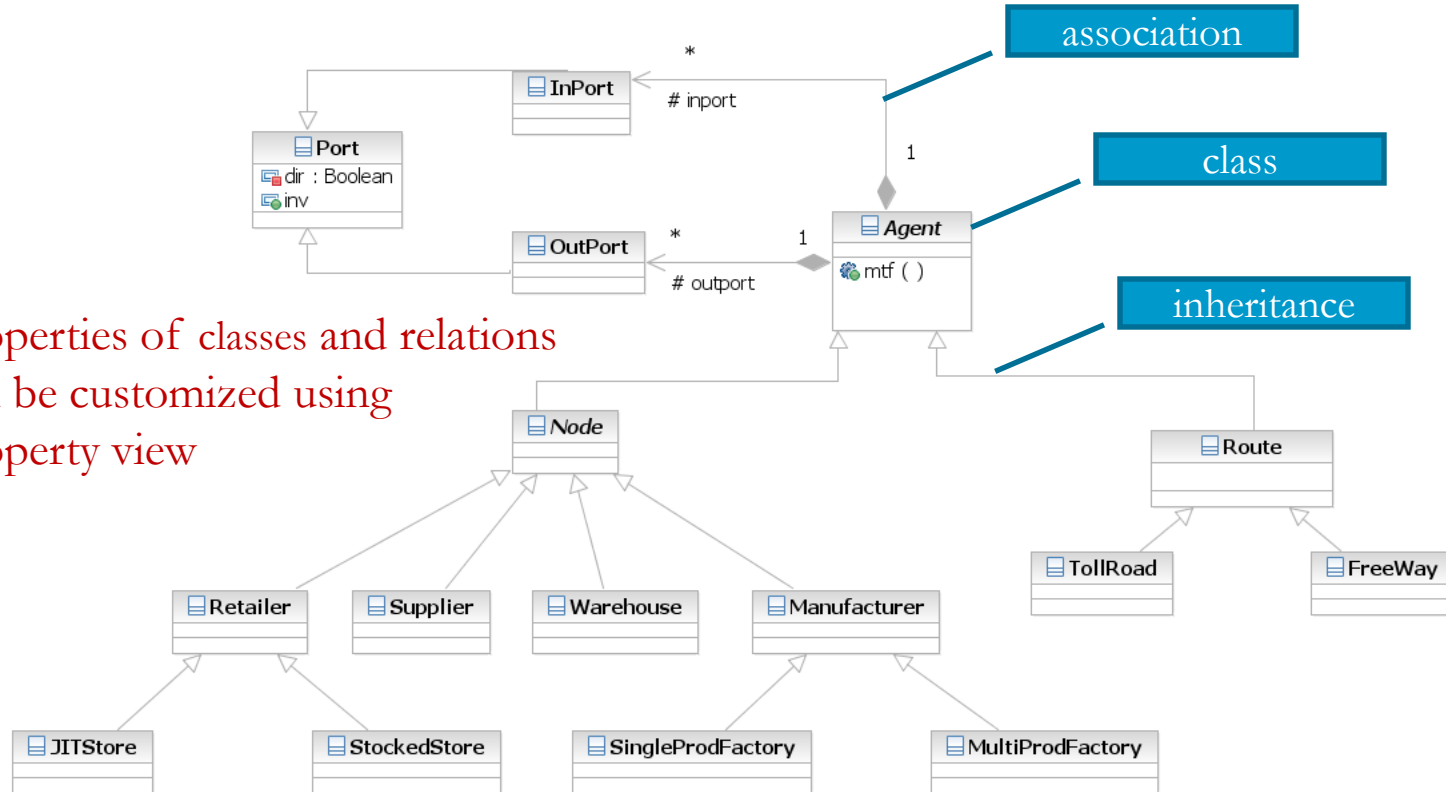
# UML modeling with Software Architect: examples



