



## UML Dictionary

*Enterprise Architect is an intuitive, flexible and powerful UML analysis and design tool for building robust and maintainable software.*

*This dictionary explains the way in which Enterprise Architect represents the UML 2.1 diagrams, elements and connectors, and its own extensions of UML.*



# Enterprise Architect - UML Dictionary

© 1998-2010 Sparx Systems Pty Ltd

All rights reserved. No parts of this work may be reproduced in any form or by any means - graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems - without the written permission of the publisher.

Products that are referred to in this document may be either trademarks and/or registered trademarks of the respective owners. The publisher and the author make no claim to these trademarks.

While every precaution has been taken in the preparation of this document, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

Printed: May 2010

## **Publisher**

*Sparx Systems*

## **Managing Editor**

*Geoffrey Sparks*

## **Technical Editors**

*Geoffrey Sparks*

*Neil Capey*

## **Special thanks to:**

*All the people who have contributed suggestions, examples, bug reports and assistance in the development of Enterprise Architect. The task of developing and maintaining this tool has been greatly enhanced by their contribution.*

# Table of Contents

Foreword	1
<b>UML Dictionary</b>	<b>2</b>
<b>UML Diagrams</b> .....	<b>4</b>
<b>Behavioral Diagrams</b> .....	<b>4</b>
Activity Diagram.....	5
Use Case Diagram.....	7
State Machine Diagrams.....	9
Regions .....	12
Pseudo-States .....	13
State Machine Table.....	14
State Machine Table Options.....	15
State Machine Table Operations.....	17
Change State Machine Table Position.....	18
Change State Machine Table Size.....	18
Insert New State.....	18
Insert Trigger.....	19
Insert/Change Transition.....	19
Reposition State or Trigger Cells.....	20
Add Legend.....	20
Find Cell in State Machine Diagram.....	20
State Machine Table Conventions.....	21
Export State Table To CSV File.....	21
Timing Diagram.....	22
Create a Timing Diagram.....	23
Set a Time Range.....	23
Edit a Timing Diagram.....	24
Add and Edit State Lifeline.....	24
Edit States In State Lifeline.....	25
Edit Transitions In State Lifeline.....	26
Add and Edit Value Lifeline.....	27
Add States In Value Lifeline.....	28
Edit Transitions In Value Lifeline.....	28
Configure Timeline - States.....	29
Configure Timeline - Transitions.....	31
Time Intervals.....	32
Time Interval Operations.....	36
Sequence Diagram.....	39
Denote Lifecycle of an Element.....	41
Layout of Sequence Diagrams.....	42
Sequence Elements.....	43
Sequence Diagrams and Version Control.....	44
Sequence Element Activation.....	45
Lifeline Activation Levels.....	46
Sequence Message Label Visibility.....	48
Change the Top Margin.....	48
Inline Sequence Elements.....	49
Communication Diagram.....	49
Communication Diagrams in Color.....	51
Interaction Overview Diagram.....	52
<b>Structural Diagrams</b> .....	<b>54</b>
Package Diagram.....	55
Class Diagram.....	56

Object Diagram.....	58
Composite Structure Diagram.....	59
Properties .....	61
Deployment Diagram.....	62
Component Diagram.....	65
<b>Extended Diagrams .....</b>	<b>67</b>
Analysis Diagram.....	67
Custom Diagram.....	69
Requirements Diagram.....	71
Maintenance Diagram.....	72
User Interface Diagram.....	73
Database Schema.....	75
Business Modeling/Interaction.....	75
<b>UML Elements .....</b>	<b>78</b>
<b>Behavioral Diagram Elements .....</b>	<b>78</b>
Action.....	79
Action Notation.....	81
Set Feature Dialog.....	85
Action Expansion Node.....	86
Action Pin .....	87
Assign Action Pins.....	88
Local Pre/Post Conditions.....	89
Activity.....	90
Activity Notation.....	91
Activity Parameter Nodes.....	91
Activity Partition.....	93
Actor.....	94
Central Buffer Node.....	95
Choice.....	95
Combined Fragment.....	96
Create a Combined Fragment.....	98
Interaction Operators.....	99
Datastore .....	102
Decision.....	102
Diagram Frame.....	104
Diagram Gate.....	105
Endpoint.....	106
Entry Point.....	107
Exception.....	107
Expansion Region.....	107
Add Expansion Region.....	110
Exit Point.....	110
Final.....	110
Flow Final.....	111
Fork/Join.....	112
Fork .....	114
Join .....	115
History.....	116
Initial.....	117
Interaction.....	118
Interaction Occurrence.....	119
Interruptible Activity Region.....	121
Add Interruptible Activity Region.....	122
Junction.....	122
Lifeline.....	123
Merge.....	124
Message Endpoint.....	124

Message Label.....	125
Note.....	126
Partition.....	126
Receive.....	127
Region.....	128
Send.....	129
State.....	129
Composite State.....	130
State/Continuation.....	132
Continuation.....	132
State Invariant.....	134
State Lifeline.....	135
State Machine.....	136
Structured Activity.....	136
Structured and Sequential Nodes.....	138
Loop and Conditional Nodes.....	139
Synch.....	142
System Boundary.....	142
Boundary Element Settings.....	144
Terminate.....	144
Trigger.....	145
Use Case.....	146
Use Case Extension Points.....	147
Rectangle Notation.....	148
Value Lifeline.....	149
<b>Structural Diagram Elements .....</b>	<b>150</b>
Artifact.....	151
Class.....	152
Active Classes.....	153
Parameterized Classes (Templates).....	154
Collaboration.....	156
Collaboration Occurrence.....	157
Component.....	158
Data Type.....	159
Deployment Spec.....	160
Device.....	161
Document Artifact.....	161
Enumeration.....	162
Execution Environment.....	162
Expose Interface.....	163
Information Item.....	163
Interface.....	164
Node.....	165
Object.....	165
Run-time State.....	166
Define a Run-time Variable.....	166
Remove a Defined Variable.....	167
Object State.....	167
Package.....	168
Part.....	168
Add Property Value.....	169
Port.....	169
Add a Port to an Element.....	170
Inherited and Redefined Ports.....	170
The Property Tab.....	172
Primitive.....	173
Qualifiers.....	173
Qualifiers Dialog.....	175

Signal.....	178
<b>Inbuilt and Extension Stereotypes .....</b>	<b>178</b>
Analysis Stereotypes.....	179
Boundary.....	179
Create a Boundary.....	180
Composite Elements.....	180
Control.....	181
Create a Control Element.....	182
Entity.....	182
Create an Entity.....	183
Event.....	183
Feature.....	184
Hyperlinks.....	184
N-Ary Association.....	187
Packaging Component.....	188
Process.....	189
Requirements.....	189
Screen.....	190
Test Case.....	191
Table.....	192
UI Control Element.....	192
Web Stereotypes.....	194
<b>UML Connectors .....</b>	<b>196</b>
<b>Aggregate .....</b>	<b>198</b>
Change Aggregation Connector Form.....	199
<b>Assembly .....</b>	<b>199</b>
<b>Associate .....</b>	<b>199</b>
<b>Association Class .....</b>	<b>200</b>
Connect New Class to Association.....	201
<b>Communication Path .....</b>	<b>202</b>
<b>Compose .....</b>	<b>202</b>
<b>Connector .....</b>	<b>203</b>
<b>Control Flow .....</b>	<b>204</b>
<b>Delegate .....</b>	<b>205</b>
<b>Dependency .....</b>	<b>205</b>
Apply a Stereotype.....	206
<b>Deployment .....</b>	<b>206</b>
<b>Extend .....</b>	<b>207</b>
<b>Generalize .....</b>	<b>207</b>
<b>Include .....</b>	<b>208</b>
<b>Information Flow .....</b>	<b>208</b>
Convey Information on a Flow.....	210
Realize an Information Flow.....	210
<b>Interrupt Flow .....</b>	<b>211</b>
<b>Manifest .....</b>	<b>211</b>
<b>Message .....</b>	<b>212</b>
Message (Sequence Diagram).....	212
Self-Message.....	215
Call.....	216
Message Examples.....	217
Change the Timing Details.....	218
General Ordering.....	220
Asynchronous Signal Message.....	221
Message (Communication Diagram).....	223
Create a Communication Message.....	224
Re-Order Messages.....	224
Message (Timing Diagram).....	226

---

Create a Timing Message.....	227
<b>Nesting .....</b>	<b>229</b>
<b>Notelink .....</b>	<b>230</b>
<b>Object Flow .....</b>	<b>230</b>
Object Flows in Activity Diagrams .....	231
<b>Occurrence .....</b>	<b>232</b>
<b>Package Import .....</b>	<b>232</b>
<b>Package Merge .....</b>	<b>232</b>
<b>Realize .....</b>	<b>233</b>
<b>Recursion .....</b>	<b>234</b>
<b>Role Binding .....</b>	<b>234</b>
<b>Represents .....</b>	<b>235</b>
<b>Representation .....</b>	<b>235</b>
<b>Trace .....</b>	<b>236</b>
<b>Transition .....</b>	<b>236</b>
<b>Use .....</b>	<b>238</b>
 <b>Index .....</b>	 <b>240</b>

# Foreword

This dictionary explains the way in which Enterprise Architect represents the UML 2.1 diagrams, elements and connectors, and its own extensions of UML.



## UML Dictionary



### The Unified Modeling Language (UML)

Enterprise Architect's modeling platform is based on the Unified Modeling Language (UML) 2.1, a standard that defines rules and notations for specifying business and software systems. The notation supplies a rich set of graphic elements for modeling object oriented systems, and the rules state how those elements can be connected and used. UML is not a tool for creating software systems; instead, it is a visual language for communicating, modeling, specifying and defining systems.

UML is not a prescriptive process for modeling software systems; it does not supply a method or process, simply the language. You can therefore use UML in a variety of ways to specify and develop your software engineering project. This language is designed to be flexible, extendable and comprehensive, yet generic enough to serve as a foundation for all system modeling requirements. With its specification, there is a wide range of elements characterized by the kinds of diagrams they serve, and the attributes they provide. All can be further specified by using stereotypes, Tagged Values and profiles. Enterprise Architect supports many different kinds of UML elements (as well as some custom extensions). Together with the connectors between elements, these form the basis of the model.

See:

- [UML Diagrams](#) 4
- [UML Elements](#) 78
- [UML Connectors](#) 198

### Wide Range of Applications

Although initially conceived as a language for software development, UML can be used to model a wide range of real world domains and processes (in business, science, industry, education and elsewhere), organizational hierarchies, deployment maps and much more. Enterprise Architect also provides additional custom diagrams and elements, to address further modeling interests. This topic is intended to provide an introduction to Enterprise Architect's diagrams, elements and connectors. It also illustrates its alignment, when applicable, to the Unified Modeling Language.

### Extending UML for New Domains

Using UML Profiles, UML Patterns (see *Extending UML with Enterprise Architect*), Grammars, Data Types, Constraints and other extensions, UML and Enterprise Architect can be tailored to address a particular modeling domain not explicitly covered in the original UML specification. Enterprise Architect makes extending UML simple and straightforward and, best of all, the extension mechanism is still part of the UML Specification.

### Find Out More

UML is an open modeling standard, defined and maintained by the Object Management Group. Further information, including the full UML 2.1.1 documentation, can be found on the OMG website at <http://www.omg.org>.

#### Tip:

If you are unfamiliar with UML, please explore the topics in this UML Dictionary and the Enterprise Architect UML Toolbox descriptions in *Using Enterprise Architect - UML Modeling Tool*, and the *EAExample* project supplied with Enterprise Architect. The online [UML Tutorial](#) (parts 1 and 2) and [UML 2.0 Tutorial](#) are also very helpful.

**Recommended Reading:**

In addition to the UML Specification available from the OMG, two books that provide excellent introductions to UML are:

- *Schaum's Outlines: UML* by Bennet, Skelton and Lunn. Published by McGraw Hill. ISBN 0-07-709673-8
- *Developing Software with UML* by Bern Oestereich. Published by Addison Wesley. ISBN 0-201-36826-5

## 1 UML Diagrams



### What is a UML Diagram?

A UML diagram is a representation of the components or elements of a system or process model and, depending on the type of diagram, how those elements are connected or how they interact from a particular perspective. For example, how and why an object changes state, or how requirements are realized by the process or a system.

### Types of Diagram

There are two major groupings of UML diagrams:

- [Structural Diagrams](#)<sup>[54]</sup> which depict the structural elements composing a system or function, reflecting the static relationships of a structure, or run-time architectures.
- [Behavioral Diagrams](#)<sup>[4]</sup> which show a dynamic view of the model, depicting the behavioral features of a system or business process.

Enterprise Architect provides the following additional diagram types that *extend* the core UML diagrams for business process modeling, formal requirements specifications and other domain-specific models:

- [Analysis](#)<sup>[67]</sup> diagrams
- [Custom](#)<sup>[69]</sup> diagrams
- [Requirements](#)<sup>[71]</sup> diagrams
- [Maintenance](#)<sup>[72]</sup> diagrams
- [User Interface](#)<sup>[73]</sup> diagrams
- [Database](#)<sup>[75]</sup> diagrams
- [Business Modeling and Business Interaction](#)<sup>[75]</sup> diagrams.

Enterprise Architect also supports diagram types specific to MDG Technologies, including integrated technologies such as Archimate, BPMN, Data Flow Diagrams, Eriksson-Penker Extensions, Iconix and Mind Mapping. For information on these MDG Technologies, see *Extending UML with Enterprise Architect*.

### Work with Diagrams

Diagrams are developed in the main workspace in which you create and connect model elements. You create them by right-clicking a package and selecting the **New Diagram** context menu option, or load them by double-clicking their diagram icon in the **Project Browser**.

For full details on how to work with diagrams, see *Diagram Tasks* in *UML Modeling with Enterprise Architect – UML Modeling Tool*.

### 1.1 Behavioral Diagrams

Behavioral diagrams depict the behavioral features of a system or business process. Behavioral diagrams include the following diagram types:

#### Activity Diagrams

[Activity diagrams](#)<sup>[5]</sup> model the behaviors of a system, and the way in which these behaviors are related in an overall flow of the system.

## Use Case Diagrams

[Use Case diagrams](#)<sup>[7]</sup> capture Use Cases and relationships among Actors and the system; they describe the functional requirements of the system, the manner in which external operators interact at the system boundary, and the response of the system.

## State Machine Diagrams

[State Machine diagrams](#)<sup>[9]</sup> illustrate how an element can move between states, classifying its behavior according to transition triggers and constraining guards.

## Timing Diagrams

[Timing diagrams](#)<sup>[22]</sup> define the behavior of different objects within a time-scale, providing a visual representation of objects changing state and interacting over time.

## Sequence Diagrams

[Sequence diagrams](#)<sup>[39]</sup> are structured representations of behavior as a series of sequential steps over time. They are used to depict work flow, message passing and how elements in general cooperate over time to achieve a result.

## Communication Diagrams

[Communication diagrams](#)<sup>[49]</sup> show the interactions between elements at run-time, visualizing inter-object relationships.

## Interaction Overview Diagrams

[Interaction Overview diagrams](#)<sup>[52]</sup> visualize the cooperation between other interaction diagrams (Timing, Sequence, Communication and Interaction Overview diagrams) to illustrate a control flow serving an encompassing purpose.

### See Also

- Behavioral Modeling (in the *Work With Elements* section of *UML Modeling With Enterprise Architect - UML Modeling Tool*)
- Code Generation from Behavioral Models (in *Code Engineering Using UML Models*)

### 1.1.1 Activity Diagram

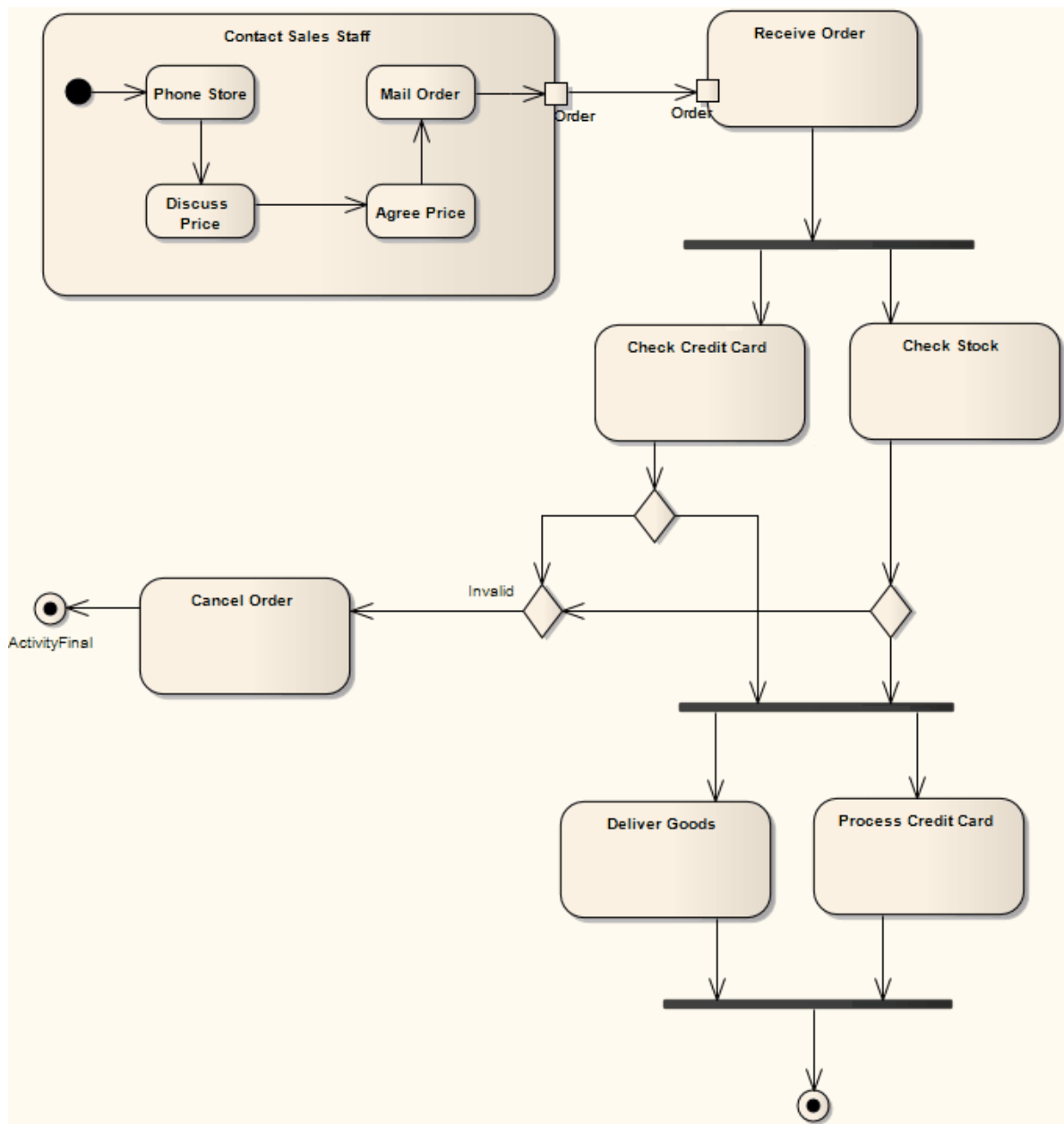
*Activity diagrams* are used to model the behaviors of a system, and the way in which these behaviors are related in an overall flow of the system. The logical paths a process follows, based on various conditions, concurrent processing, data access, interruptions and other logical path distinctions, are all used to construct a process, system or procedure.

#### Note:

You can create [Analysis diagrams](#)<sup>[67]</sup> (Simplified Activity), containing the elements most useful for business process modeling, using the **New Diagram** dialog (see *Diagram Tasks* in *UML Modeling with Enterprise Architect – UML Modeling Tool*).

## Example Diagram

The following diagram illustrates some of the features of Activity diagrams, including Activities, Actions, Start Nodes, End Nodes and Decision points.














### Toolbox Elements and Connectors

Select Activity diagram elements and connectors from the [Activity](#) pages of the Enterprise Architect UML [Toolbox](#); see *Using Enterprise Architect – UML Modeling Tool*.

**Tip:**

Click on the following elements and connectors for more information.

Activity Diagram Elements	Activity Diagram Connectors
Activity	Control Flow
Structured Activity	Object Flow

Activity Diagram Elements	Activity Diagram Connectors
 Action	 Interrupt Flow
 Partition	
 Object	
 Central Buffer Node	
 Datastore	
 Decision	
 Merge	
 Send	
 Receive	
 Synch	
 Initial	
 Final	
 Flow Final	
 Region	
 Exception	
 Fork/Join	
 Fork/Join	

### 1.1.2 Use Case Diagram

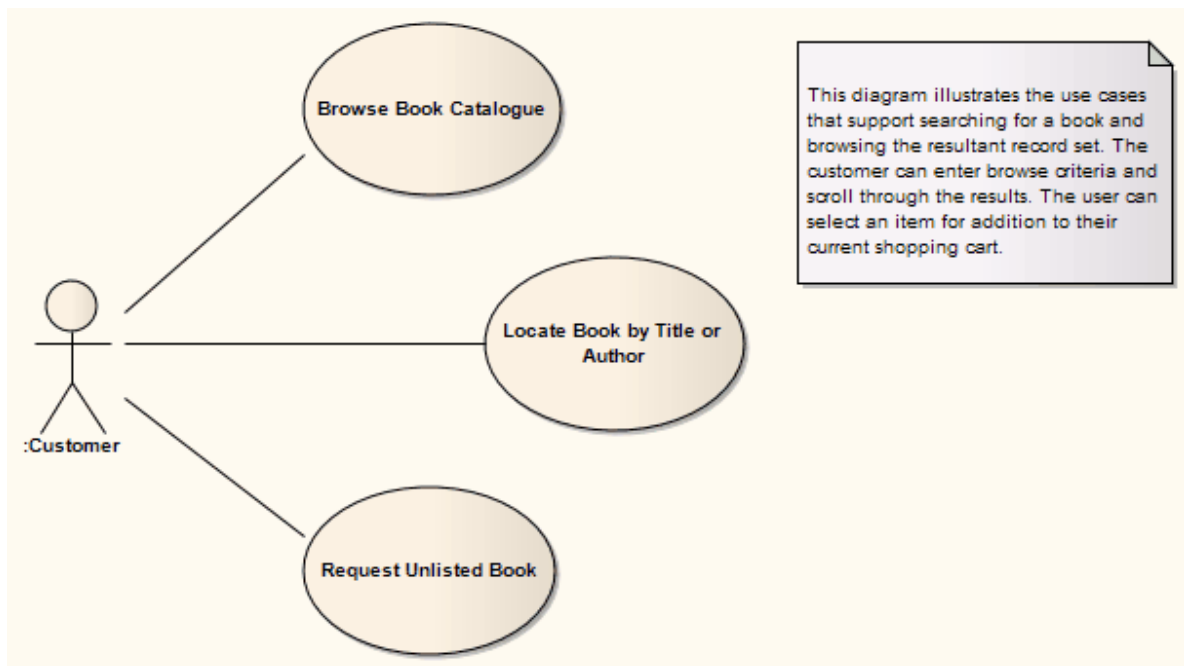
A Use Case diagram captures [Use Cases](#)<sup>[146]</sup> and relationships between [Actors](#)<sup>[94]</sup> and the subject (system). It describes the functional requirements of the system, the manner in which outside things (Actors) interact at the system boundary, and the response of the system.

In developing a Use Case diagram, also consider:

- [Use Case Extension Points](#)<sup>[147]</sup>
- [Use Rectangle Notation](#)<sup>[148]</sup>
- [Business Use Case](#)<sup>[75]</sup> (stereotyped Use Case)

#### Example Diagram

The following diagram illustrates some features of Use Case diagrams:



### Toolbox Elements and Connectors

Select Use Case diagram elements and connectors from the [Use Case](#) pages of the Enterprise Architect UML [Toolbox](#); see *Using Enterprise Architect – UML Modeling Tool*.

**Tip:**

Click on the following elements and connectors for more information.

Use Case Diagram Elements	Use Case Diagram Connectors
Actor	Use
Use Case	Associate
Test Case	Generalize
Collaboration	Include
Boundary	Extend
Package	Realize
	Invokes
	Precedes

**Note:**

*Invokes* and *Precedes* relationships are defined by the Open Modeling Language (OML). They are stereotyped *Dependency* relationships; *Invokes* indicates that Use Case A, at some point, causes Use Case B to happen, whilst *Precedes* indicates that Use Case C must complete before Use Case D can begin.

### 1.1.3 State Machine Diagrams

**Note:**

State Machine diagrams were formerly known as State diagrams.

A *State Machine* diagram illustrates how an element (often a Class) can move between states, classifying its behavior according to transition triggers and constraining guards. Other aspects of State Machine diagrams further depict and explain movement and behavior; see the *Working With Elements* section of *UML Modeling With Enterprise Architect – UML Modeling Tool*.

For information on code generation from State Machine diagrams, see the *SW Code Generation - State Machine Diagrams* and *State Machine Modeling for HDLs* topics in *Code Engineering Using UML Models*.

State Machine representations in UML are based on the *Harel State Chart Notation* (see the *OMG UML Superstructure Specification 2.1.1, section 15.1*), and therefore are sometimes referred to as *State Charts*.

You can display a State Machine as a diagram (as below) or as a [table](#)<sup>[14]</sup> in one of three relationship formats. In all formats, you use the same Enterprise Architect UML [Toolbox](#) elements and connectors; see *Using Enterprise Architect – UML Modeling Tool*.

To select the display format, follow the steps below:

1. Right-click on the diagram background to display the context menu.
2. Select the **Statechart Editor** option.
3. Select the appropriate display option:
  - **Diagram**
  - **Table (State-Next State)**
  - **Table (State-Trigger)**
  - **Table (Trigger-State)**.

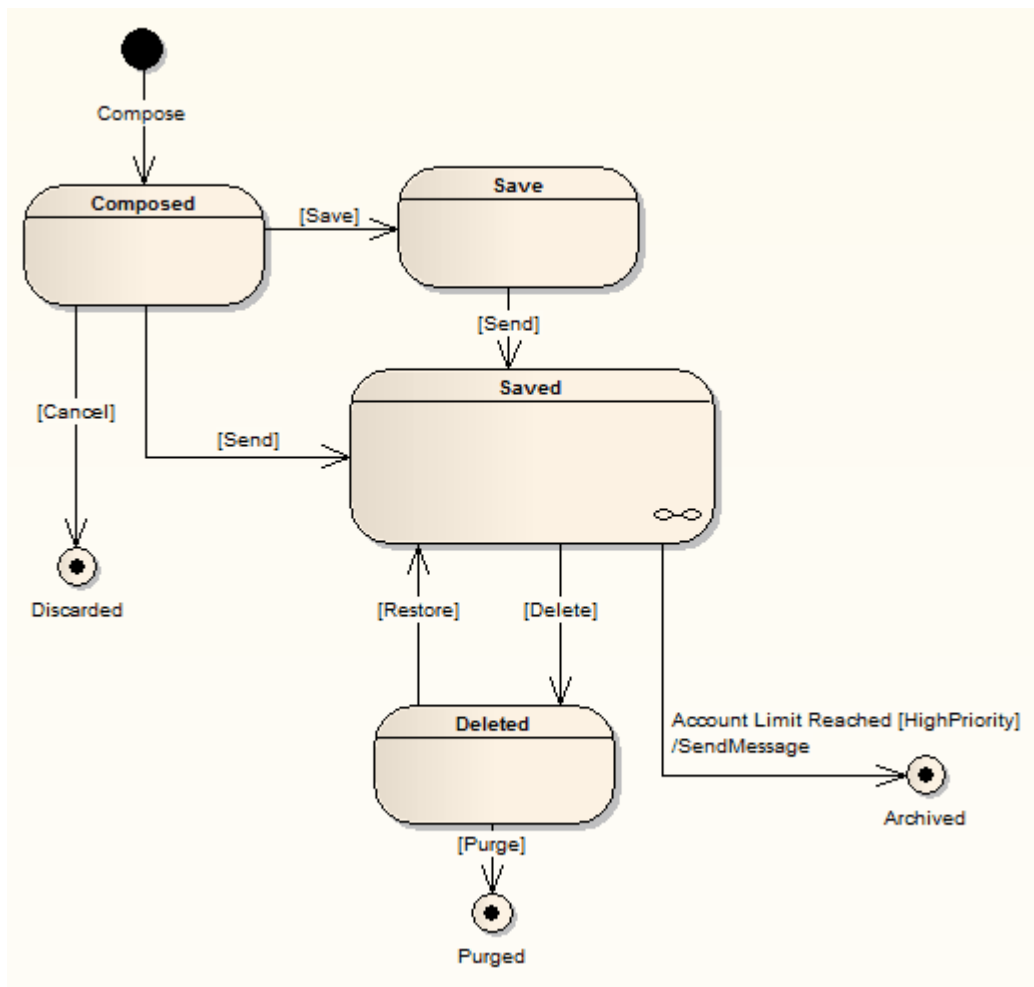
#### Example Diagram

The following diagram illustrates some features of State Machine diagrams. The *Saved State* is a [Composite](#)<sup>[130]</sup> State, and enclosed States are [sub-states](#)<sup>[130]</sup>. Initial and final [pseudo-states](#)<sup>[13]</sup> indicate the entry to and exit from the State Machine. Composite States and sub-states are both [State](#)<sup>[129]</sup> elements, a Composite State being an expanded State element that encloses other State elements, which are then referred to as sub-states. Composite States and State Machines can also contain [Regions](#)<sup>[12]</sup>.

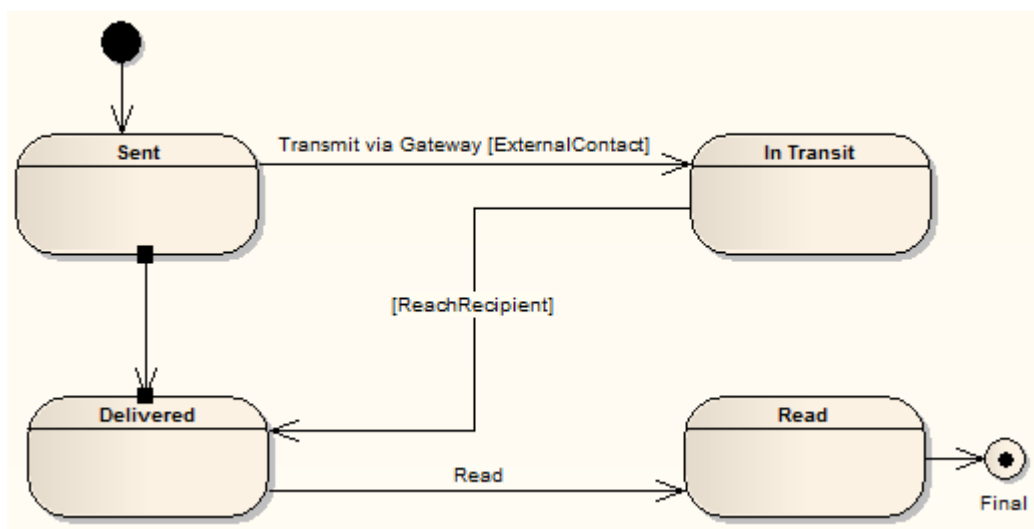
**Note:**

State elements can display either with or without a line across them. The line - as shown below - displays when the element has features such as attributes (which could be hidden) or when the **Show State Compartment** checkbox is selected in the [Objects](#) page of the [Options](#) dialog (see *Using Enterprise Architect - UML Modeling Tool*).

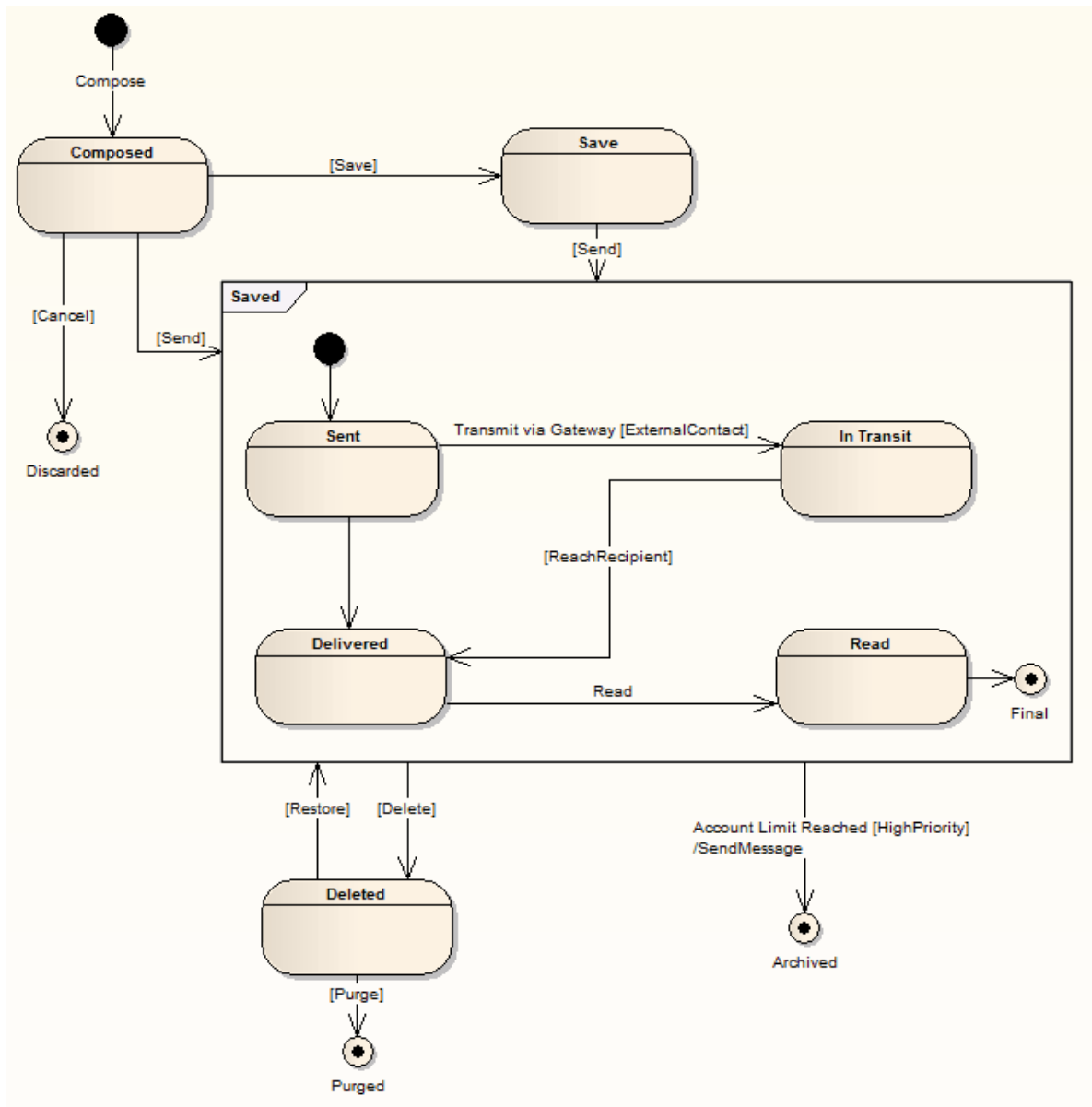




You have two options for exposing the contents of a composite State, such as *Saved*. Firstly, you can double-click on the element to display its child diagram separately, as shown below:



Alternatively, you can right-click on the composite element and select the **Advanced | Show Composite Diagram** context menu option, which displays the child diagram in the context of the parent diagram.



















### Toolbox Elements and Connectors

Select State Machine diagram elements and connectors from the **State** pages of the Enterprise Architect UML **Toolbox**; see *Using Enterprise Architect – UML Modeling Tool*.

**Tip:**

Click on the following elements and connectors for more information.

State Machine Diagram Elements	State Machine Diagram Connectors
 State	 Transition
 State Machine	 Object Flow

State Machine Diagram Elements	State Machine Diagram Connectors
 Initial	
 Final	
 History	
 Synch	
 Object	
 Choice	
 Junction	
 Entry	
 Exit	
 Terminate	
 Fork/Join	
 Fork/Join	

## OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, Section 15.3.12, p. 560*) states:

*A state machine owns one or more regions, which in turn own vertices and transitions.*

*The behaved classifier context owning a state machine defines which signal and call triggers are defined for the state machine, and which attributes and operations are available in activities of the state machine. Signal triggers and call triggers for the state machine are defined according to the receptions and operations of this classifier.*

*As a kind of behavior, a state machine may have an associated behavioral feature (specification) and be the method of this behavioral feature. In this case the state machine specifies the behavior of this behavioral feature. The parameters of the state machine in this case match the parameters of the behavioral feature and provide the means for accessing (within the state machine) the behavioral feature parameters.*

*A state machine without a context classifier may use triggers that are independent of receptions or operations of a classifier, i.e. either just signal triggers or call triggers based upon operation template parameters of the (parameterized) state machine.*

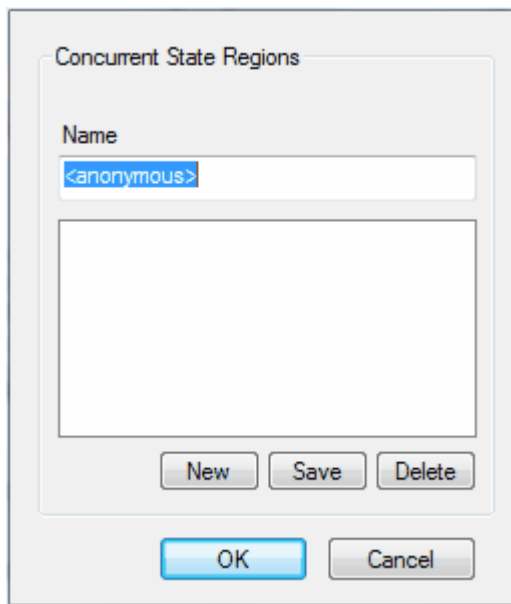
### 1.1.3.1 Regions

Regions can be created in [Composite States](#)<sup>[130]</sup> or [State Machines](#)<sup>[129]</sup> on a [State Machine diagram](#)<sup>[94]</sup>.

Regions indicate concurrency, such that a single State is active in each region. Multiple transitions can occur from a single event dispatch, so long as similarly triggered transitions are divided by Regions.

To create a Region in a Composite State or State Machine element, follow the steps below:

1. Right-click on the element, and select the **Advanced | Define Concurrent Substates** context menu option. The **State Regions** dialog displays.



2. Create the Regions of a State, which can be named or anonymous.
3. Click on the **OK** button.

### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 544*) states:

*A region is an orthogonal part of either a composite state or a state machine. It contains states and transitions.*

#### 1.1.3.2 Pseudo-States

Pseudo-states are a UML 2.1.1 abstraction for various types of transient vertices used in [State Machine](#)<sup>[9]</sup> diagrams. Pseudo-states are used to express complex transition paths. The following types of pseudo-state are available:

- [Initial](#)<sup>[117]</sup>
- [Entry Point](#)<sup>[107]</sup>
- [Exit Point](#)<sup>[110]</sup>
- [Choice](#)<sup>[95]</sup>
- [Junction](#)<sup>[122]</sup>
- [History](#)<sup>[116]</sup>
- [Terminate](#)<sup>[144]</sup>
- [Final](#)<sup>[110]</sup>
- [Fork](#)<sup>[114]</sup>
- [Join](#)<sup>[115]</sup>

### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 536*) states:

*A pseudostate is an abstraction that encompasses different types of transient vertices in the state machine graph... Pseudostates are typically used to connect multiple transitions into more complex state transitions paths. For example, by combining a transition entering a fork pseudostate with a set of transitions exiting the fork pseudostate, we get a compound transition that leads to a set of orthogonal target states.*

### 1.1.4 State Machine Table

A *State Machine table* is one of two variants of a State Machine (the other is the [State Machine diagram](#)). It displays the information of the State Machine in table form, and is a method of specifying the discrete behavior of a finite state-transition system; that is, what state the State Machine moves to and the conditions under which the transition takes place.

You can display the state transition as one of two different relationships:

- **State - Trigger:** The rows indicate the current states and the columns indicate trigger events (or the other way around if you prefer, in a **Trigger - State** format). The cell at the intersection of a row and column identifies the target state in the transition if the trigger occurs, and the condition (or guard) of the transition.

		Trigger		Event1	Event2	Event3	Event4	<None>
		State		E0	E1	E2	E3	E4
Initial		S0						S1
State1		S1					S2	
State2		S2	S6	<u>[Guard]</u> S4				
	SubState1	S3		S4				
	SubState2	S4			<u>[Cond]</u> S2			
	SubState3	S5						
State3		S6						S7
Final		S7						

- **State - Next State:** The rows and columns both indicate states, and the cell at the intersection of a row and column indicates the event that triggers a transition from the current (row) state to the next (column) state, the condition (or guard) of the event, and the effect of the transition.

Next State		Initial	State1	State2			State3	Final	
					SubState1	SubState2			SubState3
State		S0	S1	S2	S3	S4	S5	S6	S7
Initial		S0	_____						
State1		S1		Event4 _____					
State2		S2				Event2 [Guard] _____		Event1 _____	
	SubState1	S3				Event2 _____			
	SubState2	S4		Event3 [Cond] _____					
	SubState3	S5							
State3		S6							_____
Final		S7							

## Select Format

You can display a State Machine as a diagram or table, and as a table in one of three relationship formats.

To select the display format, follow the steps below:

1. Right-click on the diagram background to display the context menu.
2. Select the **Statechart Editor** option.
3. Select the appropriate display option:
  - **Diagram**
  - **Table (State-Next State)**
  - **Table (State-Trigger)**
  - **Table (Trigger-State)**

To define the State Machine Table further, see:

- [State Machine Table Options](#)<sup>[15]</sup>
- [State Machine Table Operations](#)<sup>[17]</sup>

### 1.1.4.1 State Machine Table Options

You can choose the [State Machine table](#)<sup>[14]</sup> layout and set other options from the **State Machine Diagram: Options** dialog, which you display by either:

- Double-clicking on the State Machine table background or
- Right-clicking on the background and selecting the **State Table Options** context menu option.

Table Format: State - Next State

**Cell Size**

Transition Cell Width:

Transition Cell Height:

Left Edge Cell Width:

Top Edge Cell Height:

**Display Options**

Always Display an Empty State Zone

Enable State Enumeration  
Prefix:

Enable Event Enumeration  
Prefix:

**Cell Color**

State/Trigger Cell:

State/Trigger Enumeration:

Transition Cell:

**Sample State Table**

		Trigger		
		Trigger0	Trigger1	Trigger2
State		E0	E1	E2
State0	P0			
State1	P1		P2	
State2	P2			

Option	Use to
<b>Table Format</b>	Select the required table format: <ul style="list-style-type: none"> <li><b>State - Trigger:</b> rows represent States, each state name in a left edge cell; columns represent Triggers, each trigger name in a column header cell; the intersection of a row and column identifies the Transition (if there is one); the Transition cell displays information about the next State and the condition (guard) of the Transition</li> <li><b>Trigger - State:</b> as above, except that rows represent triggers and columns represent states</li> <li><b>State - Next State:</b> both rows and columns represent states; intersection of row and column defines the transition (if there is one) from the row state to the column state.</li> </ul>
<b>Cell Size</b>	
<b>Transition Cell Width</b>	Specify the width of the transition cells (that is, the column width).
<b>Transition Cell Height</b>	Specify the height of the transition cells (that is, the row height).
<b>Left Edge Cell Width</b>	Specify the width of the left edge (row title) cells.
<b>Top Edge Cell Height</b>	Specify the height of the top edge (column title) cells.
<b>Cell Color</b>	

Option	Use to
State/Trigger Cell	Select the color of the row and column title cells.
State/Trigger Enumeration	Select the color of the enumeration (row/column numbering) cells. <b>Note:</b> You must select at least one of the <b>Enable State Enumeration</b> and <b>Enable Event Enumeration</b> checkboxes to set this color.
Transition Cell	Select the color of the transition cells (in the main body of the table).
<b>Highlight Options</b>	
Highlight Zones Related to Selected Transition	Highlight the cells for all elements involved in a selected transition - the initial state, the target state, and the trigger.
Highlight Color	Select the color of the highlight.
Use Different Color for Target State	Highlight the cell for the target element in a transition in a different color to the cell for the source element.
Target Zone Color	Select the color of the highlight.
<b>Display Options</b>	
Always Display an Empty State Zone	Add an empty row (and, on a <i>State - Next State</i> table, an empty column) to the end of the table.  The title cell contains an ellipsis (...). You can click twice (not double-click) on the ellipsis to edit it and identify a new state. In this case, another empty state zone is automatically added.
Enable State Enumeration	Add a cell to each state title cell, to number the state. Numbering starts at 0.
Prefix	If required, type a prefix for the state number or delete the default <b>S</b> to have no prefix.
Enable Event Enumeration	Add a cell to each event or trigger title cell, to number the event. Numbering starts at 0.
Prefix	If required, type a prefix for the event number or delete the default <b>E</b> to have no prefix.
Sample State Table	Display a preview of the table format as you define it.
Advanced	Define diagram options. The <b>State Machine Diagram Properties</b> dialog displays. (See <i>UML Modeling with Enterprise Architect – UML Modeling Tool</i> for information on the <b>Diagram Properties</b> dialog.)
Restore Defaults	Reapply the State Table diagram default values.
Apply	Apply changed options to the State Table diagram.

### 1.1.4.2 State Machine Table Operations

#### Overview

As a [State Machine table](#)<sup>[14]</sup> is a variant of a UML [State Machine diagram](#)<sup>[9]</sup>, most of the operations for manipulating the data are the same as for State Machine diagrams. These include operations to:

- Create new items by drag-and-dropping a specified object from the Enterprise Architect UML **Toolbox** to the current diagram



- Delete an item
- Apply to the diagram elements in the **Project Browser**
- Display or change the properties of the State, Trigger or Transition
- Apply to the diagram, such as **Lock Diagram**, **Zoom**, and *in place editing* of the element.

The operations specific to State Machine tables are described in the following topics:

- [Change Position of State Machine Table](#) <sup>[18]</sup>
- [Change Size of State Machine Table](#) <sup>[18]</sup>
- [Insert New State \(and Substate\)](#) <sup>[18]</sup>
- [Insert Trigger](#) <sup>[19]</sup>
- [Insert/Change Transition](#) <sup>[19]</sup>
- [Reposition State/Trigger Cells](#) <sup>[20]</sup>
- [Add Legend](#) <sup>[20]</sup>
- [Find Cell in State Machine Diagram](#) <sup>[20]</sup>
- [State Machine Table Conventions](#) <sup>[21]</sup>
- [Export State Table To CSV File](#) <sup>[21]</sup>

#### 1.1.4.2.1 Change State Machine Table Position

If necessary, you can move the State Machine table around in the **Diagram View**. To change the position of the State Machine table, follow the steps below:

1. Press **[Ctrl]+[A]** or double click on the top left cell to select the whole State Machine table.
2. Drag and drop the State Machine table to the required position. Alternatively, use **[Shift]+[→]**, **[←]**, **[↑]** or **[↓]** to move the State Machine table.

#### 1.1.4.2.2 Change State Machine Table Size

There are three ways to change the size of the State Machine table:

- Change the cell size on the [State Machine Diagram: Options](#) <sup>[15]</sup> dialog.
- Press **[Ctrl]+[A]** or double click on the top left cell to select the whole State Machine table, then press **[Ctrl]+[→]**, **[←]**, **[↑]** or **[↓]** to change the size.
- Select the State Machine table, then drag the shape handles to change the size.

#### 1.1.4.2.3 Insert New State

You can insert a new State in the State Machine table, using one of following methods:

- In the top left cell in the State Machine table, move the cursor to the word **State** to display a **+** at the end of the word; click on the **+** to create a new State
- Right-click in the top left cell in the State Machine table to display the context menu, and select the **Add State** menu option
- Right-click on an existing State cell in the State Machine table to display the context menu, and select the
  - **Insert New State Before** option to insert a new State before the current State, or
  - **Insert New State After** option to insert a new State after the current State
- Click on an existing State cell in the State Machine table, and press **[Insert]** to create and insert a new State above the selected State
- In the Enterprise Architect UML **Toolbox**, on the **State Elements** page, click on an element and then click on:
  - the diagram background to add a new State to the end of the table, or
  - an existing State cell to add the new State just above it.

#### Note:

From the **State Elements** page of the Enterprise Architect UML **Toolbox** you can insert *State*, *Initial*, *Final*, *Entry*, *Exit* and *Terminate* elements.

## Add a Substate

To add a Substate to a selected State, follow the steps below:

1. Right-click on the required State cell in the State Machine table. The context menu displays.
2. Select the **Add Substate** menu option. Enterprise Architect adds the Substate to the State.

### Note:

If the selected State does not allow a Substate, then the **Add Substate** menu option is grayed out.

You can also drag one existing State over another. If the second State allows Substates, the dragged State then becomes its Substate.

Similarly, you can change the parent State of a Substate by dragging the Substate from the original parent State to a different State.

## Remove Parent Relation of a Substate

To remove the parent relation of a Substate and make it a separate State, follow the steps below:

1. Right-click on the Substate in the State Machine table. The context menu displays.
2. Select the **Remove Parent Relation** menu option. The Substate cell becomes a State cell.

You can also drag and drop the Substate onto the top left cell of the State Machine table. The dragged Substate again becomes a State cell.

### 1.1.4.2.4 Insert Trigger

If the State Machine table format is either *State-Trigger* or *Trigger-State*, you can use one of the following methods to insert a new Trigger:


- In the top left cell in the State Machine table, move the cursor to the word **Event** to display a **+** at the end of the word; click on the **+** to create a new Trigger
- In the top left cell in the State Machine table, right-click to display the context menu and select the **Add Trigger** menu option to create a new Trigger
- Select an existing Trigger in the State Machine table, then press **[Insert]** to insert a new Trigger before the existing Trigger
- Click on an existing Trigger in the State Machine table, right-click to display the context menu and select either the:
  - **Insert New Trigger Before** option to insert a new Trigger before the current Trigger, or
  - **Insert New Trigger After** option to insert a new Trigger after the current Trigger.

### 1.1.4.2.5 Insert/Change Transition

You can insert a new Transition using one of the following methods:

- Right-click on the cell in which to create a Transition, to display the context menu
  - If the State Machine table format is *State-Trigger* or *Trigger-State*, the context menu lists the States you can choose as the target of the Transition; click on the required State name to create the Transition
  - If the State Machine table format is *State-Next State*, click on the **Insert Transition** menu option to create the Transition.
- In the **State Relationships** page of the Enterprise Architect UML **Toolbox**, select the *Transition* element, then click on the cell in the State Machine table in which to create the Transition. Double-click on the Transition to define it in the **Transition Properties** dialog.

## Change the Transition

As for the [State Chart](#)  diagram, to change the properties of a Transition double-click the Transition cell and edit the details on the **Transition Properties** dialog.

## Change Transition States

You can change the source and target of the Transition by right-clicking the Transition and selecting the **Advanced | Set Source and Target** context menu option.

Alternatively, you can change the Transition source, target or Trigger by clicking on the Transition and dragging it to a different cell.

If the State Machine table format is either **State-Trigger** or **Trigger-State**, you can change the target state of a transition by:

1. Highlighting the target state name in the Transition cell and clicking on it to display a list of the states in the table.
2. Clicking on the preferred target state name.

## Highlight States and Trigger Related to Transition

You can select options to highlight the source State, target State and Trigger cells associated with a Transition, using the **Highlight Options** panel on the [State Machine Diagram Options](#) <sup>[15]</sup> dialog. When you click on the Transition cell its associated State and Trigger cells are highlighted.

Alternatively, click on the Transition cell and press and hold **[L]**.

### 1.1.4.2.6 Reposition State or Trigger Cells

You can change the position of a selected State or Trigger cell in one of the following ways:

- Right-click on the State or Trigger title cell and select the appropriate **Order | Move xxx** context menu option
- Click on the cell and press **[Shift]+[→]**, **[←]**, **[↑]** or **[↓]**.

### 1.1.4.2.7 Add Legend

You can add a simple legend to any State Machine Table cell that has no transition. The two legend symbols are:

- **I** - Ignore
- **N** - Never Happen

To assign a legend symbol to a State Machine Table cell, follow the steps below:

1. Right-click on the cell to which to assign the legend. The context menu displays.
2. Select the appropriate menu option:
  - **Legend | Ignore**
  - **Legend | Never Happen.**

The required symbol displays in the center of the cell.

To remove a legend symbol from a cell, right-click on the cell and select the **Legend | Remove Legend** context menu option.

### 1.1.4.2.8 Find Cell in State Machine Diagram

On the State Machine table you can select a State or Trigger element and locate it in a State Machine diagram, by selecting the **Find | Locate in State Chart** context menu option. Enterprise Architect switches to the State Machine diagram and highlights the selected element. You can locate a Transition relationship in a similar way, by selecting the **Locate in State Chart** context menu option.

#### Note:

A Trigger on a State Machine table might or might not exist on the corresponding State Machine diagram. If the Trigger does not exist on the State Machine diagram, the **Locate in State Chart** option is disabled.

Conversely, on the State Machine diagram, you can select a State or Trigger element and locate it on the corresponding State Machine table, by selecting the **Find | Locate in State Table** context menu option.

Enterprise Architect switches to the State Machine table and highlights the selected element. You can locate a Transition relationship in a similar way, by selecting the **Locate in State Table** context menu option.

#### 1.1.4.2.9 State Machine Table Conventions

##### Trigger

- Deleting a Trigger removes it completely from the model, therefore you cannot UNDO a Trigger deletion
- There is a <None> column at the end of the **Event** heading row. This is for Transitions that have no Trigger information.

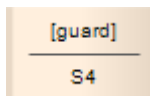
##### State

From the Enterprise Architect UML **Toolbox** you can insert the following *State* element types only (although the State Machine table might pick up and display other types, such as *Submachine State*):

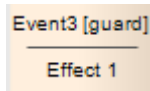
- State
- Initial
- Final
- Entry
- Exit
- Terminate.

##### Transition

The Transition cell displays its properties in one of two ways, depending on the State Machine table format. If the State Machine table format is *State - Trigger* or *Trigger - State*, the Transition cell displays the *Guard* and *Target* as shown below:



If the State Machine table format is *State - Next State*, then the Transition cell displays the *Trigger*, *Guard* and *Effect* as shown below:









The State Machine table enables you to edit the *Guard* and *Effect* in place. If the *Guard* or *Effect* is empty for your selected Transition cell, the cell displays an ellipsis [ ... ] instead. Click twice (not double-click) on the ellipsis to type in the *Guard* and *Effect* names.

#### 1.1.4.2.10 Export State Table To CSV File

To export a State Machine Table to a CSV file, follow the steps below:

1. Open the required State Machine Table.
2. Right-click on the diagram background and select the **Export Statechart to CSV file** context menu option.
3. The **Save As** browser dialog displays. Select the appropriate directory location and type in the .CSV filename.
4. Click on the **Save** button.



Timing Diagram Elements	Timing Diagram Message
 State Lifeline	 Message
 Value Lifeline	
 Message Label	
 Message Endpoint	
 Diagram Gate	

### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification*, v2.1.1, p. 517) states:

*Timing Diagrams are used to show interactions when a primary purpose of the diagram is to reason about time. Timing diagrams focus on conditions changing within and among Lifelines along a linear time axis.*

*Timing diagrams describe behavior of both individual classifiers and interactions of classifiers, focusing attention on time of occurrence of events causing changes in the modeled conditions of the Lifelines.*

The OMG UML specification (*UML Superstructure Specification*, v2.1.1, p. 519) also states:

*The primary purpose of the timing diagram is to show the change in state or condition of a lifeline (representing a Classifier Instance or Classifier Role) over linear time. The most common usage is to show the change in state of an object over time in response to accepted events or stimuli. The received events are annotated as shown when it is desirable to show the event causing the change in condition or state.*

#### 1.1.5.1 Create a Timing Diagram

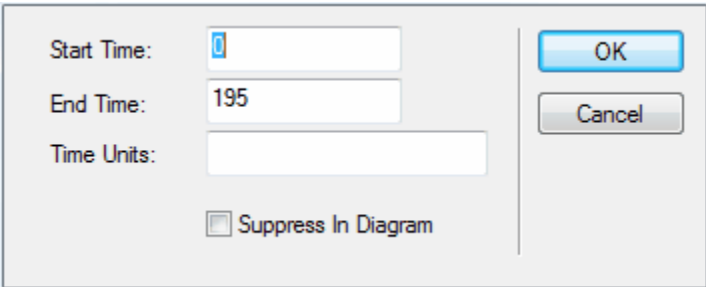
To create a Timing diagram, follow the steps below:

1. Right-click on a package in the **Project Browser**. The context menu displays.
2. Select the **Add | Add Diagram** menu option. The **New Diagram** dialog displays.
3. In the **Select From** panel, select **UML Behavioral**.
4. In the **Diagram Types** panel, select **Timing**.
5. Click on the **OK** button. The **Diagram** view displays, on which you create the Timing elements for the diagram. See [Set a Time Range](#)<sup>[23]</sup> and [Edit a Timing Diagram](#)<sup>[24]</sup>.

#### 1.1.5.2 Set a Time Range

Before adding Lifeline elements to your Timing diagram, set a time range. To do this, follow the steps below:

1. Right-click on the diagram. The context menu displays.
2. Select the **Set Timeline Range** option. The **Set Timeline Range** dialog displays.



The dialog box has the following fields and controls:

- Start Time:
- End Time:
- Time Units:
- Suppress In Diagram
- OK button
- Cancel button

3. In the **Start Time** and **End Time** fields, type the numeric values for the start and end points of the

timeline; for example, set the range **0** to **100**.

**Note:**

The start time must be less than the end time.

4. In the **Time Units** field, type the unit in which the time is measured; for example, **seconds** or **minutes**.
5. If it is not necessary to show the time range on the diagram, select the **Suppress In Diagram** checkbox.
6. Click on the **OK** button. If you have not suppressed it, the time range displays underneath the Lifeline elements that you create on the diagram.

### 1.1.5.3 Edit a Timing Diagram

On a Timing Diagram, you can add *State Lifeline* elements and *Value Lifeline* elements. You can maintain the *states* and *transitions* on these Lifeline elements either on the diagram itself or via the **Configure Timeline** dialog. See the following topics:

- [Add and Edit a State Lifeline Element](#) <sup>[24]</sup>
- [Edit States in a State Lifeline Element](#) <sup>[25]</sup>
- [Edit Transitions in a State Lifeline Element](#) <sup>[26]</sup>
- [Add and Edit a Value Lifeline Element](#) <sup>[27]</sup>
- [Add States in a Value Lifeline Element](#) <sup>[28]</sup>
- [Edit Transitions in a Value Lifeline Element](#) <sup>[28]</sup>
- [Configure Timeline dialog - States Tab](#) <sup>[29]</sup>
- [Configure Timeline dialog - Transitions Tab](#) <sup>[31]</sup>


#### 1.1.5.3.1 Add and Edit State Lifeline

From the **Timing** elements page of the Enterprise Architect UML **Toolbox** drag a [State Lifeline](#) <sup>[135]</sup> element onto your diagram. The element displays on the diagram.

To define the name of the State Lifeline, follow the steps below:

1. Right-click on the element. The context menu displays.
2. Select the **Other Properties** option. The **Timeline <name>** dialog displays, showing the **General** tab.
3. Overtyping the **Name** field.
4. Click on the **Apply** button and the **OK** button.

### Sizing and Scale

In the top left corner of a selected Lifeline element are the left and right *quick sizing buttons* (). These buttons increase or decrease the width of the Lifeline element, which in turn controls the scale width of each time unit. By increasing the width of the element you increase the resolution when adding transitions, which makes them easier to edit.

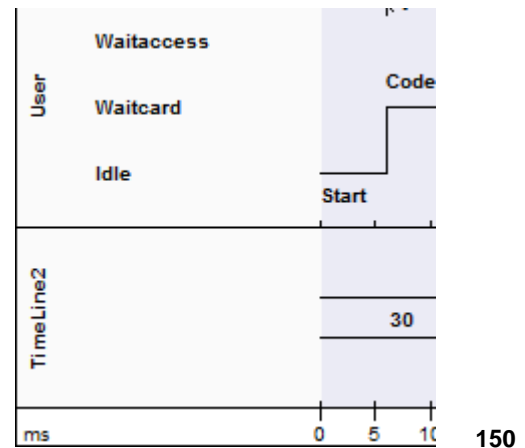
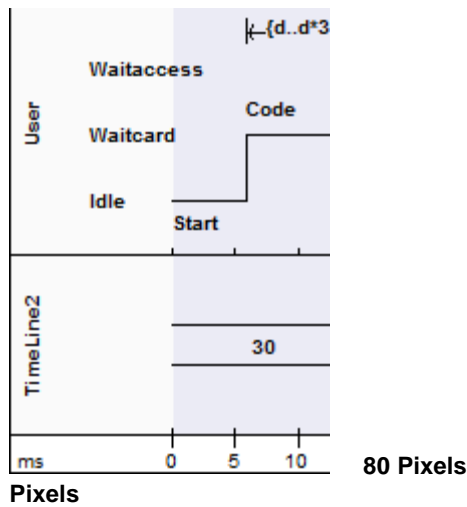
**Note:**

In order to edit the State Lifeline element, you must click on it to select it.

### Set Timeline Start Position

You might require more space at the start of your timelines; for example, to use long state names. To insert this space in all the timelines on a diagram, follow the steps below:


1. Right-click on the diagram background to display the context menu.
2. Select the **Set Timeline Start Position** menu option. The **Set Timeline Start Position** dialog displays.
3. The **Value 80 to 300** field defaults to **80** as the minimum distance in pixels between the start of the timeline element and the start of the timeline itself. Type a new value up to 300 pixels and click on the **OK** button to increase the space at the start of the timeline, as shown in the following diagrams.



You now edit the [states](#) <sup>[25]</sup> and [transitions](#) <sup>[26]</sup> in the State Lifeline.

### 1.1.5.3.2 Edit States In State Lifeline


#### Add States

1. Click on the *State Lifeline* element. The **New State** button (  ) and **Edit States** button (  ) display at the bottom left of the element.
2. Click on the **New State** button. The **New State** dialog displays.

3. In the **State** field, type the name of the state.
4. Click on the **OK** button.

#### Note:

You must add at least two states; for example, **On** and **Off**.



5. As you add states, increase the height of the element by dragging a handle-box (  ) on the edge of the element.

#### Note:

You can also add states using the **States** tab of the **Configure Timeline** dialog. Add either:

- Discrete states to the Timeline as described in [Add a New State](#) <sup>[30]</sup>, or
- A continuous range of numeric states as described in [Numeric Range Generator](#) <sup>[30]</sup>.

#### Edit States

1. Click on the State Lifeline element and click on the required state. The **Edit State** dialog displays.
2. In the **State** field, change the name as required.
3. Click on the **OK** button.
4. If necessary, change the order of the states by either:
  - Clicking on the up or down arrows (  and  ) beside each state name, or



- Right-clicking on the state name and selecting the **Move Up** or **Move Down** context menu options.

**Note:**

You can also edit the states using the [States](#) <sup>29</sup> tab of the [Configure Timeline](#) dialog.

**Delete States**




1. Right-click on the state name. The context menu displays.
2. Select the **Delete** option.

Alternatively:



1. Click on the State Lifeline element.
2. Hold down **[Ctrl]** and move the cursor over the state name. The cursor changes form (↔).
3. Click the mouse button. The state name is deleted.

**1.1.5.3.3 Edit Transitions In State Lifeline****Add and Move Transitions**

After you have added states, you can add transitions via the diagram. As you move the cursor over the timeline, the cursor changes to one of three shapes:

- The *move* cursor () displays when it is directly over the timeline. Hold down the mouse button and drag the line to move the timeline to a state above or below the current position. You can move the transition more than one state up or down, if necessary.
- The *new transition up* cursor () displays when it is just below the timeline, and there is another state above the line. Press and hold **[Alt]**; the cursor changes () . Click to create a new transition to the state above the line. To push the transition up more than one state, then move the cursor onto the line and drag it up. The transition is for one interval unit; to make it longer, see *Change the Transition Time* below.

If you do not hold **[Alt]**, the cursor does not change and the whole timeline from the transition onwards moves up.

- The *new transition down* cursor () displays when it is just above the transition line, and there is another state below the line. Press and hold **[Alt]**; the cursor changes () . Click to create a new transition to the state below the line. To push the transition down more than one state, then move the cursor onto the line and drag it down. The transition is for one interval unit; to make it longer, see *Change the Transition Time* below.

If you do not hold **[Alt]**, the cursor does not change and the whole timeline from the transition onwards moves down.

As you move the cursor over the vertical line of a transition, the time at which the transition occurs displays next to the line.

**Edit Transitions**

Follow the steps below:

1. Click directly on the appropriate transition line, after the transition begins. Alternatively, right-click on the transition line to display the context menu, and select the **Edit** menu option.

The [Edit Transition](#) dialog displays. The fields in this dialog are all optional.

At Time: 6

Transition To: Waitcard

Event: Code

Duration Constraint: d..d\*3

Time Constraint:

OK Cancel

2. In the **At Time** field, type the point on the timescale at which the transition occurs.
3. In the **Transition To** field, type the name of the state to which the transition occurs.
4. In the **Event** field, type the name of the event that the transition represents; this displays on the Timeline element just above the transition line.
5. In the **Duration Constraint** field, type any constraint on the duration of the transition; this displays on the Timeline element, along the top of the element over the transition.
6. In the **Time Constraint** field, type any constraint on the start of the transition. This displays on the Timeline element at the start of the transition.
7. Click on the **OK** button.

#### Notes:

- Once **Event**, **Duration Constraint** or **Time Constraint** are displayed on the diagram, you can edit them directly by clicking on them to display their specific dialog. You can also delete them by pressing and holding **[Ctrl]** as you click on them; the cursor changes form when you press **[Ctrl]**.
- You can also edit transitions using the [Transitions](#)<sup>[31]</sup> tab of the **Configure Timeline** dialog.

## Change the Transition Time

Move the cursor over one or other of the vertical transition lines and drag the line left or right to change the time of the transition. While on the line, the cursor shape changes to the horizontal movement cursor (←→).

## Merge Transitions

If necessary, you can 'push' a transition to merge it with the next or previous transition point on any Lifeline element on the diagram.

Position the cursor off the appropriate side of the transition line; the cursor changes form (← or →). Click the mouse button. The system locates the nearest transition in the required direction, on any element on the diagram, and merges the current transition with that transition.

## Delete Transitions

Transitions are automatically deleted when you move the transition to the same state as the previous transition state, and release the cursor.

Alternatively, right-click on the transition line to display the context menu, and select the **Delete** menu option.

### 1.1.5.3.4 Add and Edit Value Lifeline

From the Enterprise Architect UML **Toolbox** drag a [Value Lifeline](#)<sup>[149]</sup> element onto your diagram. The element displays on the diagram.

To edit the Value Lifeline name, follow the steps below:

1. Right-click on the element. The context menu displays.
2. Select the **Other Properties** option. The **Timeline <name>** dialog displays, showing the **General** tab.

3. Overtyping the **Name** field.
4. Click on the **Apply** button and the **OK** button.

## Sizing and Scale

In the top left corner of a selected Lifeline element are the left and right *quick sizing* buttons (↔ ↔). These buttons increase or decrease the width of the Lifeline element, which in turn controls the scale width of each time unit. By increasing the width of the element you increase the resolution when adding transitions, which makes them easier to edit.

### Note:

In order to edit the Value Lifeline element, you must click on it to select it.

You now [add states](#) <sup>[28]</sup> and [edit transitions](#) <sup>[28]</sup> on the Value Lifeline.

### 1.1.5.3.5 Add States In Value Lifeline

#### Add States

This is similar to adding states to a [State Lifeline](#) <sup>[25]</sup> element.


### Notes:

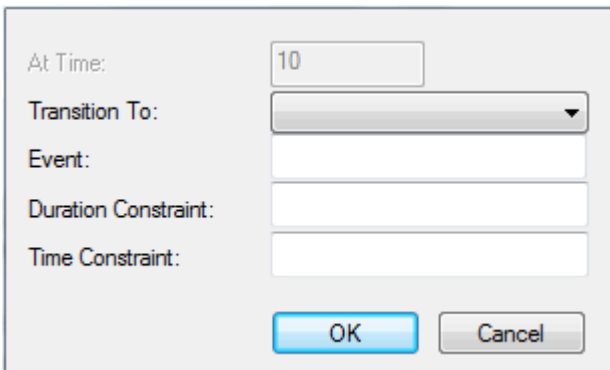
- For a Value Lifeline, only the first state displays on the diagram. The other states are added to a list to access when creating transitions; they only display on the Lifeline element as you create transitions to those states.
- You can only edit or delete states in a Value Lifeline element using the [States](#) <sup>[29]</sup> tab of the **Configure Timeline** dialog.

### 1.1.5.3.6 Edit Transitions In Value Lifeline

#### Add Transitions

After you have added states to the Value Lifeline element, you can add transitions via the diagram. To do this, follow the steps below:

1. Move the cursor above the transition line. The cursor changes form (⌘). 
2. Click the mouse button. The **New Transition Event** dialog displays.



3. In the **Transition To** field, click on the drop-down arrow and select a state from the list of available states; this displays on the Lifeline element within the transition box. The remaining fields on the dialog are optional.
4. In the **Event** field, type the name of the event that the transition represents; this displays on the Lifeline element just below and at the start of the transition line.

5. In the **Duration Constraint** field, type any constraint on the duration of the transition; this displays on the Lifeline element, along the top of the element over the transition.
6. In the **Time Constraint** field, type any constraint on the start of the transition. This displays on the Lifeline element at the start of the transition, just after the Event name.
7. Click on the **OK** button to create the new transition.

### Edit Transitions

To edit a transition, follow the steps below:

1. Click on the state name in the transition. Alternatively, right-click on the state name to display the context menu, and select the **Edit** menu option.

The **Edit Transition** dialog displays; this is the same as the **New Transition Event** dialog, except that the **At Time** field is enabled.


2. If necessary, overwrite the **At Time** field to define a different start point.

#### Note:


You cannot change the **At Time** field for the first state in the timeline; this is always **0**.

3. Edit the remaining fields as necessary.
4. Click on the **OK** button to save the changes.

### Change the Transition Time

To change the start or end time of a transition, click on the start or end point of the transition and drag it to the new position. While on the line, the cursor shape changes to the horizontal movement cursor ()


### Delete Transitions

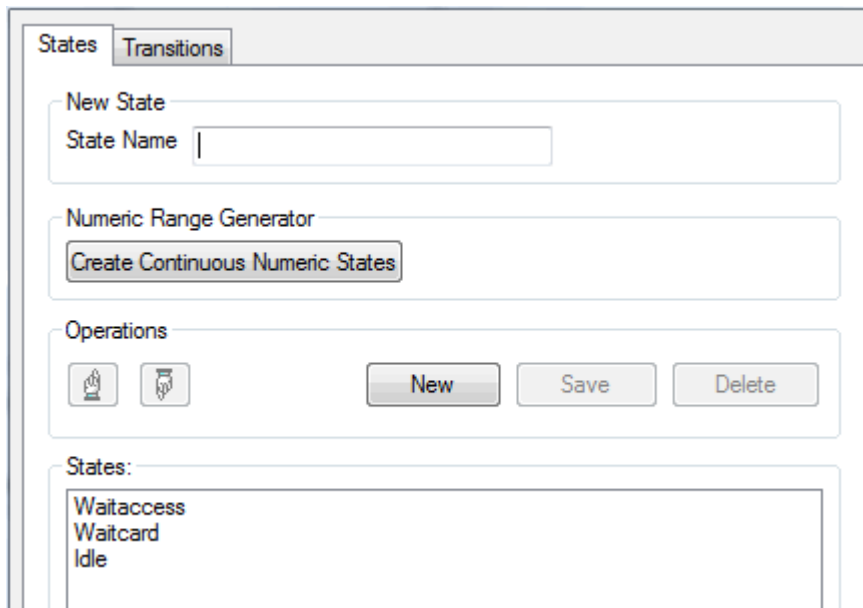
To delete a transition, press and hold **[Ctrl]** and click on the transition state name. While you hold **[Ctrl]** on the transition state name, the cursor changes form ()

Alternatively, right-click on the state name to display the context menu, and select the **Delete** menu option.

#### 1.1.5.3.7 Configure Timeline - States

You can also manage states using the **States** tab of the **Configure Timeline** dialog. To display this, either:

- Double-click on the Lifeline element
- Right click on the Lifeline element and, from the context menu, select the **Properties** option, or
- On a Value Lifeline, click on the **Edit States** button ()



The **Configure Timeline** dialog defaults to the **States** tab.

All states currently defined for the Lifeline element are listed in the **States** panel.

#### Add a New State:

1. In the **State Name** field, type the name of the first new state in the Lifeline element; for example, **WaitState**.
2. Click on the **Save** button. The state is added to the **States** panel and (for a State Lifeline Element) to the diagram.
3. Click on the **New** button.
4. In the **State Name** field, type the name of the next state in the Lifeline element.
5. Repeat steps 2 to 5 until you have added all required states (you must add at least three to the Lifeline element).
6. When you have added all the required states, click on the **OK** button to close the **Configure Timeline** dialog.



#### Edit an Existing State:

1. Click on the state in the **States:** list.
2. In the **State Name** field, change the name of the state.
3. Click on the **Save** button.

#### Delete an Existing State:

1. Click on the state in the **States:** list.
2. Click on the **Delete** button.

#### Change the Order of States:

1. Click on the state in the **States:** list.
2. Click on the  or  buttons to move the state up or down the sequence.

### Numeric Range Generator

You can also use the **Configure Timeline** dialog to create a range of states having numeric values to be applied to the Timeline.

**Important:**

This operation deletes all existing states and transitions for the Timeline element.

1. Display the **Configure Timeline** dialog.
2. Click on the **Create Continuous Numeric States** button. The **Numeric Range Generator** dialog displays.
3. In the **High Value** and **Low Value** fields, type the upper and lower values of the range.
4. In the **Step Value** field, type the increase interval.


**Note:**

Nonsense values do not parse; **Low Value** must be less than **High Value**, and **Step Value** must be a positive value smaller than the total range.

5. In the **Units** field, type the name of the measurement unit; for example, **minutes**.
6. Click on the **OK** button. Enterprise Architect displays a warning that existing states and transitions will be deleted.
7. Click on the **Yes** button. The **Configure Timeline** dialog redisplay, with the defined range of states listed in the **States** panel.
8. Click on the **OK** button. For a:
  - Value Lifeline, the first state is shown on the Timeline for the full time range of the Timeline.
  - State Lifeline, the range of states is displayed as the y-axis of the Timeline.

#### 1.1.5.3.8 Configure Timeline - Transitions

You can also manage transitions using the **Transitions** tab of the **Configure Timeline** dialog. To display this, either:

- Double-click on the Lifeline element
- Right click on the Lifeline element and, from the context menu, select the **Properties** option, or
- On a Value Lifeline, click on the **Edit States** button (  ).

The **Configure Timeline** dialog defaults to the **States** tab. Click on the **Transitions** tab.

Time	State	Event	Duration C...	Time Const...
0	Idle	Start		
6	Waitcard	Code	d..d*3	
15	Waitaccess			
18	Idle	OK		t..t+3

All transitions defined for the Timeline element are listed in the **Transition Points** panel.

### Add a New Transition

1. Click on the **New** button.
2. In the **New Transition** panel, type the details of the transition.
3. Click on the **Save** button.

### Edit a Transition

1. Click on a transition in the list.
2. In the **Edit Transition** panel, edit the fields for the transition as required.
3. Click on the **Save** button.

### Delete a Transition

1. Click on a transition in the list.
2. Click on the **Delete** button. The transition is removed from the dialog and the Lifeline.
3. Click on the **OK** button.

#### 1.1.5.4 Time Intervals

You create and manage Time Intervals using the *Interval Bar* (the pale line along the top of each selected Lifeline element). Time Intervals enable you to perform various operations on transitions, such as copy and paste. They also enable you to compress sections of the timeline so that they are not visible.

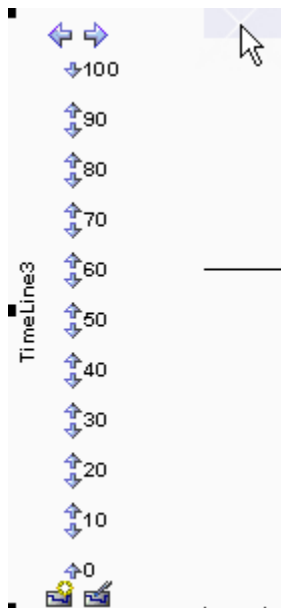
Each Time Interval displays across all Timeline elements down to the last element on the diagram.

### Create Time Intervals

To create a Time Interval, follow one of the three sets of steps below:

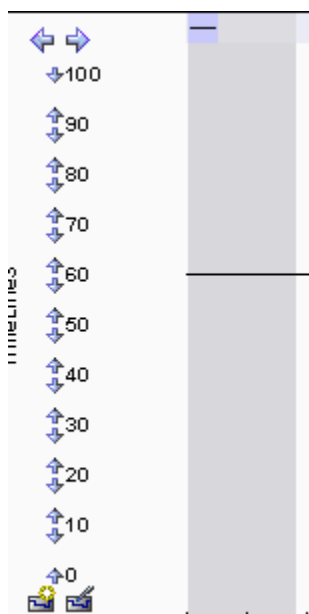
### Interval Bar - Context Menu

1. Right-click on the Interval Bar at approximately the point at which to start or finish the Time Interval.



The context menu displays.

2. Select the **Create Time Interval** option. The Time Interval displays down all the timeline elements, as a narrow pale band with a blue compression box at the top.



3. Move the cursor to the edge of the Time Interval in the Interval Bar so that the cursor changes to the drag form ( $\leftarrow\rightarrow$ ) and drag the edge to the correct start or end point.

### Interval Bar - [Shift] key

1. Move the cursor over the Interval Bar and press **[Shift]**. The cursor changes shape ( $\leftarrow\rightarrow$ ).
2. Click to create the Time Interval.
3. Move the cursor to the edge of the Time Interval in the Interval Bar so that the cursor changes to the drag form ( $\leftarrow\rightarrow$ ) and drag the edge to the correct start or end point.



**Timeline - Context menu**

1. Right-click on the timeline just after a transition. The context menu displays.
2. Click on the **Select** menu option. Enterprise Architect creates a Time Interval covering the period from the selected transition up to the next transition.

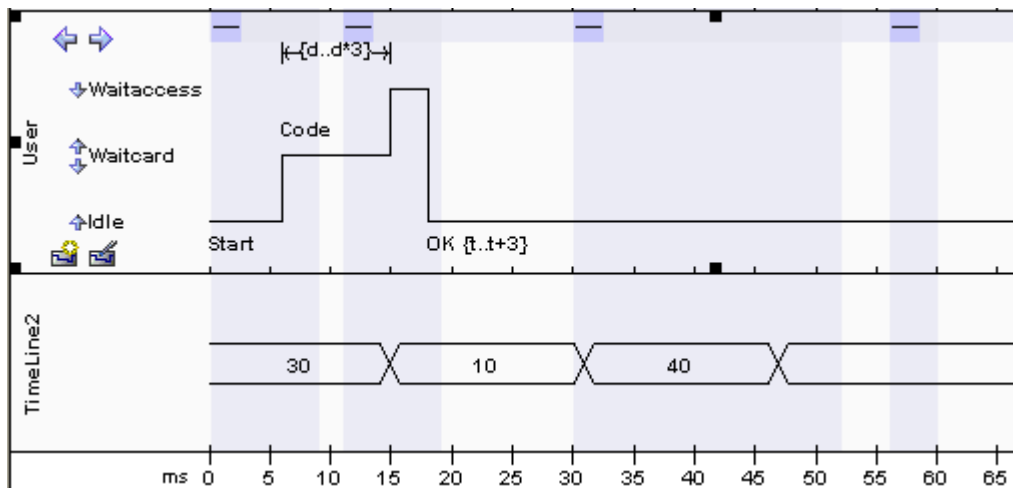
**Notes:**

- If there are other Time Intervals in this period, Enterprise Architect replaces them with the single Time Interval for the transition state. You should consider this when creating the Time Interval, as it extends across the other Timeline Elements in the diagram.
- A value of this method is that it creates a Time Interval for a period in which no transitions occur, which could be lengthy. You can then compress this Time Interval (see below) to hide the period of inactivity. See also [Compress Timeline](#).

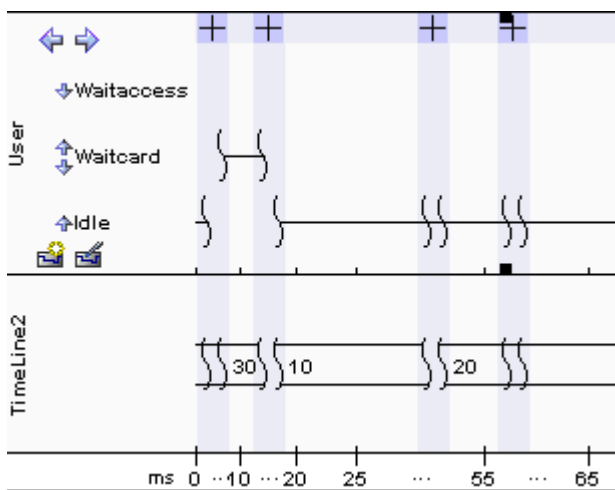
**Compress Time Intervals**

You can compress Time Intervals to conserve space on long timelines.




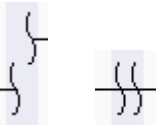
**Uncompressed Time Intervals**



**Compressed Time Intervals**



Notice:

Item	Description
	The compression toggle boxes: <ul style="list-style-type: none"> <li>•  is expanded, click on this to compress the selected time interval</li> <li>•  is compressed, click on this to expand the selected time interval again.</li> </ul>
	The compressed sections of the timelines themselves, in all elements. If there is space between the paired symbols, there are transitions within the compressed section. If the timeline continues through the paired symbols there are no transitions in the compressed section.
25 ... 55	The compressed sections in the time range underneath the elements.

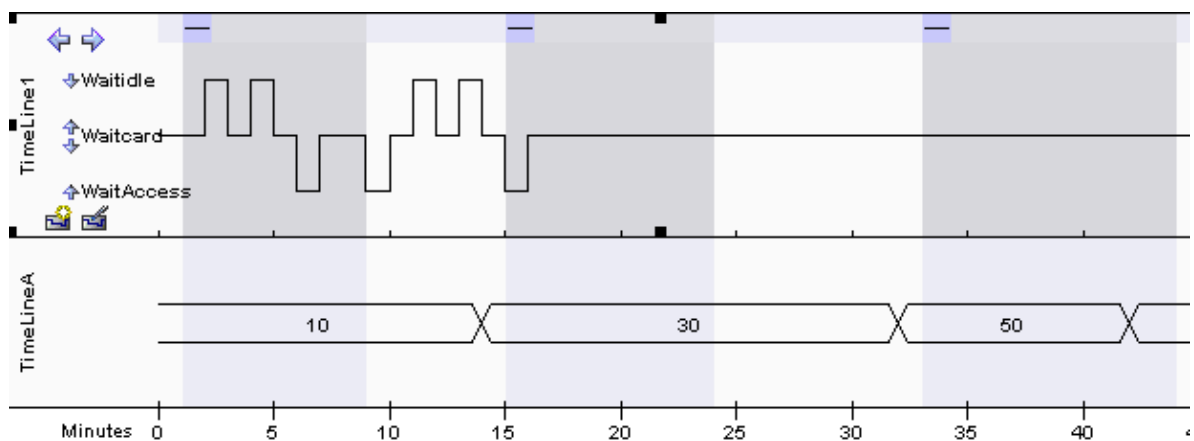
You can also compress and expand Time Intervals using context menu options; see [Time Interval Operations on Transitions](#)<sup>[36]</sup>.

### Select Time Intervals

- To select a Time Interval across all elements on the diagram, click on the Interval Bar within the Time Interval.
- To select a number of individual Time Intervals, press and hold **[Ctrl]** while clicking on the Interval Bar within each Time Interval.
- To select all Time Intervals in a range, click on the Interval Bar within the first Time Interval in the range, then press and hold **[Shift]** and click on the Interval Bar within the last Time Interval in the range. All Time Intervals between the two are selected.

After you have selected one or more Time Intervals, you can modify the selection in the following ways:

- To exclude Lifeline elements from the selection, press and hold **[Ctrl]** and click on any part of the selection within that element. In the diagram below, the Value Lifeline is excluded from selection.



Repeat the step to toggle the selection and re-include the element. See also [Toggle Interval Selection](#)<sup>[36]</sup>.

- To select only one Lifeline element and exclude all others, press and hold **[Shift]** and click on any part of the selection within that element.

#### Note:

Selection is useful for cutting, copying and pasting transitions.

### Move and Resize Time Intervals

To move a Time Interval, move the cursor over the Interval bar within the Time Interval, hold down the mouse button and drag the interval left or right.

To resize a Time Interval, move the cursor over the Interval Bar at the start or end edge of the Time Interval,

hold down the mouse button and move the edge left or right.

**Note:**

Time Intervals can meet, but cannot overlap.

### Delete Time Intervals

Select [\[35\]](#) each Time Interval to be deleted and press **[Delete]**.

**Note:**

Deleting the Time Interval does not delete transitions within that interval.

#### 1.1.5.4.1 Time Interval Operations

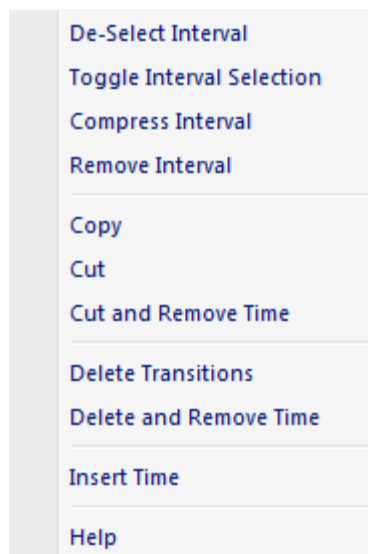
You can operate on selected [Time Intervals](#) [\[32\]](#), or all Time Intervals in the diagram.

### Selected Intervals

**Note:**

The Copy, Cut and Delete operations act on all selected Time Intervals over the whole diagram, not just the current one.

To select and update specific Time Intervals, right-click on the Interval Bar within an interval. The following context menu displays.



Option	Use to
<b>Select Interval</b> <b>Deselect Interval</b>	Select the Time Interval or, if the interval is already selected, deselect it. You can select several Time Intervals in this way, accessing the menu separately on each interval.
<b>Toggle Interval Selection</b>	Switch the selection or deselection of the Time Interval within the selected Timeline element. You select or deselect a Time Interval across all Timeline Elements, but the Toggle option acts only on the element in which you access the menu. See also <a href="#">Select Time Intervals</a> <a href="#">[35]</a> .

Option	Use to
<b>Compress Interval</b>	Compress the Time Interval, and hide all transitions within that Time Interval. This is also useful for hiding long sections of inactivity on the time line. Also see <a href="#">Compress Timeline</a> <sup>[37]</sup> , below.
<b>Remove Interval</b>	Delete the Time Interval.
<b>Copy</b>	Copy the transitions for all selected Time Intervals.
<b>Cut</b>	Copy and delete the selected transitions from the diagram.
<b>Cut and Remove Time</b>	Copy and delete the transitions that lie in the selected Time Intervals from the diagram.  This option also removes time from the timeline, the amount being the duration of the Time Interval. All transitions and Time Intervals to the right of the selected time interval are moved left.
<b>Delete</b>	Delete the selected transitions from the diagram.
<b>Delete and Remove Time</b>	Delete the transitions that lie in the selected Time Intervals from the diagram.  This option also removes time from the timeline, the amount being the duration of the Time Interval. All transitions and Time Intervals to the right of the current Time Interval are moved left.
<b>Insert Time</b>	Add time to the timeline and move all transitions and time intervals to the right. Also expand the duration of the current Time Interval.

### Compress Timeline

The Compression toggle boxes and **Compress Interval** menu option operate on the Time Interval and compress the timeline and all transitions within the Interval. You have an alternative option that operates on the timeline and compresses a single transition state.

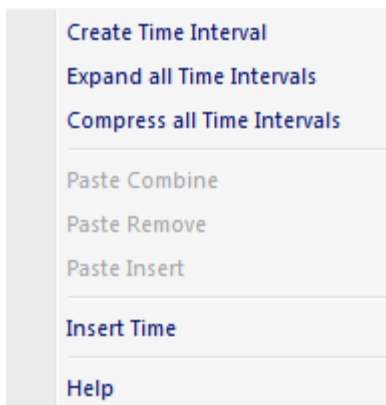
1. Right-click on the timeline (rather than the Interval Bar) just after a transition. The context menu displays.
2. Click on the **Compress** menu option. Enterprise Architect creates a new Time Interval covering the period from the selected transition up to the next transition, and then compresses that Time Interval.

#### Notes:


- If there are other Time Intervals in this period, Enterprise Architect replaces them with the single Time Interval for the transition state. You should consider this when creating and compressing the Time Interval, as it extends across the other Timeline elements in the diagram.
- A value of this method is that it creates a Time Interval for a period in which no transitions occur, which could be lengthy, and then compresses this Time Interval to hide the period of inactivity.

### All Time Intervals in the Diagram

To create a new Time Interval or work across all Time Intervals in the diagram, right-click on the Interval Bar between Time Intervals. The following context menu displays.


**Note:**

The **Paste** menu options become active after transitions have been copied.


Menu Option	Use to
Create Time Interval	<a href="#">Create a single Time Interval</a> 
Expand all Time Intervals	Expand all Time Intervals over the whole diagram.
Compress all Time Intervals	Compress all Time Intervals over the whole diagram.
Paste Combine	Paste copied transitions over any existing transitions within the copied time frame.  <b>Note:</b> The diagram does not allow two consecutive transitions to the same state, and removes the second transition automatically.
Paste Remove	Delete all the transitions and then pastes the copied transition within the copied time frame.
Paste Insert	Insert time, moving all transitions and Time Intervals to the right to make room to paste in the copied transitions.
Insert Time	Add time to the timeline and move all transitions and Time Intervals to the right. This option does not change the duration of any Time Interval.

### Copy and Paste Transitions From One Timeline Element to Another

A special mode enables you to copy transitions from one Timeline element to another. Any states that don't exist in the Timeline element you are pasting to are created.

1. Press and hold **[Shift]** and select the Timeline element within a Time Interval to copy or cut.
2. Right-click on the Interval Bar (it doesn't matter which element you select). The context menu displays.
3. [Copy or cut](#)  the transitions. You can also cut and remove time.
4. Select the timeline to paste transitions to and right-click on the Interval Bar. The context menu displays.
5. Select one of the paste operations. Note that states are created if they don't already exist in the timeline.

**Note:**

Any new states created might be in the wrong order. You can change the order via the diagram [quick buttons](#) .

## Shift Transitions Left or Right

You can move transitions within a selected Time Interval or multiple selected Time Intervals.

1. Select all the Time Intervals containing the transitions to be shifted; see [Select Time Intervals](#)<sup>[35]</sup>.
2. Press and hold **[Shift]** and click on the Interval Bar (it doesn't matter which Timeline element you select) and move the transition left or right.

### Note:

You cannot drag transitions over other transitions; the move stops when the moved transition collides with a stationary transition.

### Tip:

If having collision problems, use **[Shift]+select** to shift transitions for a single Timeline element.

## 1.1.6 Sequence Diagram

A *Sequence* diagram is one of four types of *Interaction* diagram. (The other three are [Timing Diagram](#)<sup>[22]</sup>, [Interaction Overview Diagram](#)<sup>[52]</sup> and [Communication Diagram](#)<sup>[49]</sup>.)

A Sequence diagram is a structured representation of behavior as a series of sequential steps over time. It is used to depict work flow, message passing and how elements in general cooperate over time to achieve a result.

- Each [sequence element](#)<sup>[43]</sup> is arranged in a horizontal sequence, with messages passing back and forward between elements.
- Messages on a Sequence diagram can be of several types; the Messages can also be configured to reflect the operations and properties of the source and target elements (see the *Notes* in the [Message](#)<sup>[212]</sup> topic).
- An Actor element can be used to represent the user initiating the flow of events.
- Stereotyped elements, such as [Boundary](#)<sup>[179]</sup>, [Control](#)<sup>[181]</sup> and [Entity](#)<sup>[182]</sup>, can be used to illustrate screens, controllers and database items, respectively.
- Each element has a dashed stem called a lifeline, where that element exists and potentially takes part in the interactions.

To configure a Sequence diagram, see the following topics:

- [Denote the Lifecycle of an Element](#)<sup>[41]</sup>
- [Layout of Sequence Diagrams](#)<sup>[42]</sup>
- [Sequence Element Activation](#)<sup>[45]</sup>
- [Lifeline Activation Levels](#)<sup>[46]</sup>
- [Message Label Visibility](#)<sup>[48]</sup>
- [Change the Top Margin](#)<sup>[48]</sup>
- [Change the Timing Details](#)<sup>[218]</sup>.

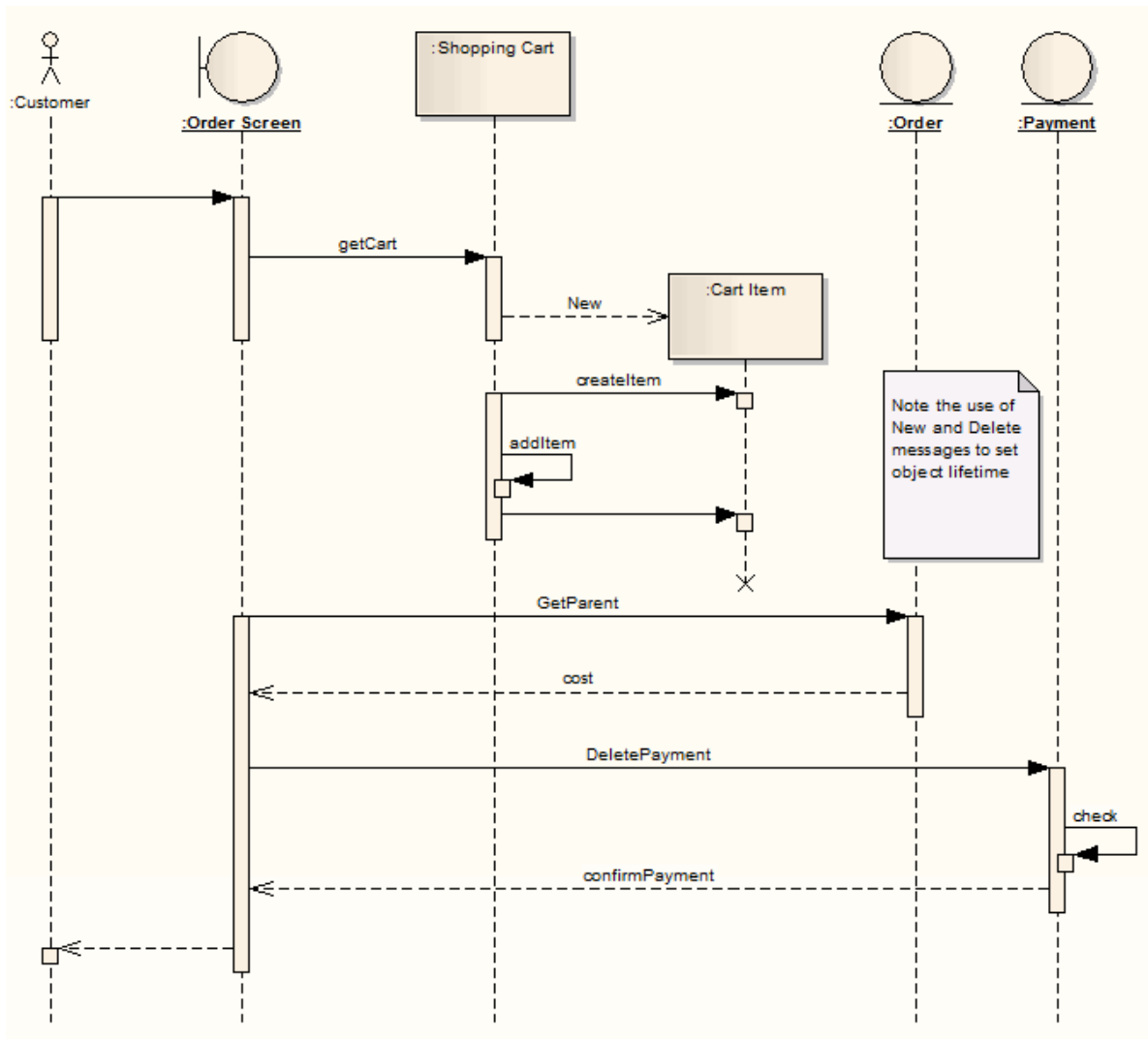
Also take note of the important information in the [Sequence Diagrams and Version Control](#)<sup>[44]</sup> topic.

*Robustness diagrams*, used extensively in *ICONIX*, can be created as Sequence diagrams.

To toggle the numbering of messages on a Sequence diagram, select or deselect the **Show Sequence Numbering** checkbox on the **Options** dialog; see *Using Enterprise Architect – UML Modeling Tool*.

## Example Diagram

The following example Sequence diagram demonstrates several different elements:



### Toolbox Elements and Connectors








Select Sequence diagram elements and connectors from the [Interaction](#) pages of the Enterprise Architect UML [Toolbox](#); see *Using Enterprise Architect – UML Modeling Tool*.

Enterprise Architect also supports a number of [stereotyped elements](#) <sup>75</sup> to represent various entities in business modeling.

**Tip:**

Click on the following elements and connectors for more information.

Sequence Diagram Elements	Sequence Diagram Connectors
Actor	Message
Lifeline	Self-Message
Boundary	Recursion

Sequence Diagram Elements	Sequence Diagram Connectors
 Control	 Call
 Entity	
 Fragment	
 Endpoint	
 Diagram Gate	
 State/Continuation	

### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 503*) states:

*A sequence diagram describes an Interaction by focusing on the sequence of Messages that are exchanged, along with their corresponding OccurrenceSpecifications on the Lifelines.*

#### 1.1.6.1 Denote Lifecycle of an Element

You can capture element lifetimes using messages that are denoted as *New* or *Delete* message types. To do this, follow the steps below:

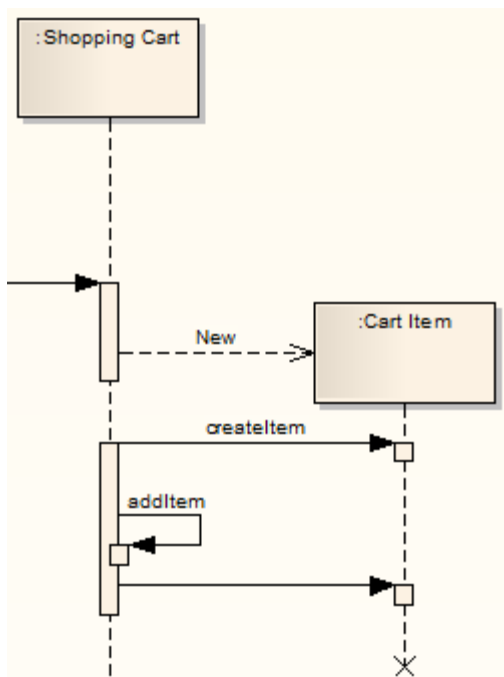
1. Double-click on a message within a Sequence diagram to display the **Message Properties** dialog.
2. In the **Lifecycle** field, click on the drop-down arrow and select **New** or **Delete**.
3. Click on the **OK** button to save the changes.

The example below shows two elements that have specific creation and deletion times.

#### Note:

To show the termination **X** on the lifeline in the following example diagram, you must switch on garbage collection: **Tools | Options | Diagram | Sequence | Garbage Collect**.