# Class Diagram



# Class Diagram



Properties of classes and relations can be customized using Property view

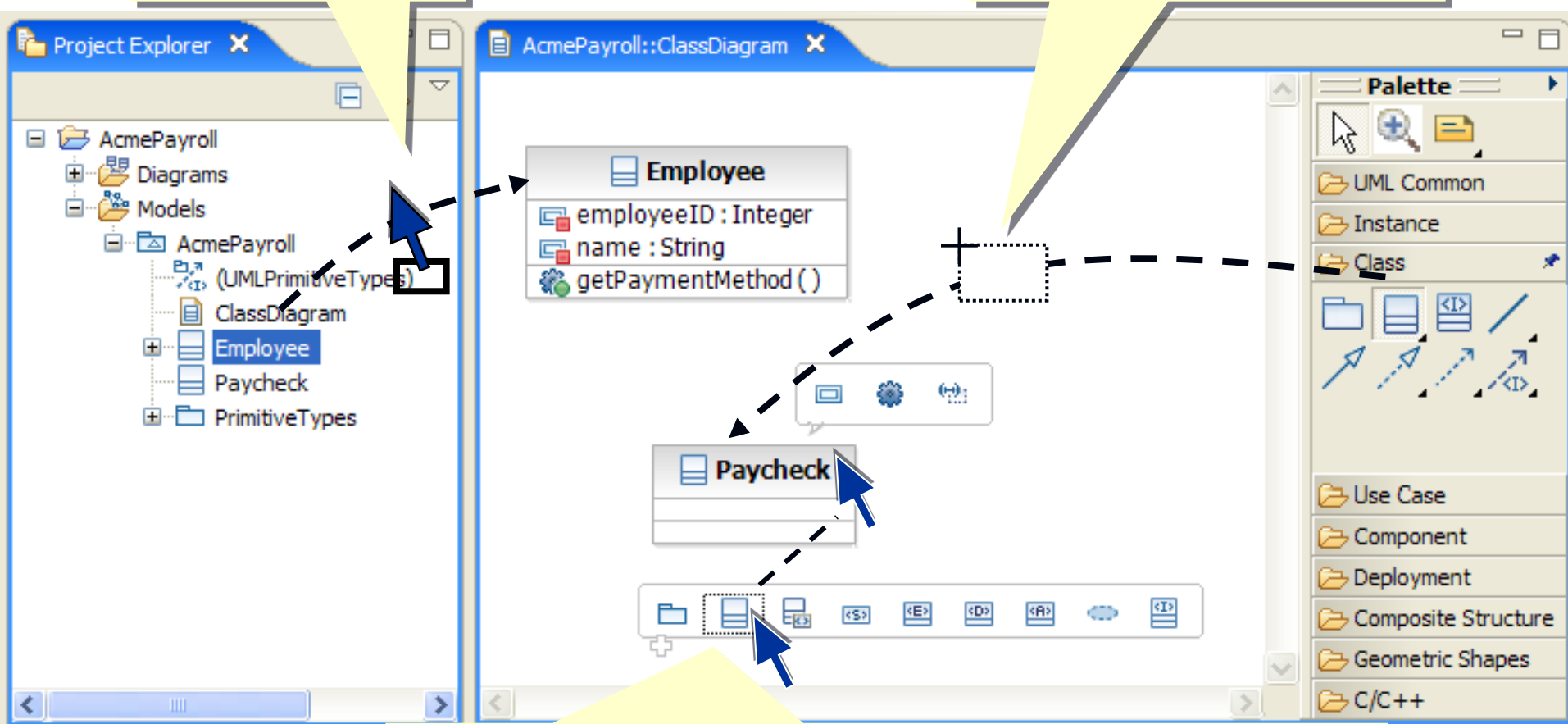
# Adding Elements

1

Drag an existing element from the Project Explorer view.