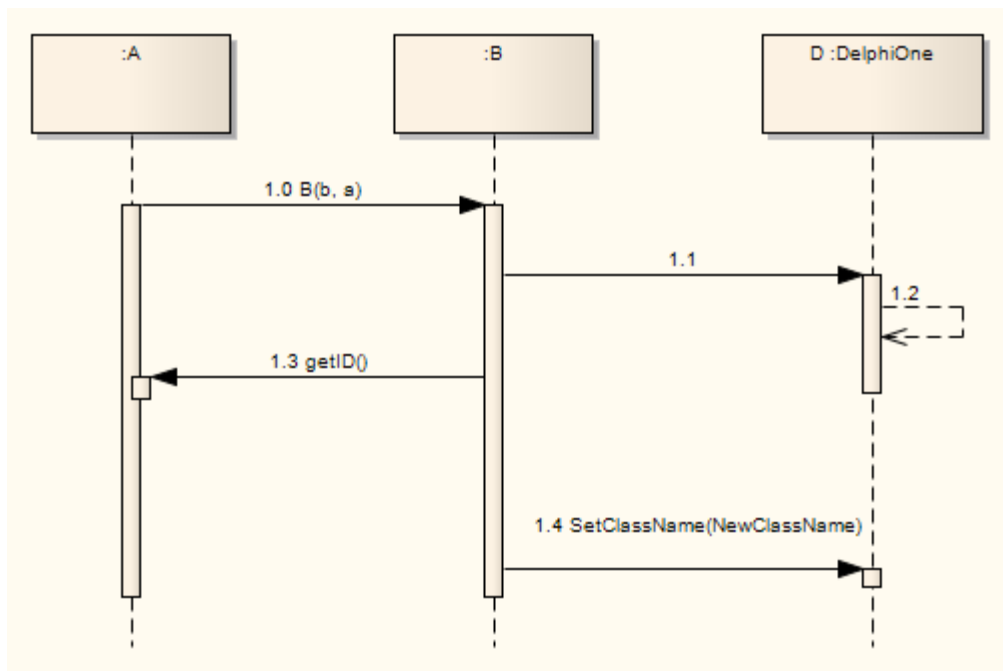
### 1.1.6.2 Layout of Sequence Diagrams

You can modify the vertical height of sequence messages to get an attractive and effective layout. To offset message positions, follow the steps below:

1. Select the appropriate message in a Sequence diagram.
2. Use the mouse to drag the message up or down as required.

As you drag a message up or down a lifeline, any messages or fragments below that message are shifted up or down the same amount. However, be aware that if you drag up or down past the next or previous message, Enterprise Architect interprets that as the requirement to swap positions, rather than simply offset a message position.

The example below shows an economical use of space in a Sequence diagram.



### 1.1.6.3 Sequence Elements

A [Sequence diagram](#) <sup>39</sup> models a dynamic view of the interactions between model elements at runtime.

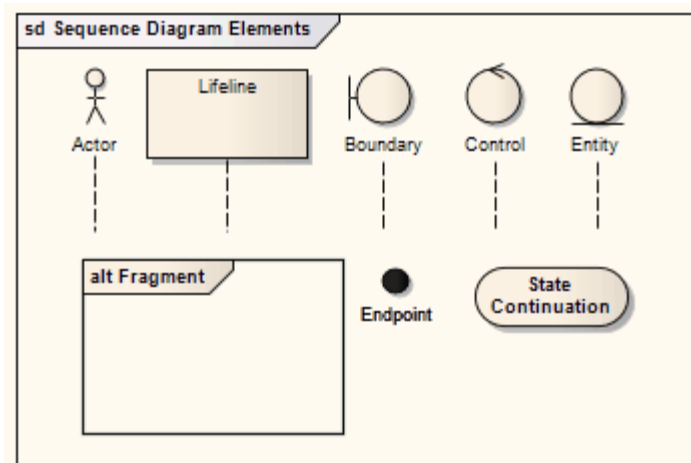
Sequence diagrams are commonly used as explanatory models for Use Case scenarios. By creating a Sequence diagram with an Actor and elements involved in the Use Case, you can model the sequence of steps the user and the system undertake to complete the required tasks. An element in a Sequence diagram is usually either an Actor (the stimulus that starts the interaction) or a collaborating element.

#### Note:

A Sequence diagram is often attached directly under the Use Case to which it refers. This helps keep elements together, both in the model and when documentation is produced. To do this, right-click the Use Case on the diagram and select the **Advanced | Make Composite** context menu option.

The example below shows some possible elements of Sequence diagrams and their stereotyped display.

- *Actor* - An instance of an actor at runtime.
- *Lifeline* - An Object element with the stereotype Lifeline.
- *Boundary* - Represents a user interface screen or input/output device.
- *Entity* - A persistent element - typically implemented as a database table or element.
- *Control* - The active component that controls what work gets done, when and how.

**Tip:**

Use Sequence diagrams early in analysis to capture the flow of information and responsibility throughout the system. Messages between elements eventually become method calls in the Class model.

#### 1.1.6.4 Sequence Diagrams and Version Control

You might create Sequence diagrams that use elements from other packages as the Lifelines within the diagram. In such cases, the diagrams could be corrupted when the element packages are checked in and out under version control. This is because during checkout the elements are first deleted from the model and then re-imported, and although they are reinstated in the diagrams, any Messages connecting them are not.

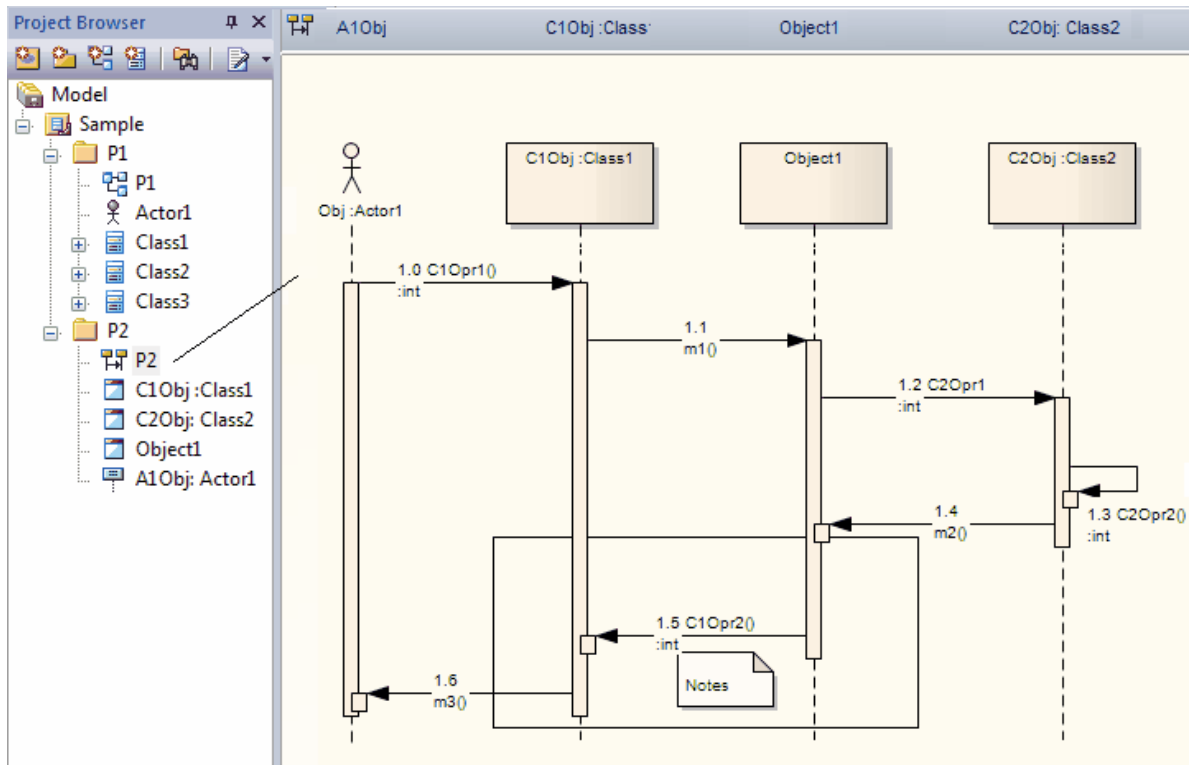
So, if the diagram and its elements reside in different packages, a round-trip of the element package through version control might damage the Sequence diagram.

The solution is to drag-and-drop each Class onto the Sequence diagram as an *object* - when you drop the Class onto the Sequence diagram, in the **Paste Element** dialog select the **as Instance of Element (Object)** option. This creates a new object in the *diagram's* parent package, based on the selected Class element. You then create the Messages between the objects.

Therefore, to ensure that a Sequence diagram is not damaged by round-trips of other packages through version control, remember that:

- The Lifelines must be objects (even though Enterprise Architect allows you to drop elements as Lifelines onto a Sequence diagram, it is not a strictly UML compliant construct)
- The Lifelines must be in the same package as the diagram.

The following illustration shows the **Project Browser** with two packages: *P1*, containing the elements, and *P2*, containing a Sequence diagram that uses those elements. The diagram itself is also shown.

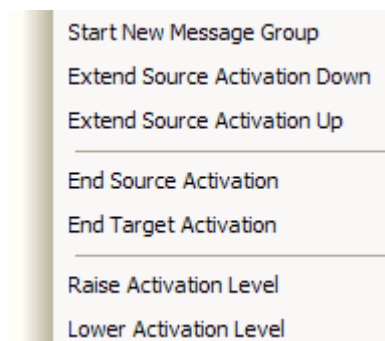


This diagram will not be damaged, because all the Lifelines are objects and these objects reside in the same package as the Sequence diagram.

### 1.1.6.5 Sequence Element Activation

Sequence elements in a [Sequence diagram](#) have *Activation rectangles* drawn along their lifelines. These rectangles describe the time the element is active during the overall period of processing. This visual representation can be suppressed by right-clicking the Sequence diagram, and selecting the **Suppress Activations** context menu option.

In general, Enterprise Architect calculates the period of activation for you, but in some cases you might want to fine tune the rectangle length. There are several context menu options on a sequence message that you can use to accomplish this. To access the following context menu, right-click on the message and select the **Activations** context menu option.

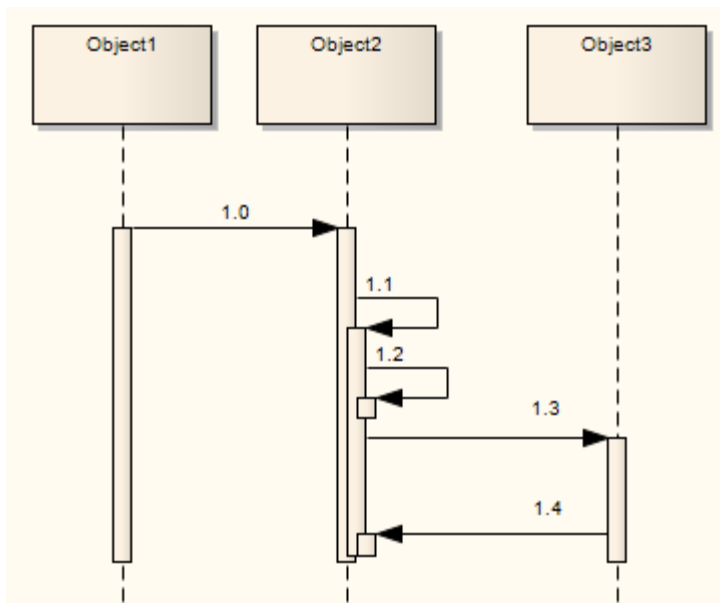


- **Start New Message Group:** Starts off a new round of processing in the current diagram. This enables you to describe more than one processing scenario in a single diagram.
- **Extend Source Activation Down:** Forces an element to stay active beyond the normal processing period. This could be used to express an element that continues its own processing concurrently with other processes.

- **Extend Source Activation Up:** Forces an element's activation upwards.
- **End Source Activation:** Truncates the activation of the source element after the current message. This is useful for expressing an asynchronous message after which the source element becomes idle.
- **End Target Activation:** Ends a Forced Activation started by the **Extend Source Activation** options.

The **Raise Activation Level** and **Lower Activation Level** options display on the context menu only where their use is appropriate. For example, after a self-message the next message starts by default at a lower activation level but the **Raise Activation Level** command displays on the context menu to enable you to raise its level.

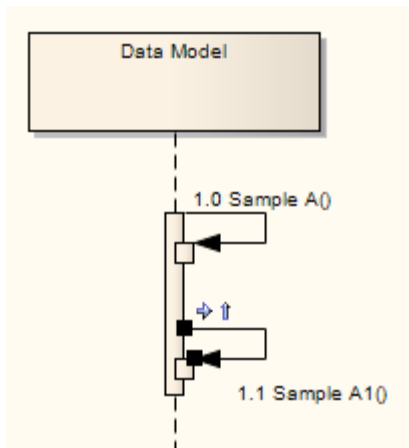
A more convenient way to change activation levels is directly on the diagram. Whenever appropriate, left and/or right arrows display on specific connectors. In the following diagram, see connector 1.3. Click on the arrow to raise or lower the activation level.

**Note:**

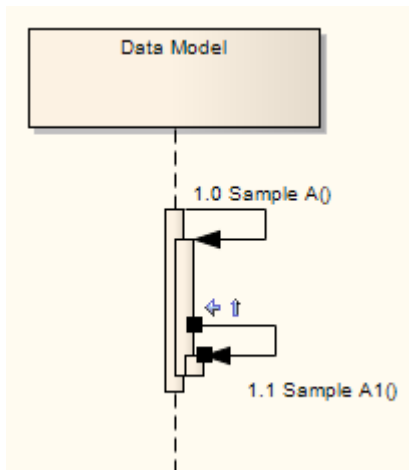
Program flow can more accurately be depicted with nested activation levels for callback messages.

### 1.1.6.6 Lifeline Activation Levels

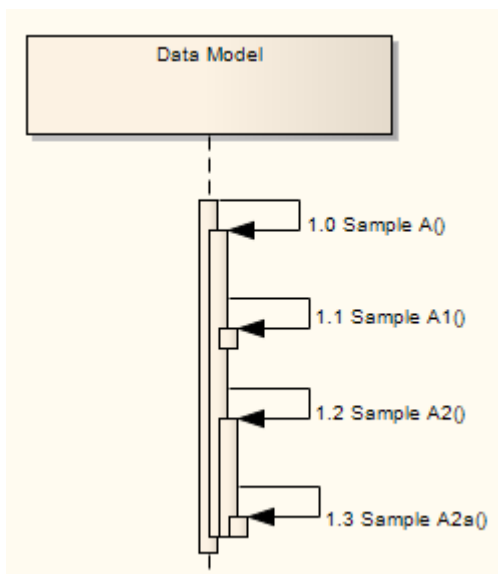
Complicated processing systems can be easily negotiated and reflected in Sequence diagrams, by adding activation layers on a single lifeline. For example, a Class invokes the method *Sample A*, which in turn calls *Sample A1*. To produce the arrangement in the diagram, select the **More tools | Interaction** menu option, click on the **Self-message** icon in the **Interaction Relationships** panel and then click on the lifeline.



In order to raise the Activation level of *Sample A1*, click on the *raise arrow* of the selected connector. The lifeline now visually depicts that method *Sample A1* is called during the processing of *Sample A*.



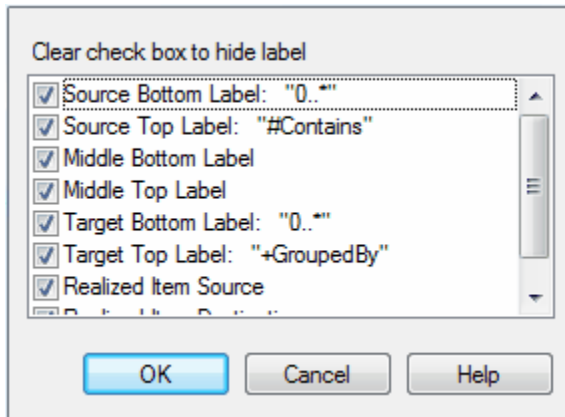
In the example below, a few more self-messages have been added. The message *Sample A2a* is called from *Sample A2* which in turn is called from *Sample A* (not *Sample A1*). *Sample A1* is called from *Sample A*.



### 1.1.6.7 Sequence Message Label Visibility

On Sequence messages, you can control label visibility using the message context menu. To hide and show the labels used in Sequence messages, follow the steps below:

1. Right-click on the message within the Sequence diagram. The message context menu displays.
2. Select the **Set Label Visibility** menu option. The **Label Visibility** dialog displays.



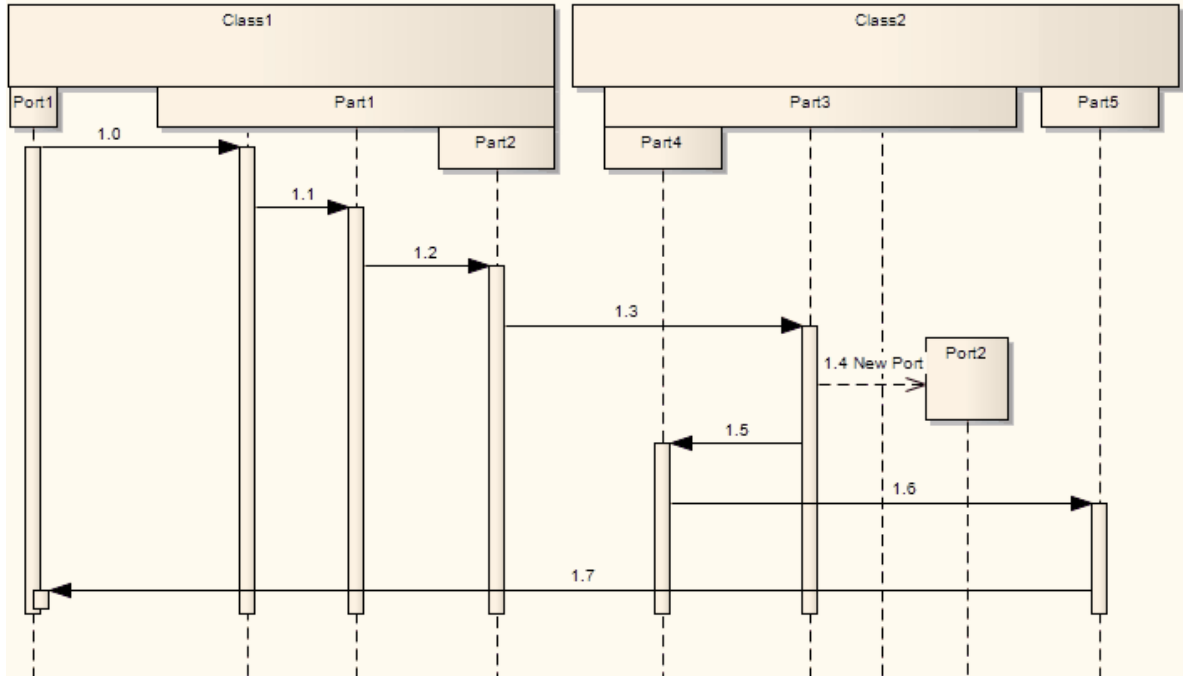
3. Select or clear the checkbox against each message label to display or hide, respectively.
4. Click on the **OK** button to save the settings.

### 1.1.6.8 Change the Top Margin

In order to change the top margin of a [Sequence diagram](#) from the default 50 units, right-click on the diagram to display the context menu and select the **Set Top Margin** menu option. You can set the top margin to any value between 30 and 250 units.

### 1.1.6.9 Inline Sequence Elements

It is possible to represent [Part](#)<sup>[166]</sup> and [Port](#)<sup>[169]</sup> elements on a [Sequence diagram](#)<sup>[39]</sup>. Child Parts and Ports appear as inline sequence elements under their parent Class sequence element.



1. Right-click on the sequence elements containing the child Ports or Parts, to display the context menu.
2. Select the **Embedded Elements | Embedded Elements** menu option.
3. Select the checkbox against each Part or Port to show, and click on the **Close** button.

### 1.1.7 Communication Diagram

One of four types of *Interaction* diagram. (The other three are [Timing Diagrams](#)<sup>[22]</sup>, [Sequence Diagrams](#)<sup>[39]</sup> and [Interaction Overview Diagrams](#)<sup>[52]</sup>.)

A *Communication diagram* shows the interactions between elements at run-time in much the same manner as a Sequence diagram. However, Communication diagrams are used to visualize inter-object relationships, while Sequence diagrams are more effective at visualizing processing over time.

Communication diagrams employ ordered, labeled associations to illustrate processing. Numbering is important to indicate the order and nesting of processing. A numbering scheme could be:

1  
 1.1  
 1.1.1  
 1.1.2  
 1.2, and so on.

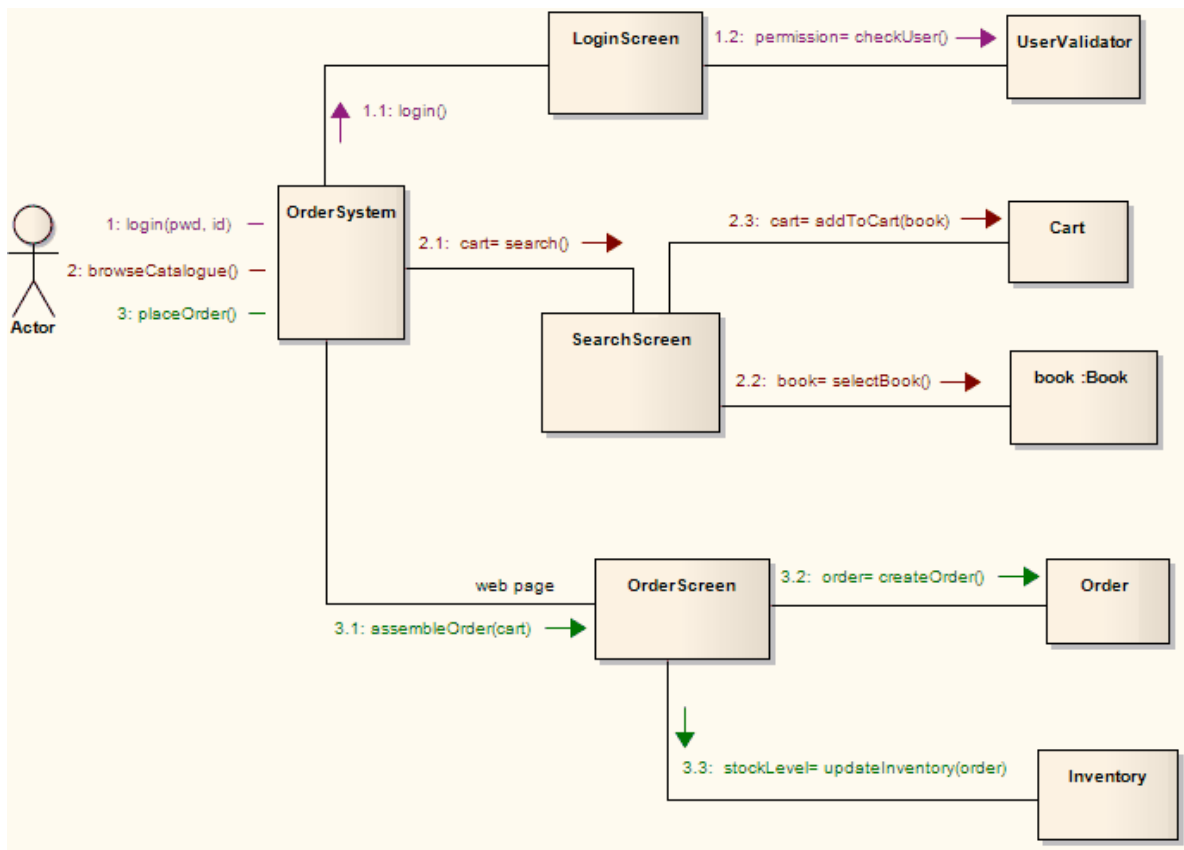
A new number segment begins for a new layer of processing, and would be equivalent to a method invocation.

*Robustness diagrams* are simplified Communication diagrams, but can be created in any diagram type that supports [Boundary](#)<sup>[179]</sup>, [Control](#)<sup>[181]</sup> and [Entity](#)<sup>[182]</sup> elements.

### Example Diagram

The example below illustrates a Communication diagram among cooperating object instances. Note the use of message levels to capture related flows, and the different [colors](#)<sup>[51]</sup> of the [messages](#)<sup>[223]</sup>.





### Toolbox Elements and Connectors

Select Communication diagram elements and connectors from the **Communication** pages of the Enterprise Architect UML **Toolbox**; see *Using Enterprise Architect – UML Modeling Tool*.

**Tip:**

Click on the following elements and connectors for more information.

Communication Diagram Elements	Communication Diagram Connectors
Actor	Associate
Object	Nesting
Boundary	Realize
Control	
Entity	
Package	

### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 511*) states:

Communication Diagrams focus on the interaction between Lifelines where the architecture of the internal structure and how this corresponds with the message passing is central. The sequencing of Messages is given through a sequence numbering scheme.

Communication Diagrams correspond to simple Sequence Diagrams that use none of the structuring mechanisms such as InteractionUses and CombinedFragments. It is also assumed that message overtaking (i.e., the order of the receptions are different from the order of sending of a given set of messages) will not take place or is irrelevant.

**Note:**

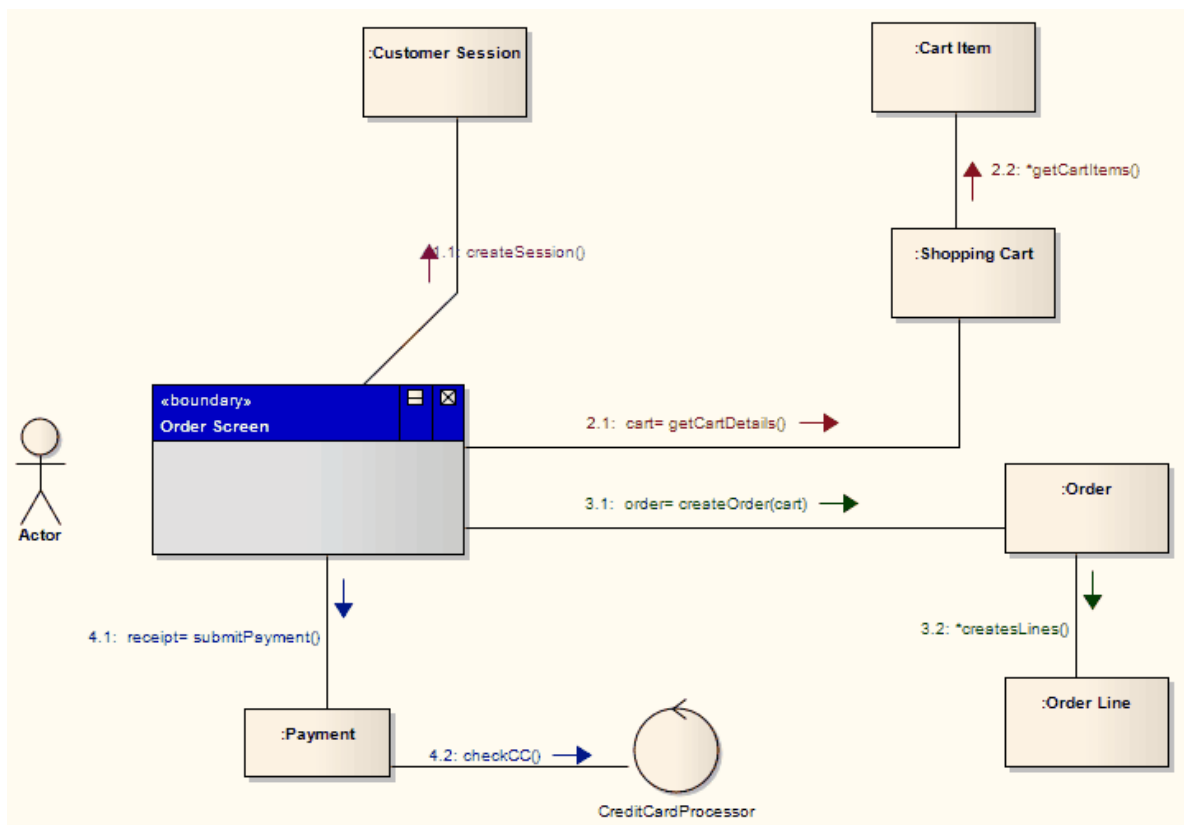
Communication diagrams were known as Collaboration diagrams in UML 1.4.

### 1.1.7.1 Communication Diagrams in Color

Enterprise Architect enables you to highlight particular message flows in a [Communication diagram](#) <sup>49</sup> using different colors for each message set.

To highlight the colors in a Communication diagram, follow the steps below:

1. Select the **Tools | Options | Communication Colors** menu option. The **Communication Message Coloring** page of the **Options** dialog displays.
2. Select the **Use Communication Color** checkbox.
3. Click on the drop-down arrow of each **Message n** field, and select the required color for each message group.
4. Click on the **Close** button. On your Communication diagram, each sequence group of messages displays in a different color as shown below.



### 1.1.8 Interaction Overview Diagram

One of four types of *Interaction* diagram. (The other three are [Timing Diagrams](#)<sup>[22]</sup>, [Sequence Diagrams](#)<sup>[39]</sup> and [Communication Diagrams](#)<sup>[49]</sup>.)

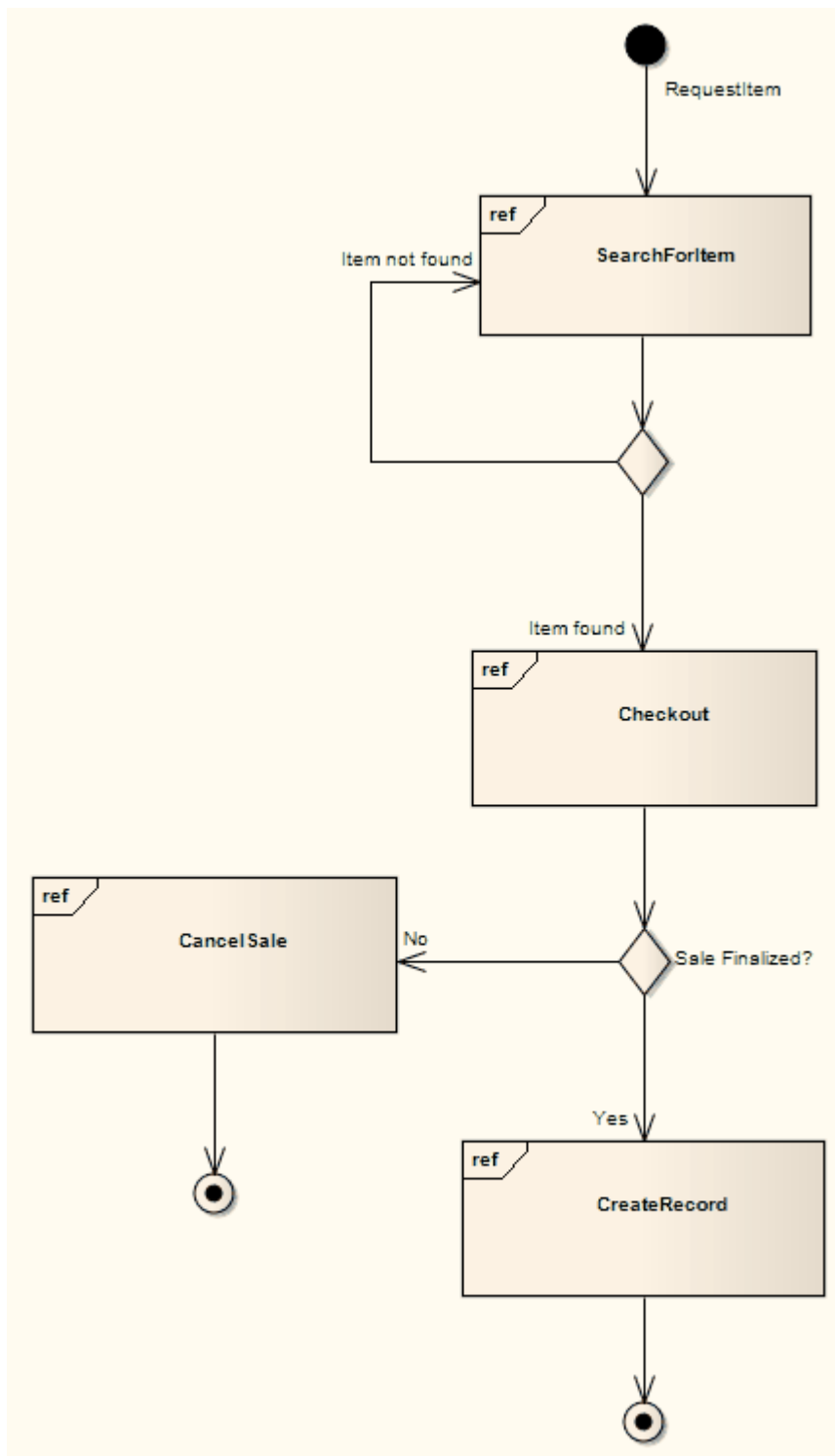
*Interaction Overview* diagrams visualize the cooperation between other interaction diagrams to illustrate a control flow serving an encompassing purpose. As *Interaction Overview* diagrams are a variant of [Activity diagrams](#)<sup>[5]</sup>, most of the diagram notation is the same, as is the process of constructing the diagram. Decision points, Forks, Joins, Start points and End points are the same. Instead of [Activity](#)<sup>[90]</sup> elements, however, rectangular elements are used. There are two types of these elements:

- *Interaction* elements display an inline *Interaction* diagram, which can be any one of the four types
- [Interaction Occurrence](#)<sup>[119]</sup> elements are references to an existing *Interaction* diagram: they are visually represented by a frame, with **ref** in the frame's title space; the diagram name is indicated in the frame contents.

To create an *Interaction Occurrence*, simply drag an *Interaction* diagram from the **Project Browser** onto your *Interaction Overview* diagram. The **ref** frame displays, encapsulating an instance of the *Interaction* diagram.

#### Example Diagram

The following example depicts a sample sale process, shown in an *Interaction Overview* diagram, with sub-processes abstracted within *Interaction Occurrences*. The diagram appears very similar to an *Activity* diagram, and is conceptualized the same way; as the flow moves into an *interaction*, the respective *interaction's* process must be followed before the *Interaction Overview's* flow can advance.



















### Toolbox Elements and Connectors

Select Interaction Overview diagram elements and connectors from the [Activity](#) pages of the Enterprise Architect UML [Toolbox](#); see *Using Enterprise Architect – UML Modeling Tool*.

**Tip:**

Click on the following elements and connectors for more information.

Interaction Overview Diagram Elements	Interaction Overview Diagram Connectors
 Partition	 Control Flow
 Decision	 Object Flow
 Send	 Interrupt Flow
 Receive	
 Synch	
 Initial	
 Final	
 Flow Final	
 Region	
 Exception	
 Merge	
 Fork/Join	
 Fork/Join	

## OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 514*) states:

*Interaction Overview Diagrams define Interactions (described in Chapter 14, "Interactions") through a variant of Activity Diagrams (described in Chapter 6, "Activities") in a way that promotes overview of the control flow.*

*Interaction Overview Diagrams focus on the overview of the flow of control where the nodes are Interactions or InteractionUses. The Lifelines and the Messages do not appear at this overview level.*

## 1.2 Structural Diagrams

*Structural* diagrams depict the structural elements composing a system or function. These diagrams reflect the static relationships of a structure, such as Class or Package diagrams, or run-time architectures such as Object or Composite Structure diagrams.

Structural diagrams include the following diagram types:

### Class Diagrams

[Class diagrams](#) <sup>[56]</sup> capture the logical structure of the system, the Classes and objects that make up the model, describing what exists and what attributes and behavior it has.

### Composite Structure Diagrams

[Composite Structure diagrams](#) <sup>[59]</sup> reflect the internal collaboration of Classes, Interfaces and Components (and their properties) to describe a functionality.

## Component Diagrams

[Component diagrams](#) [65] illustrate the pieces of software, embedded controllers and such that make up a system, and their organization and dependencies.

## Deployment Diagrams

[Deployment diagrams](#) [62] show how and where the system is to be deployed; that is, its execution architecture.

## Object Diagrams

[Object diagrams](#) [58] depict object instances of Classes and their relationships at a point in time.

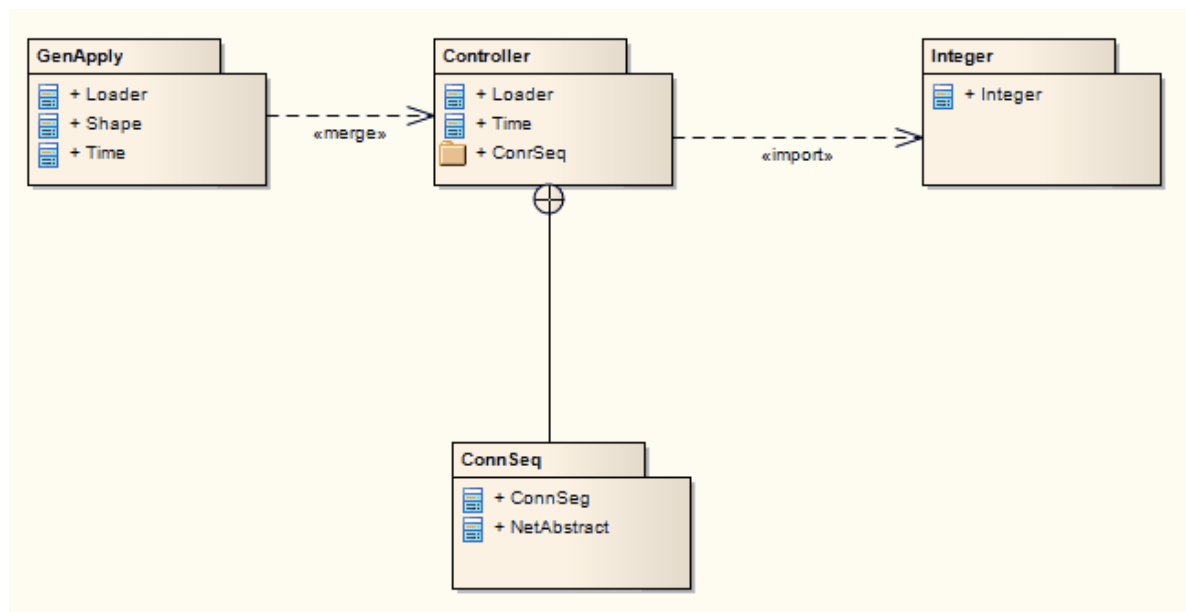
## Package Diagrams

[Package diagrams](#) [55] depict the organization of model elements into packages and the dependencies amongst them.

### 1.2.1 Package Diagram

*Package diagrams* depict the organization of model elements into packages and the dependencies amongst them, including package imports and package extensions. They also provide a visualization of the corresponding namespaces.

The following example demonstrates a basic Package diagram.



The nesting connector between *ConnSeq* and *Controller* reflects what the package contents reveal. Package contents can be listed by clicking on the diagram background to display the diagram's **Properties** dialog (see *UML Modeling with Enterprise Architect – UML Modeling Tool* for information on the **Diagram Properties** dialog), selecting the **Elements** tab and selecting the **Package Contents** checkbox.

The `«import»` connector indicates that the elements within the target *Integer* package, which in this example is the single Class *Integer*, are imported into the package *Controller*. The *Controller*'s namespace gains access to the *Integer* Class; the *Integer* namespace is not affected.

The `«merge»` connector indicates that the package *Controller*'s elements are imported into *GenApply*, including *Controller*'s nested and imported contents. If an element already exists within *GenApply*, such as *Loader* and *Time*, these elements' definitions are expanded by those included in the package *Controller*. All elements added or updated by the merge are noted by a generalization relationship back to that package.

**Notes:**


















- Private elements within a package cannot be imported or merged.
- If you click on an element listed in a package, and then double-click, you can display and edit the element properties (see the *Work With Elements* section of *UML Modeling with Enterprise Architect – UML Modeling Tool*).

**Toolbox Elements and Connectors**

Select Package diagram elements and connectors from the **Class** pages of the Enterprise Architect UML **Toolbox**; see *Using Enterprise Architect – UML Modeling Tool*.

**Tip:**

Click on the following elements and connectors for more information.

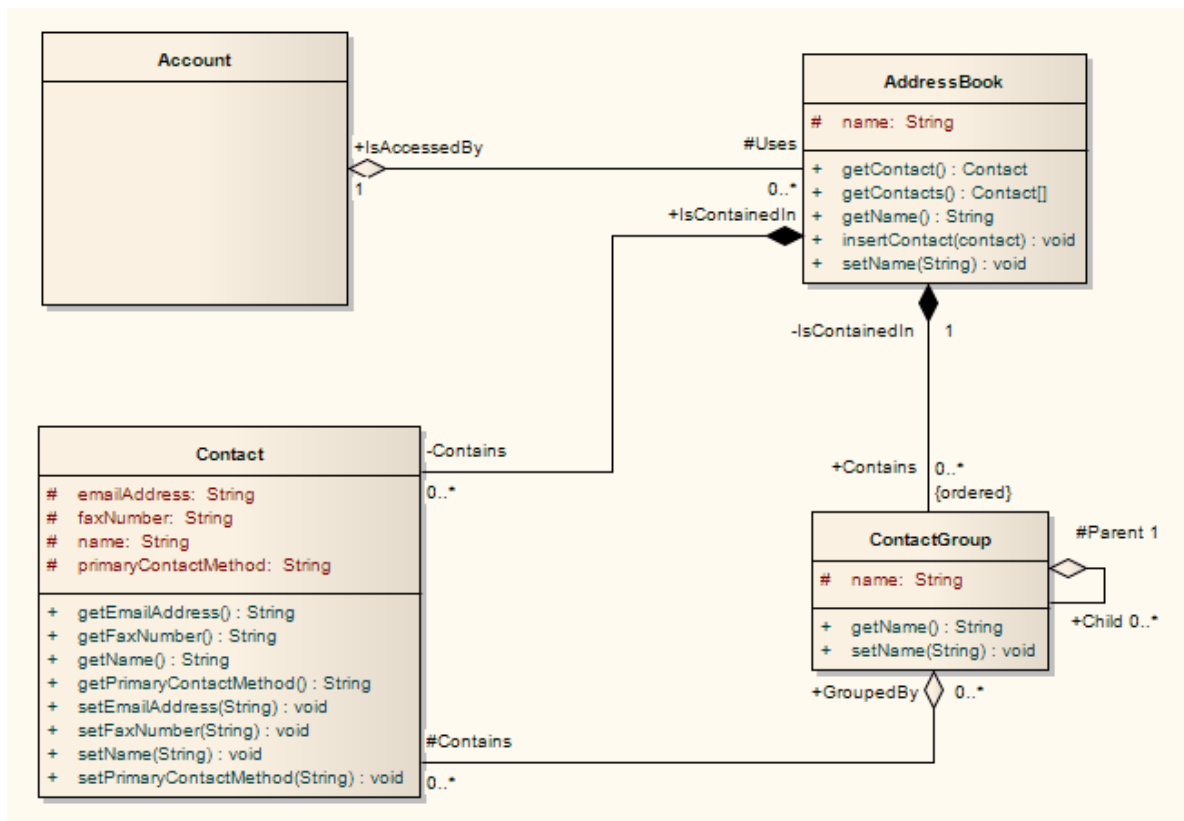
Package Diagram Elements	Package Diagram Connectors
 Package	 Associate
 Class	 Generalize
 Interface	 Compose
 Table	 Aggregate
 Enumeration	 Association Class
 Signal	 Assembly
 Association	 Realize
	 Nesting
	 Package Merge
	 Package Import

**1.2.2 Class Diagram**

The *Class* diagram captures the logical structure of the system: the [Classes](#)<sup>[152]</sup> - including [Active](#)<sup>[153]</sup> and [Parameterized](#)<sup>[154]</sup> (template) Classes - and things that make up the model. It is a static model, describing what exists and what attributes and behavior it has, rather than how something is done. Class diagrams are most useful to illustrate relationships between Classes and Interfaces. Generalizations, Aggregations and Associations are all valuable in reflecting inheritance, composition or usage, and connections, respectively.

**Example Diagram**

The pale [Aggregation](#)<sup>[198]</sup> relationship indicates that the Class *Account* uses *AddressBook*, but does not necessarily contain *AddressBook*. The dark [Composite Aggregation](#)<sup>[198]</sup> connectors indicate ownership or containment by the target Classes (at the diamond end) of the source Classes.



### Toolbox Elements and Connectors

Select Class diagram elements and connectors from the [Class](#) pages of the Enterprise Architect UML [Toolbox](#); see *Using Enterprise Architect – UML Modeling Tool*.






Enterprise Architect also supports a number of [stereotyped Class](#) <sup>[75]</sup> elements to represent various entities in web page modeling.

#### Tip:

Click on the following elements and connectors for more information.

Class Diagram Elements	Class Diagram Connectors
Package	Associate
Class	Generalize
Interface	Compose
Data Type	Aggregate
Enumeration	Association Class
Primitive	Assembly
Table	Realize



Class Diagram Elements	Class Diagram Connectors
 Signal	 Nesting
 Association	 Package Merge
	 Package Import

### 1.2.3 Object Diagram

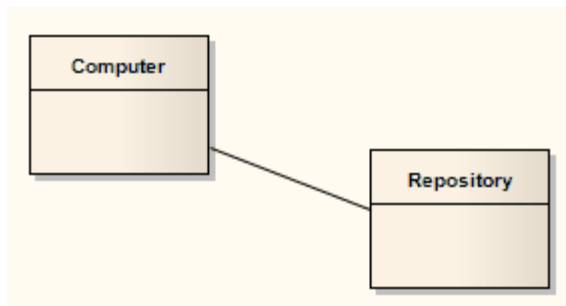
An *Object diagram* is closely related to a [Class diagram](#)<sup>[56]</sup>, with the distinction that it depicts object instances of Classes and their relationships at a point in time. This might appear similar to a [Composite Structure](#)<sup>[59]</sup> diagram, which also models run-time behavior; the difference is that Object diagrams exemplify the static Class diagrams, whereas Composite Structure diagrams reflect run-time architectures different from their static counterparts. Object diagrams do not reveal architectures varying from their corresponding Class diagrams, but reflect multiplicity and the roles instantiated Classes could serve. They are useful in understanding a complex Class diagram, by creating different cases in which the relationships and Classes are applied. An Object diagram can also be a kind of [Communication diagram](#)<sup>[49]</sup>, which also models the connections between objects, but additionally sequences events along each path.

#### Note:

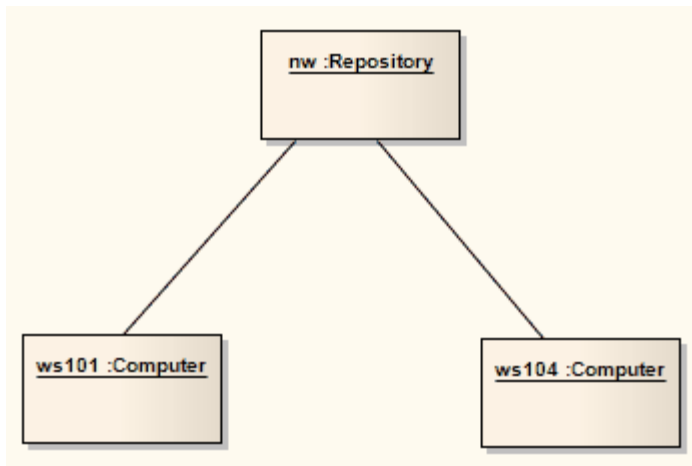
Communication diagrams were known as Collaboration diagrams in UML 1.4.

### Example Diagram

The following example first shows a simple Class diagram, with two [Class](#)<sup>[152]</sup> elements connected.



The Classes above are instantiated below as Objects in an Object diagram. There are two instances of *Computer* in this model, which can prove useful for considering the relationships and interactions Classes play in practice, as Objects.













### Toolbox Elements and Connectors

Select Object diagram elements and connectors from the **Object** pages of the Enterprise Architect UML **Toolbox**; see *Using Enterprise Architect – UML Modeling Tool*.

Enterprise Architect also supports a number of [stereotyped Object](#)<sup>[75]</sup> elements to represent various entities in business modeling.

#### Tip:

Click on the following elements and connectors for more information.

Object Diagram Elements	Object Diagram Connectors
 Actor	 Information Flow
 Object	 Associate
 Collaboration	 Dependency
 Information Item	
 Boundary	
 Control	
 Entity	

### 1.2.4 Composite Structure Diagram

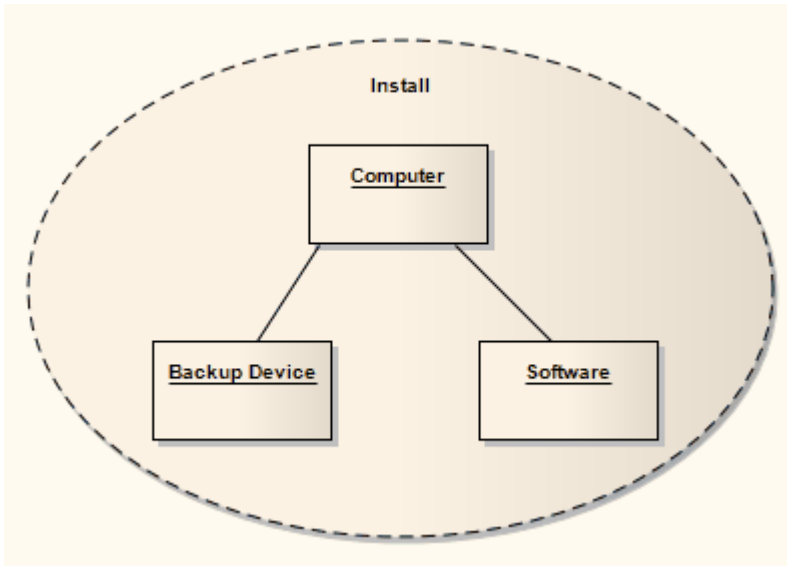
A *Composite Structure* diagram reflects the internal collaboration of [Classes](#)<sup>[152]</sup>, [Interfaces](#)<sup>[164]</sup> or [Components](#)<sup>[158]</sup> (and their [Properties](#)<sup>[61]</sup>) to describe a functionality. Composite Structure diagrams are similar to [Class diagrams](#)<sup>[56]</sup>, except that they model a specific usage of the structure. Class diagrams model a static view of Class structures, including their attributes and behaviors. A Composite Structure diagram is used to express run-time architectures, usage patterns and the participating elements' relationships, which might not be reflected by static diagrams.