2

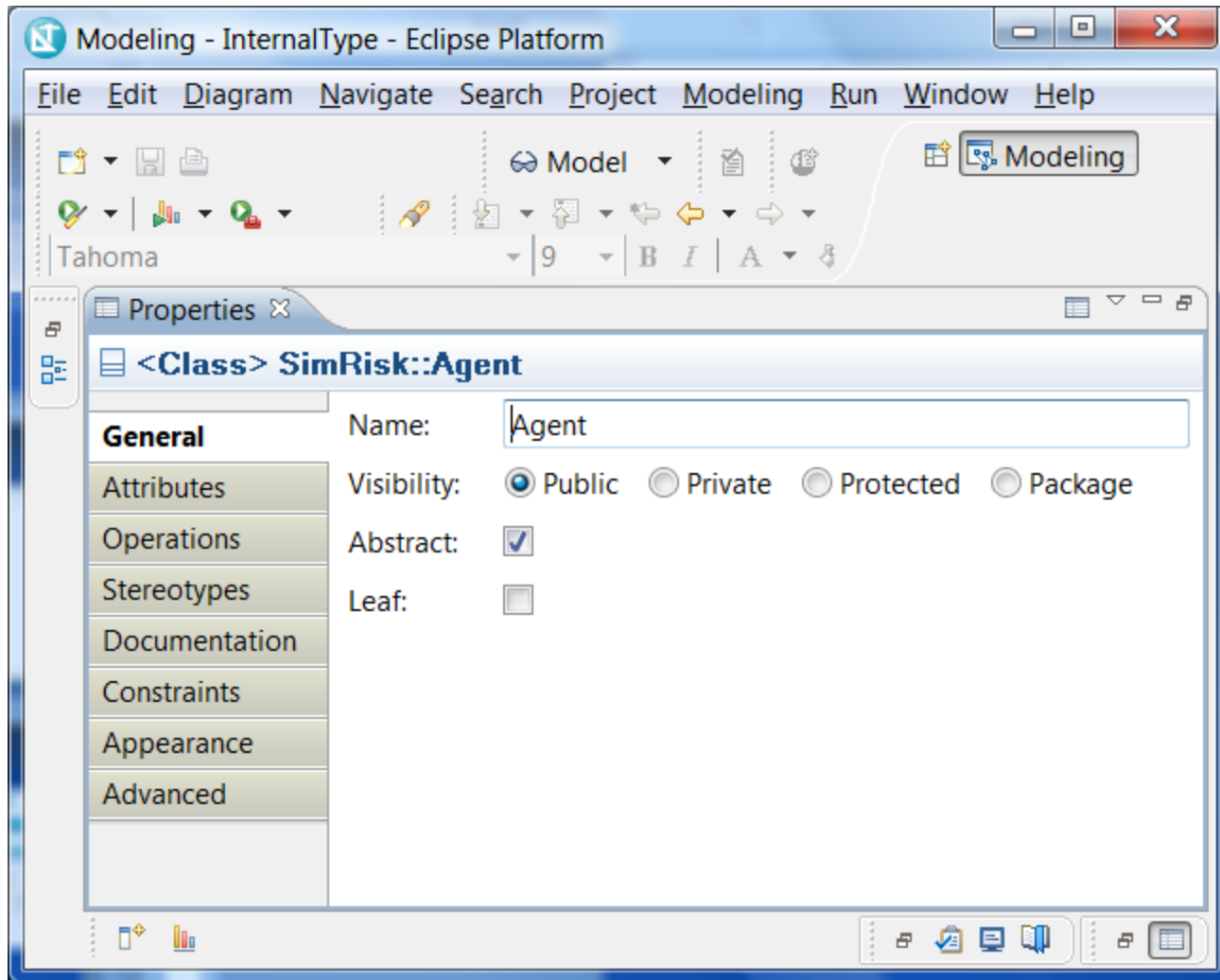
Click a tool in the diagram editor's tool palette and then click inside the diagram.



3

Hover or click the mouse in the diagram to make the action bar appear. Select the element to insert. Hover over the new model element to use the action bar to add details (attributes and operations).

# Property of Class



# Property of Class

The screenshot shows the Eclipse IDE interface with the Properties view open. The Properties view is titled "Properties" and shows the details for the class "<Class> SimRisk::Agent". The view is organized into sections: General, Attributes, Operations, Stereotypes, Documentation, Constraints, Appearance, and Advanced. The "Attributes" section is currently selected and expanded, displaying a table of class attributes.

Name	Type	Default Value	Visibility
outport	<Class> OutPort		Protected



# Property of Association

The screenshot displays the Eclipse Modeling Platform interface. The main window is titled "Modeling - InternalType - Eclipse Platform". The menu bar includes File, Edit, Navigate, Search, Project, Modeling, Run, Window, and Help. The toolbar contains various modeling tools. The Properties view is open, showing the configuration for an association named "<Association> (outport:OutPort)".

The association diagram shows a directed association between the class **Agent** and the class **OutPort**. The association line has an arrowhead pointing towards **OutPort**. The role name "outport" is written near the arrowhead. The multiplicity at the **Agent** end is "1", and the multiplicity at the **OutPort** end is "\*".

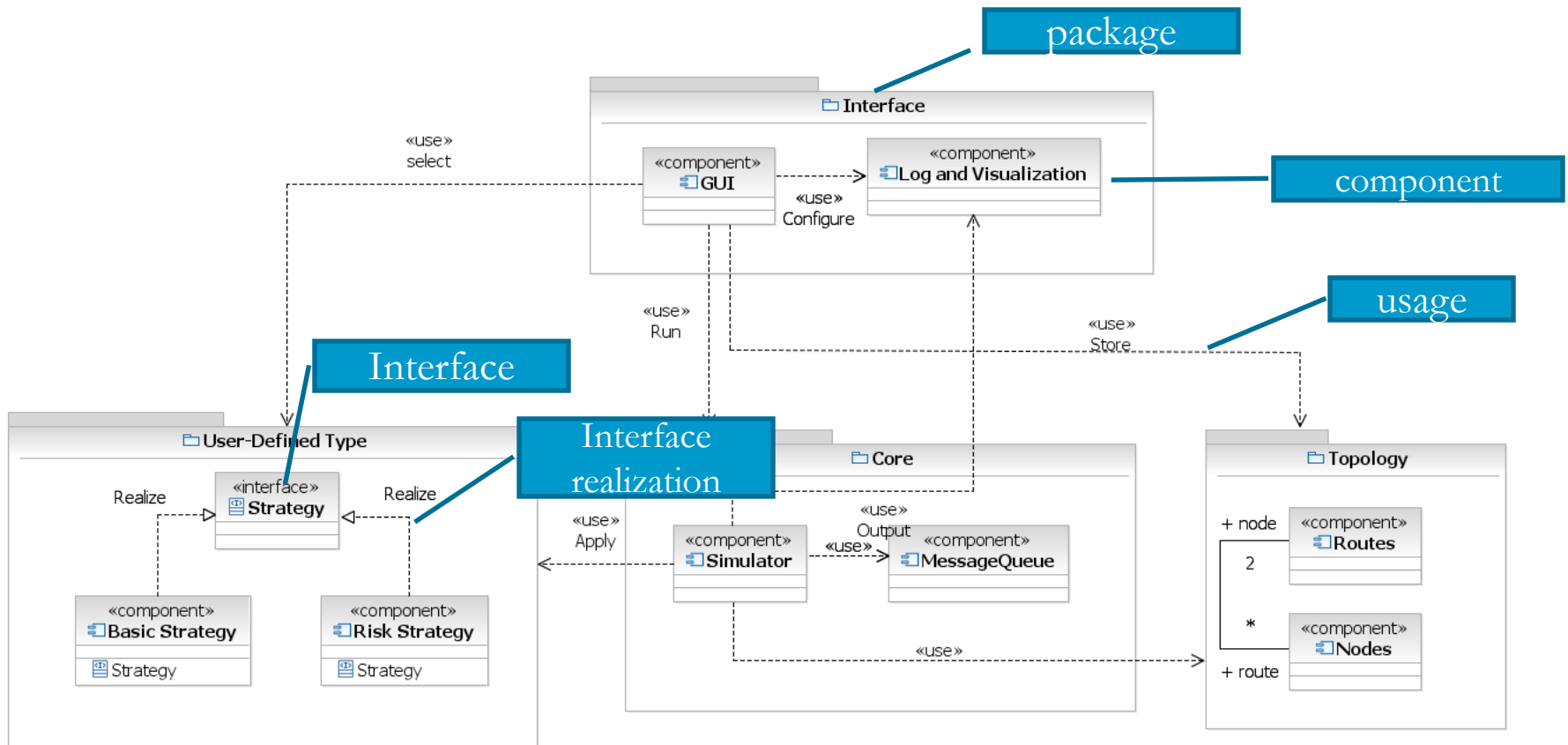
The Properties view is divided into sections:

- General**: Shows the association name and a "Change Direction" button.
- Stereotypes**: Empty.
- Documentation**: Empty.
- Constraints**: Empty.
- Appearance**: Empty.
- Advanced**: Empty.

The configuration for the two ends of the association is as follows:

End	Role	Multiplicity	Is navigable	Leaf	Aggregation
Agent		1	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="radio"/> Composite
OutPort	outport	*	<input checked="" type="checkbox"/>		<input checked="" type="radio"/> None

# Component Diagram





# Property Views

The screenshot displays the Rational Software Architect interface. The main workspace shows a UML diagram with a component labeled 'Component' containing two packages, 'Core' and 'Topolo'. A 'Configure' component is shown with dashed arrows indicating dependencies: one labeled '<<use>> Run' pointing to 'Core', and another labeled '<<use>> Store' pointing to 'Topolo'. The 'Properties' view at the bottom is titled '<Usage> Store' and shows a dependency from a 'GUI' component to a 'Topology' component. The 'Name' field is set to 'Store'. The Project Explorer on the left shows a project structure with 'simrisk 36' containing 'Diagrams', 'SimRisk 36', 'Component 37', 'InternalType 29', and 'Models'. The status bar at the bottom indicates 'Not connected' and '<No Current Work>'.

# Property Views

The screenshot displays the Rational Software Architect interface. The main window shows a UML diagram titled "InternalType" within a "User-Defined Type" container. The diagram features an interface named "Strategy" and two components, "Basic Strategy" and "Risk Strategy", both of which realize the "Strategy" interface. A "Simulator" component is also shown, which uses the "Strategy" interface and is associated with the "Risk Strategy" component via a "use" relationship.

The "Properties" view at the bottom of the window is titled "<Interface Realization> Realize". It shows a relationship between the "Risk Strategy" component and the "Strategy" interface. The "Name" field is set to "Realize", and the "Mapping" field is currently empty. The "General" tab is selected in the left-hand pane of the properties view.