In a Composite Structure diagram, Classes are accessed as [Parts](#)<sup>[168]</sup> or run-time instances fulfilling a particular role. These Parts can have multiplicity, if the role filled by the Class requires multiple instances. [Ports](#)<sup>[169]</sup> defined by a Part's Class should be represented in the composite structure, maintaining that all

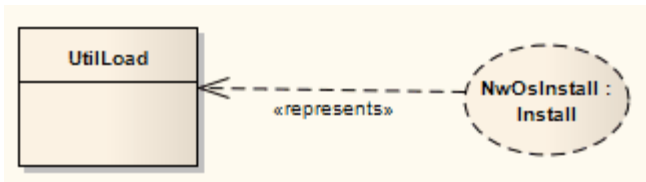
connecting Parts provide the required interfaces specified by the Port. There is extensive flexibility, and an ensuing complexity, that come with modeling composite structures. To optimize your modeling, consider building [Collaborations](#)<sup>[156]</sup> to represent reusable patterns responding to your design issues.

### Example Diagram

The following diagram shows a Collaboration used in Composite Structure diagrams to model common patterns. This particular example shows a relationship for performing an installation.



The following diagram uses the *Install* Collaboration in a [Collaboration Occurrence](#)<sup>[157]</sup>, and applies it to the *UtilLoad* Class via a «represents» relationship. This indicates that the classifier UtilLoad uses the collaboration pattern within its implementation.



For further examples of Composite Structure diagrams, see the [Toolbox](#) elements listed below.

### Toolbox Elements and Connectors









Select Composite Structure diagram elements and connectors from the [Composite](#) pages of the Enterprise Architect UML [Toolbox](#); see *Using Enterprise Architect – UML Modeling Tool*.

Enterprise Architect also supports a stereotyped Collaboration to represent a [Business Use Case Realization](#)<sup>[75]</sup> in business modeling.

**Tip:**

Click on the following elements and connectors for more information.

Composite Structure Diagram Elements	Composite Structure Diagram Connectors
Class	Connector
Interface	Assembly

Composite Structure Diagram Elements	Composite Structure Diagram Connectors
 Part	 Role Binding
 Port	 Represents
 Collaboration	 Occurrence
 Expose Interface	 Delegate

## OMG UML Specification

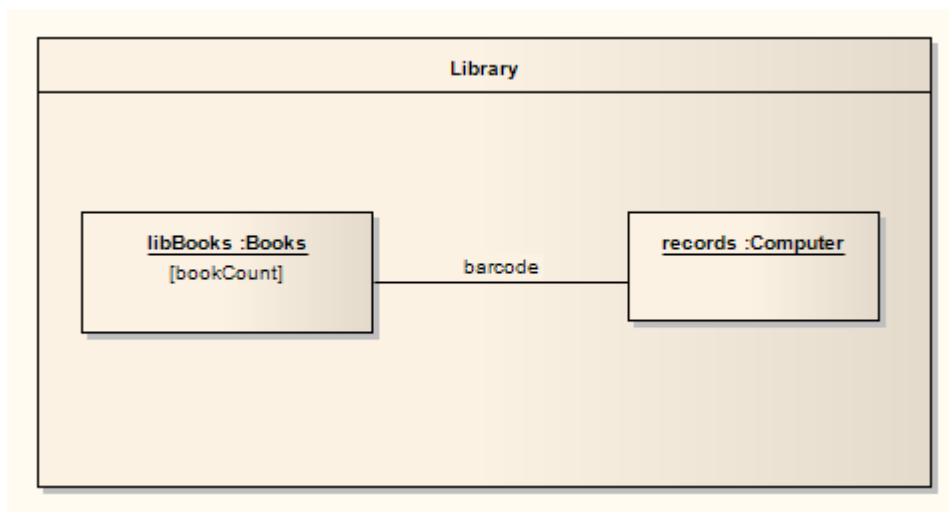
The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 193*) states:

*A composite structure diagram depicts the internal structure of a classifier, as well as the use of a collaboration in a collaboration use.*

### 1.2.4.1 Properties

A *property* is a nested structure within a classifier, which is usually a [Class](#)<sup>[152]</sup> or an [Interface](#)<sup>[164]</sup> on a [Composite Structure diagram](#)<sup>[59]</sup>. The contained structure reflects instances and relationships reflected within the containing classifier. Properties can have multiplicity.

To demonstrate properties, consider the following diagram, which demonstrates some properties of the *Library* Class.



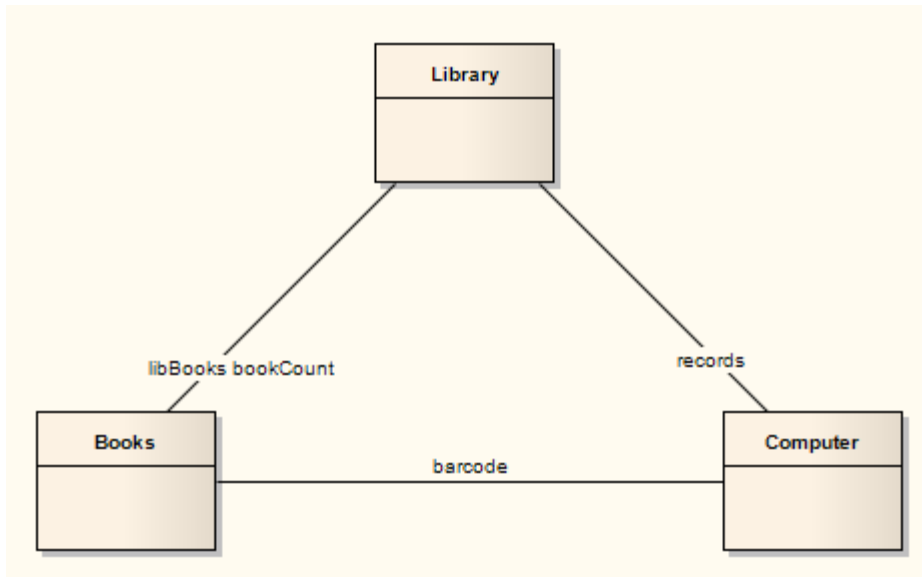
There are two [Parts](#)<sup>[168]</sup>, *libBooks* and *records*, which are instances corresponding to the Classes *Books* and *Computer* respectively. After dragging Parts from the Enterprise Architect UML **Toolbox** out to the workspace, right-click on a Part and select the **Advanced | Set Property Type** context menu option to connect to a classifier.

#### Note:

If Parts disappear when dragged onto the Class, adjust the Z-order of the Class (right-click on it and select the **Z-Order** context menu option).

The relationship between the two Parts is indicated by the connector, reflecting that communication between the Parts is via the *barcode*. This contained structure and its Parts are properties owned by the Library Class. To indicate a property that is not owned by composition to the containing classifier, use a box symbol with a dashed outline, indicating *association*. To do this, right-click on the Part and select the **Advanced | Custom Properties** context menu option. Set the **IsReference** option to **true**.

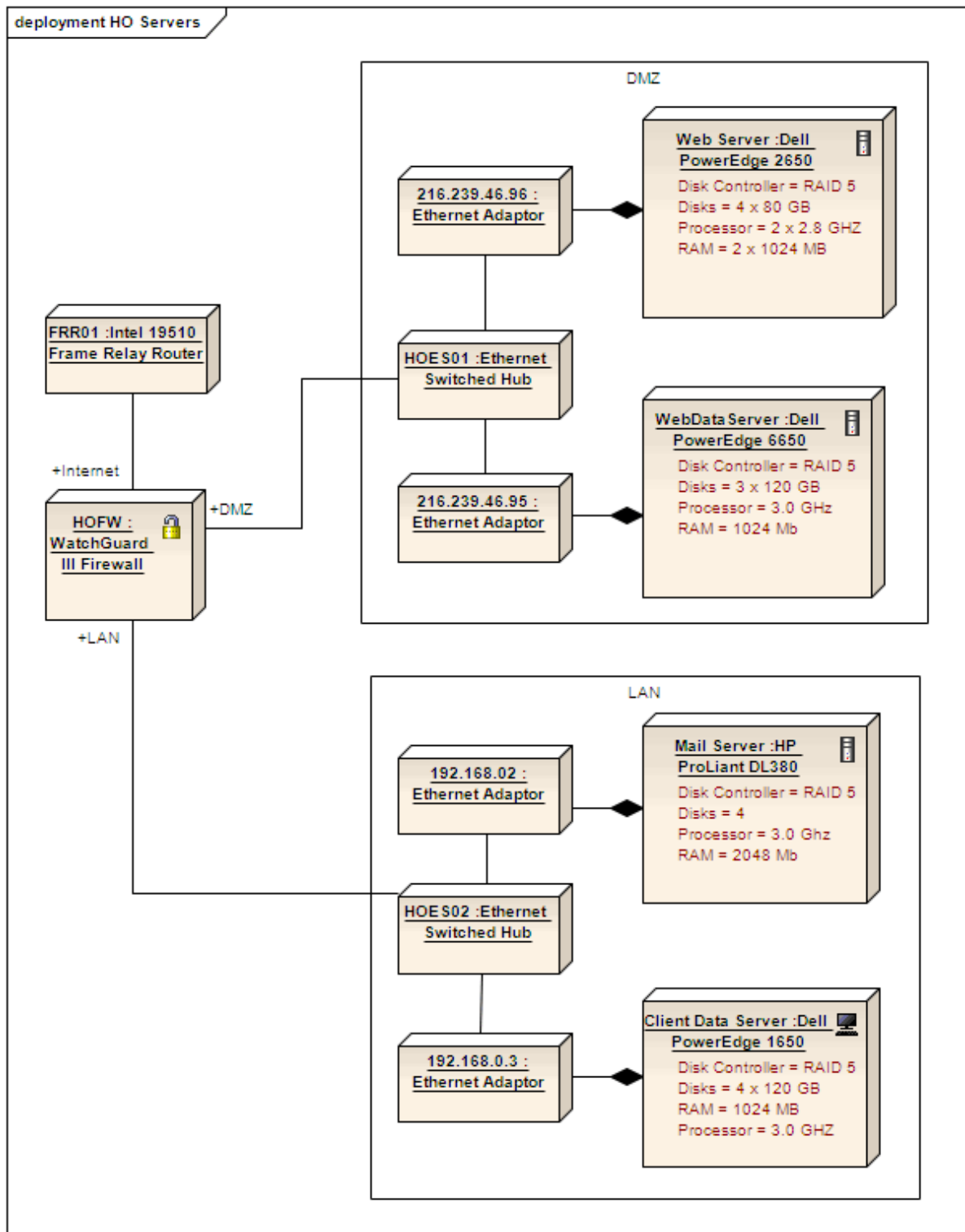
Properties can also be reflected using a normal composite structure (without containing it in a Class), with the appropriate connectors, parts and relationships indicated through connections to the Class. This alternative representation is shown in the following diagram. However, this depiction fails to express the ownership immediately reflected by containing properties within a classifier.



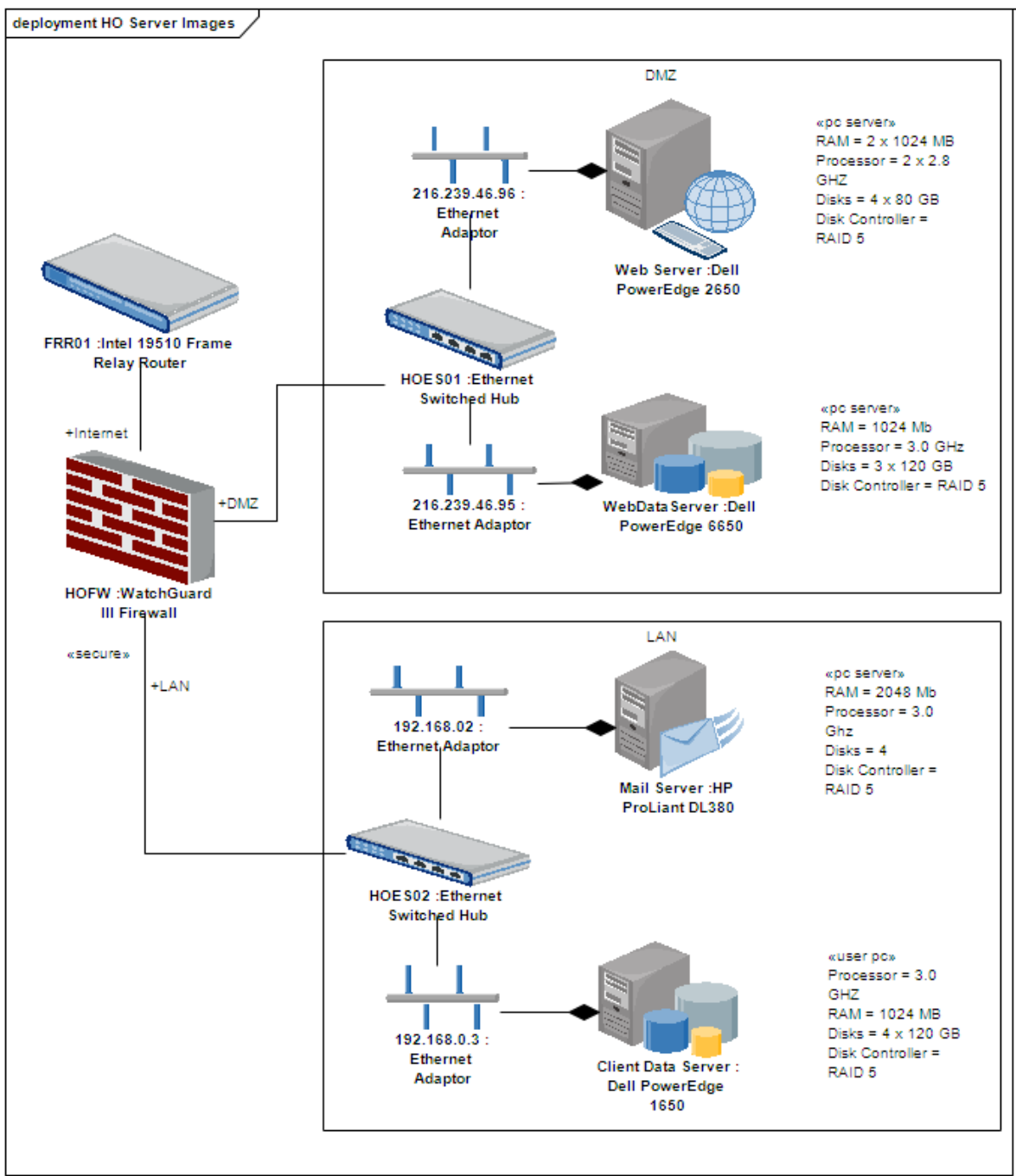
### 1.2.5 Deployment Diagram

A *Deployment diagram* shows how and where the system is to be deployed; that is, its execution architecture. Hardware devices, processors and software execution environments (system [Artifacts](#)<sup>[15†]</sup>) are reflected as [Nodes](#)<sup>[16†]</sup>, and the internal construction can be depicted by embedding or nesting Nodes. [Deployment](#)<sup>[20†]</sup> relationships indicate the deployment of Artifacts, and [Manifest](#)<sup>[21†]</sup> relationships reveal the physical implementation of components. As Artifacts are allocated to Nodes to model the system's deployment, the allocation is guided by the use of *deployment specifications*.

A simple Deployment diagram is shown below, representing the arrangement of servers at a head office. The servers are represented by Nodes linked by either simple or aggregate Association relationships.




















Deployment diagrams are ideal for using alternative images for the objects that the elements represent; see *Modeling With Enterprise Architect – UML Modeling Tool*. Such images can be substituted for the elements in the above diagram, as shown below:



### Toolbox Elements and Connectors

Select Deployment diagram elements and connectors from the [Deployment](#) pages of the Enterprise Architect UML [Toolbox](#); see *Using Enterprise Architect – UML Modeling Tool*.

**Tip:**  
Click on the following elements and connectors for more information.

Deployment Diagram Elements	Deployment Diagram Connectors
 Node	 Associate
 Device	 Communication Path
 Execution Environment	 Association Class
 Component	 Generalize
 Interface	 Realize
 Artifact	 Deployment
 Document Artifact	 Manifest
 Deployment Specification	 Nesting
 Package	

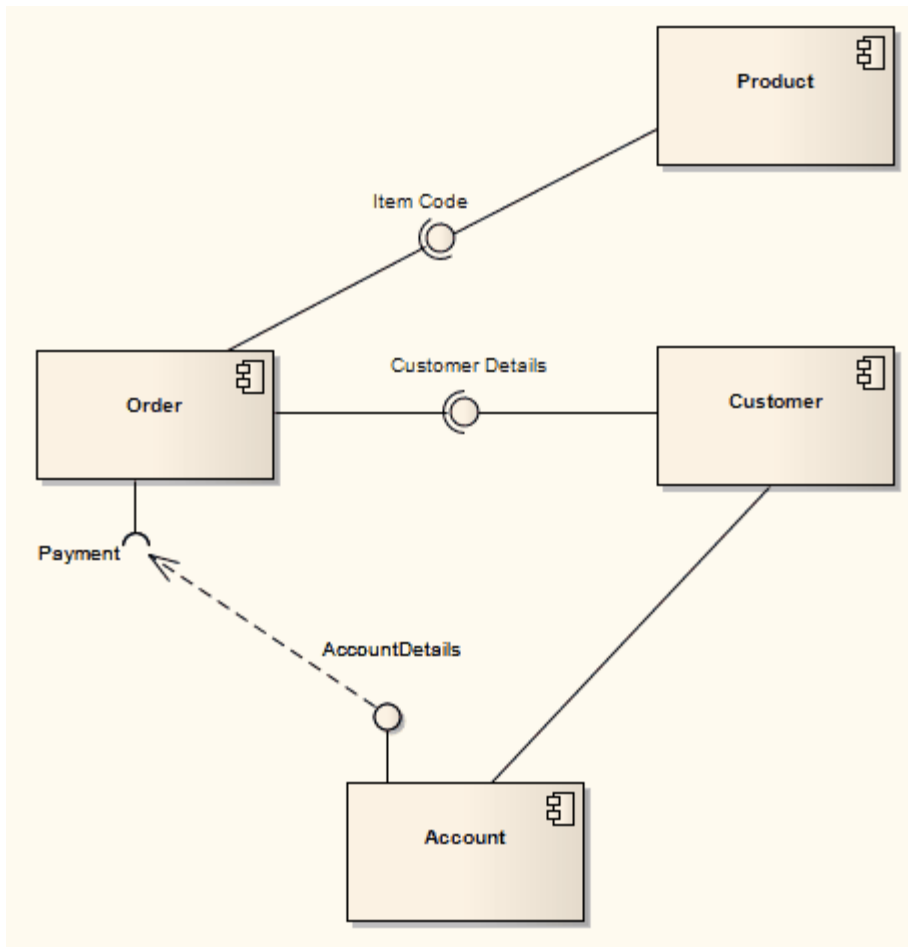
### 1.2.6 Component Diagram

A *Component* diagram illustrates the pieces of software, embedded controllers and such that make up a system, and their organization and dependencies. A Component diagram has a higher level of abstraction than a [Class diagram](#)<sup>[56]</sup>; usually a component is implemented by one or more [Classes](#)<sup>[152]</sup> (or [Objects](#)<sup>[165]</sup>) at runtime. They are building blocks, built up so that eventually a component can encompass a large portion of a system.

#### Example Diagram

The following diagram demonstrates some components and their inter-relationships. [Assembly](#)<sup>[199]</sup> connectors connect the provided interfaces supplied by *Product* and *Customer* to the required interfaces specified by *Order*. A [Dependency](#)<sup>[205]</sup> relationship maps a customer's associated account details to the required interface *Payment*, indicated by *Order*.









### Toolbox Elements and Connectors

Select Component diagram elements and connectors from the **Component** pages of the Enterprise Architect UML **Toolbox**; see *Using Enterprise Architect – UML Modeling Tool*.

**Tip:**

Click on the following elements and connectors for more information.

Component Diagram Elements	Component Diagram Connectors
Package	Assembly
Packaging Component	Delegate
Component	Associate
Class	Realize
Interface	Generalize
Object	

Component Diagram Elements	Component Diagram Connectors
 Port	
 Expose Interface	
 Artifact	
 Document Artifact	

### 1.3 Extended Diagrams

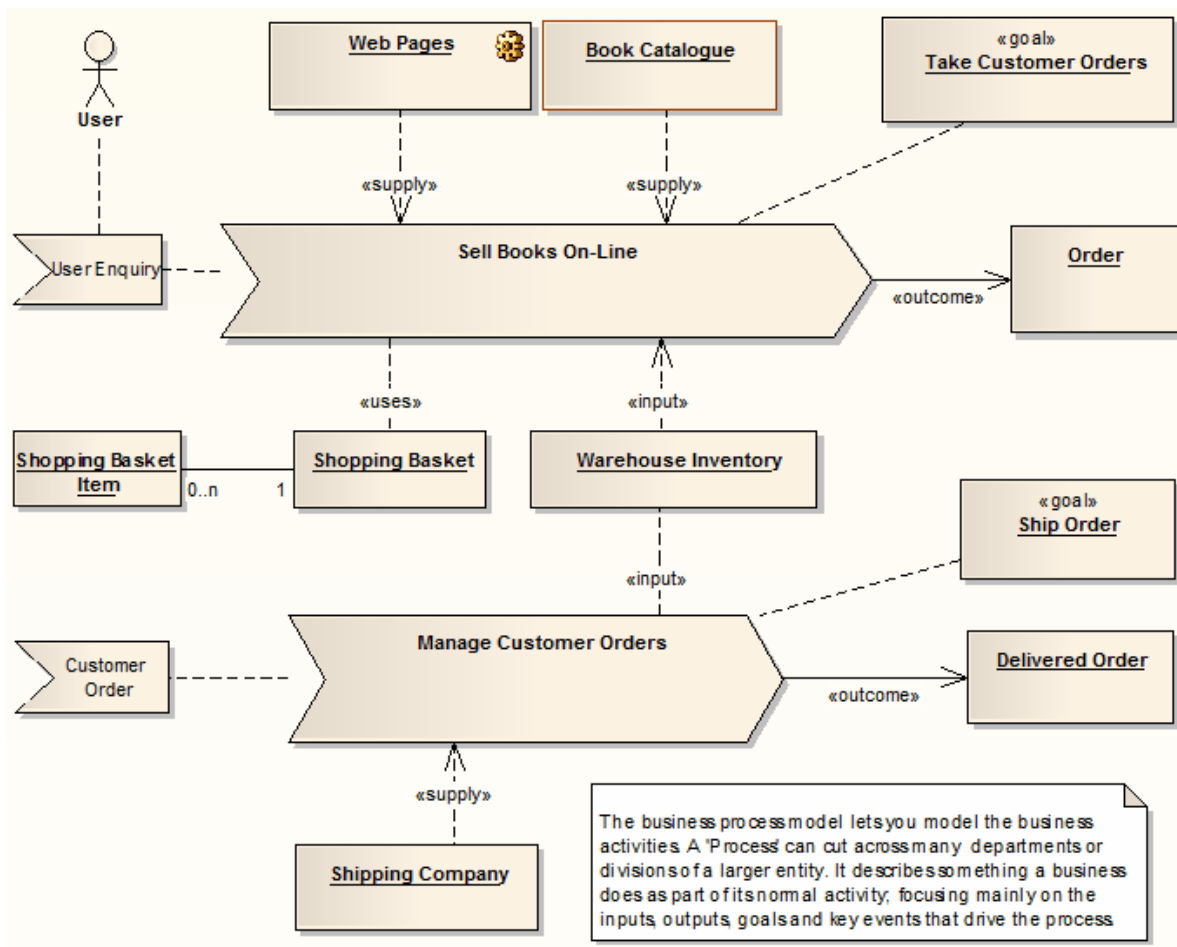
In addition to diagrams defined by the UML, Enterprise Architect provides some extended diagram platforms to model business processes or develop custom diagrams.

- [Analysis Diagram](#) <sup>[67]</sup>
- [Custom Diagram](#) <sup>[69]</sup>
- [Requirements Diagram](#) <sup>[71]</sup>
- [Maintenance Diagram](#) <sup>[72]</sup>
- [User Interface Diagram](#) <sup>[73]</sup>
- [Database Schema](#) <sup>[75]</sup>
- Documentation (see *Virtual Documents in Report Creation in UML Models*)
- [Business Modeling and Business Interaction](#) <sup>[75]</sup>

#### 1.3.1 Analysis Diagram

An *Analysis diagram* is a simplified [Activity diagram](#) <sup>[5]</sup>, which is used to capture high level business processes and early models of system behavior and elements. It is less formal than some other diagrams, but provides a good means of capturing the essential business characteristics and requirements.

Enterprise Architect supports some of the *Eriksson-Penker Business Extensions* that facilitate business process modeling; see *Extending UML With Enterprise Architect*. The complete Eriksson-Penker Business Extensions UML Profile can also be loaded into Enterprise Architect and used to create detailed process models.



Robustness diagrams, used extensively in ICONIX, can be created as Analysis diagrams.









### Toolbox Elements and Connectors

Select Analysis diagram elements and connectors from the [Analysis](#) pages of the Enterprise Architect UML [Toolbox](#); see *Using Enterprise Architect – UML Modeling Tool*.

**Tip:**

Click on the following elements and connectors for more information. The *Information* element is a simple flow-chart representation of data or input/output.

Analysis Diagram Elements	Analysis Diagram Connectors
Actor	Information Flow
Object	Object Flow
Process	Associate
Collaboration	Realize
Send	Representation

Analysis Diagram Elements	Analysis Diagram Connectors
 Receive	
 Information	
 Information Item	
 Decision	
 Merge	
 Boundary	
 Control	
 Entity	

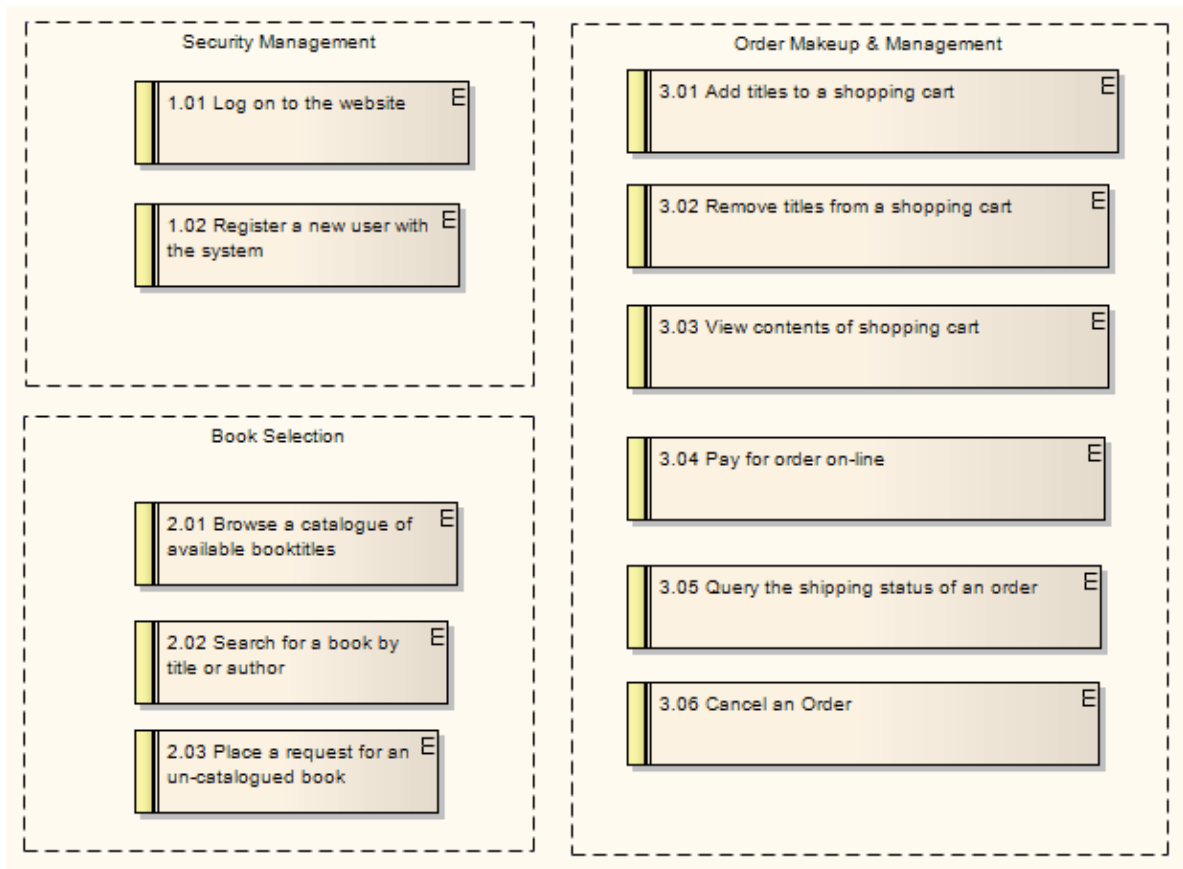
### 1.3.2 Custom Diagram

A *Custom* diagram is an extended [Class diagram](#)<sup>[56]</sup> that is used to capture requirements, user interfaces or custom-design models.

The below example reflects a [Requirements diagram](#)<sup>[71]</sup>. [Requirement elements](#)<sup>[189]</sup> can be linked back to [Use Cases](#)<sup>[146]</sup> and [Components](#)<sup>[158]</sup> in the system to illustrate how a particular system requirement is met. Change and Defect (Issue) elements (see *Project Management with Enterprise Architect*) look the same as Requirement elements and can be coded and managed in the same way.

Screen design is supported through a stereotyped [Screen](#)<sup>[190]</sup> element and [UI Controls](#)<sup>[192]</sup>. Use this model to design high level system prototypes.

Custom models provide a few extensions to the UML model and enable some exploratory and non-rigorous experimentation with model elements and diagrams.




### Toolbox Elements and Connectors

Select Custom diagram elements and connectors from the **Custom** pages of the Enterprise Architect UML **Toolbox**; see *Using Enterprise Architect – UML Modeling Tool*. Issue and Change elements are discussed in *Project Management With Enterprise Architect*.

**Tip:**

Click on the following elements and connectors for more information.

Custom Diagram Elements	Custom Diagram Connectors
Package	Associate
Requirement	Aggregate
Issue	Generalize
Change	Realize
Screen	Nesting
UI Control	
Test Case	

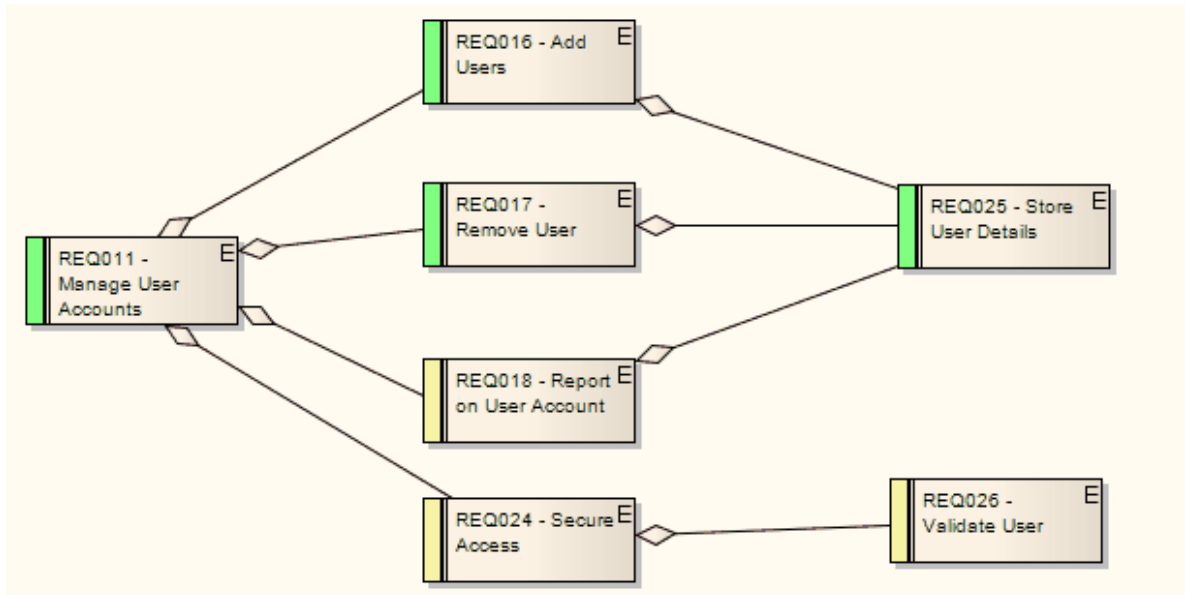
Custom Diagram Elements	Custom Diagram Connectors
 Entity	

### 1.3.3 Requirements Diagram

A *Requirements* diagram is a custom diagram used to describe a system's requirements or features as a visual model.

Requirements are defined using [Requirement elements](#)<sup>[189]</sup> (*Custom* elements of type *Requirement*). To view the detailed description of a Requirement, double-click on the element to display its properties. Requirement elements can be linked back to [Use Cases](#)<sup>[146]</sup> and [Components](#)<sup>[158]</sup> in the system to illustrate how a particular system requirement is met.

Requirements models provide extensions to the UML model and enable traceability between specifications and design requirements, and the model elements that realize them. (See *UML Model Management*.)







Requirements can have relationships with other elements such as other Requirements and Use Cases. To view the traceability of a requirement, use the **Traceability** window (see *Using Enterprise Architect - UML Modeling Tool*), which you access using the **View | Traceability** menu option (or press **[Ctrl]+[Shift]+[4]**).





### Toolbox Elements and Connectors

Select Requirements diagram elements and connectors from the **Requirements** pages of the Enterprise Architect UML **Toolbox**; see *Using Enterprise Architect – UML Modeling Tool*.

#### Tip:

Click on the following elements and connectors for more information.

Requirements Diagram Elements	Requirements Diagram Connectors
 Package	 Aggregate
 Requirement	 Inheritance

Requirements Diagram Elements	Requirements Diagram Connectors
 Feature	 Associate
 Object	 Implements

### 1.3.4 Maintenance Diagram

A *Maintenance diagram* is a custom diagram used to describe change requests and issue items within a system model.

An example Maintenance diagram is shown below. *Change*, *Task* and *Issue* elements can be linked back to other model elements in the system to illustrate how they must be modified, fixed or updated.

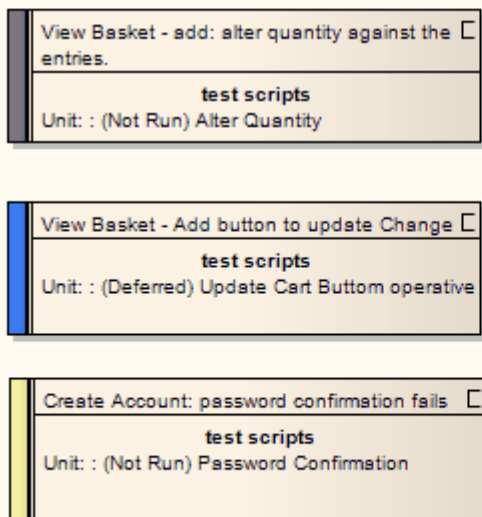
Maintenance models provide extensions to the UML model and enable change management of change items, and of the model elements that require the changes to be made to them.

## Changes

EA supports custom Elements of type 'Change'. These can be linked to other elements in the repository or used as a separate lists of any changes proposed for the model.

Below are a set of Change elements for the shopping basket with test definitions listed against them. They are ready for checking once they are confirmed as corrected.

The color markings reflect the Status of the element.









### Toolbox Elements and Connectors

Select Maintenance diagram elements and connectors from the [Maintenance](#) pages of the Enterprise Architect UML [Toolbox](#); see *Using Enterprise Architect – UML Modeling Tool*. Issue and Change elements are discussed in *Project Management With Enterprise Architect*.

**Tip:**

Click on the following elements and connectors for more information.

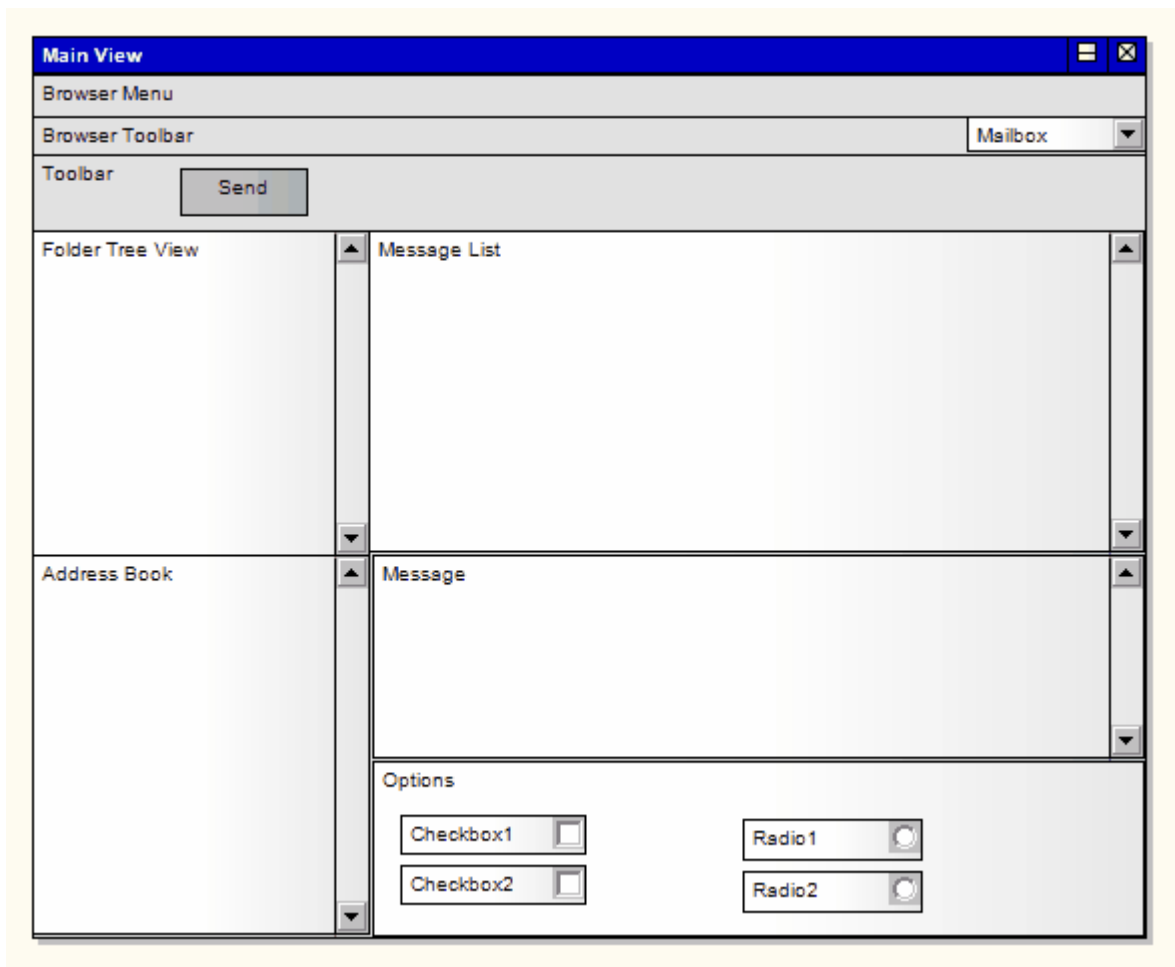
Maintenance Diagram Elements	Maintenance Diagram Connectors
 Package	 Aggregate
 Issue	
 Change	
 Test Case	
 Entity	

### 1.3.5 User Interface Diagram

*User Interface Diagrams* are custom diagrams used to visually mock-up a system's user interface using forms, controls and labels.

In the example *User Interface* diagram below, forms, controls and labels are arranged on the diagram to describe its appearance. [UI elements](#)<sup>[192]</sup> can also be traced to other model elements linking the UI with the underlying implementation.













## Toolbox Elements and Connectors

Select User Interface diagram elements and connectors from the [User Interface](#) pages of the Enterprise Architect UML [Toolbox](#); see *Using Enterprise Architect – UML Modeling Tool*.

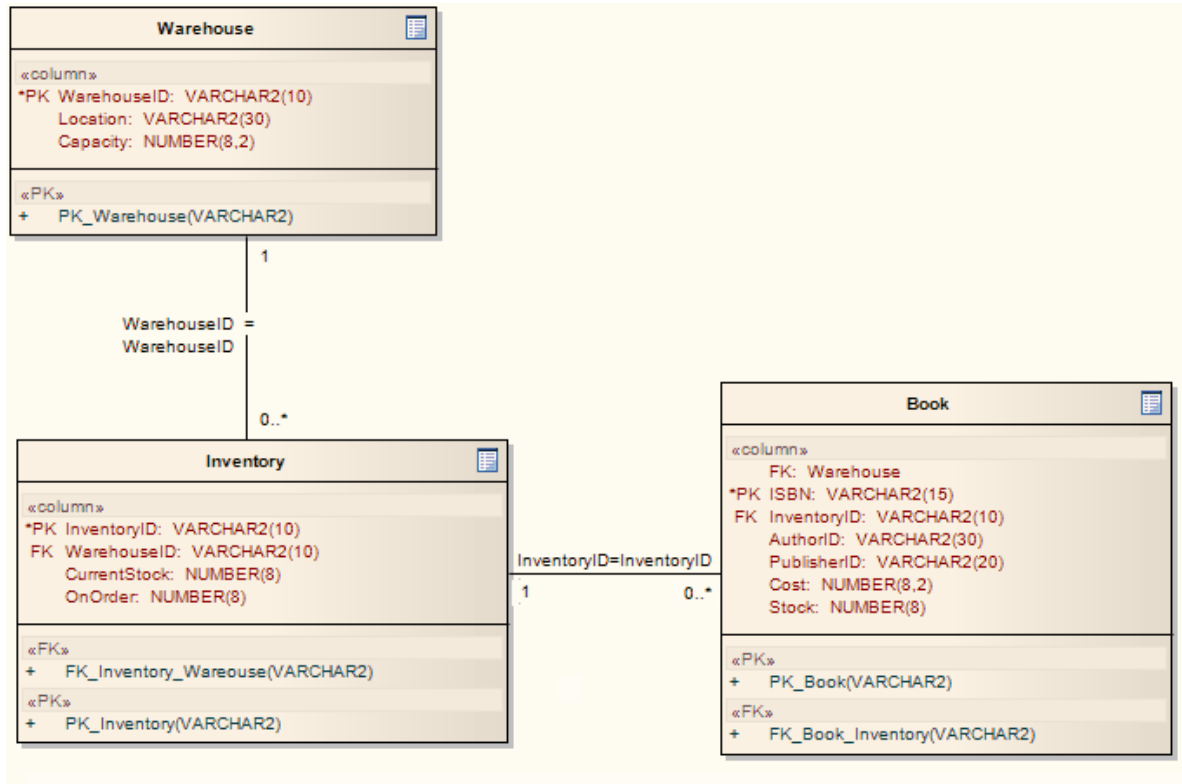
### Note:

Click on the following elements and connectors for more information.

User Interface Diagram Elements	User Interface Diagram Connectors
 Package	 Associate
 Screen	 Aggregate
 UI Control	 Generalize
 Object	 Realize

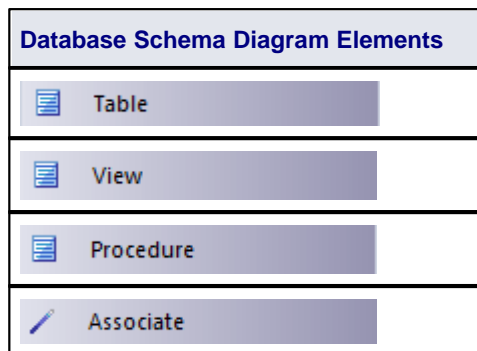
### 1.3.6 Database Schema

The following diagram shows an example Database Schema, used in Data Modeling. (See *Code Engineering Using UML Models*.)



### Toolbox Elements and Connectors

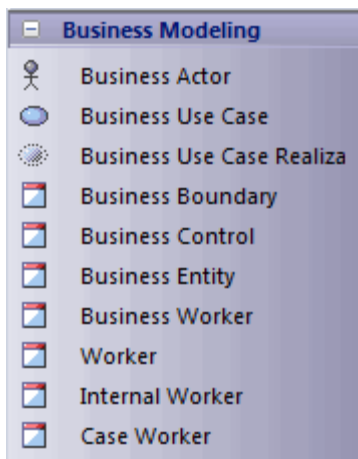
Select Database Schema diagram elements from the **Data Modeling** pages of the Enterprise Architect UML **Toolbox**; see *Using Enterprise Architect – UML Modeling Tool* and *Code Engineering Using UML Models*.



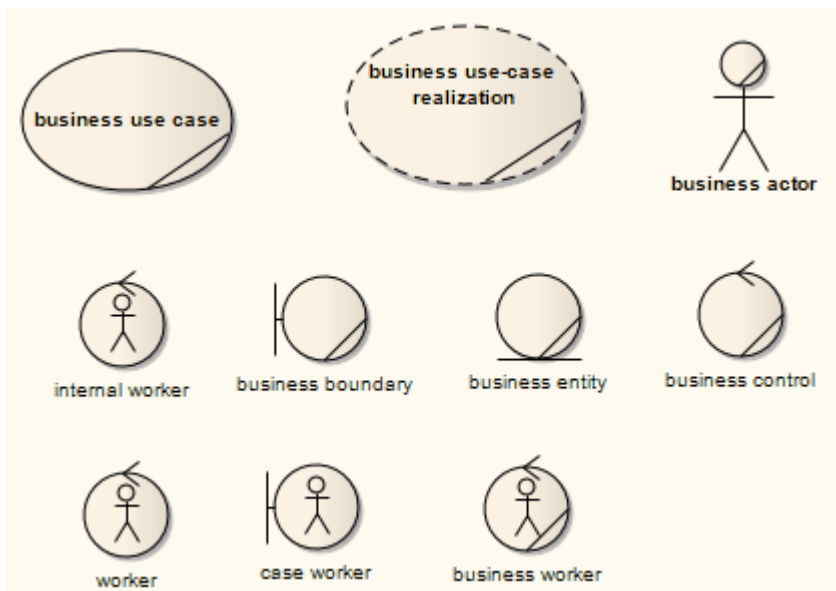
### 1.3.7 Business Modeling/Interaction

*Business Modeling* diagrams and *Business Interaction* diagrams enable you to model both the structure and behavior of a business system.

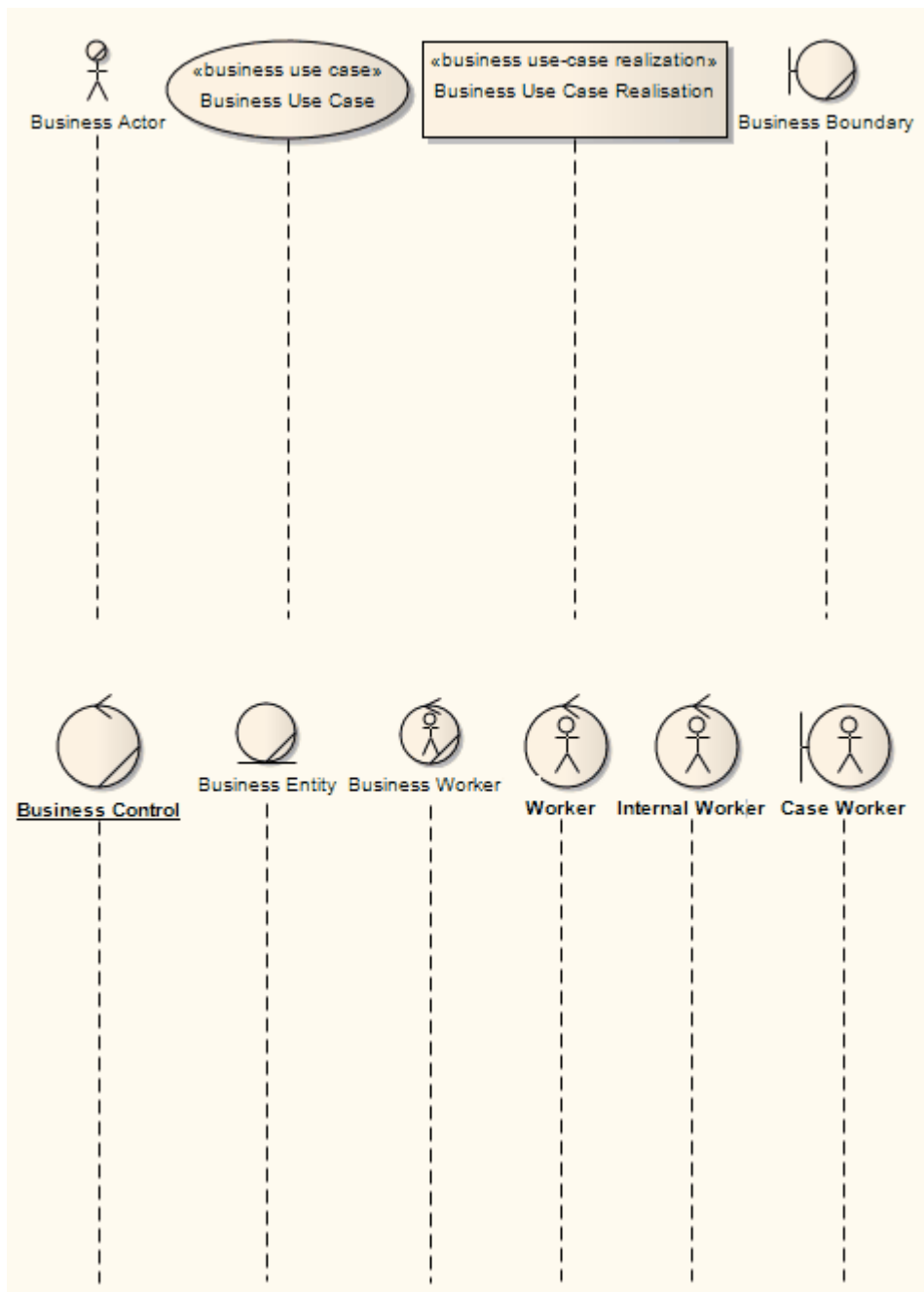
Business Modeling diagrams are based on a Class (UML Structural) diagram, whilst Business Interaction diagrams are based on a Sequence (UML Behavioral) diagram. Both diagram types have the same default Enterprise Architect UML **Toolbox**, which consists of a **Business Modeling** element page. The available elements include stereotyped **Objects**<sup>[165]</sup>, and a stereotyped **Actor**<sup>[94]</sup> (*Business Actor*), **Use Case**<sup>[146]</sup> (*Business Use Case*) and **Collaboration**<sup>[156]</sup> (*Business Use Case Realization*).



The following diagram shows the appearance of the elements when dragged and dropped onto a Business Modeling diagram:



The following diagram shows the appearance of the elements when dragged and dropped onto a Business Interaction diagram:



## 2 UML Elements



Models in UML are constructed from elements such as [Classes](#)<sup>[152]</sup>, [Objects](#)<sup>[165]</sup>, [Interfaces](#)<sup>[164]</sup>, [Use Cases](#)<sup>[146]</sup>, [Components](#)<sup>[158]</sup> and [Nodes](#)<sup>[165]</sup>, each of which has a different purpose, different rules and different notation. Model elements are used at different stages of the design process for different purposes.

This topic provides an introduction to elements defined by UML, which together compose the backbone of modeling. Most conceivable modeling elements are stereotypes or extensions of the elements introduced in this topic.

- During early analysis, Use Cases, Activities, Business Processes, Objects and Collaborations are used to capture the problem domain
- During elaboration, Sequence diagrams, Objects, Classes and State Machines are used to refine the system specification
- Components and Nodes are used to model larger parts of the system as well as the physical entities that are created and deployed into a production environment.

UML elements can be divided into two categories: those used on [Behavioral Diagrams](#)<sup>[78]</sup> and those used on [Structural Diagrams](#)<sup>[150]</sup>. This basic set can be [extended](#)<sup>[178]</sup> almost without limit using Stereotypes and UML Profiles (see *Extending UML With Enterprise Architect*).

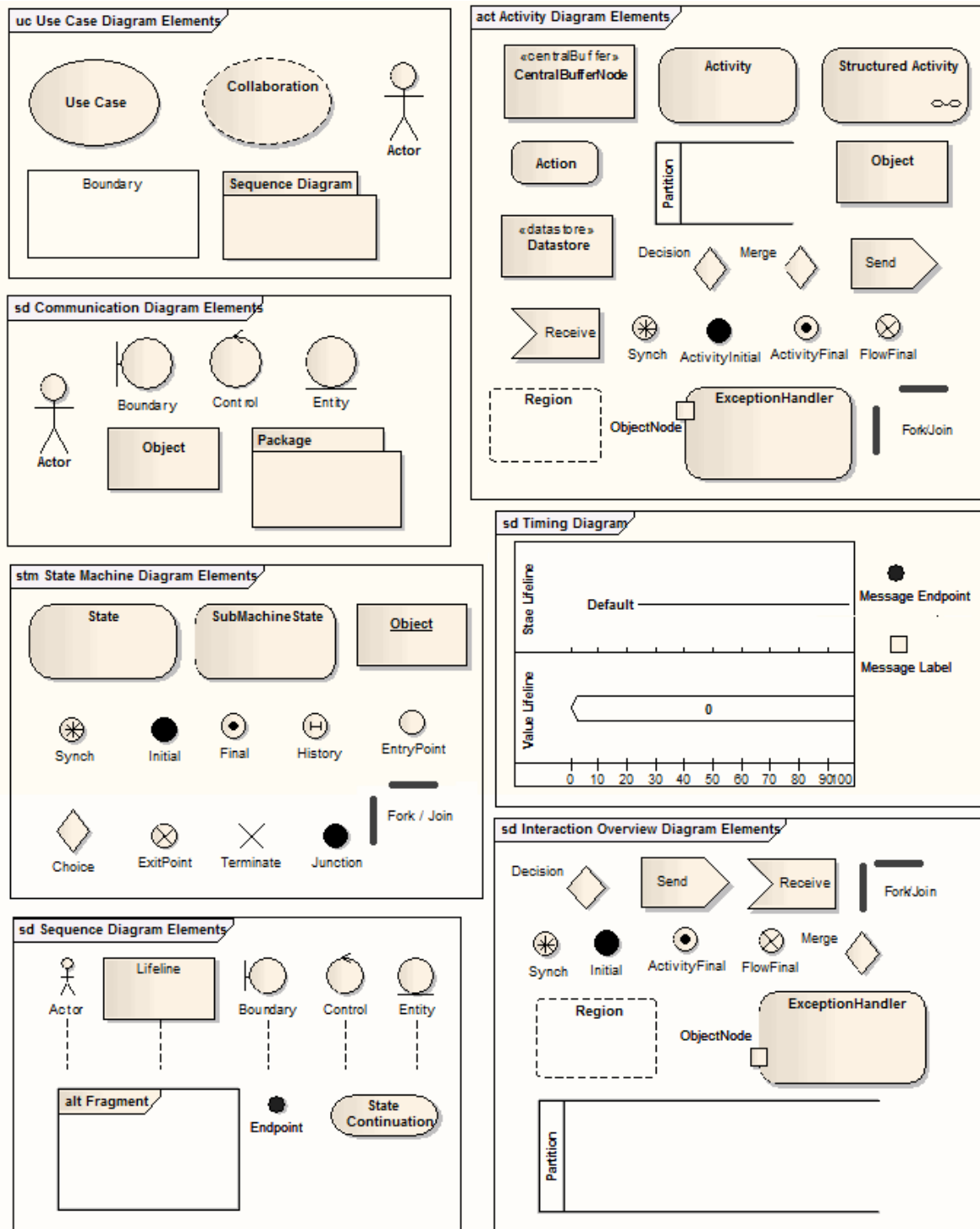
### 2.1 Behavioral Diagram Elements

The following figure illustrates the main [UML elements](#)<sup>[78]</sup> that are used in [Behavioral Diagrams](#)<sup>[4]</sup>. For more information on using each element, click on the element name in this list:

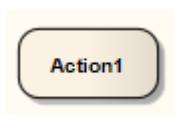
- [Action](#)<sup>[79]</sup>, [Activity](#)<sup>[90]</sup>, [Actor](#)<sup>[94]</sup>
- [Central Buffer Node](#)<sup>[95]</sup>, [Choice](#)<sup>[95]</sup>, [Collaboration](#)<sup>[156]</sup>, [Combined Fragment](#)<sup>[96]</sup>
- [Datastore](#)<sup>[102]</sup>, [Decision](#)<sup>[102]</sup>, [Diagram Frame](#)<sup>[104]</sup>, [Diagram Gate](#)<sup>[105]</sup>
- [Endpoint](#)<sup>[106]</sup>, [Entry Point](#)<sup>[107]</sup>, [Exception](#)<sup>[107]</sup>, [Expansion Region](#)<sup>[107]</sup>, [Exit Point](#)<sup>[110]</sup>
- [Final](#)<sup>[110]</sup>, [Flow Final](#)<sup>[111]</sup>, [Fork](#)<sup>[112]</sup>
- [History](#)<sup>[116]</sup>
- [Initial](#)<sup>[117]</sup>, [Interaction](#)<sup>[118]</sup>, [Interaction Occurrence](#)<sup>[119]</sup>, [Interruptible Activity Region](#)<sup>[121]</sup>
- [Join](#)<sup>[112]</sup>, [Junction](#)<sup>[122]</sup>
- [Lifeline](#)<sup>[123]</sup>
- [Note](#)<sup>[126]</sup>
- [Object](#)<sup>[165]</sup>
- [Package](#)<sup>[168]</sup>, [Partition](#)<sup>[126]</sup>
- [Receive](#)<sup>[127]</sup>, [Region](#)<sup>[128]</sup>
- [Send](#)<sup>[129]</sup>, [State](#)<sup>[129]</sup>, [State/Continuation](#)<sup>[132]</sup>, [State Lifeline](#)<sup>[135]</sup>, [State Machine](#)<sup>[136]</sup>, [Structured Activity](#)<sup>[138]</sup>, [Synch](#)<sup>[142]</sup>, [System Boundary](#)<sup>[142]</sup>
- [Terminate](#)<sup>[144]</sup>, [Trigger](#)<sup>[145]</sup>
- [Use Case](#)<sup>[146]</sup>
- [Value Lifeline](#)<sup>[149]</sup>

#### Note:

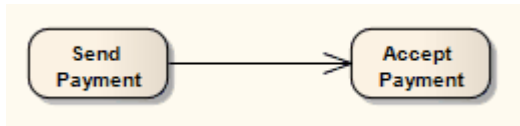
Actor, Collaboration, Note, Object and Package elements are used in both Behavioral diagrams and Structural diagrams.



### 2.1.1 Action

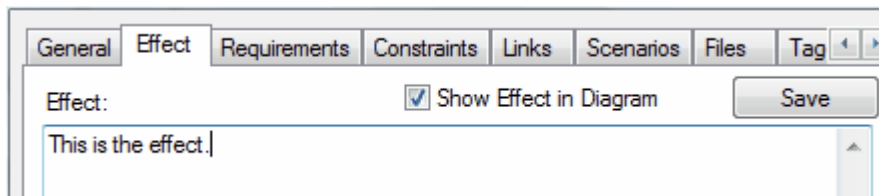


An *Action* element describes a basic process or transformation that occurs within a system. It is the basic functional unit within an [Activity diagram](#)<sup>[54]</sup>. Actions can be thought of as children of [Activities](#)<sup>[90]</sup>. Both represent processes, but *Activities* can contain multiple steps or decomposable processes, each of which can be embodied in an Action. An Action cannot be further broken down or decomposed.



An Action can be further defined with [pre-condition and post-condition](#)<sup>[89]</sup> notes, and certain properties can be [graphically depicted](#)<sup>[81]</sup> on the Action (Enterprise Architect prompts you to define the type of Action you are creating when you first drag the **Action** icon from the **Toolbox**). The data values passed out of and into an Action can be represented by [Action Pins](#)<sup>[87]</sup>. For a named Action (that is, other than a basic Action) you can also [assign](#)<sup>[88]</sup> Action Pins to represent specific properties.

For a basic (*Atomic*) Action, you can define the effect of the Action on the **Effect** tab of the element **Properties** dialog, and select to display the effect on the diagram.

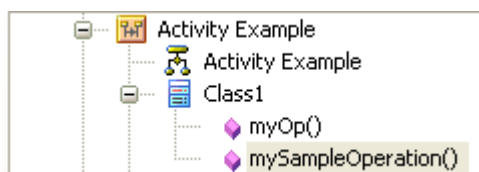


An Action can also be depicted as an [Expansion Node](#)<sup>[86]</sup> to indicate that the Action comprises an [Expansion Region](#)<sup>[107]</sup>.

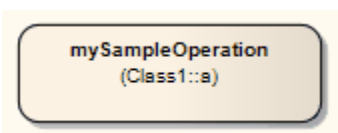
## Class Operations in Activity Diagrams

Operations from Classes can be displayed on Activity diagrams as Actions. When an operation is shown as an Action, the notation of the Action displays the name of the Class that features the operation. To add an operation to an Activity diagram follow the steps below:

1. Open an Activity diagram.
2. From the **Project Browser** open a Class and locate the operation to be added to the Activity diagram.
3. Drag the operation on to the diagram.



4. When the operation has been added to the Activity diagram the Action displays the name of the Class that features the operation.

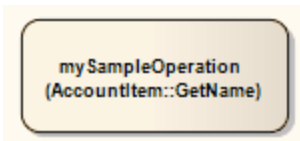


If you right-click on the Action in the diagram, you can locate the behavior classifier (CallBehavior Activity) or call operation (CallOperation Activity) in the **Project Browser** using the **Find | Locate Classifier in Project Browser** and **Find | Locate Operation in Project Browser** context menu options.

If it becomes necessary to change the operation that this Action refers to, follow the steps below:

1. Right-click on the Action. The context menu displays.

2. Select the **Advanced | Set Operation** menu option. The [Set Operation dialog](#)<sup>85</sup> displays.
3. If necessary, in the **In Namespace** field, select the model that contains the required operation.
4. Double-click on the required operation. The Action updates to show the new classifier and operation.



### Toolbox Icon



### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 241*) states:

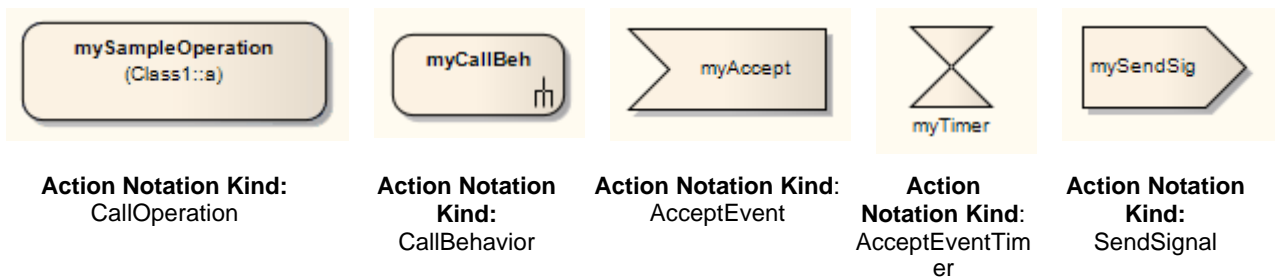
*An action is a named element that is the fundamental unit of executable functionality. The execution of an action represents some transformation or processing in the modeled system, be it a computer system or otherwise.*

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 313*) also states:

*An action may have sets of incoming and outgoing activity edges that specify control flow and data flow from and to other nodes. An action will not begin execution until all of its input conditions are satisfied. The completion of the execution of an action may enable the execution of a set of successor nodes and actions that take their inputs from the outputs of the action.*

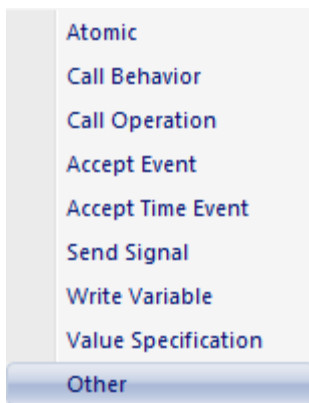
#### 2.1.1.1 Action Notation

Some properties can be graphically depicted on an [Action](#)<sup>79</sup> element, as shown in the examples below.

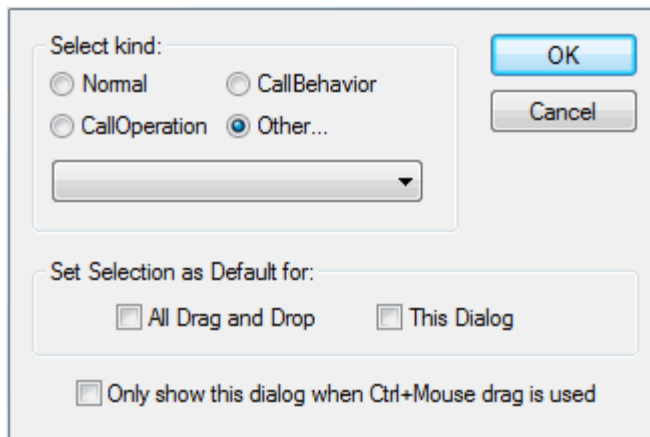


When you drag the **Action** icon from the **Activity** page of the **Toolbox** onto your diagram, a selection list displays showing the commonest types of Action to create. (If this list does not display, press **[Ctrl]** as you drag the icon.)



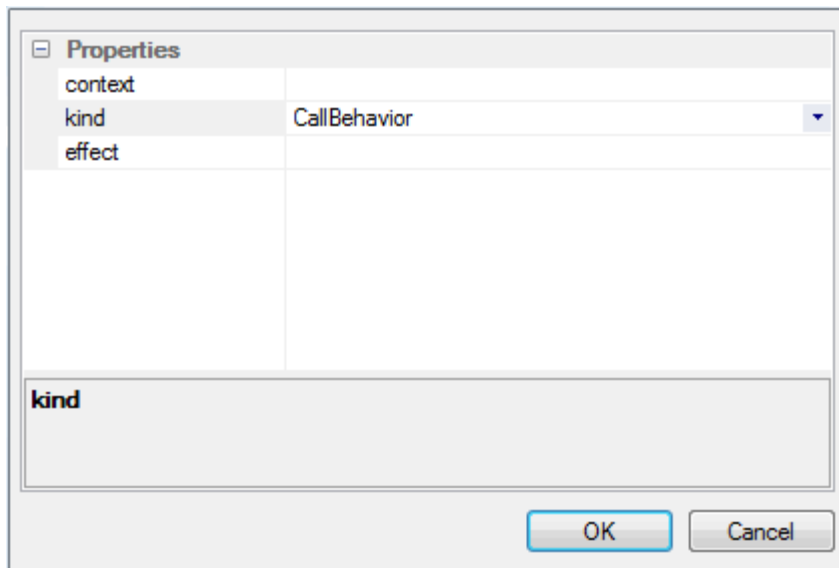


When you click on one of the specific types, that type of Action element displays on diagram. If you click on the **Other** option, the **New Action** dialog displays:



You can again select to create a normal (Atomic) [Action](#)<sup>79</sup> element, a CallOperation or a CallBehavior, or you can select the **Other** radio button and click on the drop-down arrow in the blank field to select the Action type from an extensive list.

If you later decide that the Action type is not appropriate, you can change it by right-clicking on the Action and selecting the **Advanced | Custom Properties** context menu option, which displays the **Custom Properties** dialog. Set the Action type by selecting a value from the **Kind** drop-down list. For a Value Specification Action, you also set the value on this dialog.



### AcceptEvent Actions

For an *Accept Event* Action element, the **Properties** dialog contains a **Triggers** tab on which you define one or more triggers to denote the type of events accepted by the Action, as defined in the following table:

Option	Use to
<b>Name</b>	Specify the name of the trigger.
<b>Type</b>	<p>Specify the type of trigger: <b>Call</b>, <b>Change</b>, <b>Signal</b> or <b>Time</b>.</p> <ul style="list-style-type: none"> <li>• <b>Call</b> - specifies that the event is a CallEvent, which sends a message to the associated object by invoking an operation.</li> <li>• <b>Change</b> - specifies that the event is a ChangeEvent, which indicates that the transition is the result of a change in value of an attribute.</li> <li>• <b>Signal</b> - specifies that the event is a SignalEvent, which corresponds to the receipt of an asynchronous signal instance.</li> <li>• <b>Time</b> - corresponds to a TimeEvent; which specifies a moment in time.</li> </ul> <p><b>Note:</b></p> <p>Code generation for State Machines currently supports Change and Time trigger events only, and expects a specification value.</p>
<b>Specification</b>	Specify the event instigating the Transition.

### SendSignal Action & BroadcastSignal Action

For a *SendSignal* or *BroadcastSignal* Action element, you can model the signal to be sent and the associated arguments to be conveyed, using the **Signal** tab of the element **Properties** dialog.

The screenshot shows the 'Signal' tab of a dialog box. At the top, there are several tabs: 'General', 'Signal', 'Requirements', 'Constraints', 'Links', 'Scenarios', 'Files', and 'Tagged Values'. The 'Signal' field is set to 'Signal1'. Below it, the 'Argument' section is expanded, showing an 'Attribute' dropdown menu with 'iCyclesCompleted' selected and a 'Value' field containing '10'. A 'Save' button is located to the right of the 'Value' field. Underneath, the 'Arguments:' section contains a list with one entry: 'Development Model::Activity Example::LoopNode1.Action12: Activity6.ActionPinA: Boolean'. There are 'Add' and 'Del' buttons below the list. At the bottom of the dialog are 'OK', 'Cancel', 'Apply', and 'Help' buttons.

To complete this tab, follow the steps below:

1. In the **Signal** field, click on the [ ... ] button and select the required signal from the **Select Signal** dialog.
2. In the **Attribute** field, click on the drop-down arrow and select the attribute (as previously created in the Signal element) with which the arguments are to be associated.
3. In the **Value** field, type the appropriate value for the attribute.
4. Identify the arguments (as [Action Pins](#)<sup>87</sup>) for the Signal; click on the **Add** button under the **Arguments** panel, and select the appropriate Pins from the **Select Pin** dialog.

**Note:**

To assign more than one Pin, press **[Ctrl]** whilst you select each Pin.

5. Click on the **Save** button.

### Structural Feature Actions

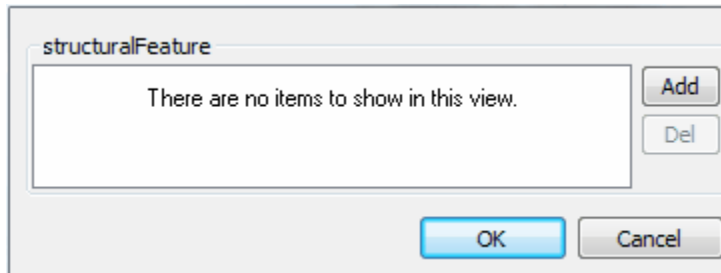
Enterprise Architect supports the following types of *Structural Feature* Actions:

- AddStructuralFeatureValue
- ClearStructuralFeature

- ReadStructuralFeature
- RemoveStructuralFeatureValue
- WriteStructuralFeature

These actions can take Ports, Parts or Attributes as the target structural feature. To set the appropriate feature, follow the steps below:

1. Right-click on the Action element in the diagram, and select the **Advanced | Set Structural Feature** context menu option. The **Set Structural Feature** dialog displays.



2. To locate the structural feature, click on the **Add** button. The **Select Property** dialog displays (a variant of the **Select <Item>** dialog).
3. Browse or search for the appropriate structural feature, and double-click on it. The feature name and location displays in the **structuralFeature** field of the **Set Structural Feature** dialog.
4. Click on the **OK** button to save the setting.

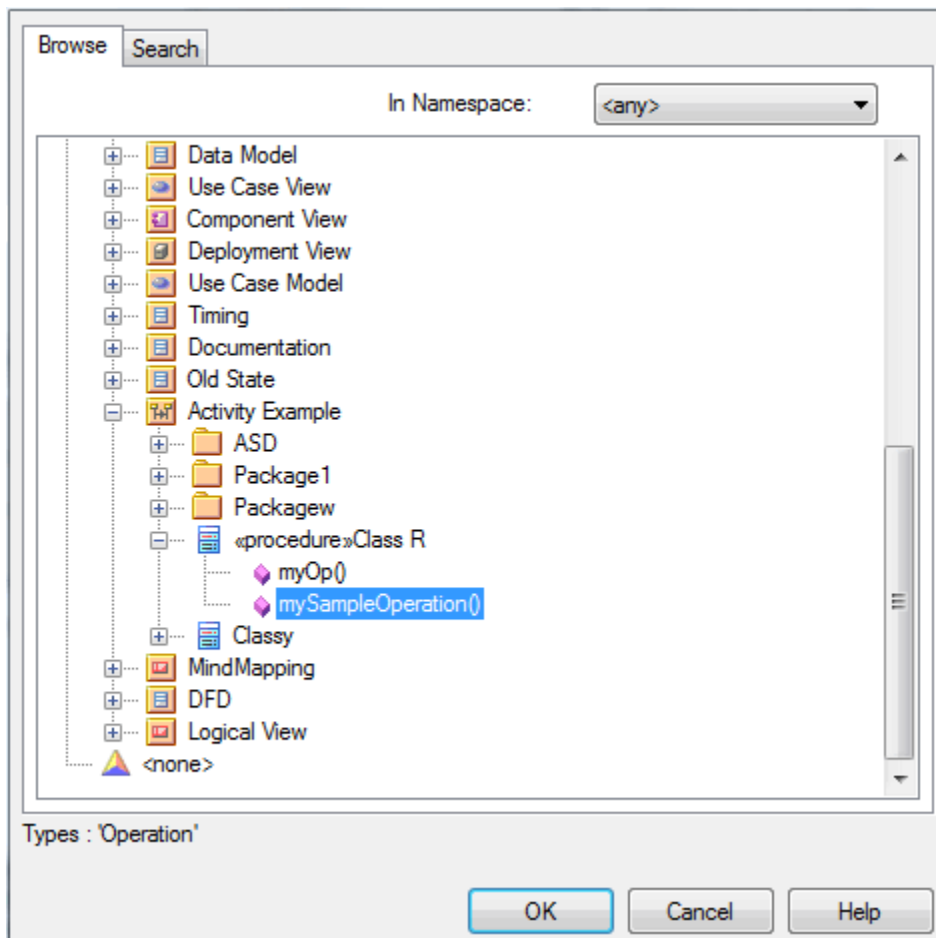
#### 2.1.1.1.1 Set Feature Dialog

The **Set Feature** dialog is the **Set Operation** dialog used to change the [operation represented by an Action](#)<sup>[80]</sup> on an Activity diagram.

As the **Set Operation** or **Set Attribute** dialog, it is also used to set the *Value* operation or attribute for Tagged Values of type RefGUID (see the *SDK For Enterprise Architect*) or for the target of a [hyperlink](#)<sup>[184]</sup> in a diagram.

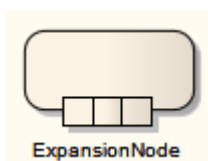
To use this dialog, follow the steps below:

1. The **Set Operation** (or **Set Attribute**) dialog displays, with the model hierarchy opened at the point at which you selected the original operation or attribute.



2. If required, in the **In Namespace** field, click on the drop-down arrow and select another model that contains the required operation or attribute. The package hierarchy for that model displays.
3. Browse through the hierarchy, or use the **Search** tab to locate the required operation or attribute, then double-click on the item to select it.

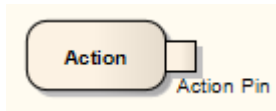
### 2.1.1.2 Action Expansion Node



Representing an [Action](#) <sup>[79]</sup> as an *Expansion Node* is a shorthand notation to indicate that the Action comprises an [Expansion Region](#) <sup>[107]</sup>.

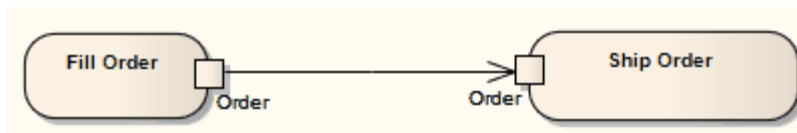
To specify an Action as an Expansion Node, right-click on the Action to display the context menu and select the **Embedded Elements | Add Expansion Node** menu option. After designating an Action as an Expansion Node, you can modify or delete it using the **Embedded Elements | Embedded Elements** menu option.

### 2.1.1.3 Action Pin



An *Action Pin* is used to define the data values passed out of and into an [Action](#)<sup>[79]</sup>. An *input pin* provides values to the Action, whereas an *output pin* contains the results from that Action.

Action Pins are used below to connect two Actions:



See *UML Superstructure Specification, v2.1.1, Figure 12.110, p. 391*.

Action Pins can be further characterized as defining exception parameters, streams, or states. Associating a state with a Pin defines the state of input or output values. For instance, the Pin could be called *Orders*, but the state could be *Validated* or *Canceled*.

To *add* an Action Pin to an Action, right-click on the Action to display the context menu and select the **Embedded Elements | Add Action Pin** menu option. (You can also [assign](#)<sup>[88]</sup> Action Pins, to define specific properties of the Action.)

The **Properties** dialog of an Action Pin has a **Pin** tab on which you define the specific actions of the Pin.

A Pin serves as an argument for Call Behavior Actions and Call Operation Actions. When an Action is associated with a valid behavior in the model, the associated behavior's parameters are listed in the **Parameter** field drop-down list to facilitate a one-to-one mapping between the argument and the parameter. The fields in the **Argument** panel are enabled only for Pins belonging to Call Actions, and only when the Action is associated with a valid behavior with valid parameters.

You can also change certain properties of an Action Pin on the **Custom Properties** dialog: right-click on the Pin and select the **Advanced | Custom Properties** context menu option. The following properties can be set:

Properties	
isOrdered	True
isUnique	True
objectState	
kind	input

#### 2.1.1.4 Assign Action Pins

Apart from *adding* Action Pins to any Action, you can *assign* specialized input or output Action Pins to Actions that have a specific type (that is, those that are not Basic or Atomic Actions). These input/output Pins signify various *properties* of the Action - they are not visible as structures on the diagram unless they have previously been [added](#) <sup>[87]</sup>, but are listed in the [Project Browser](#) as properties of the Action.

You can only assign Pins that have already been added or assigned to the Action, or that are being created specifically to be assigned to the Action.

To assign Pins to an Action, follow the steps below:

1. Right-click on the Action in the diagram, and select the **Advanced | Assign Action Pins** context menu option. The [Assign Action Pins to <ActionName>](#) dialog displays.

The dialog box is titled 'Assign Action Pins to <ActionName>'. It contains three sections, each with a text field and two buttons ('Add' and 'Del').

- first**: Text field contains 'Model::Test::TestIdentity.first'. Buttons: 'Add', 'Del'.
- result**: Text field contains 'Model::Test::TestIdentity.result'. Buttons: 'Add', 'Del'.
- second**: Text field contains 'Model::Test::TestIdentity.second'. Buttons: 'Add', 'Del'.

At the bottom of the dialog are 'OK' and 'Cancel' buttons.

The format of this dialog depends on the type of Action: for a *SendObject* Action the dialog has two fields (**request** and **target**); for the above *TestIdentity* Action, three; and for a *CallBehavior* Action, one (**result**). The fields are populated in exactly the same way.

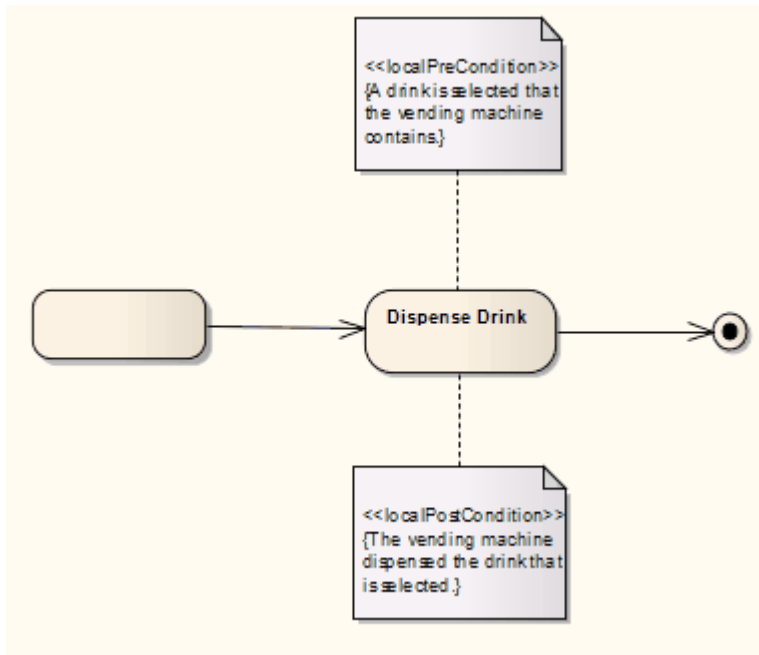
2. The mandatory number and type of Pins are automatically selected (if they exist) or created. To change or add a Pin in a field, click on the corresponding **Add** button. The [Select Pins](#) dialog displays (a variant of the [Select <Item>](#) dialog), showing the selected Action and listing all the input Pins currently owned by the Action.
3. Double-click on one of the Pins (or, depending on the multiplicity of the Pin, **[Ctrl]+click** on several Pins). Alternatively, if no suitable Pin exists, click on the **Add New** button and then click on the newly-created Pin. The selected Pin is identified in the field on the [Assign Action Pins to <ActionName>](#) dialog.
4. Click on the **OK** button.

To check the exact location of an assigned Action Pin, you can right-click on the Pin name in the dialog and

select the **Find in Project Browser** context menu option.

### 2.1.1.5 Local Pre/Post Conditions

[Actions](#) can be further defined with *pre-condition* and *post-condition* notes, which constrain an Action's entry and exit. These notes can be added to an Action as defined below.



See *UML Superstructure Specification, v2.1.1, Figure 12.32, p. 316.*

### Create a Constraint

To attach a constraint to an Action follow the steps below:

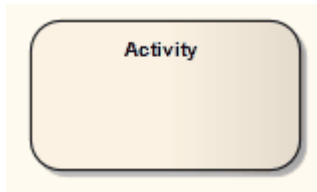
1. Right-click on the Action. The context menu displays:
2. Select the **Add | Constraint** menu option. A *Note* is created on the diagram, connected to the Action.
3. Right-click on the *Note*. The context menu displays.
4. Select the **View Properties** menu option. The **Constraint** dialog displays.

The screenshot shows the "Constraint" dialog box. It has a "Constraint Type:" label followed by a dropdown menu currently showing "localPostCondition". Below this is a "Constraint:" label followed by a text area containing the text "The vending machine dispensed the selected drink." At the bottom right of the dialog are two buttons: "OK" and "Cancel".

5. In the **Constraint Type** field, click on the drop-down arrow and select the required constraint type.
6. In the **Constraint** field, type the text for the constraint.
7. Click on the **OK** button to save the constraint.

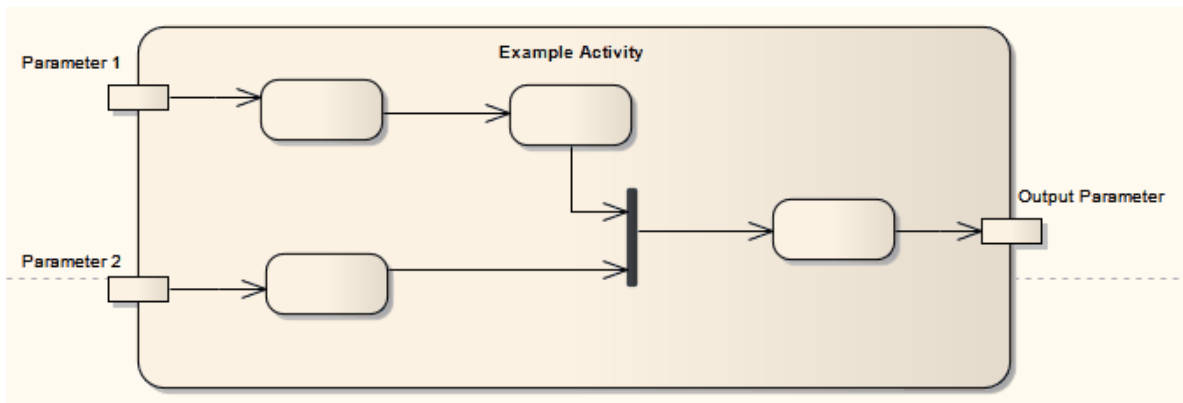


## 2.1.2 Activity



An *Activity* organizes and specifies the participation of subordinate behaviors, such as *sub-Activities* or *Actions*<sup>[79]</sup>, to reflect the control and data flow of a process. Activities are used in *Activity diagrams*<sup>[5]</sup> for various modeling purposes, from procedural-type application development for system design, to business process modeling of organizational structures or work flow.

The following simple diagram of an Activity contains Action elements and includes [input parameters and output parameters](#)<sup>[91]</sup>.



You can define an Activity as a *composite element*<sup>[180]</sup>, either during creation or during later edits. When creating a composite Activity element, it is simpler to apply the mechanism for creating *Structured Activity* elements, which reduces the number of steps to work through. See the *Structured Activity*<sup>[138]</sup> topic. If converting an existing Activity element, right-click on the element and select the **Advanced | Make Composite** context menu option.

Certain properties can be [graphically depicted](#)<sup>[91]</sup> on an Activity. The Actions in an Activity can be further organized by *Activity Partitions*<sup>[93]</sup>.

### Toolbox Icon



### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 318*) states:

*An activity specifies the coordination of executions of subordinate behaviors, using a control and data flow model. The subordinate behaviors coordinated by these models may be initiated because other behaviors in the model finish executing, because objects and data become available, or because events occur external to the flow. The flow of execution is modeled as activity nodes connected by activity edges. A node can be the execution of a subordinate behavior, such as an arithmetic computation, a call to an operation, or manipulation of object contents. Activity nodes also include flow-of-control constructs, such as synchronization, decision, and concurrency control. Activities may form invocation hierarchies invoking other activities, ultimately resolving to individual actions. In an object-oriented model, activities are usually invoked indirectly as methods bound to operations that are directly invoked.*

*Activities may describe procedural computation. In this context, they are the methods corresponding to operations on classes. Activities may be applied to organizational modeling for business process engineering*

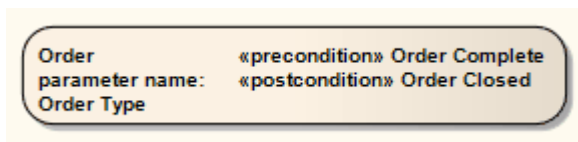
and workflow. In this context, events often originate from inside the system, such as the finishing of a task, but also from outside the system, such as a customer call. Activities can also be used for information system modeling to specify system level processes. Activities may contain actions of various kinds:

- Occurrences of primitive functions, such as arithmetic functions.
- Invocations of behavior, such as activities.
- Communication actions, such as sending of signals.
- Manipulations of objects, such as reading or writing attributes or associations.

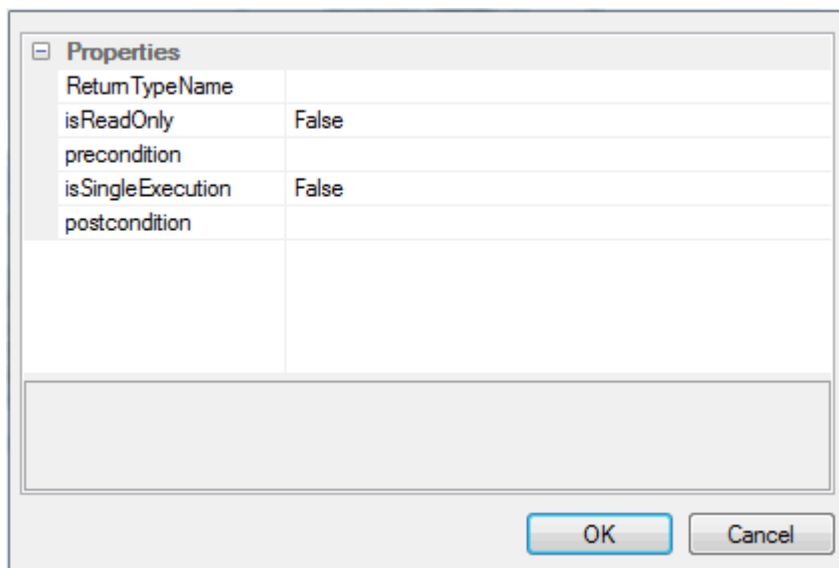
Actions have no further decomposition in the activity containing them. However, the execution of a single action may induce the execution of many other actions. For example, a call action invokes an operation that is implemented by an activity containing actions that execute before the call action completes.

### 2.1.2.1 Activity Notation

Certain properties can be graphically depicted on an [Activity](#) <sup>90</sup> element, as shown below.



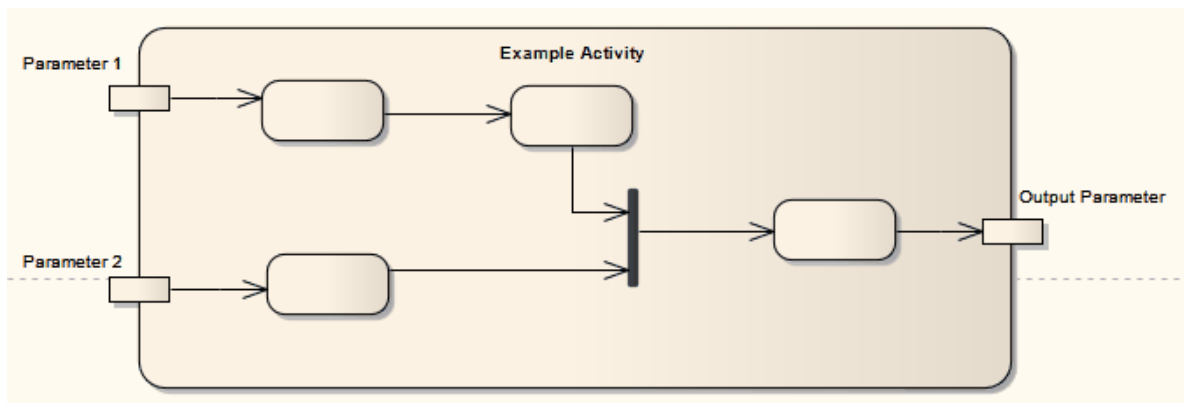
To define these properties, right-click on the Activity and select the **Advanced | Custom Properties** context menu option. The following dialog displays:



### 2.1.2.2 Activity Parameter Nodes

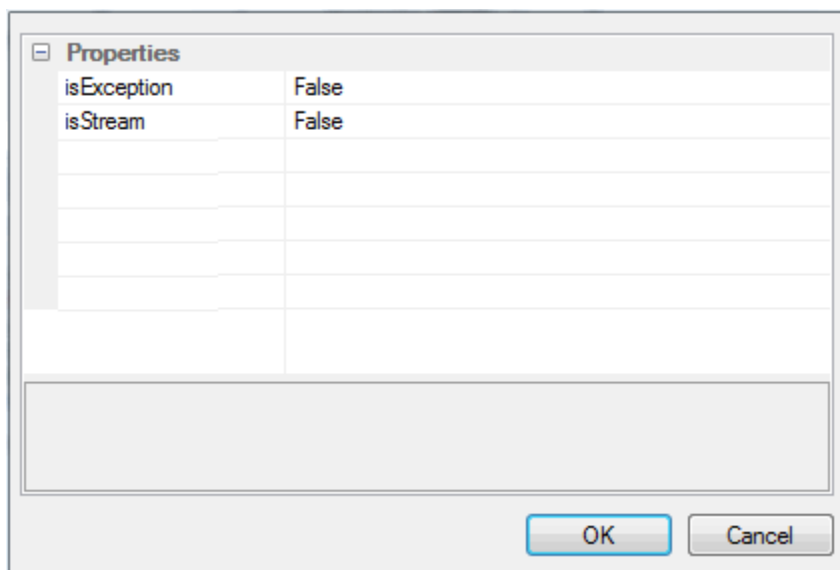
An *Activity Parameter Node* accepts input to an [Activity](#) <sup>90</sup> or provides output from an Activity.

The following example depicts two entry parameters and one output parameter defined for the Activity.



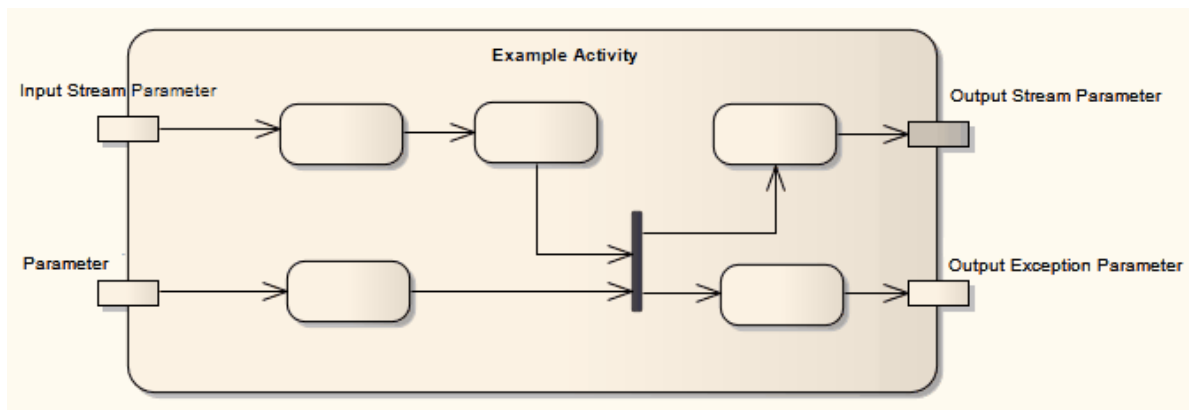
To define an Activity Parameter Node for an Activity, follow the steps below:

1. Right-click on the element and select the **Embedded Elements | Add Activity Parameter** context menu option.
2. The **Properties** dialog displays, which prompts for the **Name** and other properties of the embedded element.
3. After closing this dialog, you can further define the new Activity Parameter; right-click on it and select the **Advanced | Custom Properties** context menu option. The following dialog displays:



Similar to characterizing [Action Pins](#)<sup>[87]</sup>, Activity Parameter Nodes also have the *isException* and *isStream* options. *isException* indicates that a parameter can emit a value at the exclusion of other outputs, usually because of some error. *isStream* indicates whether or not a parameter can accept or post values during the execution of the Activity.

The following example uses the above settings:



### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 338*) states:

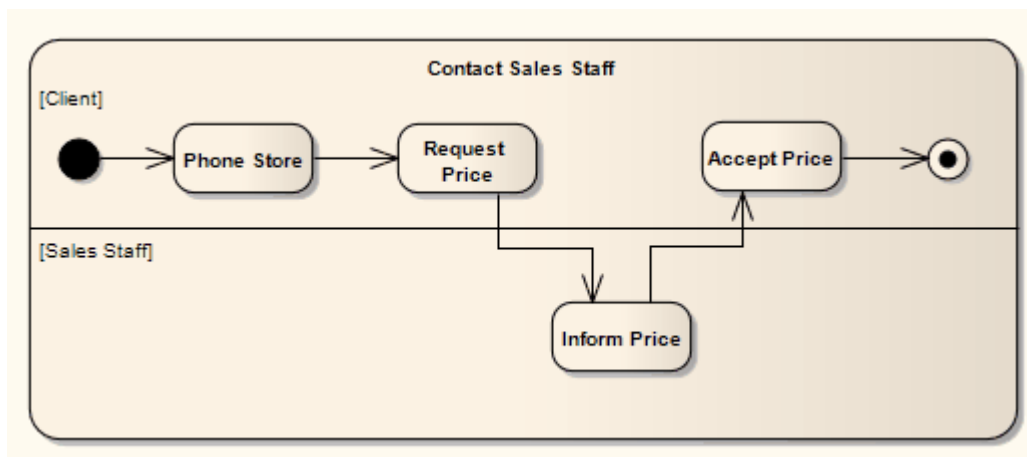
*An activity parameter node is an object node for inputs and outputs to activities.*

*... Activity parameter nodes are object nodes at the beginning and end of flows that provide a means to accept inputs to an activity and provide outputs from the activity, through the activity parameters.*

*Activity parameters inherit support for streaming and exceptions from Parameter.*

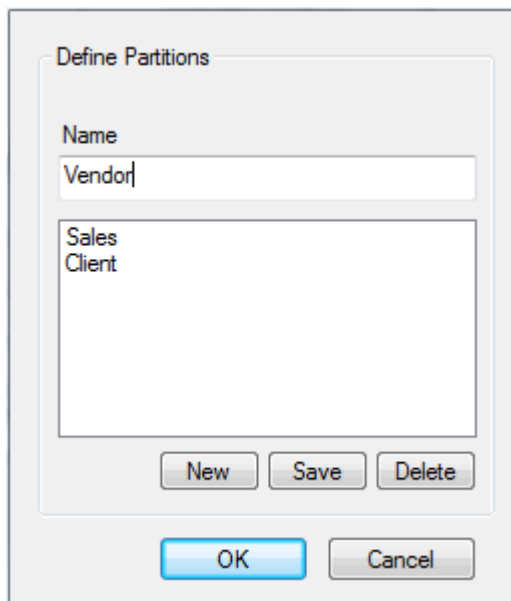
#### 2.1.2.3 Activity Partition

*Activity Partitions* are used to logically organize an [Activity](#)<sup>[90]</sup>. They do not affect the token flow of an Activity diagram, but help structure the view or parts of an Activity. An example of a partitioned Activity is shown below:



To define Partitions:

1. Right-click on the Activity element. The context menu displays.
2. Select the **Advanced | Partition Activity** menu option. The **Activity Partitions** dialog displays.



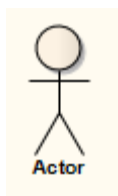
3. In the **Name** field, type the name of a partition. Click on the **Save** button.
4. Repeat step 3 for each partition to be created.

### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 341*) states:

*Partitions divide the nodes and edges to constrain and show a view of the contained nodes. Partitions can share contents. They often correspond to organizational units in a business model. They may be used to allocate characteristics or resources among the nodes of an activity.*

### 2.1.3 Actor



An *Actor* is a user of the system; *user* can mean a human user, a machine, or even another system or subsystem in the model. Anything that interacts with the system from the outside or system boundary is termed an Actor. Actors are typically associated with [Use Cases](#)<sup>[146]</sup>.

Actors can use the system through a graphical user interface, through a batch interface or through some other media. An Actor's interaction with a Use Case is documented in a Use Case scenario, which details the functions a system must provide to satisfy the user requirements.

Actors also represent the role of a user in [Sequence Diagrams](#)<sup>[39]</sup>. Enterprise Architect supports a stereotyped Actor element for [business modeling](#)<sup>[75]</sup>. The business modeling elements also represent Actors as stereotyped Objects.

### Toolbox Icon

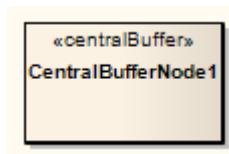


## OMG UML Specification

The OMG UML specification (*UML Superstructure Specification*, v2.1.1, p. 584) states:

*An actor models a type of role played by an entity that interacts with the subject (e.g. by exchanging signals and data), but which is external to the subject. ... Actors may represent roles played by human users, external hardware, or other subjects. Note that an actor does not necessarily represent a specific physical entity but merely a particular facet (i.e., "role") of some entity that is relevant to the specification of its associated Use Cases. Thus, a single physical instance may play the role of several different actors and, conversely, a given actor may be played by multiple different instances.*

### 2.1.4 Central Buffer Node



A *Central Buffer Node* is an object node for managing flows from multiple sources and destinations, represented in an [Activity diagram](#)<sup>[54]</sup>. It acts as a buffer for multiple in-flows and out-flows from other object nodes, but does not connect directly to Actions.