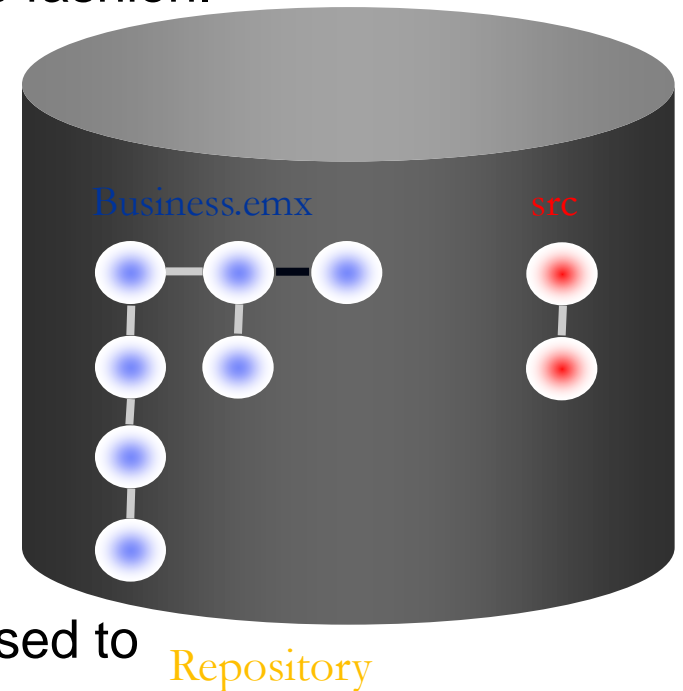
The Project Explorer on the left shows the project structure for "simrisk 36", including "Diagrams", "SimRisk 36", "Component 37", and "InternalType 29". The status bar at the bottom indicates "Not connected" and "<No Current Work>".

# **RSA in action**

**Integration with software development process**

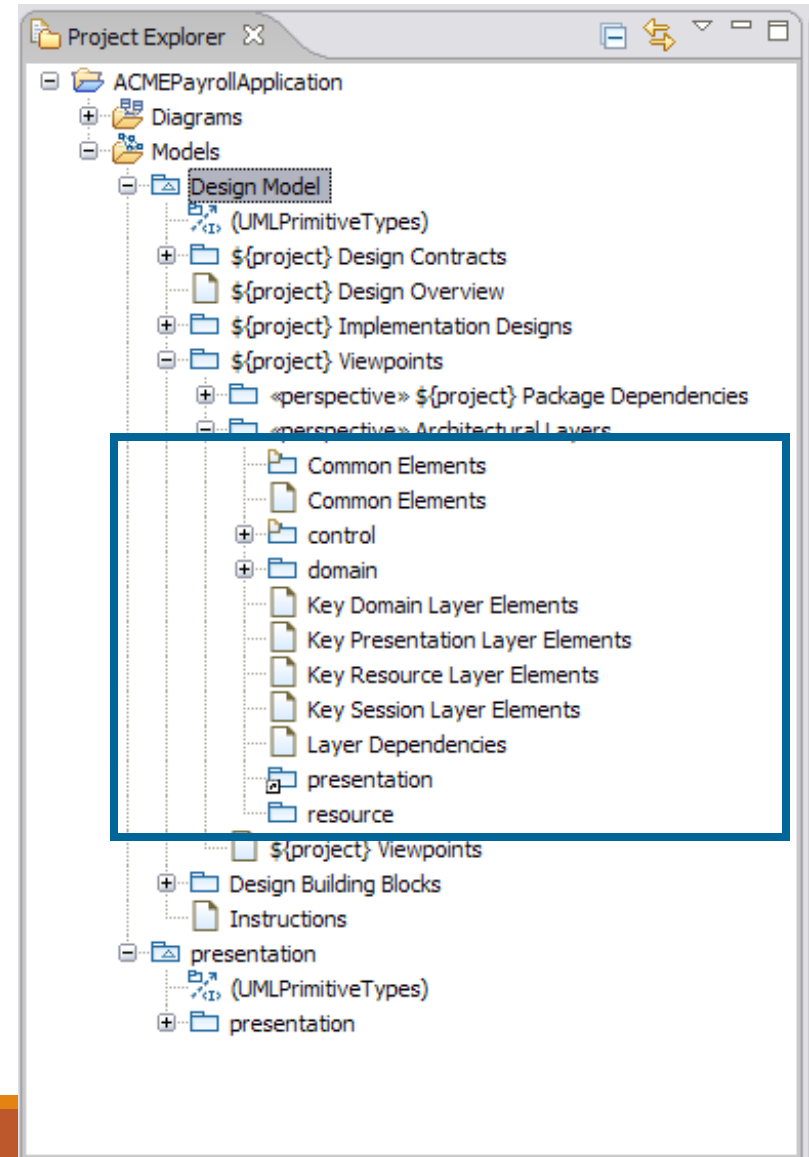
# Configuration Management (CM)

- CM allows change in software assets to occur in a structured, controlled, and repeatable fashion.
- A CM tool can:
  - Give team members simultaneous access to models
  - Control who can update different model elements
  - Help introduce changes in a controlled manner
  - Maintain the evolutionary history of a model and its elements
- A CM process defines how tools will be used to manage change in a project.
- We use SVN for CM.





# SCM Best Practices: Model Partitioning

- Partition a model to avoid unnecessary merges
- Factors to consider when deciding how to partition a model:
  - Stabilize abstraction levels
  - Minimize dependencies between models
  - Establish ownership policies
  - Avoid broken references

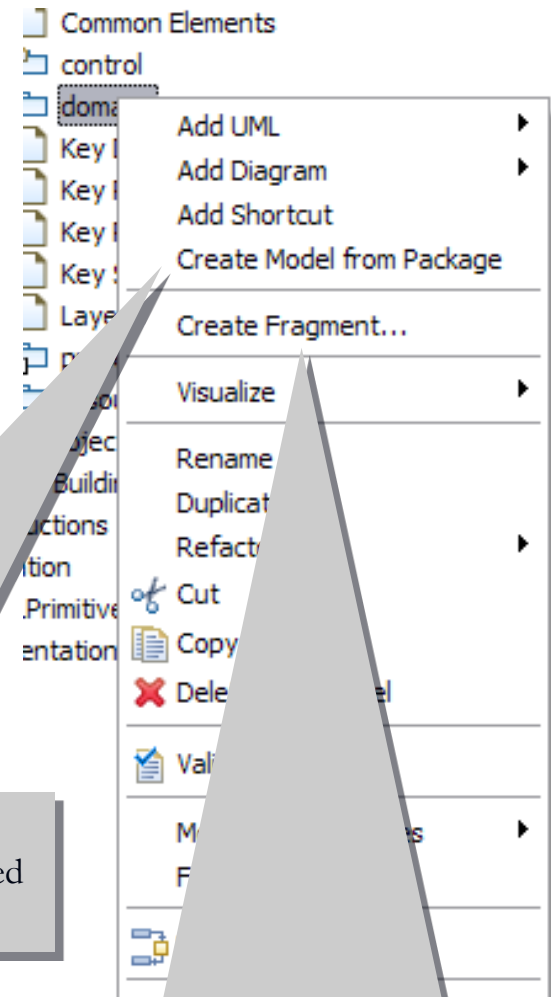


# Model Partitioning

- To manage partitions, you can:
  - Create fragments
    - An element that has been converted into a fragment has this icon: 
  - Create new models from packages
    - A package that has been made into a model has this icon: 
  - Absorb fragments back into the model
  - Copy packages from partitions back into the main model

Creating a model from a package automatically leaves a **shortcut** reference to the new model where the package used to be.

Creating a fragment from a model element adds an adornment to the element and creates a separate file for that element.

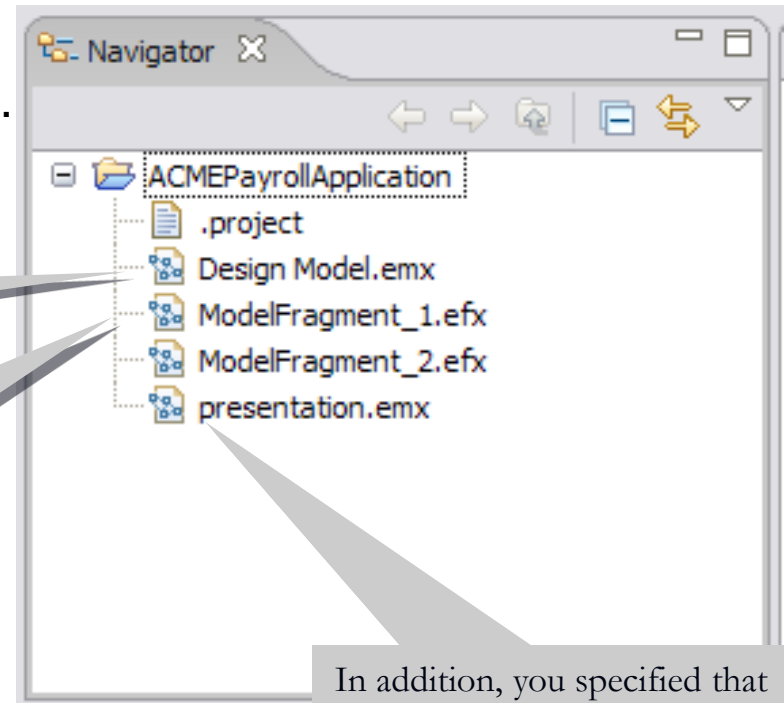


# Partition and Fragment Files

- After you have created a partition and a fragment, Rational Software Architect will create new files to represent these elements. You can view all of the files in the Navigator view.

**Design Model.emx** is the original model that you started with.

you have **two fragments** that have been created in relationship to this model. So you have two .efx files within your project.



In addition, you specified that you wanted to convert the presentation package into its own model. So there is another .emx file.