#### Toolbox Icon



## OMG UML Specification

The OMG UML specification (*UML Superstructure Specification*, v2.1.1, p. 352) states:

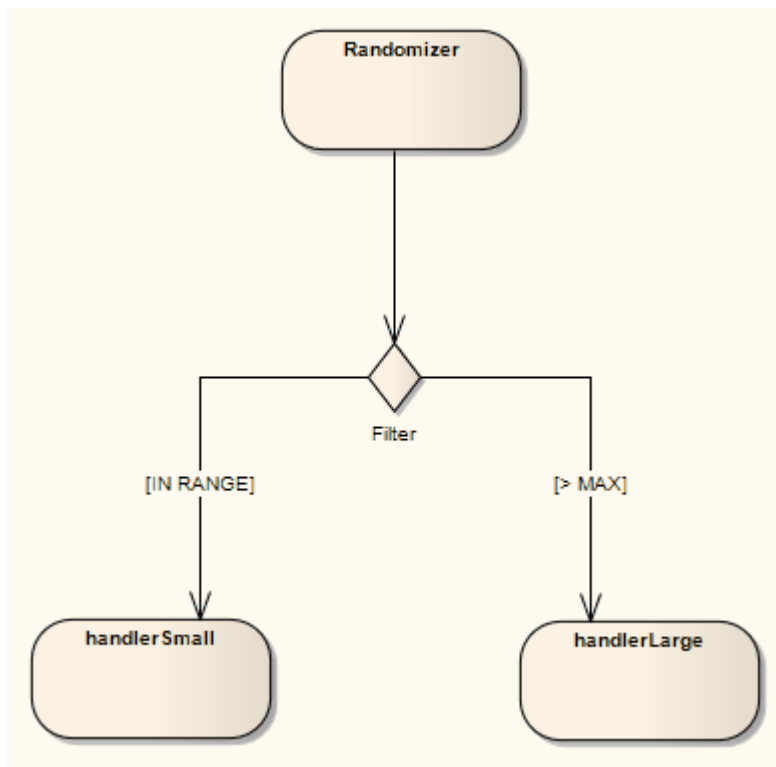
*A central buffer node is an object node for managing flows from multiple sources and destinations. ... A central buffer node accepts tokens from upstream object nodes and passes them along to downstream object nodes.*

### 2.1.5 Choice



The *Choice pseudo-state*<sup>[13]</sup> is used to compose complex transitional paths in, for example, a [State Machine diagram](#)<sup>[9]</sup>, where the outgoing transition path is decided by dynamic, run-time conditions. The run-time conditions are determined by the actions performed by the [State Machine](#)<sup>[129]</sup> on the path leading to the choice.

The following example depicts the Choice element. Upon reaching the *Filter* pseudo-state, a transition fires to the appropriate state based on the run-time value passed to the Filter. Very similar in form to a [Junction](#)<sup>[122]</sup> pseudo-state, the Choice pseudo-state's distinction is in deciding transition paths at run-time.



### Toolbox Icon

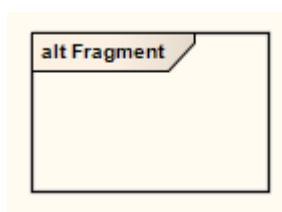


### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 538*) states:

*...choice vertices which, when reached, result in the dynamic evaluation of the guards of the triggers of its outgoing transitions. This realizes a dynamic conditional branch. It enables splitting of transitions into multiple outgoing paths such that the decision on which path to take may be a function of the results of prior actions performed in the same run-to-completion step. If more than one of the guards evaluates to true, an arbitrary one is selected. If none of the guards evaluates to true, then the model is considered ill-formed. (To avoid this, it is recommended to define one outgoing transition with the predefined "else" guard for every choice vertex.) Choice vertices should be distinguished from static branch points that are based on junction points.*

### 2.1.6 Combined Fragment

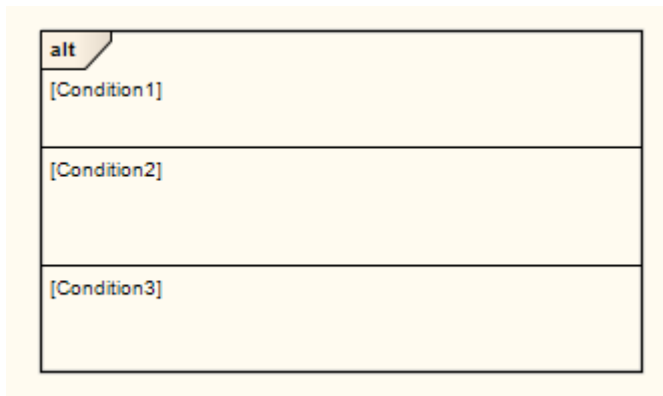


A *Combined Fragment* reflects a piece or pieces of interaction (called *interaction operands*) controlled by an [interaction operator](#)<sup>[99]</sup>, whose corresponding boolean conditions are known as *interaction constraints*. It displays as a transparent window, divided by horizontal dashed lines for each operand.

The following diagram illustrates the use of Combined Fragments, with a [Sequence diagram](#)<sup>[39]</sup> modeling a

simplified purchasing process. A loop fragment is created to iterate through an unknown number of items for purchase, after which the cashier requests payment. At this point, two payment options are considered and an alternative fragment is [created](#)<sup>98</sup>, divided to show the two operands: cash and credit card. After the fragment completes its trace, the cashier gives a receipt to the customer, under the fulfilled condition that payment requirements were met.

The order of interaction fragment conditions can be changed directly on the diagram. Select an interaction fragment with more than one condition defined. Up and down arrows appear on the right hand side of the each condition. Just click on the arrow to change the order.

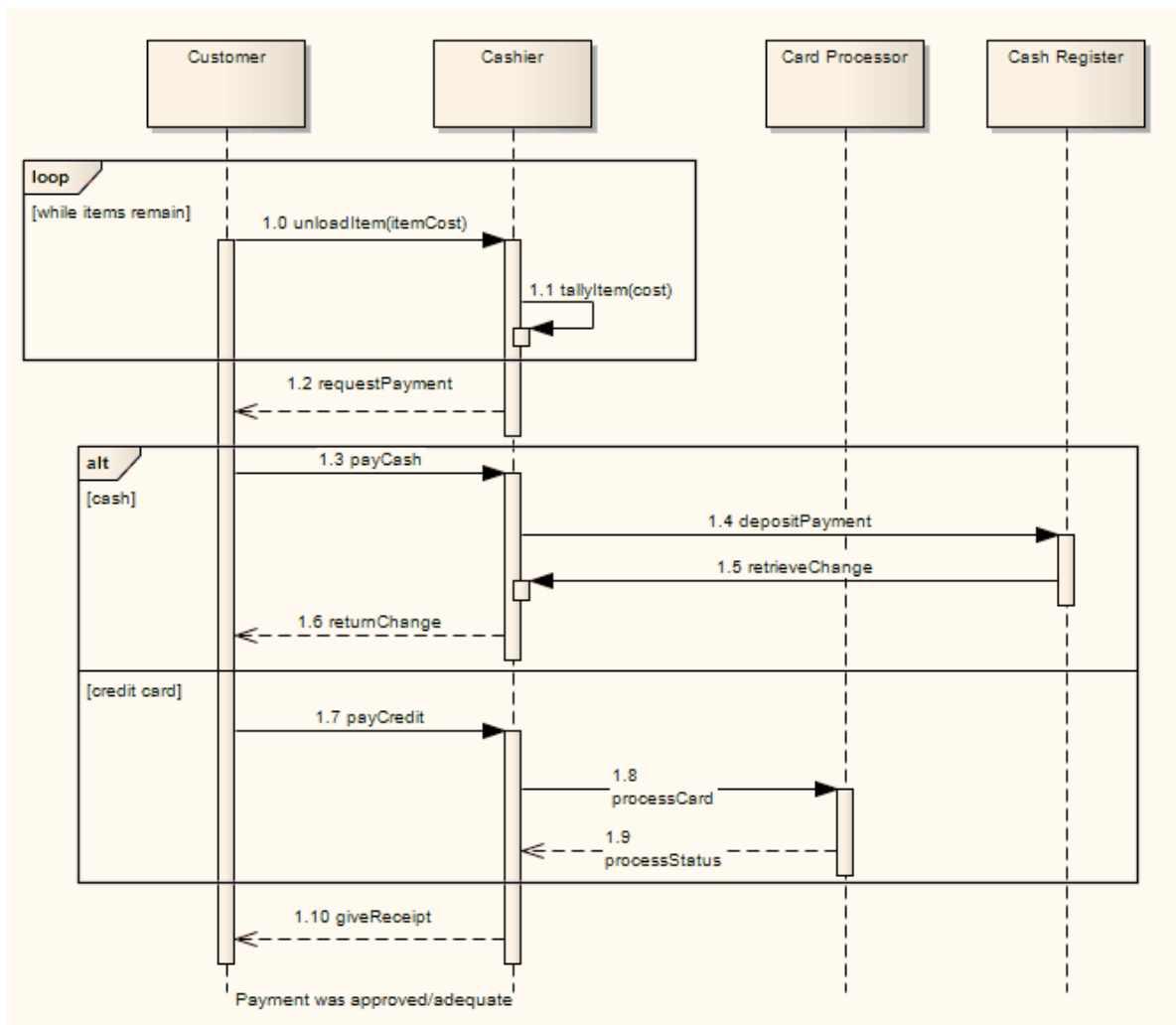
**Note:**

In order to select an interaction fragment, you must click near the inside edge or drag a selection rectangle around the fragment. This prevents accidental selection when moving connectors inside the interaction fragment.

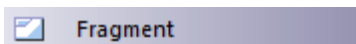
**Tip:**

Press and hold **[Alt]** to move a combined fragment independently of its contents.





## Toolbox Icon



## OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 467*) states:

*A combined fragment defines an expression of interaction fragments. A combined fragment is defined by an interaction operator and corresponding interaction operands. Through the use of CombinedFragments the user will be able to describe a number of traces in a compact and concise manner.*

### 2.1.6.1 Create a Combined Fragment

Use the following guidelines to create a [Combined Fragment](#) <sup>[96]</sup> in Enterprise Architect.

1. Drag the *Fragment* element from the **Interaction Elements** page of the Enterprise Architect UML **Toolbox**. The following dialog displays:

The screenshot shows a dialog box for creating a Combined Fragment. It features a 'Type' dropdown menu with 'alt' selected, a 'Name' text field, and a 'Condition' text field containing the text 'else'. Below the condition field is a large rectangular frame intended for defining interaction operands. At the bottom of the dialog are buttons for 'Delete', 'New', 'Save', 'OK', and 'Cancel'.

2. In the **Type** field, click on the drop-down arrow and select the interaction operator. See the [Interaction Operators](#)<sup>[99]</sup> topic for an explanation of the various types of Combined Fragments.
3. In the **Condition** field, specify a condition or interaction constraint for each operand.
4. A rectangular frame displays, partitioned by dashed lines into segments for each operand.
5. Adjust the frame to encompass the required event occurrences for each operand.

### 2.1.6.2 Interaction Operators

When creating [Combined Fragments](#)<sup>[96]</sup>, you must apply an appropriate interaction operator to characterize the fragment. The following table provides guidance on the various operators, and their corresponding descriptions.

Interaction Operator	Use to
<b>alt</b>	Divide up interaction fragments based on Boolean conditions.
<b>opt</b>	Enclose an optional fragment of interaction.
<b>par</b>	Indicate that operands operate in parallel.
<b>loop</b>	Indicate that the operand repeats a number of times, as specified by interaction constraints.
<b>critical</b>	Indicate a sequence that cannot be interrupted by other processing.
<b>neg</b>	Assert that a fragment is invalid, and implies that all other interaction is valid.
<b>assert</b>	Specify the only valid fragment to occur. Often enclosed within a <i>consider</i> or <i>ignore</i> operand.
<b>strict</b>	Indicate that the behaviors of the operands must be processed in strict sequence.
<b>seq</b>	Indicate that the Combined Fragment is weakly sequenced. This means that the ordering within operands is maintained, but the ordering between operands is undefined, so long as an event occurrence of the first operand precedes that of the second operand, if the event occurrences are on the same lifeline.
<b>ignore</b>	Indicate which messages should be ignored during execution, or can appear anywhere

Interaction Operator	Use to
	in the execution trace.
<b>consider</b>	Specify which messages should be considered in the trace. This is often used to specify the resulting event occurrences with the use of an <b>assert</b> operator.
<b>ref</b>	Provide a reference to another diagram.  <b>Note:</b> The ref fragment is not created using the method described in the <a href="#">Create a Combined Fragment</a> topic. To create a ref fragment, simply drag an existing diagram from the <b>Project Browser</b> onto the current diagram.

## OMG UML Specification

The OMG UML specification (*UML Superstructure Specification*, v2.1.1, p. 468-471) states:

*The semantics of a CombinedFragment is dependent upon the interactionOperator as explained below.*

### Alternatives

*The interactionOperator **alt** designates that the CombinedFragment represents a choice of behavior. At most one of the operands will be chosen. The chosen operand must have an explicit or implicit guard expression that evaluates to true at this point in the interaction. An implicit true guard is implied if the operand has no guard.*

*The set of traces that defines a choice is the union of the (guarded) traces of the operands.*

*An operand guarded by **else** designates a guard that is the negation of the disjunction of all other guards in the enclosing CombinedFragment.*

*If none of the operands has a guard that evaluates to true, none of the operands are executed and the remainder of the enclosing InteractionFragment is executed.*

### Option

*The interactionOperator **opt** designates that the CombinedFragment represents a choice of behavior where either the (sole) operand happens or nothing happens. An option is semantically equivalent to an alternative CombinedFragment where there is one operand with non-empty content and the second operand is empty.*

### Break

*The interactionOperator **break** designates that the CombinedFragment represents a breaking scenario in the sense that the operand is a scenario that is performed instead of the remainder of the enclosing InteractionFragment. A **break** operator with a guard is chosen when the guard is true and the rest of the enclosing Interaction Fragment is ignored. When the guard of the **break** operand is false, the **break** operand is ignored and the rest of the enclosing InteractionFragment is chosen. The choice between a **break** operand without a guard and the rest of the enclosing InteractionFragment is done non-deterministically.*

*A CombinedFragment with interactionOperator **break** should cover all Lifelines of the enclosing InteractionFragment.*

### Parallel

*The interactionOperator **par** designates that the CombinedFragment represents a parallel merge between the behaviors of the operands. The OccurrenceSpecifications of the different operands can be interleaved in any way as long as the ordering imposed by each operand as such is preserved.*

*A parallel merge defines a set of traces that describes all the ways that OccurrenceSpecifications of the operands may be interleaved without obstructing the order of the OccurrenceSpecifications within the operand.*

### Weak Sequencing

*The interactionOperator **seq** designates that the CombinedFragment represents a weak sequencing between the behaviors of the operands.*

Weak sequencing is defined by the set of traces with these properties:

1. The ordering of OccurrenceSpecifications within each of the operands is maintained in the result.
2. OccurrenceSpecifications on different lifelines from different operands may come in any order.
3. OccurrenceSpecifications on the same lifeline from different operands are ordered such that an OccurrenceSpecification of the first operand comes before that of the second operand.

Thus weak sequencing reduces to a parallel merge when the operands are on disjunct sets of participants. Weak sequencing reduces to strict sequencing when the operands work on only one participant.

### Strict Sequencing

The interactionOperator **strict** designates that the CombinedFragment represents a strict sequencing between the behaviors of the operands. The semantics of strict sequencing defines a strict ordering of the operands on the first level within the CombinedFragment with interactionOperator **strict**. Therefore OccurrenceSpecifications within contained CombinedFragment will not directly be compared with other OccurrenceSpecifications of the enclosing CombinedFragment.

### Negative

The interactionOperator **neg** designates that the CombinedFragment represents traces that are defined to be invalid.

The set of traces that defined a CombinedFragment with interactionOperator negative is equal to the set of traces given by its (sole) operand, only that this set is a set of invalid rather than valid traces. All InteractionFragments that are different from Negative are considered positive meaning that they describe traces that are valid and should be possible.

### Critical Region

The interactionOperator **critical** designates that the CombinedFragment represents a critical region. A critical region means that the traces of the region cannot be interleaved by other OccurrenceSpecifications (on those Lifelines covered by the region). This means that the region is treated atomically by the enclosing fragment when determining the set of valid traces. Even though enclosing CombinedFragments may imply that some OccurrenceSpecifications may interleave into the region, such as with **par**-operator, this is prevented by defining a region.

Thus the set of traces of enclosing constructs are restricted by critical regions.

### Ignore / Consider

(p. 473) The interactionOperator **ignore** designates that there are some message types that are not shown within this combined fragment. These message types can be considered insignificant and are implicitly ignored if they appear in a corresponding execution. Alternatively one can understand **ignore** to mean that the messages that are ignored can appear anywhere in the traces.

Conversely the interactionOperator **consider** designates which messages should be considered within this CombinedFragment. This is equivalent to defining every other message to be ignored.

### Assertion

The interactionOperator **assert** designates that the CombinedFragment represents an assertion. The sequences of the operand of the assertion are the only valid continuations. All other continuations result in an invalid trace. Assertions are often combined with Ignore or Consider.

### Loop

The interactionOperator **loop** designates that the CombinedFragment represents a loop. The **loop** operand will be repeated a number of times.

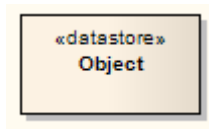
The Guard may include a lower and an upper number of iterations of the loop as well as a Boolean expression. The semantics is such that a loop will iterate minimum the 'minint' number of times (given by the iteration expression in the guard) and at most the 'maxint' number of times. After the minimum number of iterations have executed, and the boolean expression is false the loop will terminate. The loop construct represent a recursive application of the **seq** operator where the **loop** operand is sequenced after the result of earlier iterations.

### The Semantics of Gates

The gates of a *CombinedFragment* represent the syntactic interface between the *CombinedFragment* and its surroundings, which means the interface towards other *InteractionFragments*.

The only purpose of gates is to define the source and the target of messages.

### 2.1.7 Datastore

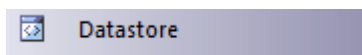


A *Datastore* is an element used to define permanently stored data. A token of data that enters into a *Datastore* is stored permanently, updating tokens for data that already exists. A token of data that comes out of a *Datastore* is a copy of the original data.

Use *Object Flow* <sup>[230]</sup> connectors to connect elements (such as *Activities* <sup>[907]</sup>) to *Datastores*, as values and information are being passed between nodes. Selection and transformation behavior, together composing a sort of query, can be specified as to the nature of data access. For instance, selection behavior determines which objects are affected by the connection to the *Datastore*. Transformation behavior might then further specify the value of an attribute pertaining to a selected object.

To define the behavior of access to a *Datastore*, attach a note to the *Object Flow* connector. To do this, right-click on the *Object Flow* and select the **Attach Note or Constraint** context menu option. A dialog indicates other flows in the *Activity diagram* <sup>[54]</sup>, to which you can attach the note (if the behavior applies to multiple flows). To comply with UML 2, preface behavior with the notation «*selection*» or «*transformation*».

#### Toolbox Icon



#### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification*, v2.1.1, p. 360) states:

*A data store node is a central buffer node for non-transient information... A data store keeps all tokens that enter it, copying them when they are chosen to move downstream. Incoming tokens containing a particular object replace any tokens in the object node containing that object.*

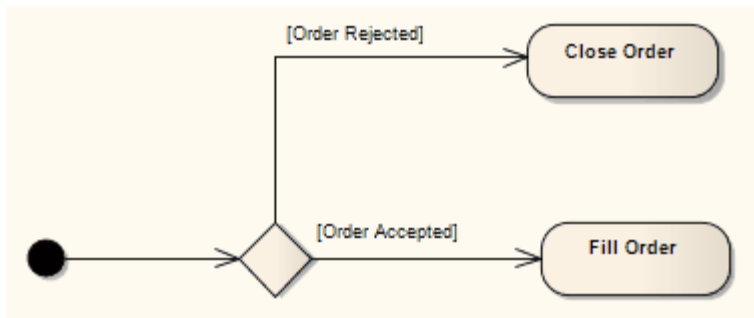
### 2.1.8 Decision



A *Decision* is an element of an *Activity diagram* <sup>[54]</sup> or *Interaction Overview diagram* <sup>[527]</sup> that indicates a point of conditional progression: if a condition is true, then processing continues one way; if not, then another.

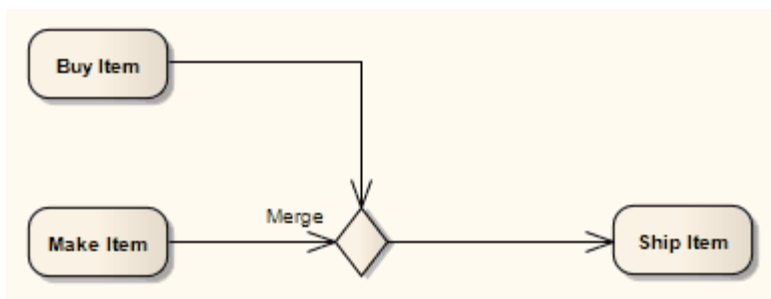
This can also be used as a *Merge node* <sup>[124]</sup> in that multiple alternative flows can be merged (but not synchronized) to form one flow. The following examples show both of these modes of using the decision element.

**Used as a decision:**



See *UML Superstructure Specification, v2.1.1, figure 12.77, p. 363.*

**Used as a merge:**

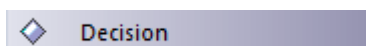


See *UML Superstructure Specification, v2.1.1, figure 12.106, p. 388.*

#### Note:

Moving a diagram generally does not affect the location of elements in packages. If you move a diagram out of one package into another, all the elements in the diagram remain in the original package. However, Decision elements are used only within one diagram, have no meaning outside that diagram, and are never re-used in any other diagram. Therefore, if you move a diagram containing these elements, they **are** moved to the new parent package with the diagram.

### Toolbox Icon



### OMG UML Specification

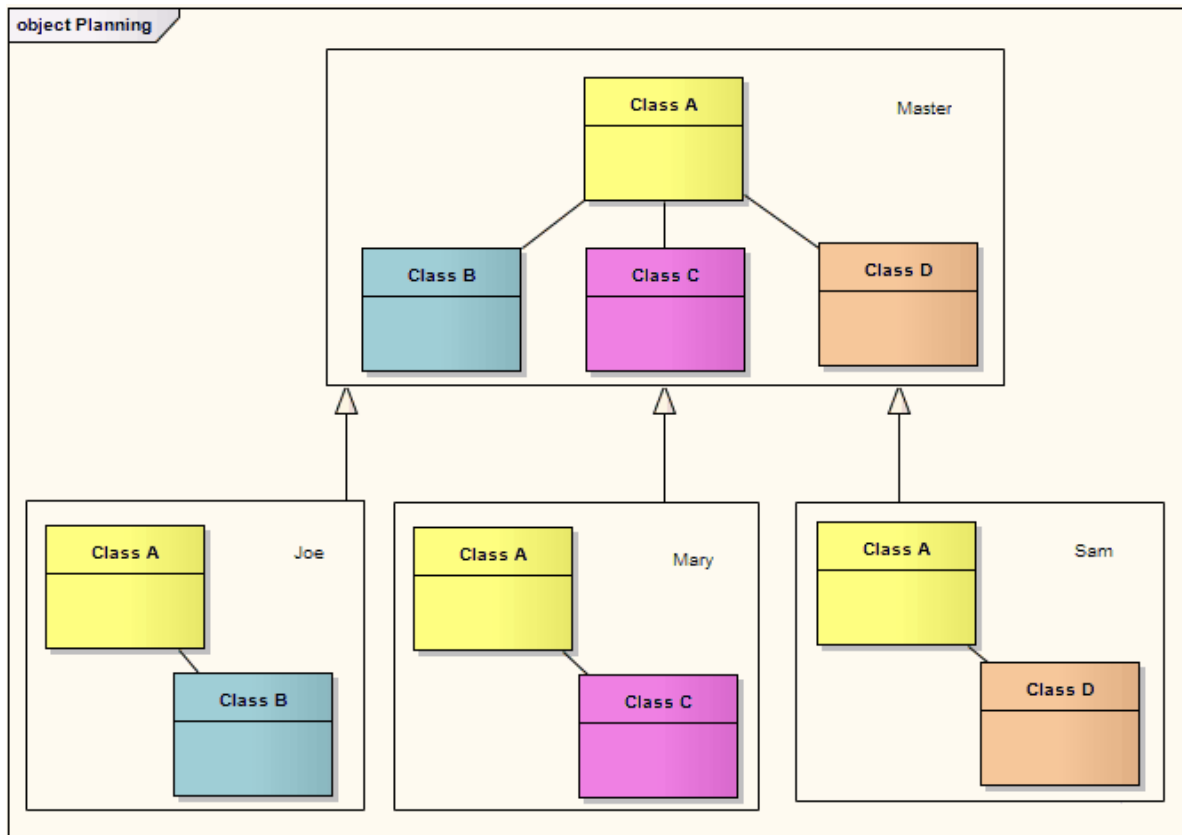
The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 361 (Decision symbol)*) states:

*A decision node is a control node that chooses between outgoing flows. A decision node has one incoming edge and multiple outgoing activity edges.*

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 387 (Merge symbol)*) also states:

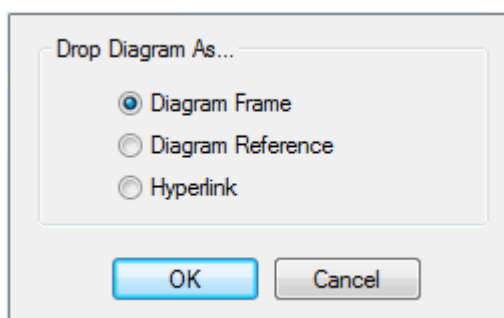
*A merge node is a control node that brings together multiple alternate flows. It is not used to synchronize concurrent flows but to accept one among several alternate flows... A merge node has multiple incoming edges and a single outgoing edge.*

## 2.1.9 Diagram Frame



A *Diagram Frame* element is a rendition of a diagram dropped from the **Project Browser** into another diagram. It is a type of **Combined Fragment**<sup>[96]</sup> with the **Interaction Operator**<sup>[100]</sup> ref. However, it can be created on any type of diagram, and is not created in the same way as other Combined Fragments.

When you drop the diagram from the **Project Browser** onto the open diagram, the following prompt displays:



If you click on the **Diagram Frame** radio button, a Diagram Frame is inserted into the diagram, containing an image of the dropped diagram.

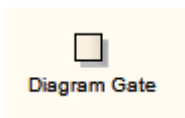
If you select the **Diagram Reference** option, an empty frame is inserted with the name of the dropped diagram in the frame label. If you select the **Hyperlink** radio button, a diagram icon is inserted with no frame, and with the parent package and diagram name next to it.

In all three cases, the object acts as a hyperlink to the real referenced diagram. You can also define properties for the objects, as for other elements, by right-clicking on the object and selecting the element **Properties** context menu option.

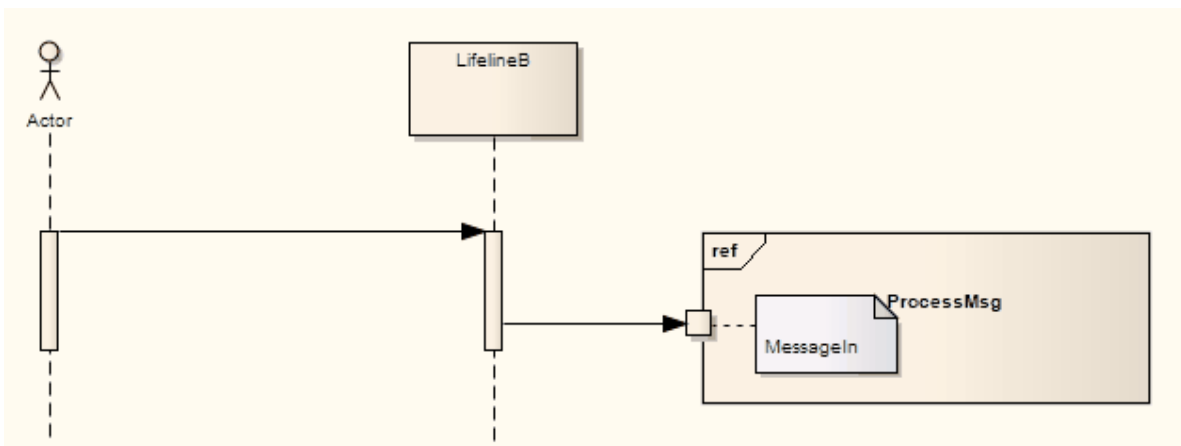
**Notes:**

- You can change the size of all three objects, but you cannot reduce a Diagram Frame to less than the size of the enclosed diagram.
- You cannot change the diagram within a Diagram Frame. To edit the diagram, double-click within the frame and edit the original diagram.
- The Diagram Frame element looks identical to but is **not the same** as a diagram frame *border*, which you can set automatically on new *images* of diagrams using the **Tools | Options | Diagram** option, and selecting the appropriate checkboxes in the **Diagram Frames** panel. These options set frames on print-outs of diagrams, images of diagrams copied to file, and images of diagrams copied to the clipboard. If you paste the image from the clipboard into another diagram, the image initially looks the same as the Diagram Frame element but it is actually a discreet unit that you manipulate using the Image Manager see *UML Modeling With Enterprise Architect - UML Modeling Tool*.

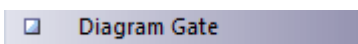
### 2.1.10 Diagram Gate



A *Diagram Gate* is a simple graphical way to indicate the point at which messages can be transmitted into and out of interaction fragments. A fragment might be required to receive or deliver a message; internally, an ordered message reflects this requirement, with a gate indicated on the boundary of the fragment's frame. Any external messages 'synching' with this internal message must correspond appropriately. Gates can appear on Interaction diagrams ([Sequence](#)<sup>[41]</sup>, [Timing](#)<sup>[22]</sup>, [Communication](#)<sup>[49]</sup> or [Interaction Overview](#)<sup>[52]</sup>), [interaction occurrences](#)<sup>[119]</sup> and [combined fragments](#)<sup>[96]</sup> (to specify the expression).



#### Toolbox Icon



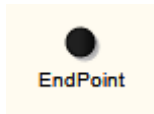
#### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 480*) states:

*A Gate is a connection point for relating a Message outside an InteractionFragment with a Message inside the InteractionFragment ... Gates are connected through Messages. A Gate is actually a representative of an OccurrenceSpecification that is not in the same scope as the Gate. Gates play different roles: we have formal gates on Interactions, actual gates on InteractionUses, expression gates on CombinedFragments.*



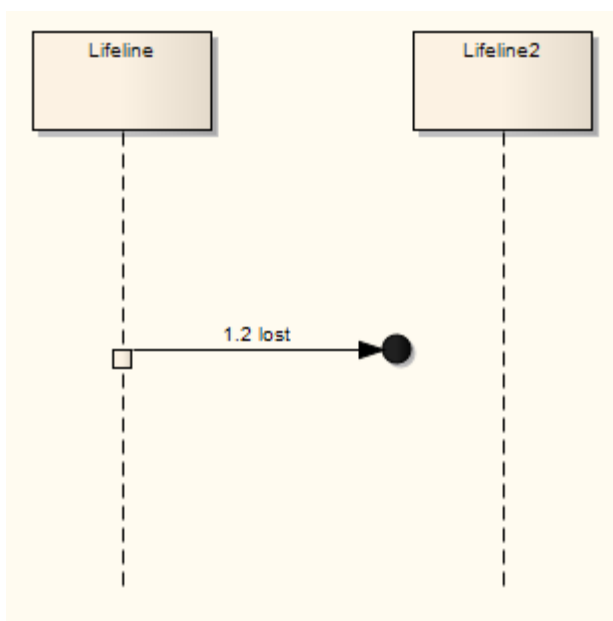
### 2.1.11 Endpoint



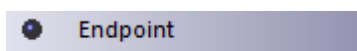
An *Endpoint* is used in Interaction diagrams ([Sequence](#)<sup>[41]</sup>, [Timing](#)<sup>[22]</sup>, [Communication](#)<sup>[49]</sup> or [Interaction Overview](#)<sup>[52]</sup>) to reflect a lost or found message in sequence. To model this, drag an *Endpoint* element onto the workspace.

With [Sequence diagrams](#)<sup>[39]</sup>, drag a message from the appropriate lifeline to the Endpoint. With [Timing diagrams](#)<sup>[22]</sup>, the message connecting the lifeline to the Endpoint requires some timing specifications to draw the connection.

The following example depicts a lost message in a Sequence diagram.



#### Toolbox Icon



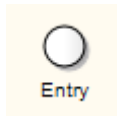
#### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification*, v2.1.1, p. 492) states:

*A lost message is a message where the sending event occurrence is known, but there is no receiving event occurrence. We interpret this to be because the message never reached its destination.*

*A found message is a message where the receiving event occurrence is known, but there is no (known) sending event occurrence. We interpret this to be because the origin of the message is outside the scope of the description. This may, for example, be noise or other activity that we do not want to describe in detail.*

### 2.1.12 Entry Point



*Entry Point pseudo-states* <sup>[13]</sup> are used to define the beginning of a *State Machine* <sup>[9]</sup>. An Entry Point exists for each region, directing the initial concurrent state configuration.

#### Toolbox Icon

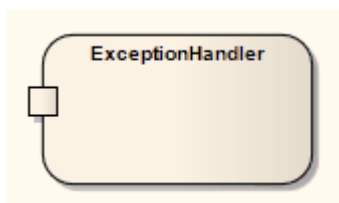


#### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 471*) states:

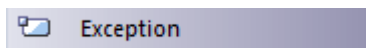
*An entry point pseudostate is an entry point of a state machine or composite state. In each region of the state machine or composite state it has a single transition to a vertex within the same region.*

### 2.1.13 Exception



The *Exception Handler* element defines the group of operations to carry out when an exception occurs. In an *Activity diagram* <sup>[5]</sup>, the protected element can contain a set of operations and is connected to the exception handler via an *Interrupt Flow* <sup>[21]</sup> connector. Any defined error contained within an element's parts can trigger the flow to move to an exception.

#### Toolbox Icon

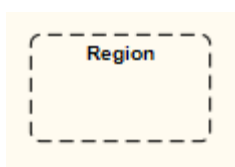


#### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 364*) states:

*An exception handler is an element that specifies a body to execute in case the specified exception occurs during the execution of the protected node.*

### 2.1.14 Expansion Region

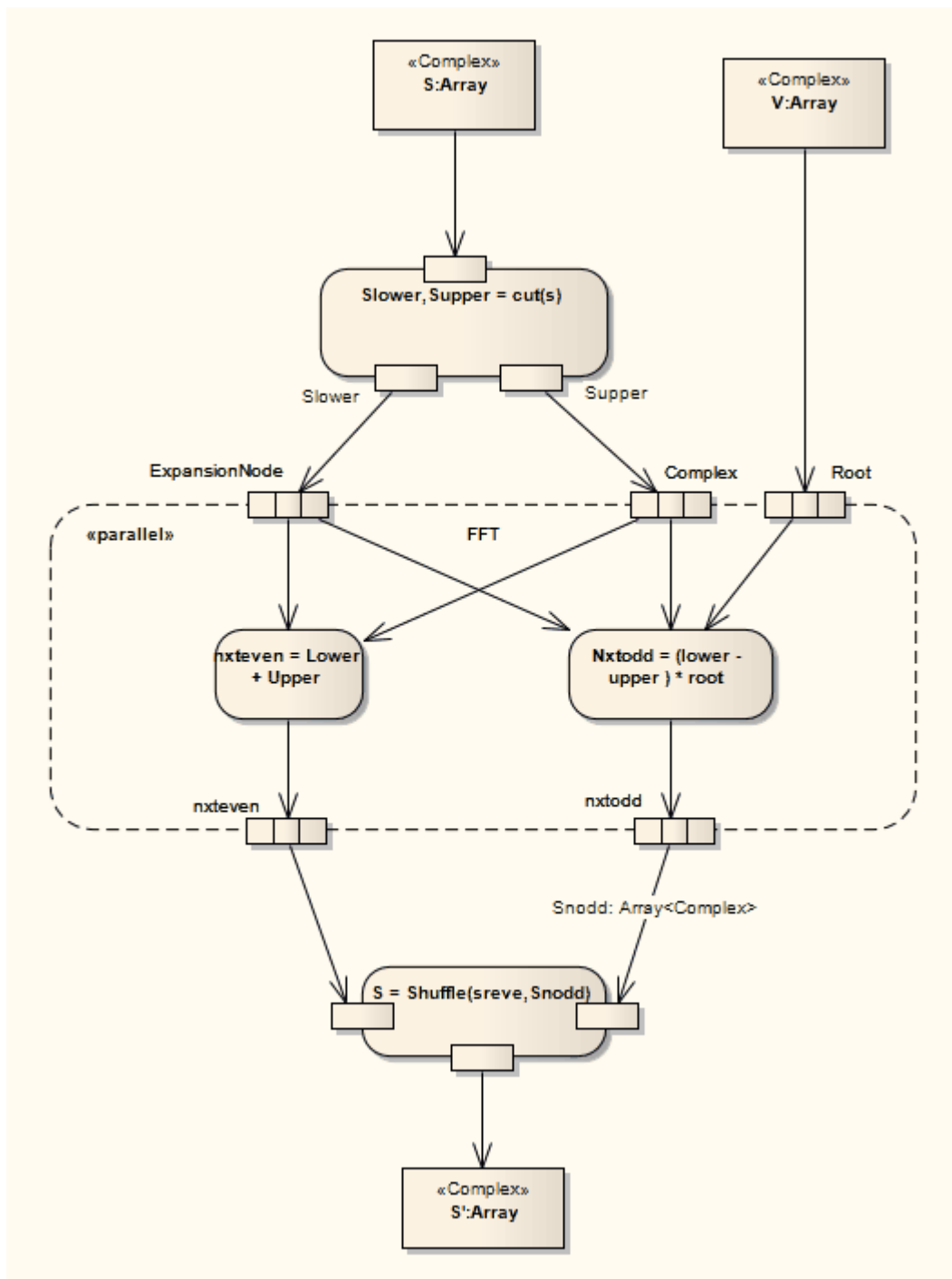


You [create](#)<sup>[110]</sup> an *Expansion Region* as one variant of a [Region](#)<sup>[128]</sup> (the other is an [Interruptible Activity Region](#)<sup>[127]</sup>).

On an [Activity diagram](#)<sup>[5]</sup>, an Expansion Region surrounds a process to be imposed multiple times on the incoming data, once for every element in the input collection. If there are multiple inputs, the collection sizes must match, and the elements within each collection must be of the same type. Similarly, any outputs must be in the form of a collection matching the size of the inputs.

The concurrency of the Expansion Region's multiple executions can be specified as type *parallel*, *iterative*, or *stream*. Parallel reflects that the elements in the incoming collections can be processed at the same time or overlapping, whereas an iterative concurrency type specifies that execution must occur sequentially. A stream-type Expansion Region indicates that the input and output come in and exit as streams, and that the Expansion Region's process must have some method to support streams.

To modify the mode of an Expansion Region, right-click on it and select the **Advanced | Custom Properties** context menu option.



See *UML Superstructure Specification, v2.1.1, figure 12.87, p. 372.*

### Toolbox Icon



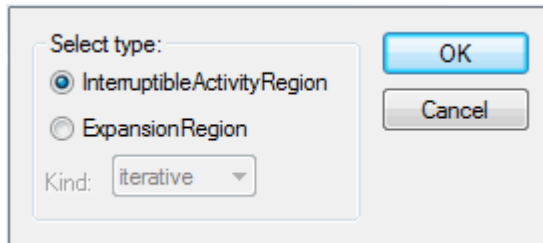
### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 367*) states:

An expansion region is a structured activity region that executes multiple times corresponding to elements of an input collection.

### 2.1.14.1 Add Expansion Region

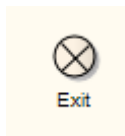
When you add a [Region](#)<sup>[128]</sup> element to a diagram, the following prompt displays:



The **Select type** defaults to **InterruptibleActivityRegion**.

1. Select the type [ExpansionRegion](#)<sup>[107]</sup>.
2. In the **Kind** field, click on the drop-down arrow and select the concurrency attribute.

### 2.1.15 Exit Point



*Exit Points* are used in [Submachine states](#)<sup>[136]</sup> and [State Machines](#)<sup>[129]</sup> to denote the point where the machine is exited and the transition sourcing this exit point, for Submachines, is triggered. Exit points are a type of [pseudo-state](#)<sup>[13]</sup> used in the [State Machine](#)<sup>[9]</sup> diagram.

#### Toolbox Icon

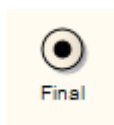


#### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1 p. 538*) states:

*An exit point pseudostate is an exit point of a state machine or composite state. Entering an exit point within any region of the composite state or state machine referenced by a submachine state implies the exit of this composite state or submachine state and the triggering of the transition that has this exit point as source in the state machine enclosing the submachine or composite state.*

### 2.1.16 Final

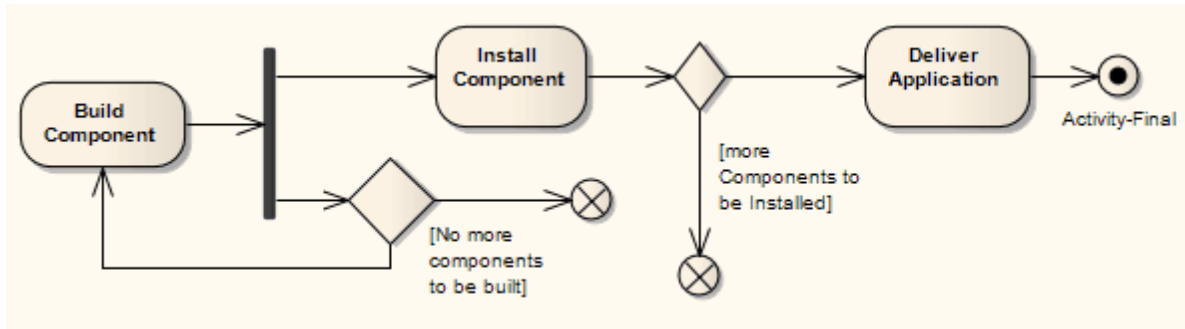


There are two nodes used to define a *Final* state in an [Activity](#)<sup>[90]</sup>, both defined in UML 2.1 as of type *Final Node*. The *Activity Final* element, shown above, indicates the completion of an Activity; upon reaching the Final, all execution in the [Activity diagram](#)<sup>[5]</sup> is aborted. The other type of final node, [Flow Final](#)<sup>[111]</sup>, depicts

an exit from the system that has no effect on other executing flows in the Activity.

The following example illustrates the development of an application. The process comes to a Flow Final node when there are no more components to be built; note that the [Fork](#)<sup>[114]</sup> element indicates a concurrent process with the building of new components and installation of completed components. The Flow Final terminates only the sub-process building components. Similarly, only those tokens entering the decision branch for the installation of further components terminate with the connecting Flow Final (that is, stop installing this component, but keep on installing other components). It is only after the *Deliver Application* activity is completed, after the control flow reaches the Final node, that all flows stop.

The node that initiates a flow is the [Initial](#)<sup>[117]</sup> node.



See *UML Superstructure Specification, v2.1.1, figure 12.91, p. 374.*

#### Note:

Moving a diagram generally does not affect the location of elements in packages. If you move a diagram out of one package into another, all the elements in the diagram remain in the original package. However, Final elements are used only within one diagram, have no meaning outside that diagram, and are never re-used in any other diagram. Therefore, if you move a diagram containing these elements, they **are** moved to the new parent package with the diagram.

### Toolbox Icon

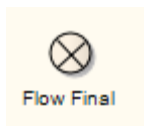


### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 332*) states:

*An activity may have more than one activity final node. The first one reached stops all flows in the activity.*

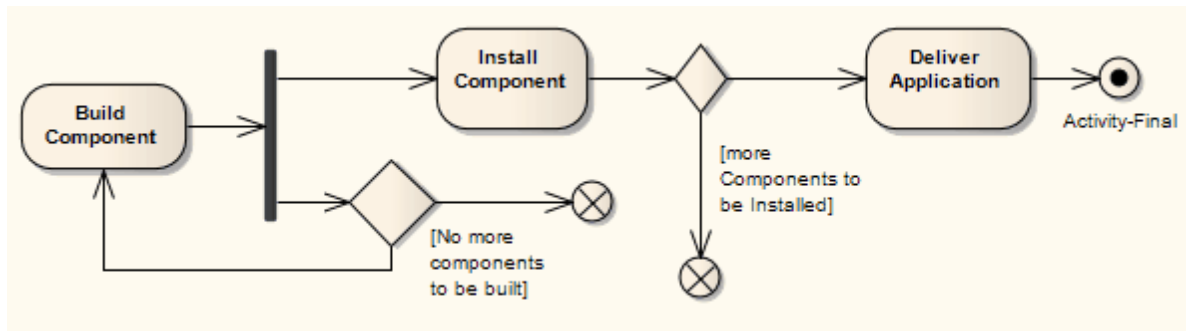
#### 2.1.17 Flow Final



There are two nodes used to define a final state in an Activity, both defined in UML 2.1 as of type *Final Node*. The *Flow Final* element depicts an exit from the system, as opposed to the [Activity Final](#)<sup>[110]</sup>, which represents the completion of the Activity. Only the flow entering the Flow Final node exits the Activity; other flows continue undisturbed.

The following example [Activity Diagram](#)<sup>[5]</sup> illustrates the development of an application. The process comes to a Flow Final node when there are no more components to be built; note that the [Fork](#)<sup>[114]</sup> element indicates a concurrent process with the building of new components and installation of completed components. The Flow Final terminates only the sub-process building components. Similarly, only those tokens entering the

decision branch for the installation of further components terminate with the connecting Flow Final (that is, stop installing this component, but keep on installing other components). It is only after the *Deliver Application* activity is completed, after the control flow reaches the Final node, that all flows stop.

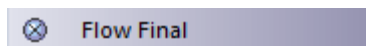


See *UML Superstructure Specification, v2.1.1, figure 12.91, p. 374.*

#### Note:

Moving a diagram generally does not affect the location of elements in packages. If you move a diagram out of one package into another, all the elements in the diagram remain in the original package. However, Flow Final elements are used only within one diagram, have no meaning outside that diagram, and are never re-used in any other diagram. Therefore, if you move a diagram containing these elements, they **are** moved to the new parent package with the diagram.

### Toolbox Icon



### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 375*) states:

*A flow final destroys all tokens that arrive at it. It has no effect on other flows in the activity.*

#### 2.1.18 Fork/Join



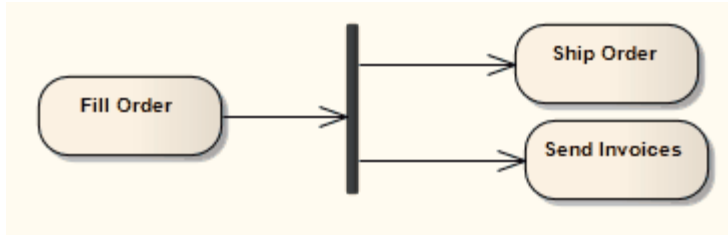
The *Fork/Join* elements have different modes of use, as follows:

- To fork or split the flow into a number of concurrent flows.
- To join the flow of a number of concurrent flows.
- To both join and fork a number of incoming flows to a number of outgoing flows.

These elements are used in both [Activity](#)<sup>[5]</sup> and [State Machine](#)<sup>[9]</sup> diagrams. With respect to State Machine diagrams, [Forks](#)<sup>[114]</sup> and [Joins](#)<sup>[115]</sup> are used as [pseudo-states](#)<sup>[13]</sup>. Other pseudo-states include [history states](#)<sup>[116]</sup>, [entry points](#)<sup>[107]</sup> and [exit points](#)<sup>[110]</sup>. Forks are used to split an incoming transition into concurrent multiple transitions leading to different target states. Joins are used to merge concurrent multiple transitions into a single transition leading to a single target. They are semantic inverses. To learn more about these individual elements see their specific topics.

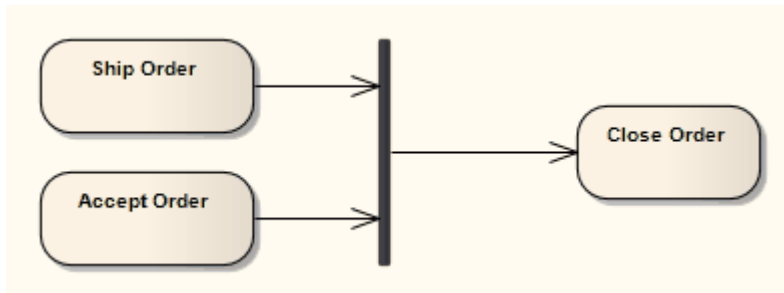
Some examples of Fork/Join nodes include:

Fork or split the flow into a number of concurrent flows:



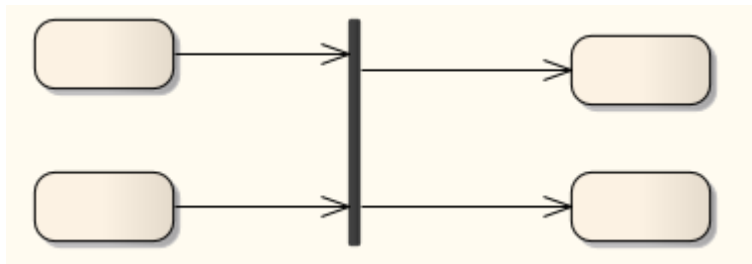
See *UML Superstructure Specification, v2.1.1, figure 12.95 p. 377.*

Join the flow of a number of concurrent flows:



See *UML Superstructure Specification, v2.1.1, figure 12.103, p. 384.*

Join and Fork a number of incoming flows to a number of outgoing flows:



### Toolbox Icon



### OMG UML Specification

#### Fork

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 376*) states:

*A fork node is a control node that splits a flow into multiple concurrent flows... A fork node has one incoming edge and multiple outgoing edges.*

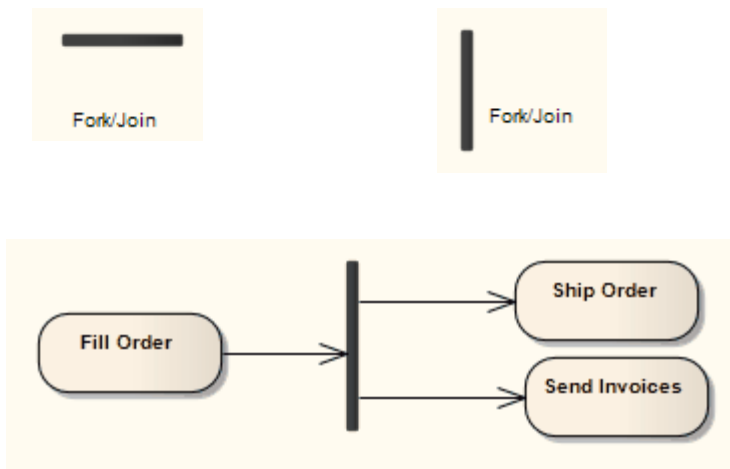
#### Join

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 381-382*) states:

*A join node is a control node that synchronizes multiple flows... A join node has multiple incoming edges and one outgoing edge.*

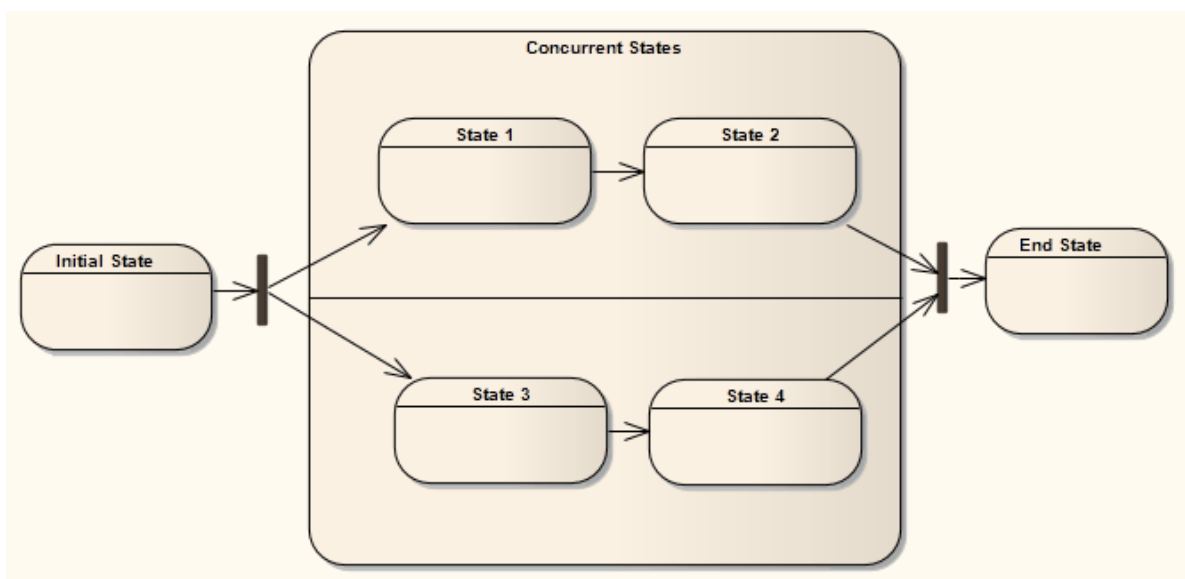


### 2.1.18.1 Fork



See *UML Superstructure Specification, v2.1.1, figure 12.95 p. 377.*

These elements are used in both [Activity](#)<sup>[54]</sup> and [State Machine](#)<sup>[94]</sup> diagrams. With respect to State Machine diagrams, a [Fork pseudo-state](#)<sup>[137]</sup> signifies that its incoming transition comes from a single state, and it has multiple outgoing transitions. These transitions must occur concurrently, requiring the use of concurrent [regions](#)<sup>[128]</sup>, as depicted below in the [Composite State](#)<sup>[130]</sup>. Unlike [Choice](#)<sup>[95]</sup> or [Junction](#)<sup>[122]</sup> pseudo-states, Forks must not have triggers or guards. The following diagram demonstrates a Fork pseudo-state dividing into two concurrent regions, which then return to the *End State* via the [Join](#)<sup>[115]</sup> pseudo-state.

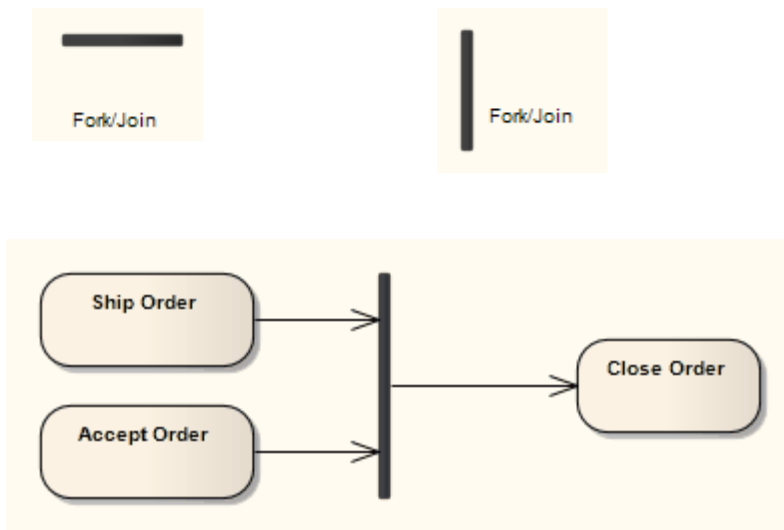


### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 538*) states:

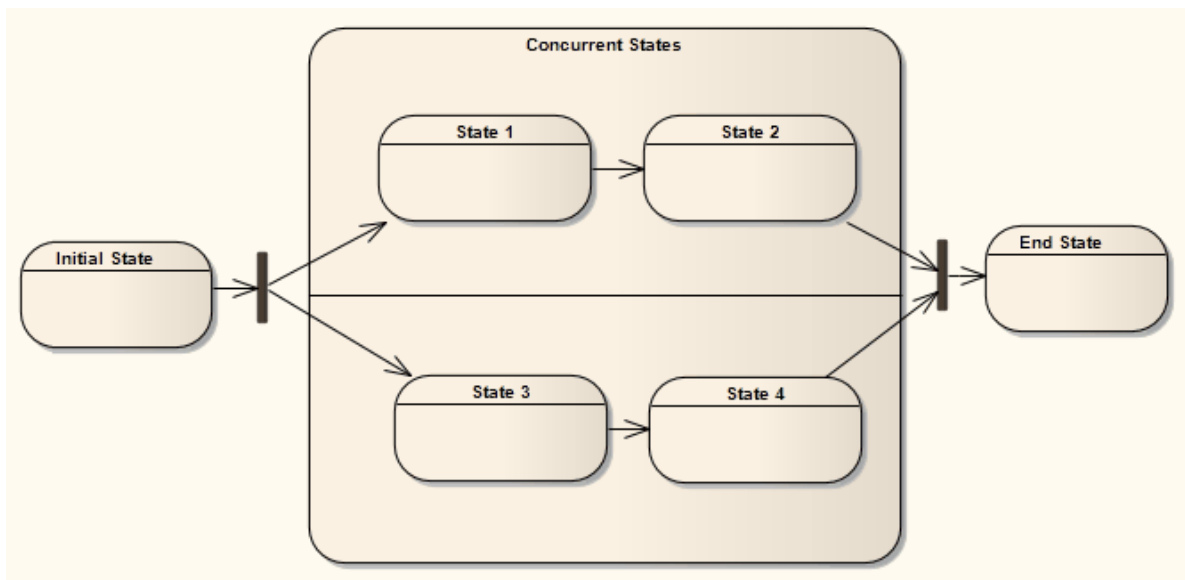
*Fork vertices serve to split an incoming transition into two or more transitions terminating on orthogonal target vertices (i.e. vertices in different regions of a composite state). The segments outgoing from a fork vertex must not have guards or triggers.*

## 2.1.18.2 Join



See *UML Superstructure Specification, v2.1.1, figure 12.103, p. 384.*

The *Join* element is used by [Activity](#)<sup>[5]</sup> and [State Machine](#)<sup>[9]</sup> diagrams. The above example illustrates a Join transition between Activities. With respect to State Machine diagrams, a Join *pseudo-state*<sup>[13]</sup> indicates multiple [States](#)<sup>[129]</sup> concurrently transitioning into the Join and onto a single State. Unlike [Choice](#)<sup>[95]</sup> or [Junction](#)<sup>[122]</sup> pseudo-states, Joins must not have triggers or guards. The following diagram demonstrates a [Fork](#)<sup>[114]</sup> pseudo-state dividing into two concurrent [Regions](#)<sup>[128]</sup>, which then return to the *End State* via the Join.



## OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 538*) states:

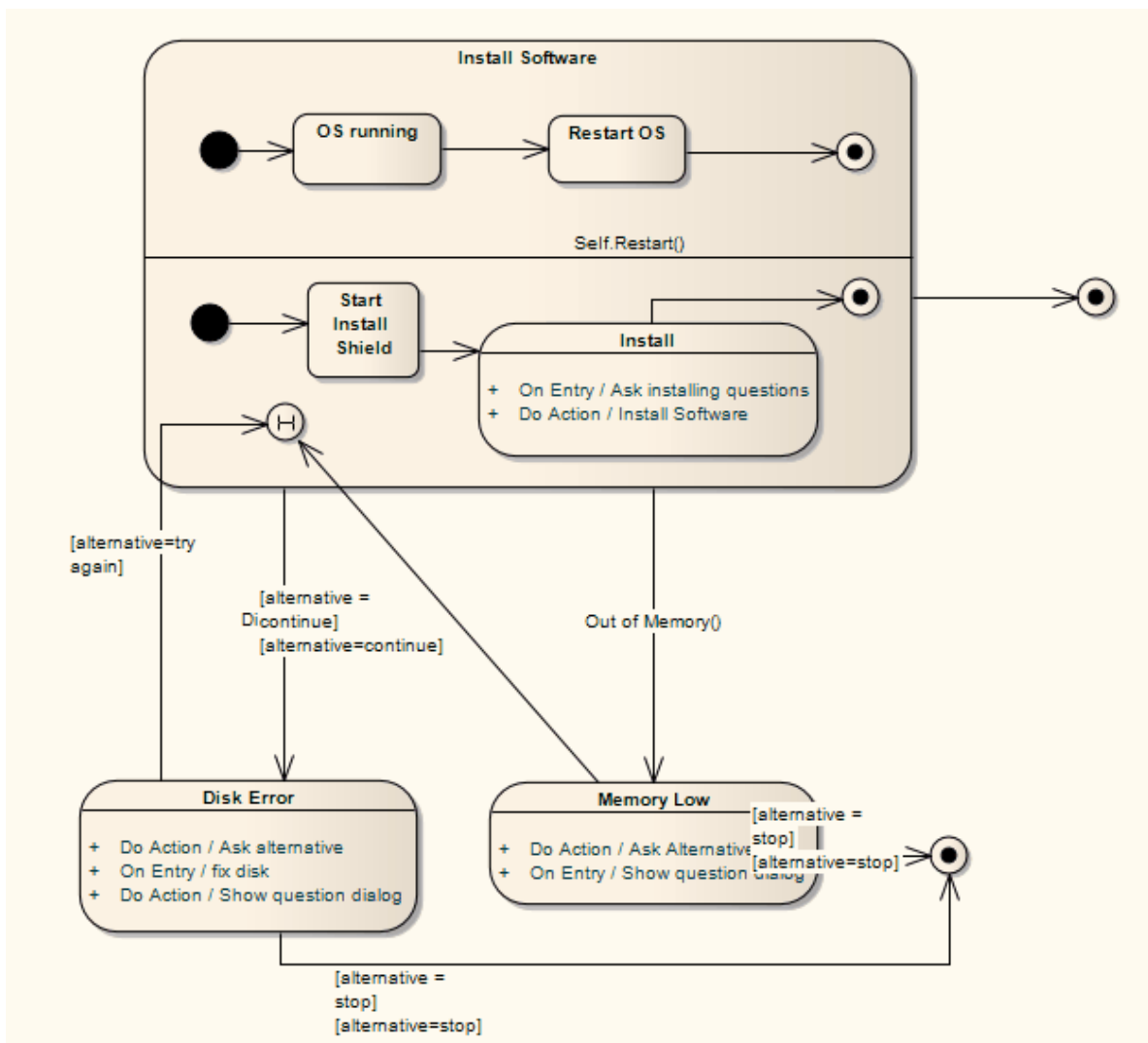
*Join vertices serve to merge several transitions emanating from source vertices in different orthogonal regions. The transitions entering a join vertex cannot have guards or triggers.*

### 2.1.19 History



There are two types of *History pseudo-state*<sup>[13]</sup> defined in UML: *shallow* and *deep history*. A shallow History sub-state is used to represent the most recently active sub-state of a *Composite State*<sup>[130]</sup>; this pseudo-state does not recurse into this sub-state's active configuration, should one exist. A single connector can be used to depict the default shallow History state, in case the Composite State has never been entered.

A deep History sub-state, in contrast, reflects the most recent active configuration of the Composite State. This includes active sub-states of all regions, and recurses into those sub-states' active sub-states, should they exist. At most one deep history and one shallow history can dwell within a composite state.



#### Toolbox Icon



## OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 537*) states:

... *deepHistory* represents the most recent active configuration of the composite state that directly contains this pseudostate (e.g., the state configuration that was active when the composite state was last exited). A composite state can have at most one deep history vertex. At most one transition may originate from the history connector to the default deep history state. This transition is taken in case the composite state had never been active before. Entry actions of states entered on the path to the state represented by a deep history are performed.

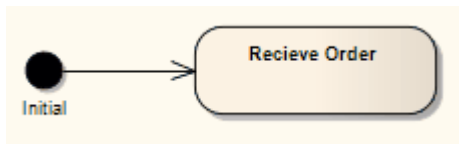
... *shallowHistory* represents the most recent active substate of its containing state (but not the substates of that substate). A composite state can have at most one shallow history vertex. A transition coming into the shallow history vertex is equivalent to a transition coming into the most recent active substate of a state. At most one transition may originate from the history connector to the default shallow history state. This transition is taken in case the composite state had never been active before. Entry actions of states entered on the path to the state represented by a shallow history are performed.

### 2.1.20 Initial



The *Initial* element is used by [Activity](#)<sup>[5]</sup> and [State Machine](#)<sup>[9]</sup> diagrams. In Activity diagrams, it defines the start of a flow when an [Activity](#)<sup>[90]</sup> is invoked. With State Machines, the Initial element is a [pseudo-state](#)<sup>[13]</sup> used to denote the default state of a [Composite State](#)<sup>[130]</sup>; there can be one Initial vertex in each [Region](#)<sup>[128]</sup> of the Composite State.

This simple example shows the start of a flow to receive an order.



See *UML Superstructure Specification, v2.1.1, Figure 12.97, p. 378*.

The activity flow is completed by a [Final](#)<sup>[110]</sup> or [Flow Final](#)<sup>[111]</sup> node.

#### Note:

Moving a diagram generally does not affect the location of elements in packages. If you move a diagram out of one package into another, all the elements in the diagram remain in the original package. However, Initial elements are used only within one diagram, have no meaning outside that diagram, and are never re-used in any other diagram. Therefore, if you move a diagram containing these elements, they **are** moved to the new parent package with the diagram.

### Toolbox Icon



## OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 537*) states:

An *initial pseudostate* represents a default vertex that is the source for a single transition to the default state of a composite state. There can be at most one initial vertex in a region. The outgoing transition from the initial

vertex may have a behavior, but not a trigger or guard.

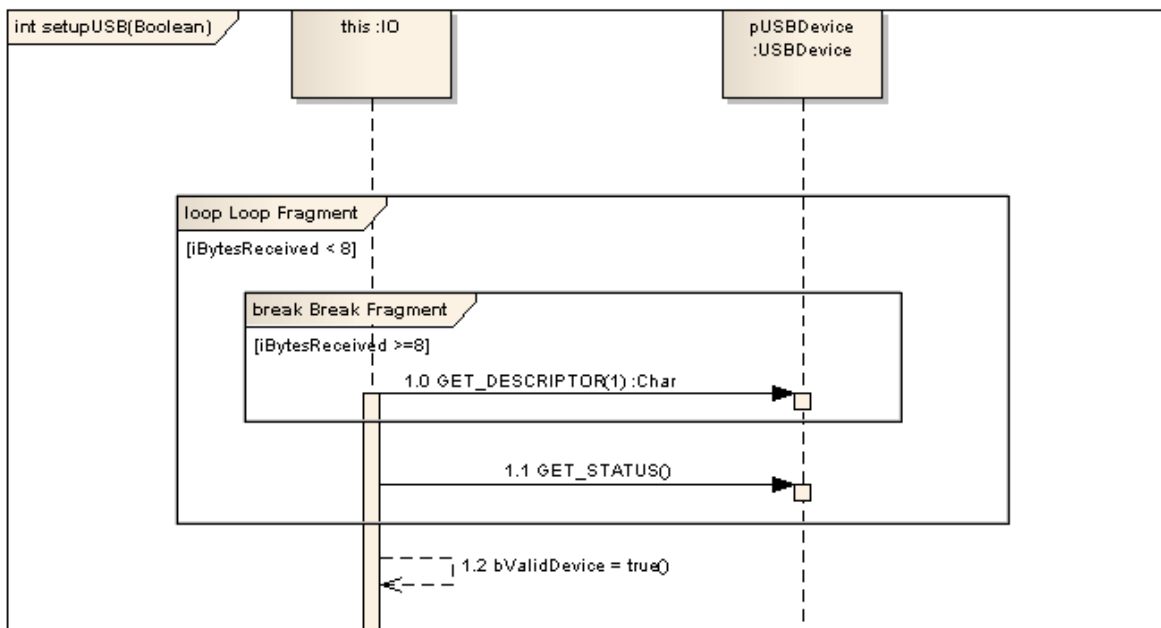
The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 378*) also states:

*An initial node is a control node at which flow starts when the activity is invoked.*

### 2.1.21 Interaction

An *Interaction* element is used to describe a system, representing its interactions at varying levels of detail, for review not only by design professionals but also by end users and stakeholders. An Interaction element can contain the following types of diagram:

- [Sequence](#) <sup>[39]</sup>
- [Interaction Overview](#) <sup>[52]</sup>
- [Communication](#) <sup>[49]</sup>
- [Timing](#) <sup>[22]</sup>.

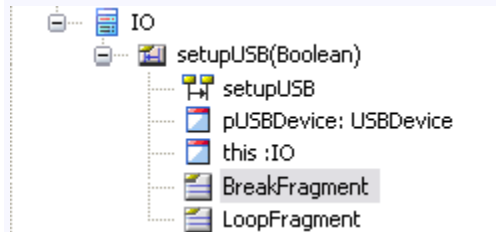


An Interaction element in Enterprise Architect is treated as a behavior of the classifier it is encapsulated within. It can have parameters and return types, which are modeled using the **Behavior** tab of the Interaction element's **Properties** dialog. The element is interpreted as a method of the containing Class in the generated code (see the *Code Generation From Behavioral Model* topic in *Code Generation from UML Models*).

An Interaction element can also be set as the classifier for an [Interaction Occurrence](#) <sup>[119]</sup> in a Sequence diagram, or for a [Call Behavior Action](#) <sup>[81]</sup> in an Activity diagram. Establishing such an association (between a *behavior* and a *behavior call*) facilitates adding *arguments* that can be individually mapped to the associated behavior's parameters (see the *Behavioral Modeling* section of *UML Modeling With Enterprise Architect - UML Modeling Tool*).

**Note:**

The behavioral code generation engine expects the Sequence diagram and all its associated messages and interaction fragments to be encapsulated within an Interaction element (such as *setupUSB* in the example below).



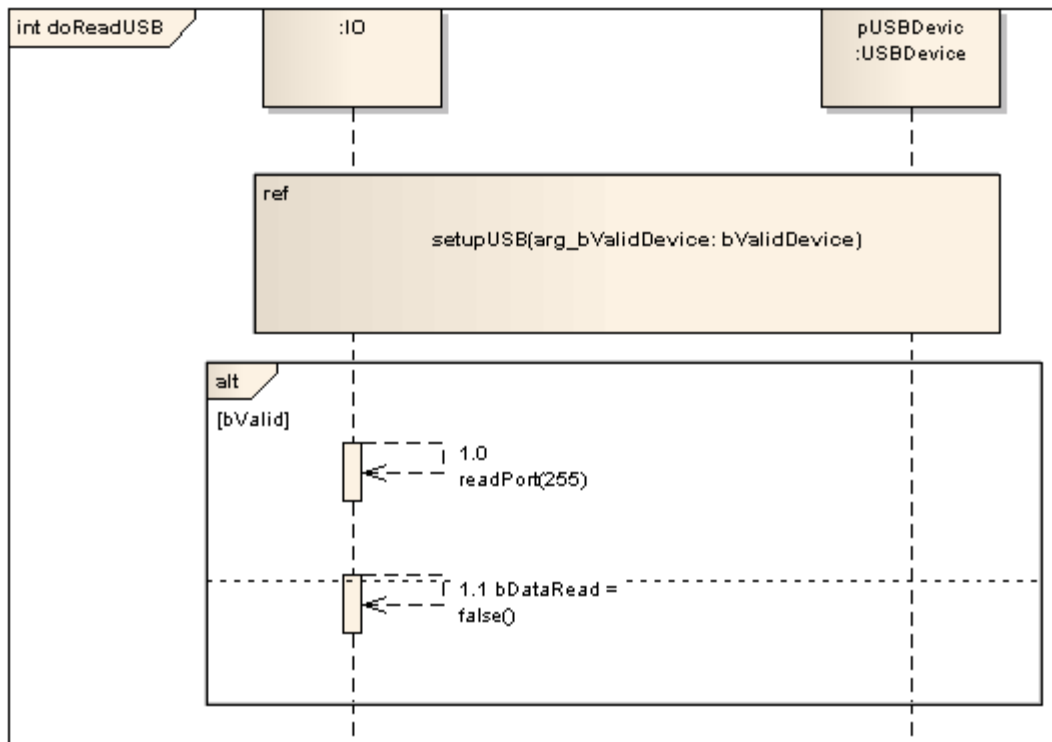
(The IO Class shown above is available in the EAExample model, under *Systems Engineering Model | Implementation Model | Software*.)

### 2.1.22 Interaction Occurrence

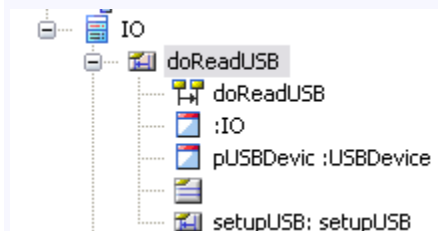


An *Interaction Occurrence* (or *InteractionUse*) is a reference to an existing *Interaction* ([Sequence](#)<sup>[41]</sup>) diagram. Interaction Occurrences are visually represented by a frame, with **ref** in the frame's title space. The diagram name is indicated in the frame contents. To create an Interaction Occurrence, simply open a Sequence diagram (preferably contained within an [Interaction element](#)<sup>[118]</sup>) and drag another Sequence diagram (also preferably contained within an Interaction element) into its workspace. A dialog displays, providing configuration options. The resulting Interaction Occurrence acts as an invocation of the original Interaction. You use the **Call** tab of the element **Properties** dialog to set up the actual arguments of the Interaction and also to change to a different associated Interaction element.

The following figure illustrates the use of an Interaction Occurrence in another Interaction (Sequence) diagram. You can display the sequence represented by the Interaction Occurrence by double-clicking on the element.

**Note:**

The behavioral code generation engine expects the Sequence diagram and all its associated messages and interaction fragments to be encapsulated within an Interaction element (such as `doReadUSB` in the example below).

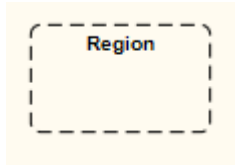
**OMG UML Specification**

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 423*) refers to an Interaction Occurrence as an **InteractionUse**, and states:

*An InteractionUse refers to an Interaction. The InteractionUse is a shorthand for copying the contents of the referred Interaction where the InteractionUse is. To be accurate the copying must take into account substituting parameters with arguments and connect the formal gates with the actual ones.*

*It is common to want to share portions of an interaction between several other interactions. An InteractionUse allows multiple interactions to reference an interaction that represents a common portion of their specification.*

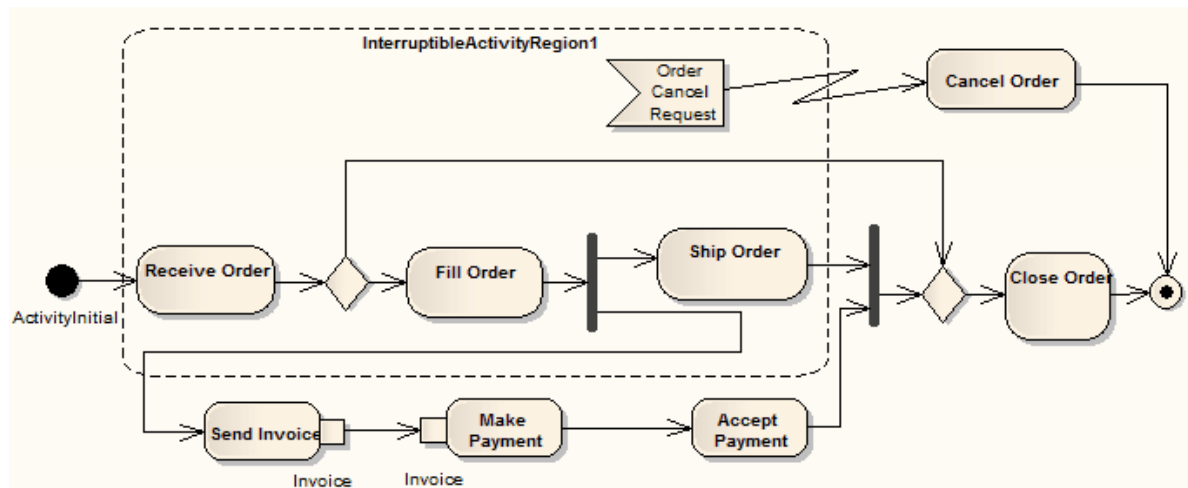
### 2.1.23 Interruptible Activity Region



You [create](#)<sup>[122]</sup> an *Interruptible Activity Region* as one variant of a [Region](#)<sup>[128]</sup> (the other is an [Expansion Region](#)<sup>[107]</sup>).

In an [Activity diagram](#)<sup>[5]</sup>, an Interruptible Activity Region surrounds a group of [Activity](#)<sup>[90]</sup> elements, all affected by certain interrupts in such a way that all tokens passing within the region are terminated should the interruption(s) be raised. Any processing occurring within the bounds of an Interruptible Activity Region is terminated when a flow is instigated across an interrupt flow to an external element.

The example below illustrates that an order cancellation kills any processing of the order at the receipt, filling or shipping stage.



See *UML Superstructure Specification, v2.1.1, figure 12.100, p. 381*.

To create an Interruptible Activity Region, click on this [Add an Interruptible Activity Region](#)<sup>[122]</sup> link.

#### Toolbox Icon



#### OMG UML Specification

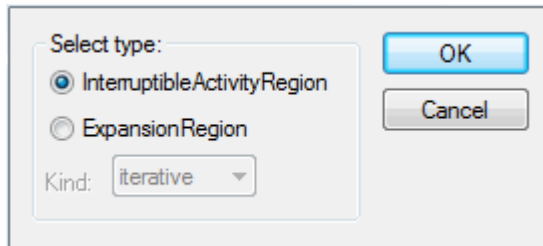
The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 380*) states:

*An interruptible region contains activity nodes. When a token leaves an interruptible region via edges designated by the region as interrupting edges, all tokens and behaviors in the region are terminated.*



### 2.1.23.1 Add Interruptible Activity Region

When you add a *Region* element to an Activity diagram, the following prompt displays:



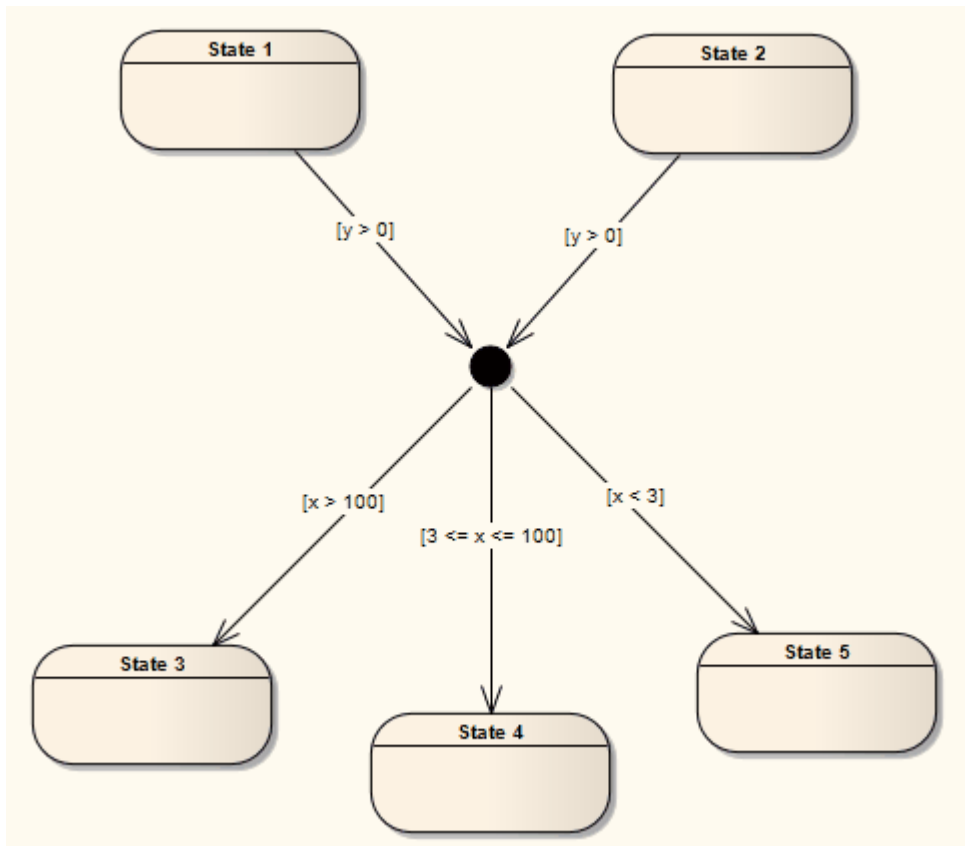
The **Select type** panel defaults to [InterruptibleActivityRegion](#)<sup>[12]</sup> and the **Kind** field is disabled. Click on the **OK** button.

### 2.1.24 Junction

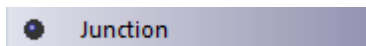


*Junction pseudo-states*<sup>[13]</sup> are used to design complex transitional paths in *State Machine*<sup>[9]</sup> diagrams. A Junction can be used to combine or merge multiple paths into a shared transition path. Alternatively, a Junction can split an incoming path into multiple paths, similar to a *Fork*<sup>[114]</sup> pseudostate. Unlike Forks or *Joins*<sup>[115]</sup>, Junctions can apply guards to each incoming or outgoing transition, such that if the guard expression is false, the transition is disabled.

The following example illustrates how guards can be applied to transitions coming into or out of a Junction pseudo-state.



### Toolbox Icon

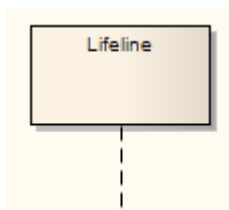


### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 538*) states:

*... junction vertices are semantic-free vertices that are used to chain together multiple transitions. They are used to construct compound transition paths between states. For example, a junction can be used to converge multiple incoming transitions into a single outgoing transition representing a shared transition path (this is known as a merge). Conversely, they can be used to split an incoming transition into multiple outgoing transition segments with different guard conditions. This realizes a static conditional branch. (In the latter case, outgoing transitions whose guard conditions evaluate to false are disabled. A predefined guard denoted "else" may be defined for at most one outgoing transition. This transition is enabled if all the guards labeling the other transitions are false.) Static conditional branches are distinct from dynamic conditional branches that are realized by choice vertices.*

#### 2.1.25 Lifeline



A *Lifeline* is an individual participant in an interaction (that is, Lifelines cannot have multiplicity). A Lifeline represents a distinct connectable element. To specify that representation within Enterprise Architect, right-click on the Lifeline and select the **Advanced | Instance Classifier** context menu option. The **Select <Item>** dialog displays which you use to locate the required project classifiers.

Lifelines are available in [Sequence](#) <sup>[39]</sup> diagrams. There are different Lifeline elements for [Timing diagrams](#) <sup>[22]</sup> ([State Lifeline](#) <sup>[135]</sup> and [Value Lifeline](#) <sup>[149]</sup> <sup>[149]</sup>); however, although the representation differs between the two diagram types, the meaning of the Lifeline is the same.

### Toolbox Icon



### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p.489*) states:

*A lifeline represents an individual participant in the Interaction. While Parts and StructuralFeatures may have multiplicity greater than 1, Lifelines represent only one interacting entity.*

*Lifeline is a specialization of NamedElement.*

*If the referenced ConnectableElement is multivalued (i.e. has a multiplicity > 1), then the Lifeline may have an expression (the 'selector') that specifies which particular part is represented by this Lifeline. If the selector is omitted this means that an arbitrary representative of the multivalued ConnectableElement is chosen.*

### 2.1.26 Merge



A *Merge Node* brings together a number of alternative flow paths in [Activity](#) <sup>[5]</sup>, [Analysis](#) <sup>[67]</sup> and [Interaction Overview](#) <sup>[52]</sup> diagrams. For example, if a [Decision](#) <sup>[102]</sup> is used after a [Fork](#) <sup>[114]</sup>, the two flows coming out of the Decision must be merged into one before going to a [Join](#) <sup>[115]</sup>; otherwise, the Join waits for both flows, only one of which arrives.

A Merge Node has multiple incoming edges and a single outgoing edge. The edges coming into and out of a Merge Node must be either all [object flows](#) <sup>[230]</sup> or all [control flows](#) <sup>[204]</sup>.

### Toolbox Icon



### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 387*) states:

*A merge node is a control node that brings together multiple alternate flows. It is not used to synchronize concurrent flows but to accept one among several alternate flows.*

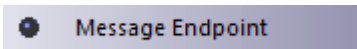
### 2.1.27 Message Endpoint



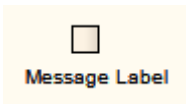
A *Message Endpoint* element defines the termination of a [State](#) <sup>[135]</sup> or [Value](#) <sup>[149]</sup> Lifeline in a [Timing diagram](#)



## Toolbox Icon

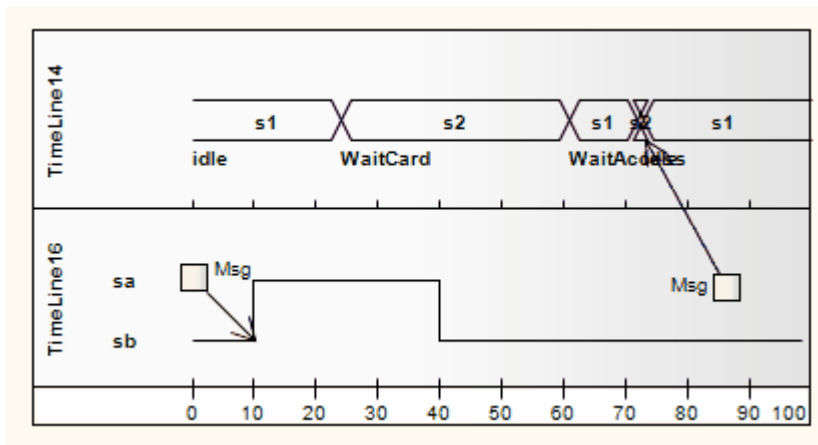


### 2.1.28 Message Label

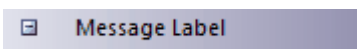


A *Message Label* is an alternative way of denoting [Messages](#) between Lifelines, which is useful for 'uncluttering' [Timing diagrams](#) strewn with messages. To indicate a Message between Lifelines, draw a connector from the source Lifeline into a Message Label. Next, draw a connector from another Message Label to the target Lifeline. Note that the label names must match to reflect that the message occurs between the two Message Labels.

The following diagram illustrates how Message Labels are used to construct a message between Lifelines.



## Toolbox Icon

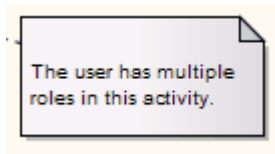


### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 518*) states:

*Labels are only notational shorthands used to prevent cluttering of the diagrams with a number of messages crisscrossing the diagram between Lifelines that are far apart. The labels denote that a Message may be disrupted by introducing labels with the same name.*

### 2.1.29 Note

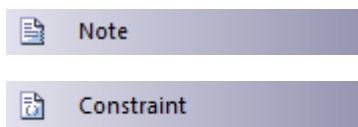


A *Note* element is a textual annotation that can be attached to a set of elements of any other type. The attachment is created separately, using a [Notelink](#)<sup>[230]</sup> connector. Both Note and Notelink are available in any Enterprise Architect diagram, through the **Common** pages of the Enterprise Architect UML **Toolbox** (see *Using Enterprise Architect - UML Modeling Tool*).

A Note is also called a *Comment*.

A *Constraint* is a form of Note, identifying a constraint on other elements. As for a Note, you can connect the Constraint element to other elements using a Notelink connector. This element is just a means of documenting the fact that there are constraints; it has no impact on the other elements. You define the types of constraint in the project reference data (see *UML Model Management*), apply them to the element in the element **Properties** dialog (see *UML Modeling with Enterprise Architect – UML Modeling Tool*), and manage them through the **Scenarios & Requirements** window (see *Using Enterprise Architect - UML Modeling Tool*).

#### Toolbox Icon



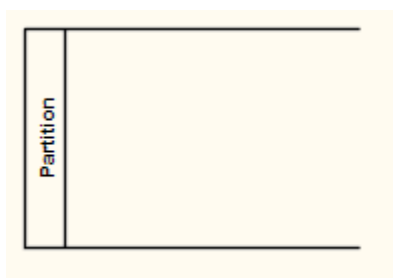
#### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 59*) states:

*A comment gives the ability to attach various remarks to elements. A comment carries no semantic force, but may contain information that is useful to a modeler.*

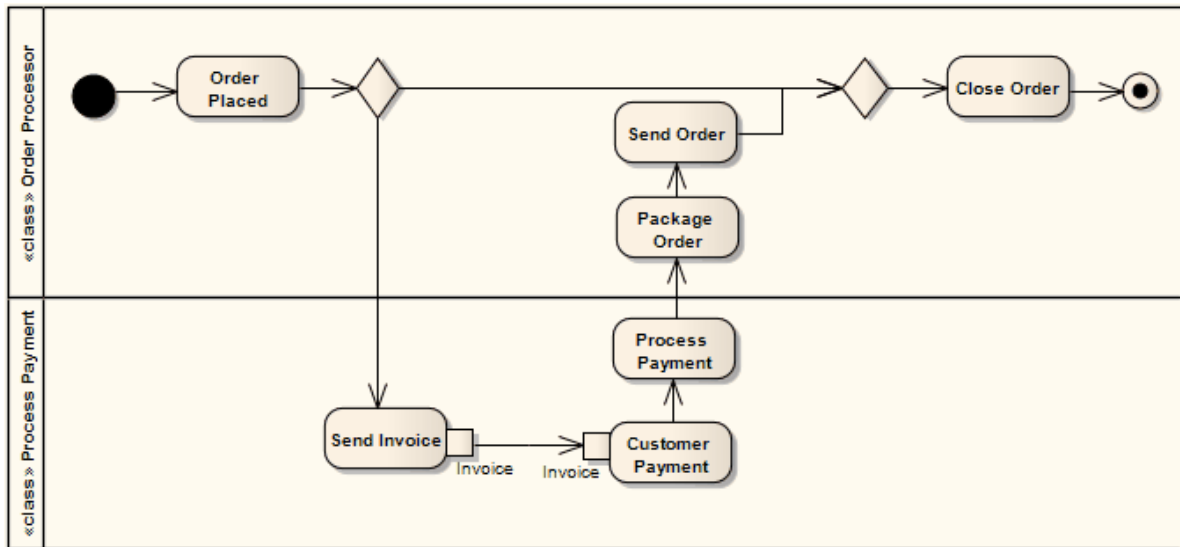
*A comment can be owned by any element.*

### 2.1.30 Partition



[Activity Partitions](#)<sup>[93]</sup> are used to logically organize an [Activity](#)<sup>[90]</sup>. They do not affect the token flow of an [Activity diagram](#)<sup>[54]</sup>, but help structure all or parts of the view.

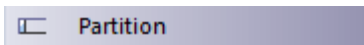
The following example depicts the partitioning between the Classes *Process Payment* and *Order Processor*.



The Partition orientation defaults to horizontal. To turn it into a vertical Partition, right-click on it and select the **Advanced | Vertical Partition** context menu option.

You can neatly align and join the Activity Partitions on a diagram using the element context menu **Dockable** option. For Partitions, the option defaults to selected (see *UML Modeling with Enterprise Architect - UML Modeling Tool*).

### Toolbox Icon



### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 341*) states:

*Partitions divide the nodes and edges to constrain and show a view of the contained nodes. Partitions can share contents. They often correspond to organizational units in a business model. They may be used to allocate characteristics or resources among the nodes of an activity.*

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 341*) also states:

*An activity partition is a kind of activity group for identifying actions that have some characteristic in common.*

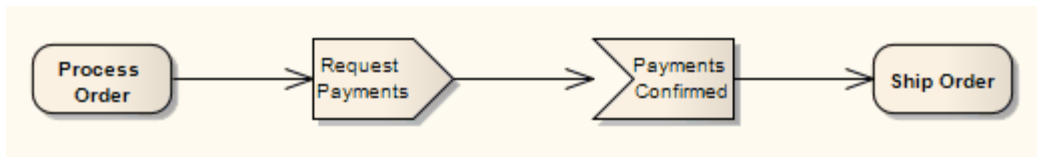
#### 2.1.31 Receive



A *Receive* element is used to define the acceptance or receipt of a request, in an [Activity diagram](#)<sup>[5]</sup>. Movement from a *Receive* element occurs only once receipt is fulfilled according to its specification. The *Receive* element comes in two forms:

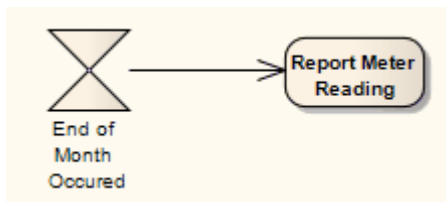
- *Accept Event Action* element (pennant shape)
- *Accept Time Event Action* element (hourglass shape)

The following example reflects a payment process on an order. Upon receiving the payment (from *Request Payments*, a [Send](#)<sup>[129]</sup> element), the payment is confirmed and the flow continues to ship the order.



See *UML Superstructure Specification, v2.1.1, figure 12.26, p. 312.*

To depict an Accept Time Event, use the standard Receive element from the Enterprise Architect UML **Toolbox**. Right-click on this element, and select the **Advanced | Accept Time Event context** menu option. The following example shows the hourglass-shaped Accept Time Event Action:



See *UML Superstructure Specification, v2.1.1, figure 12.27, p. 312.*

### Toolbox Icon

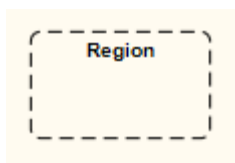


### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 239*) states:

*AcceptEventAction is an action that waits for the occurrence of an event meeting specified conditions.*

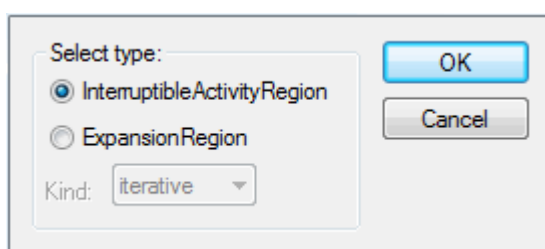
### 2.1.32 Region



Enterprise Architect supports two types of *Region* element:

- [Expansion Region](#)<sup>[107]</sup>
- [Interruptible Activity Region](#)<sup>[127]</sup>

When you add a Region element to an [Activity diagram](#)<sup>[57]</sup>, the following prompt appears. You use this to select the Region type.



## Toolbox Icon

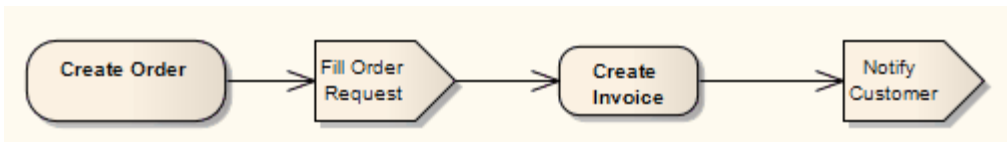


### 2.1.33 Send



The *Send* element is used to depict the action of sending a signal, in an [Activity diagram](#)<sup>[5]</sup>. It is the opposite of a [Receive](#)<sup>[12]</sup> element.

The following example shows an order being processed, where a signal is sent to fill the processed order and, upon creation of the resulting invoice, a notification is sent to the customer.



See *UML Superstructure Specification, v2.1.1, figure 12.132, p. 408.*

## Toolbox Icon



## OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 285*) states:

*SendObjectAction is an action that transmits an object to the target object, where it may invoke behavior such as the firing of state machine transitions or the execution of an activity. The value of the object is available to the execution of invoked behaviors. The requestor continues execution immediately. Any reply message is ignored and is not transmitted to the requestor.*

### 2.1.34 State



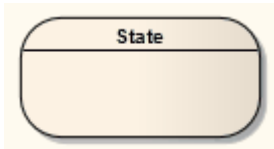
A *State* represents a situation where some invariant condition holds; this condition can be static (waiting for an event) or dynamic (performing a set of activities). State modeling is usually related to [Classes](#)<sup>[152]</sup>, and describes the enableable states a Class or element can be in and the transitions that enable the element to move there. There are two types of State: *Simple States* and [Composite States](#)<sup>[130]</sup>, both created from the State element from the Enterprise Architect UML [Toolbox](#).

Furthermore, there are [pseudo-states](#)<sup>[13]</sup>, resembling some aspect of a State but with a pre-defined implication. Pseudo-states model complex transitional paths, and classify common [State Machine](#)<sup>[9]</sup> behavior.



You can define entry, internal and exit actions for a State using operations (see *UML Modeling with Enterprise Architect - UML Modeling Tool*).

If a State element has features such as attributes or operations, the depiction of the element in a diagram has a line under the element name. This line persists if the features are hidden. The line also displays if the **Show State Compartment** checkbox is selected on the **Objects** page of the **Options** dialog (**Tools | Options | Objects**).



### Toolbox Icon



### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 546*) states:

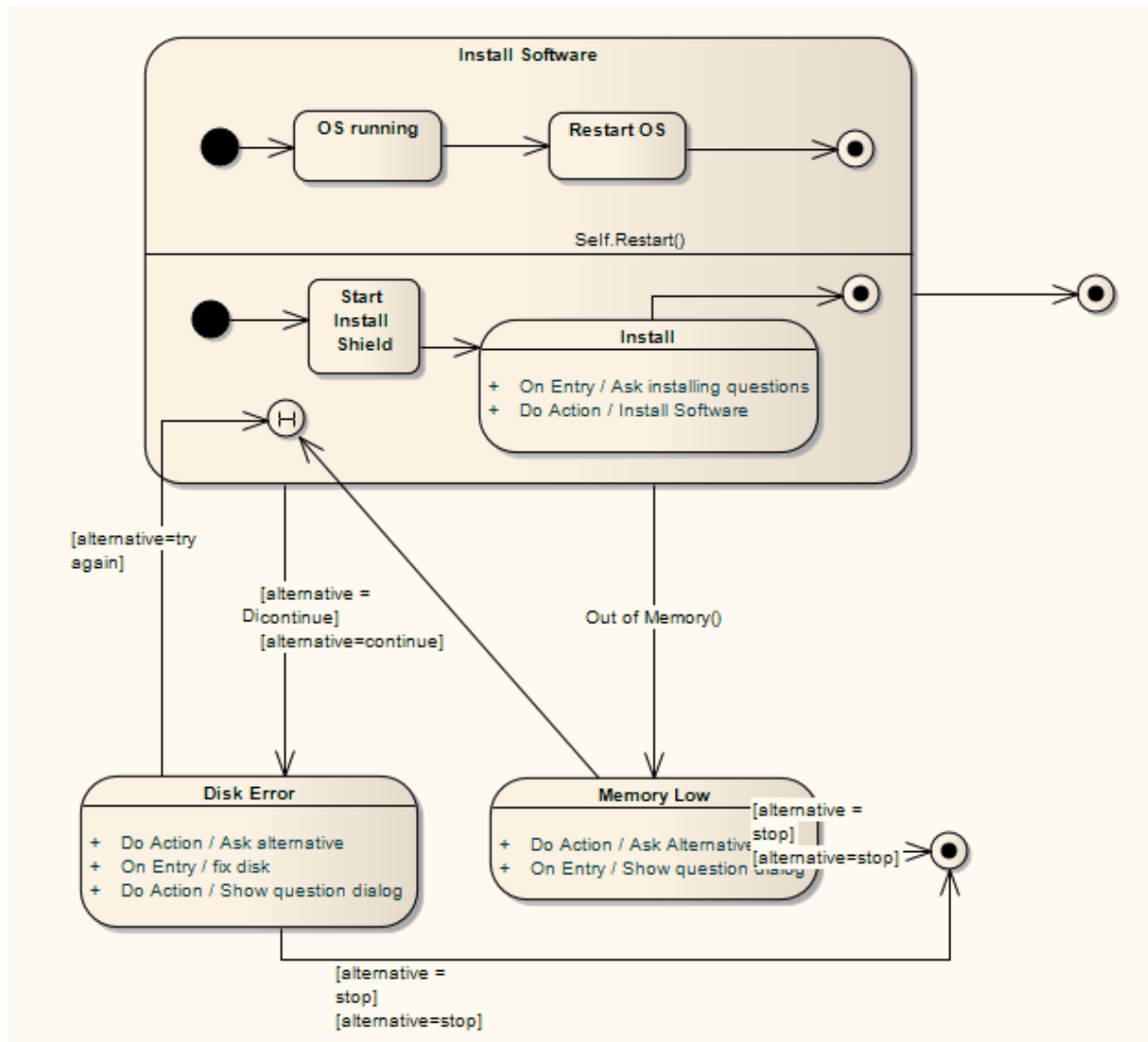
*A state models a situation during which some (usually implicit) invariant condition holds. The invariant may represent a static situation such as an object waiting for some external event to occur. However, it can also model dynamic conditions such as the process of performing some activity (i.e., the model element under consideration enters the state when the activity commences and leaves it as soon as the activity is completed).*

#### 2.1.34.1 Composite State

*Composite States* are composed *within* the [State Machine diagram](#)<sup>[9]</sup> by expanding a [State](#)<sup>[129]</sup> element, adding [Regions](#)<sup>[12]</sup> if applicable, and dragging further State elements, related elements and connectors within its boundaries. The internal State elements are then referred to as *Sub-states*.