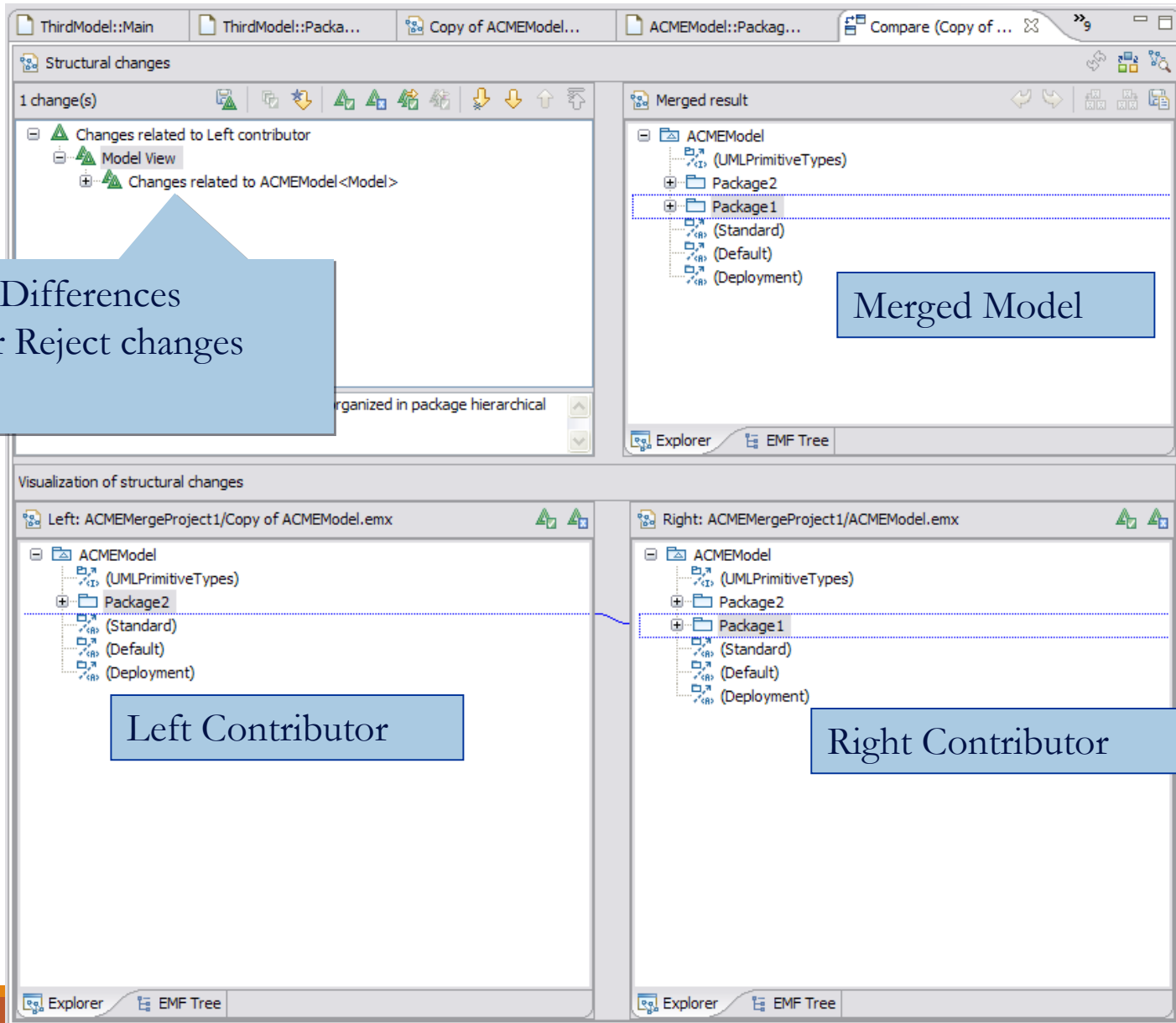
- Don't forget to add and commit your newly created files to SVN.**
  - Right-click the file, choose team/add to version control, and then commit.

# Compare and Merge Models

- In a typical team environment, team members need to constantly merge their development work to a common code base. True to designs as well.
  - Merging is almost always inevitable in a team development environment,
- Rational Software Architect allows you to merge model and diagram files using the compare and merge utility.
  - Compare models to identify changes between model versions.
  - Merge models when:
    - Parallel development occurs
    - Alternative approaches are explored
- Use wisely: avoid situations that require frequent merging.
  - Merge is expensive: think about how much self-inflicted pains you and your team may have when you have to solve merge confliction.
  - Fragmenting your model wisely to avoid unnecessary mergers.



# Compare Editor



Structural Differences

- Accept or Reject changes

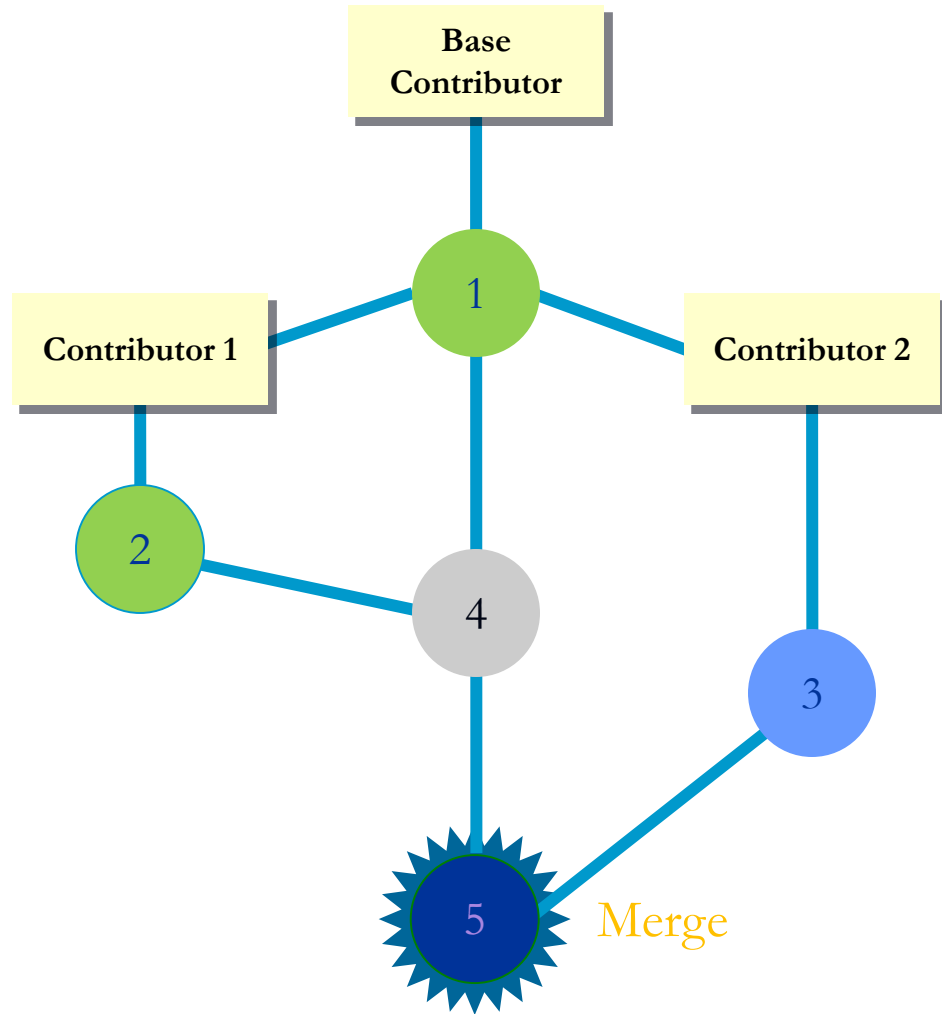
Merged Model

Left Contributor

Right Contributor

# Merging Models

- Begin with a base contributor, the **common ancestor** of the models you wish to merge.
- Have up to three contributors (modified versions of the base model) in a merge session.



\*simrisk.emx

Structural changes

0 unresolved co...sible change(s)

- Changes related to Left contributor
  - Model View
    - Changes related to SimRisk<Model>

Conflicts (0) Left Changes (1) Right Changes (0)

All changes related to model. The changes are organized in package hierarchical form.

Structural Differences

- Accept or Reject changes

Caution: always remember to save a copy, then override the existing file, and submit to SVN.  
 RSA will not commit changes automatically to SVN!

Name	Freeway
Owned Port	
Powertype Extent	
Template Parameter	
Use Case	
Visibility	public

Merged result

Properties Explorer

Visualization of structural changes

Property	Value
UML	
Classifier Beha	
Client Depend	
Is Abstract	false
Is Active	false
Is Leaf	
Name	Left Contributor
Owned Port	
Powertype Ex	
Redefined Cla	
Representatio	
Template Par	
Use Case	
Visibility	public

Left: Local File

Property	Value
Name	InterState
Owned Port	
Powertype Ex	
Redefined Cla	
Representatic	
Template Par	

Ancestor Left: Common Ancestor (31)

Property	Value
SimRisk	
Component	
InternalType	
(UMLPrimitiveTypes)	
Activity1	
Core	
Interface	

Ancestor Right: Common Ancestor (31)

Property	Value
SimRisk	
Component	
InternalType	
(UMLPrimitiveTypes)	
Activity1	
Core	
Interface	

Right: Remote File (31 [litan])

- SimRisk
  - Component
  - InternalType
  - (UMLPrimitiveTypes)
  - Activity1
  - Core
    - Interface
  - Topology
    - Agent
    - Route
    - Node
    - Retailer
    - Manufacturer
    - Supplier
    - Warehouse
    - Class1

Left Contributor

Ancestor

Right Contributor

Properties Explorer