(You can also define a State element, as with many other types of element, as a [composite element](#)<sup>[180]</sup>; this then has a hyperlink to a child diagram that can be another State Machine diagram or other type of diagram elsewhere in the model.)

Composite States can be orthogonal, if Regions are created. If a Composite State is orthogonal, its entry denotes that a single Sub-state is concurrently active in all Regions. The hierarchical nesting of Composite States, coupled with Region use, generates a situation of multiple States concurrently active; this situation is referred to as the *active State configuration*.



### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 478*) states:

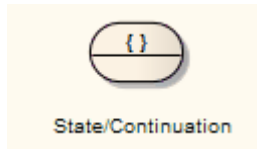
*A composite state either contains one region or is decomposed into two or more orthogonal regions. Each region has a set of mutually exclusive disjoint subvertices and a set of transitions. A given state may only be decomposed in one of these two ways.*

*Any state enclosed within a region of a composite state is called a substate of that composite state. It is called a direct substate when it is not contained by any other state; otherwise it is referred to as an indirect substate.*

*Each region of a composite state may have an initial pseudostate and a final state. A transition to the enclosing state represents a transition to the initial pseudostate in each region. A newly-created object takes its topmost default transitions, originating from the topmost initial pseudostates of each region.*

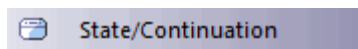
*A transition to a final state represents the completion of activity in the enclosing region. Completion of activity in all orthogonal regions represents completion of activity by the enclosing state and triggers a completion event on the enclosing state. Completion of the topmost regions of an object corresponds to its termination.*

### 2.1.35 State/Continuation



The *State/Continuation* element serves two different purposes for Interaction ([Sequence](#)<sup>[39]</sup>) diagrams, as [State Invariants](#)<sup>[134]</sup> and [Continuations](#)<sup>[132]</sup>. Enterprise Architect prompts you to identify the purpose when you create the element.

#### Toolbox Icon

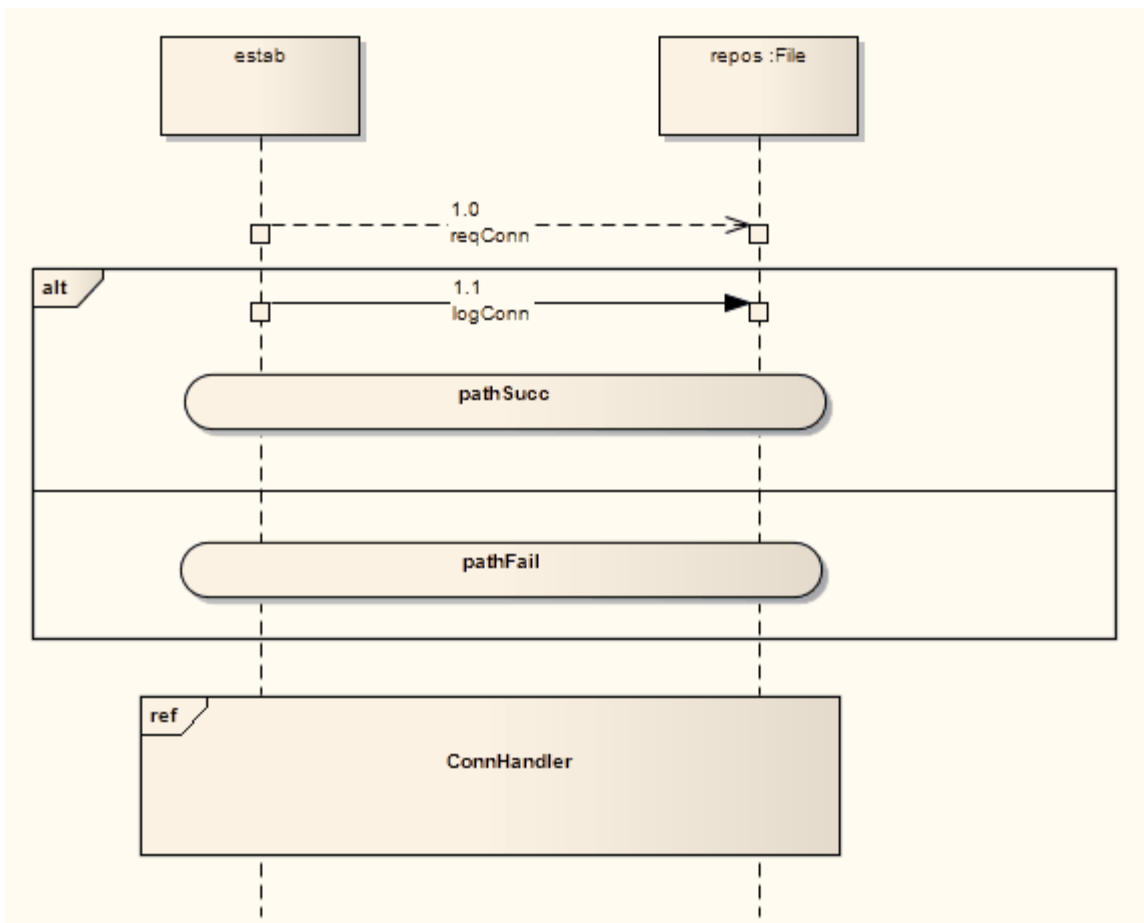


#### 2.1.35.1 Continuation

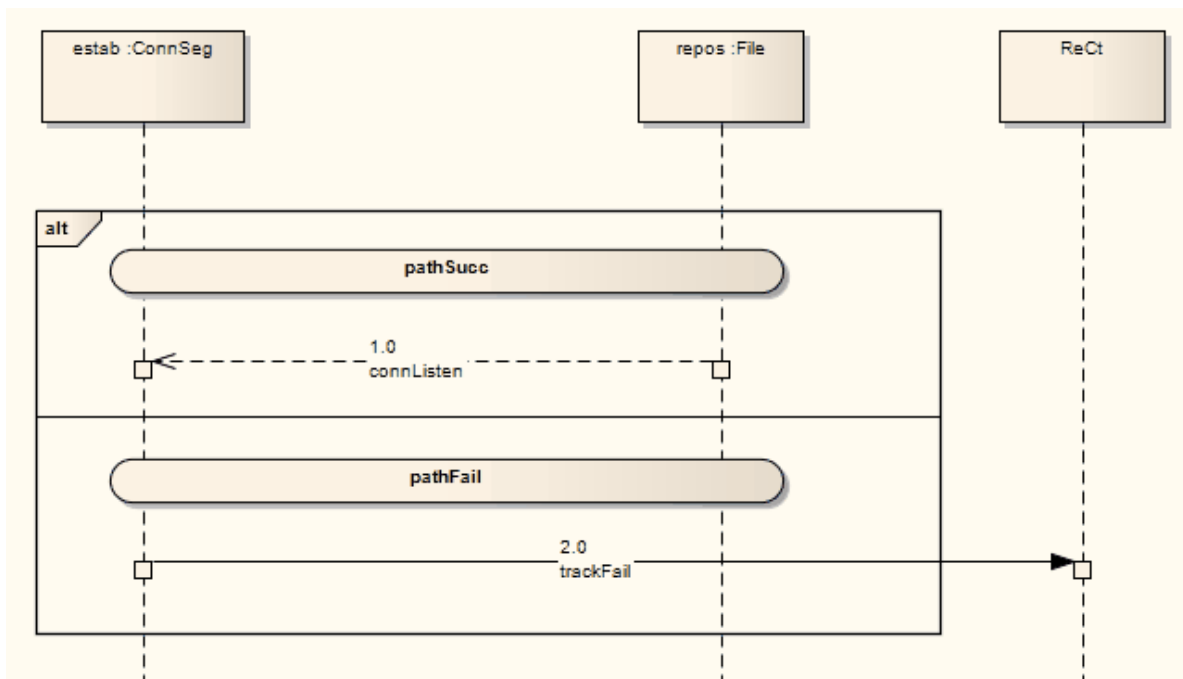
A *Continuation* is used in *seq* and *alt* [Combined Fragments](#)<sup>[96]</sup>, to indicate the branches of continuation an operand follows. To indicate a continuation, end an operand with a Continuation, and indicate the continuation branch with a matching Continuation (same name) preceding the *Interaction Fragment*.

You create a Continuation by dragging the [State/Continuation](#)<sup>[132]</sup> element onto the diagram from the **Interaction Elements** page of the Enterprise Architect UML **Toolbox**.

For the following continuation example, an *alt* Combined Fragment has Continuations *pathSucc* and *pathFail*. These Continuations are located within the [Interaction Occurrence](#)<sup>[119]</sup> *ConnHandler*, which has subsequent events based on the continuation.



The following diagram shows the interaction referenced by the *Interaction Occurrence*.



## OMG UML Specification

The OMG UML specification (*UML Superstructure Specification*, v2.1.1, p. 474) states:

*A Continuation is a syntactic way to define continuations of different branches of an Alternative CombinedFragment. Continuation is intuitively similar to labels representing intermediate points in a flow of control.*

The OMG UML specification (*UML Superstructure Specification*, v2.1.1, p. 474) also states:

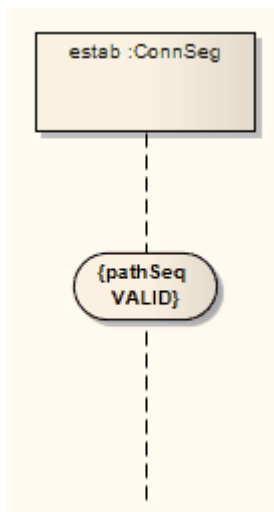
*Continuations have semantics only in connection with Alternative CombinedFragments and (weak) sequencing.*

*If an InteractionOperand of an Alternative CombinedFragment ends in a Continuation with name (say) X, only InteractionFragments starting with the Continuation X (or no continuation at all) can be appended.*

### 2.1.35.2 State Invariant

A *State Invariant* is a condition applied to a [Lifeline](#)<sup>[123]</sup>, which must be fulfilled for the Lifeline to exist. You create a State Invariant by dragging the [State/Continuation](#)<sup>[132]</sup> element onto the diagram from the [Interaction Elements](#) page of the Enterprise Architect UML [Toolbox](#).

The following diagram illustrates a State Invariant.



When a State Invariant is moved near to a Lifeline, it snaps to the center. If the sequence object is dragged left or right, the State Invariant moves with it.

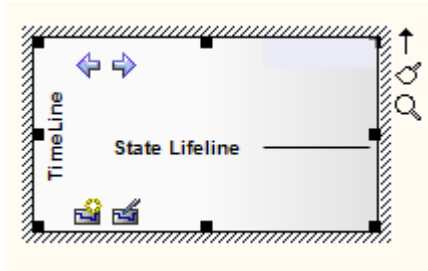
## OMG UML Specification

The OMG UML specification (*UML Superstructure Specification*, v2.1.1, p. 502) states:

*A StateInvariant is a runtime constraint on the participants of the interaction. It may be used to specify a variety of different kinds of constraints, such as values of attributes or variables, internal or external states, and so on.*

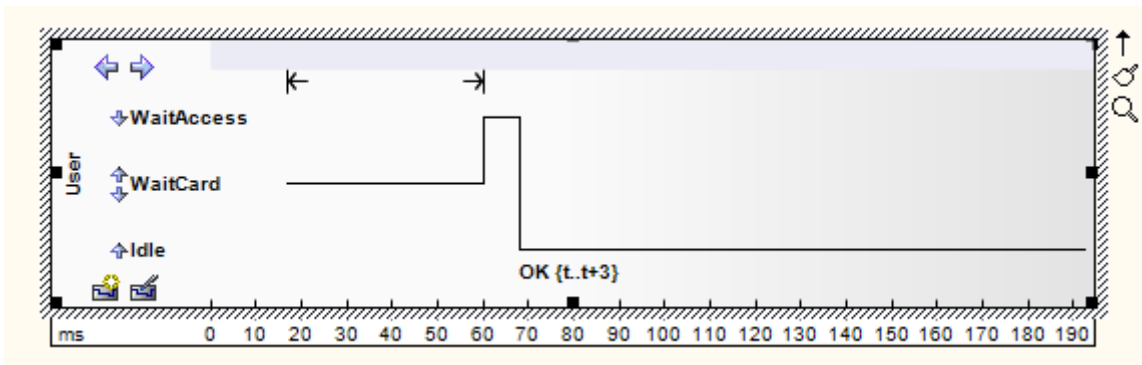
*A StateInvariant is an InteractionFragment and it is placed on a Lifeline.*

### 2.1.36 State Lifeline



A *Lifeline* is the path an object takes across a measure of time, as indicated by the x-axis. There are two sorts: *State Lifelines* (defined here) and *Value Lifelines*<sup>[149]</sup>, both used in *Timing diagrams*<sup>[22]</sup>.

A *State Lifeline* follows discrete transitions between states, which are defined along the y-axis of the timeline. Any transition has optional attributes of timing constraints, duration constraints and observations. An example of a State Lifeline is shown below:



See *UML Superstructure Specification, v2.1.1, figure 14.29, p. 519.*

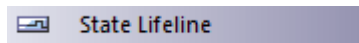
A State Lifeline consists of a set of transition points. Each transition point can be defined with the following properties:

Property	Description
<b>At time</b>	Specifies the starting time for a change of state.
<b>Transition to</b>	Indicates the state to which the lifeline changes.
<b>Event</b>	Describes the occurring event.
<b>Timing constraints</b>	Refers to the time taken for a state to change within a lifeline, or the time taken to transmit a message (e.g. $t..t+3$ ).
<b>Timing observations</b>	Provides information on the time of a state change or sent message.
<b>Duration constraints</b>	Pertains to a lifeline's period at a particular state. The constraint could be instigated by a change of state within a lifeline, or that lifeline's receipt of a message.
<b>Duration observations</b>	Indicates the interval of a lifeline at a particular state, begun from a change in state or message receipt.

In the example diagram above, the **OK** transition point has these properties:

Property	Value
At Time	18 ms
Transition to	Idle
Event	OK
Timing constraints	t..t+3
Timing observations	–
Duration constraints	–
Duration observations	–

### Toolbox Icon



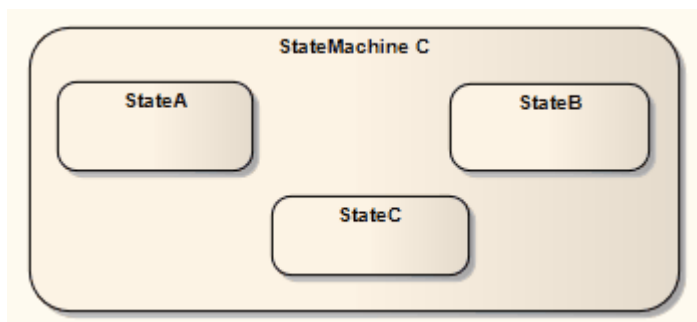
### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 518*) states:

*This is the state of the classifier or attribute, or some testable condition, such as an discrete enumerable value.*

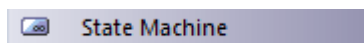
*It is also permissible to let the state-dimension be continuous as well as discrete. This is illustrative for scenarios where certain entities undergo continuous state changes, such as temperature or density.*

### 2.1.37 State Machine



A State Machine element is a container for groups of related State elements. You can create sections of a State Machine diagram, showing the organization of the inter-related State elements, and enclose each section in a State Machine element. You can also create [Regions](#)<sup>[12]</sup> on a State Machine Element.

### Toolbox Icon



### 2.1.38 Structured Activity

*Structured Activity* elements are used in [Activity diagrams](#)<sup>[5]</sup>. A Structured Activity is an activity node that can have subordinate nodes as an independent *Activity Group*. No other Activities or their side effects should interfere with this Activity's processing.

Enterprise Architect provides two forms of Structured Activity - basic and specialized. It also applies the mechanism for creating Structured Activities to creating composite [Activity](#) <sup>[90]</sup> elements quickly and simply.

The two basic Structured Activities are:

- [Structured Activity Node](#) <sup>[138]</sup> - represents an ordered arrangement of executable Activity nodes (Actions, Decisions, Merges and so on) that can include branched and nested nodes; this is the base element from which the other types of Structured Activity are derived
- [Sequential Node](#) <sup>[138]</sup> - represents a sequential arrangement of executable Activity nodes.

The two specialized Structured Activities are used to effectively model discreet patterns within an activity graph, defined in Clauses or Partitions:

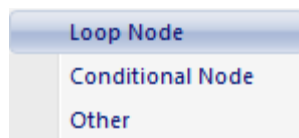
- [Conditional node](#) <sup>[139]</sup> - represents an arrangement of Actions and Activities where choice determines which Activities are performed
- [Loop node](#) <sup>[139]</sup> - represents a sequence of Actions and Activities that are - or can be - repeated on the same object.

#### Note:

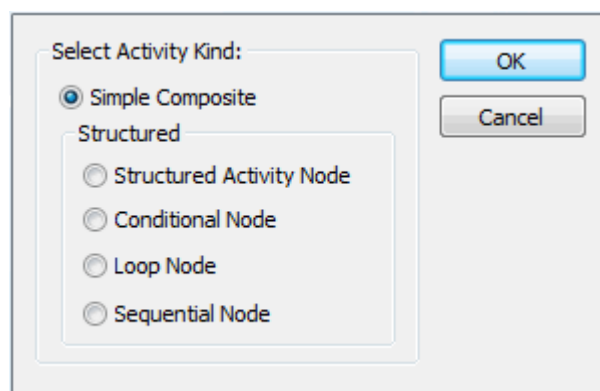
All four Structured Activity Nodes are created as [composite elements](#) <sup>[180]</sup>. However, for the Loop Node and Conditional Node elements you must create the child element structure on the parent diagram within the node element itself, as for a [Composite State](#) <sup>[130]</sup>. You cannot develop the partitioned structure of the nodes on a child diagram.

For this reason, the **Show Composite Diagram** facility is not available for the Loop Node and Conditional Node. It is also not available on the Structured Activity Node, as this is the base element for the Loop and Conditional Nodes. You can, however, use the two basic nodes as composite elements, and display the child diagram structure on the parent Sequential node.

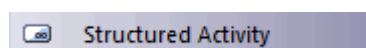
When you create a Structured Activity, by selecting the icon from the [Activity](#) page of the Enterprise Architect UML [Toolbox](#), the following context menu displays:



The first two options specifically create a Loop or Conditional Node. The **Other** option displays the [New Structured Activity](#) dialog, on which you can select to create any of the four nodes, or a simple Composite Activity element.



#### Toolbox Icon





## OMG UML Specification

### Structured Activity Node

The OMG UML specification (*UML Superstructure Specification*, v2.1.1, p. 409) states:

*A structured activity node is an executable activity node that may have an expansion into subordinate nodes as an ActivityGroup. The subordinate nodes must belong to only one structured activity node, although they may be nested.*

*A structured activity node represents a structured portion of the activity that is not shared with any other structured node, except for nesting.*

### Sequential Node

The OMG UML specification (*UML Superstructure Specification*, v2.1.1, p. 408) states:

*A sequence node is a structured activity node that executes its actions in order.*

### Loop Node

The OMG UML specification (*UML Superstructure Specification*, v2.1.1, pp. 384-385) states:

*A loop node is a structured activity node that represents a loop with setup, test, and body sections.*

*Each section is a well-nested subregion of the activity whose nodes follow any predecessors of the loop and precede any successors of the loop. The test section may precede or follow the body section. The setup section is executed once on entry to the loop, and the test and body sections are executed repeatedly until the test produces a false value. The results of the final execution of the test or body are available after completion of execution of the loop.*

### Conditional Node

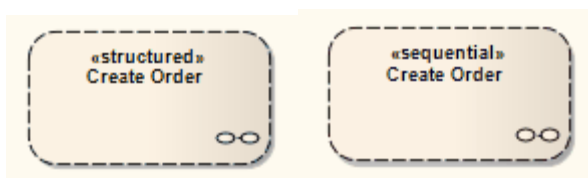
The OMG UML specification (*UML Superstructure Specification*, v2.1.1, p.355) states:

*A conditional node is a structured activity node that represents an exclusive choice among some number of alternatives.*

*A conditional node consists of one or more clauses. Each clause consists of a test section and a body section. When the conditional node begins execution, the test sections of the clauses are executed. If one or more test sections yield a true value, one of the corresponding body sections will be executed. If more than one test section yields a true value, only one body section will be executed. The choice is nondeterministic unless the test sequence of clauses is specified. If no test section yields a true value, then no body section is executed; this may be a semantic error if output values are expected from the conditional node.*

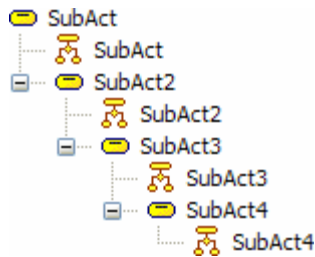
#### 2.1.38.1 Structured and Sequential Nodes

On a diagram, *Structured* and *Sequential* Activity Nodes have broken borders and composite diagram icons, as shown below:



To display the Activity diagram represented by a Structured or Sequential Activity Node element, double-click on the element.

Structured Activity Node elements can point to child diagrams that themselves contain or consist of Structured Activity elements; that is, the Structured Activity elements are nested, as shown in the section of [Project Browser](#) below.

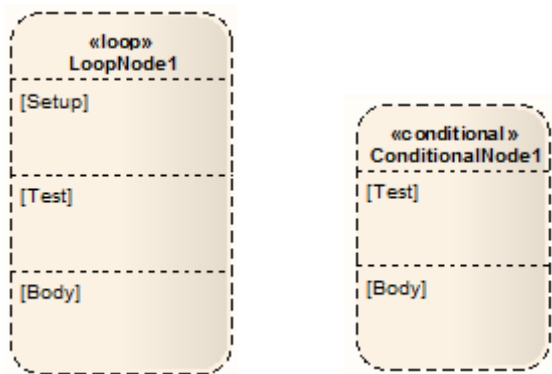


### 2.1.38.2 Loop and Conditional Nodes

A *Loop* Structured Activity Node by default has *Setup*, *Test* and *Body* partitions. The Setup partition is executed once on entry to the Loop, and the Test and Body partitions are executed repeatedly until the Test produces a false value. The results of the final execution of the Test or Body are available after execution of the Loop is complete.

A *Conditional* Structured Activity Node has a default *Clause* containing *Test* and *Body* partitions. Further Clauses can be added if required.

The two elements are depicted on an Activity diagram as shown below:



You define the Loop or Condition nodes by dragging other Activity diagram elements from the [Toolbox](#) page into the appropriate partition of the element, and linking and organizing the structure as required. The elements are aligned on the top left of the partition, so that resizing the node maintains the organization of the structure within and between the partitions. If you try to shrink the node below the structure size, the node automatically defaults to the 'best fit' size.

### Conditional Node

When you [create](#) <sup>138</sup> a Conditional Node, the element [Properties](#) dialog displays. Much of this you can complete as for any other element. However, for the Conditional Node the dialog also has a [Condition](#) tab, as shown below:

General Condition Requirements Constraints Links Scenarios Files

Must Isolate  Is Assured  Is Determinate

Result

::Activity Example::ConditionalNode3.OutputPin8

Add Del

Clause(s)

Clause Name	Predecessor	Successor
Clause1		Clause2
Clause2	Clause1	Clause3
Clause3	Clause2	

Add Delete Save

Decider Development Model::Activity Example::ConditionalNode3.OutputPin9

Body Output

::Activity Example::ConditionalNode3.OutputPin10

Add Del

Nodes

Test  Body

::Activity Example::ConditionalNode3.Action2

OK Cancel Apply Help

Add an [Action Pin](#) <sup>87</sup> as the **Result** for the node, clicking on the **Add** button to display the **Select Pins** dialog (a version of the **Select <Item>** dialog).

On creation, the Conditional Node automatically has one *Clause* containing a **Decider** and **Body Output**, and a **Test** partition and a **Body** partition. You can add further Clauses as required. For each Clause you also add an Action Pin for the **Decider** and for the **Body Output**. Click on the **Save** button to save the Clause definition.

The **Select Pin** dialog reveals only Output pins as appropriate to the context. If the required Action Pin does not already exist, you can click on the **Add New** button on the dialog to automatically create an Output pin under the appropriate parent node.

For the **Result** and **Body Output** entries, you can check on the exact location of each Action Pin by right-clicking on the entry and selecting the **Find in Project Browser** context menu option.

The **Nodes** panel, by default, lists the Actions and Activities contained in the **Test** partition. Click on the **Body** radio button to list the elements contained in the **Body** partition. An element must be completely contained in the **Body** partition to be listed there - if it overlaps with the **Test** partition in any way, it is treated as being part of the **Test** partition.

### Add or Remove Clauses

To add another Clause, click on the **Add** button underneath the **Clause(s)** list. This inserts a new Clause in the list, and identifies which is the preceding, or Predecessor, Clause and (if appropriate) which is the following, or

Successor, Clause. The remaining fields in the **Clause(s)** panel are cleared to enable you to add **Decider** and **Body Output** Action Pins. New Test and Body partitions are immediately added to the element on the diagram, and you can populate these partitions with Activity elements, which are then identified in the **Nodes** panel.

To remove a Clause, highlight it in the list and click on the **Delete** button. This immediately removes the Clause's corresponding partitions from the diagram, along with all their contained Activity elements. Removing a Clause from between two other Clauses adjusts the numerical order; for example, if Clause 2 is removed from between Clause 1 and Clause 3, Clause 3 is renamed as Clause 2, and any further Clauses are also moved up one place.

## Loop Node

When you [create](#)<sup>[136]</sup> a Loop Node, the element **Properties** dialog displays. Much of this you can complete as for any other element. However, for the Loop Node the dialog also has a **Loop** tab, as shown below:

The screenshot shows the 'Loop' tab of the Properties dialog. It includes the following elements:

- Must Isolate:**
- Tested First:**
- Decider:** Development Model::Activity Example::LoopNode2.OutputPin4
- Loop Variable Input:** ::Activity Example::LoopNode2.InputPin1
- Loop Variable:** ::Activity Example::LoopNode2.OutputPin5
- Body Output:** ::Activity Example::LoopNode2.OutputPin6
- Result:** ::Activity Example::LoopNode2.OutputPin7
- Nodes:**
  - Setup
  - Body
  - Test
  - ::Activity Example::LoopNode2.Action24
  - ::Activity Example::LoopNode2.Action23

Add an [Action Pin](#)<sup>[87]</sup> for each of the **Decider**, **Loop Variable Input**, **Loop Variable**, **Body Output** and **Result** fields for the node, in each case clicking on the **Browse** or **Add** button to display the **Select Pins** dialog (a version of the **Select <Item>** dialog). The **Select ActionPin** dialog reveals only Input pins (**Loop Variable Input**) or Output pins as appropriate to the context. If the required Action Pin does not already exist, you can click on the **Add New** button on the dialog to automatically create the Input pin or an Output pin under the appropriate parent node.

You can also check on the exact location of an existing Action Pin by right-clicking on the pin name and selecting the **Find in Project Browser** context menu option.

The **Nodes** panel, by default, lists the Actions and Activities contained in the **Setup** partition. Click on the **Body** or **Test** radio buttons to list the elements contained in the corresponding partitions. An element must be completely contained in a partition to be listed there - if it overlaps with the partition above in any way, it is treated as being part of that partition.

### 2.1.39 Synch

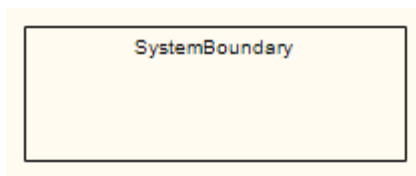


A *Synch* state is useful for indicating that concurrent paths of a [State Machine](#)<sup>[97]</sup> are synchronized. After bringing the paths to a synch state, the emerging transition indicates unison.

#### Toolbox Icon



### 2.1.40 System Boundary

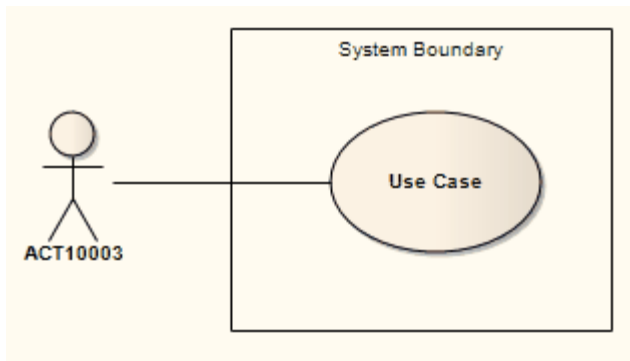


A [System Boundary](#)<sup>[142]</sup> element is a non-UML element used to define *conceptual* boundaries. You can use System Boundaries to help group logically related elements (from a visual perspective, not as part of the UML model).

In the *UML Superstructure Specification, v2.1.1*, System Boundaries are described in the sections on [Use Cases](#)<sup>[146]</sup>, because the System Boundary is often used to indicate the *application* of a Use Case to another entity. In this context, the System Boundary:

- encloses the Use Case, and
- is associated with a classifier such as a [Class](#)<sup>[152]</sup>, [Component](#)<sup>[156]</sup> or Sub-system ([Actor](#)<sup>[94]</sup>) through the **Select <Item>** dialog (see *UML Modeling With Enterprise Architect - UML Modeling Tool*).

By associating the System Boundary - and not the Use Case - with the classifier, the classifier is linked to the Use Case as a *user*, but not as an *owner*.



You can also define a Use Case as the classifier of a System Boundary element, to link the elements enclosed in the System Boundary (such as parts of an Activity diagram) to their representation in a logical Use Case. See [http://www.sparxsystems.com.au/resources/map\\_uc.html](http://www.sparxsystems.com.au/resources/map_uc.html).

The following properties of a System Boundary can be set: the name, [the border style](#)<sup>[144]</sup>, and the number of horizontal or vertical swim lanes.

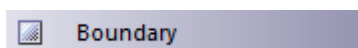
The screenshot shows a configuration dialog box for a System Boundary. It has a 'Name' field containing 'System Boundary'. Below this are radio buttons for 'Border Style' with options: Solid, Dotted, Dashed, and Solid-No Fill (which is selected). To the right of these are two input fields: 'Horizontal Swim Lanes' and 'Vertical Swim Lanes', both containing the value '1'. On the right side of the dialog are buttons for 'OK', 'Cancel', and 'Help'.

A System Boundary element can be marked as *Selectable*, using the element's context menu. When not selectable, you can click within the System Boundary space without activating or selecting the Boundary itself. This is useful when you have many elements within the Boundary and the Boundary makes their selection difficult.

#### Note:

A System Boundary can have an associated image that it displays instead of its default format. Use the **Appearance | Alternate Image** menu option to select an image. (See *UML Modeling with Enterprise Architect – UML Modeling Tool*.)

#### Toolbox Icon



#### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 594*) states:

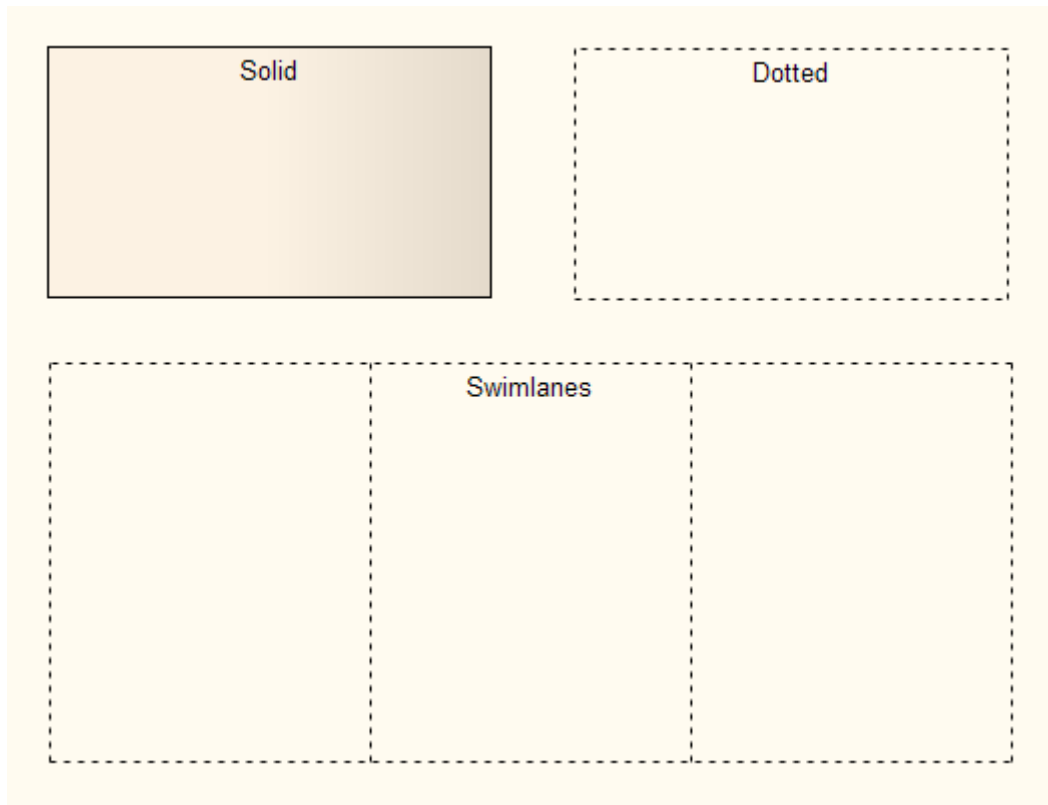
*If a subject (or system boundary) is displayed, the Use Case ellipse is visually located inside the system boundary rectangle. Note that this does not necessarily mean that the subject classifier owns the contained Use Cases, but merely that the Use Case applies to that classifier.*

### 2.1.40.1 Boundary Element Settings

#### Configure Boundary Elements

Boundary elements can be configured to display in different ways. The main differences are:

- Solid border
- Dotted border
- With horizontal or vertical 'swim lanes'; swim lanes are used to group elements in a vertical or horizontal context (for example, Client, Application and Database tiers could be represented in swim lanes).

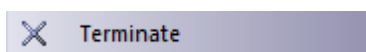


### 2.1.41 Terminate



The *Terminate pseudo-state* <sup>[13]</sup> indicates that upon entry of its pseudo-state, the *State Machine's* <sup>[9]</sup> execution ends.

#### Toolbox Icon



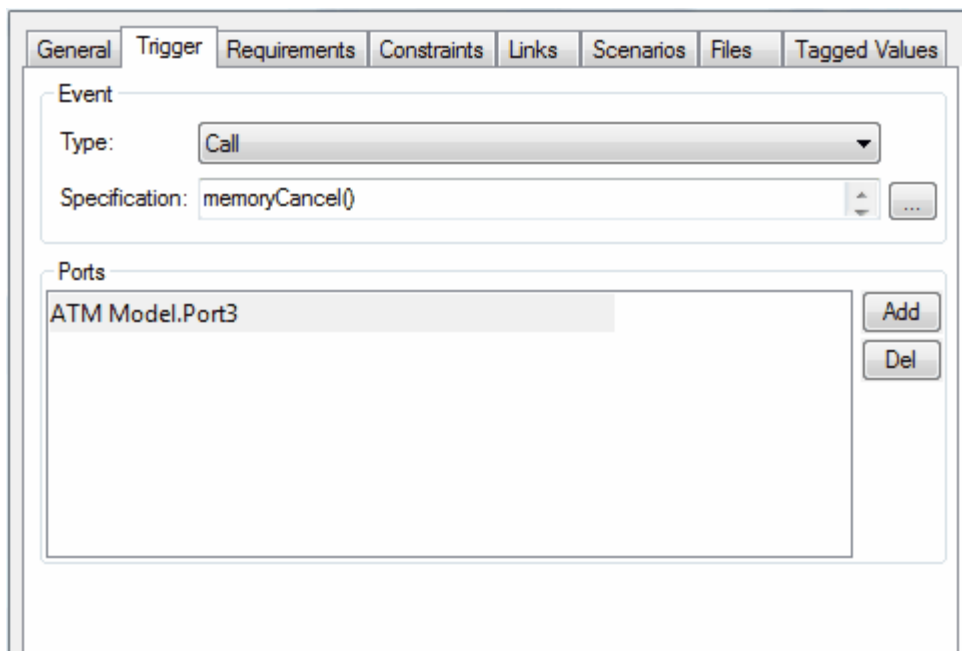
### 2.1.42 Trigger



A *Trigger* indicates an event that initiates an action (and might arise from completion of a previous action). You initially define a Trigger in one of four ways:

- As a property of a [Transition](#) <sup>[236]</sup> relationship
- As a property of an Accept Event [Action](#) <sup>[79]</sup> (on the **Triggers** tab of the element **Properties** dialog)
- As an event in a [State Machine Table](#) <sup>[19]</sup>
- Directly, as a Trigger element, through the **New Element** dialog.

When you save the Trigger, it is added to the list of elements for the parent package in the **Project Browser**. You can then right-click on it and select the **Properties** context menu option to view and, if required, edit its properties *as an element* rather than as a property itself. Triggers created as events remain as Event elements, whilst Triggers created in other ways are Trigger elements, with a **Trigger** tab in the **Properties** dialog.



Option	Use to
<b>Type</b>	If necessary, edit the type of trigger: <ul style="list-style-type: none"> <li>• <b>Call</b> - specifies that the event is a CallEvent, which sends a message to the associated object by invoking an operation.</li> <li>• <b>Change</b> - specifies that the event is a ChangeEvent, which indicates that the transition is the result of a change in value of an attribute.</li> <li>• <b>Signal</b> - specifies that the event is a SignalEvent, which corresponds to the receipt of an asynchronous signal instance.</li> <li>• <b>Time</b> - corresponds to a TimeEvent; which specifies a moment in time.</li> </ul>
<b>Specification</b>	Either type in the event instigating the Trigger, or click on the [ ... ] button and select the event (depending on the <b>Type</b> value).
<b>Ports</b>	Click on the <b>Add</b> button and select the appropriate Port from the <b>Select Port</b> dialog.



Option	Use to
	<p><b>Notes:</b></p> <ul style="list-style-type: none"> <li>• To create <i>new</i> Ports using the <b>Select Port</b> dialog, the Trigger should be created as a child of a Class or Component element.</li> <li>• To add several Ports at once, press <b>[Ctrl]</b> as you select each Port.</li> <li>• To check the exact location of a Port, right-click on the Port name and select the <b>Find in Project Browser</b> context menu option.</li> </ul>

You can also drag the Trigger element onto another diagram, although there are limited uses for the element in that context.

This element is not the same as a *Trigger Operation*, which is an operation automatically executed as a result of the modification of data in a database. (See *Code Engineering Using UML Models*.)

## OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 456*) states:

*Events may cause execution of behavior (e.g., the execution of the effect activity of a transition in a state machine). A trigger specifies the event that may trigger a behavior execution as well as any constraints on the event to filter out events not of interest.*

### 2.1.43 Use Case



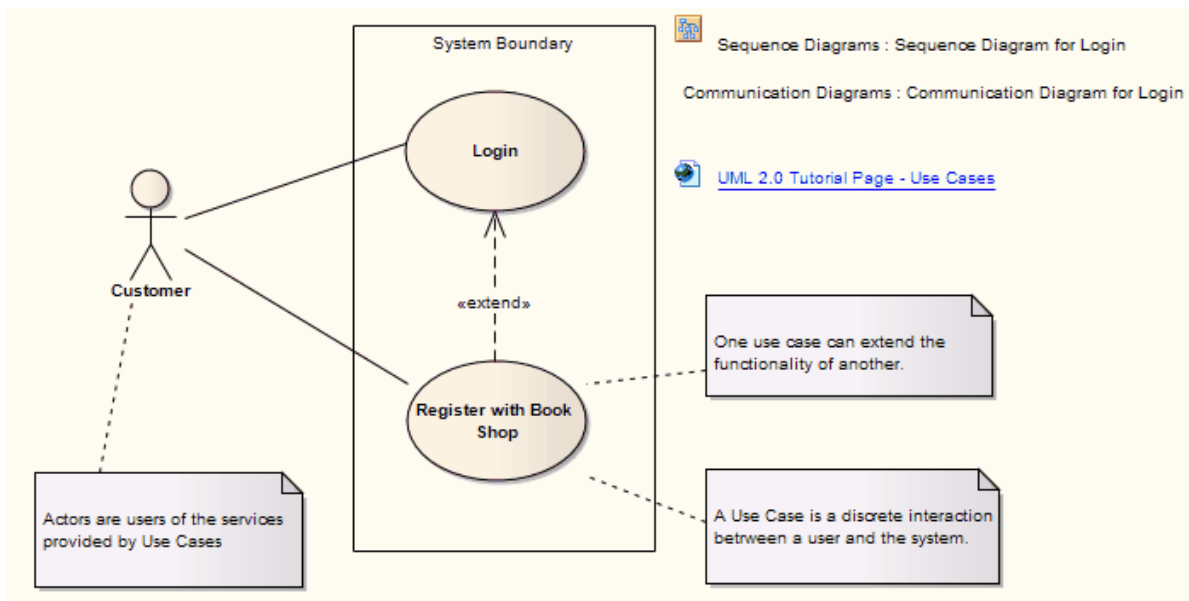
A *Use Case* is a UML modeling element that describes how a user of the proposed system interacts with the system to perform a discrete unit of work. It describes and signifies a single interaction over time that has meaning for the end user (person, machine or other system), and is required to leave the system in a complete state: the interaction either completed or rolled back to the initial state. A Use Case:

- Typically has requirements and constraints that describe the essential features and rules under which it operates
- Can have an associated [Sequence diagram](#) <sup>[39]</sup> illustrating behavior over time; who does what to whom, and when
- Typically has scenarios associated with it that describe the work flow over time that produces the end result; alternative work flows (for example, to capture exceptions) are also enabled.

**Note:**

Use a Use Case diagram and model to build up the functional requirements and implementation details of the system.

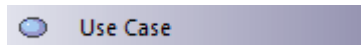
The following is an example Use Case model:



If extending a Use Case, you can specify the points of extension with [Use Case Extension Points](#)<sup>[147]</sup>. To display the attributes, operations or constraints of a Use Case on a diagram, use [Rectangle Notation](#)<sup>[148]</sup>.

Enterprise Architect also provides two stereotyped Use Cases - the [Test Case](#)<sup>[191]</sup> and the [Business Use Case](#)<sup>[75]</sup>.

### Toolbox Icon



### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 592*) states:

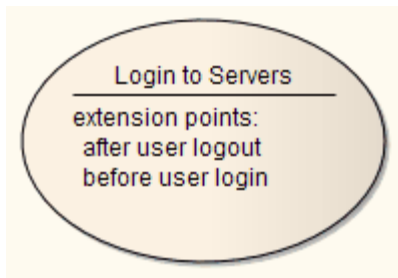
*A UseCase is a kind of behavior classifier that represents a declaration of an offered behavior. Each Use Case specifies some behavior, possibly including variants, that the subject can perform in collaboration with one or more actors.*

#### 2.1.43.1 Use Case Extension Points

Use *extension points* to specify the point of an extended [Use Case](#)<sup>[146]</sup> where an extending Use Case's behavior should be inserted. The specification text can be informal or precise to define the location of the extension point.

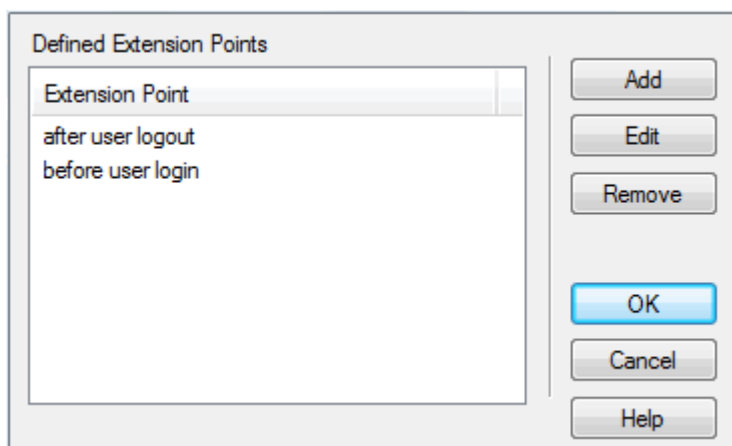
#### Note:

Conditions to apply the extending Use Case, and the extension point to use, should be attached as a note to the *extend* relationship.



To work with extension points, follow the steps below:

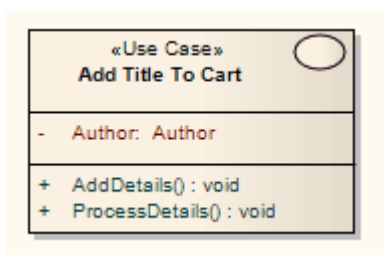
1. Right-click on the Use Case element. The context menu displays.
2. Select the **Advanced | Edit Extension Points...** menu option. The **Use Case Extension Points** dialog displays, listing defined points for that Use Case.



3. Select an extension point in the list and click on the appropriate button to edit or remove the extension point, or to add a new one.

### 2.1.43.2 Rectangle Notation

You can display a [Use Case](#)<sup>146</sup> using *rectangle notation*. This displays the Use Case in a rectangle, with an oval in the top right-hand corner. Any attributes, operations or constraints belonging to the Use Case are shown, in the same style as a [Class](#)<sup>152</sup>.

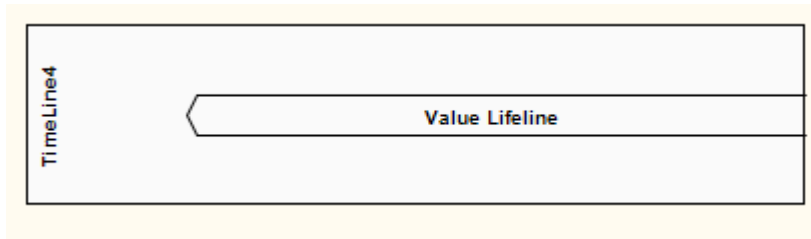


To show a Use Case using rectangle notation, right-click on the Use Case object on the diagram and select the **Advanced | Use Rectangle Notation** context menu option. This setting only applies to the selected Use Case, and can be toggled on and off.

#### Note:

[Actor](#)<sup>94</sup> elements can also be displayed using rectangle notation.

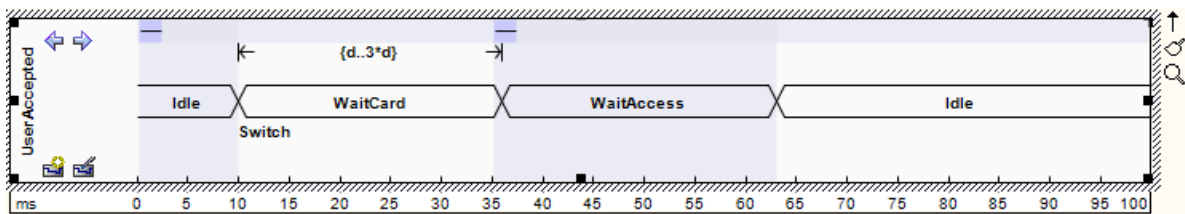
### 2.1.44 Value Lifeline



A *Lifeline* is the path an object takes across a measure of time, indicated by the x-axis. There are two sorts: *Value Lifelines* (defined here) and *State Lifelines*<sup>[135]</sup>, both used in *Timing diagrams*<sup>[22]</sup>.

A *Value Lifeline* shows the Lifeline's state across the diagram, with parallel lines indicating a steady state. A cross between the lines indicates a transition or change in state.

An example of a Value Lifeline is shown below:



See *UML Superstructure Specification, v2.1.1, Figure 14.30, p. 520*.

A Value Lifeline consists of a set of transition points. Each transition point can be defined with the following properties:

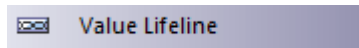
Property	Description
<b>At time</b>	Specifies the starting time for a change of state.
<b>Transition to</b>	Indicates the state to which the Lifeline will change.
<b>Event</b>	Describes the occurring event.
<b>Timing constraints</b>	Refers to the time taken for a state to change within a Lifeline, or the time taken to transmit a message.
<b>Timing observations</b>	Provides information on the time of a state change or sent message.
<b>Duration constraints</b>	Pertains to a Lifeline's period at a particular state. The constraint could be instigated by a change of state within a Lifeline, or that Lifeline's receipt of a message.
<b>Duration observations</b>	Indicates the interval of a Lifeline at a particular state, begun from a change in state or message receipt.

In the example diagram above, the **10ms** transition point has these properties:

Property	Text
<b>At Time</b>	10ms
<b>Transition to</b>	Waitcard
<b>Event</b>	Switch
<b>Timing constraints</b>	–

Property	Text
Timing observations	–
Duration constraints	d..3*d
Duration observations	–

## Toolbox Icon



## OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 518*) states:

*Shows the value of the connectable element as a function of time. Value is explicitly denoted as text. Crossing reflects the event where the value changed.*

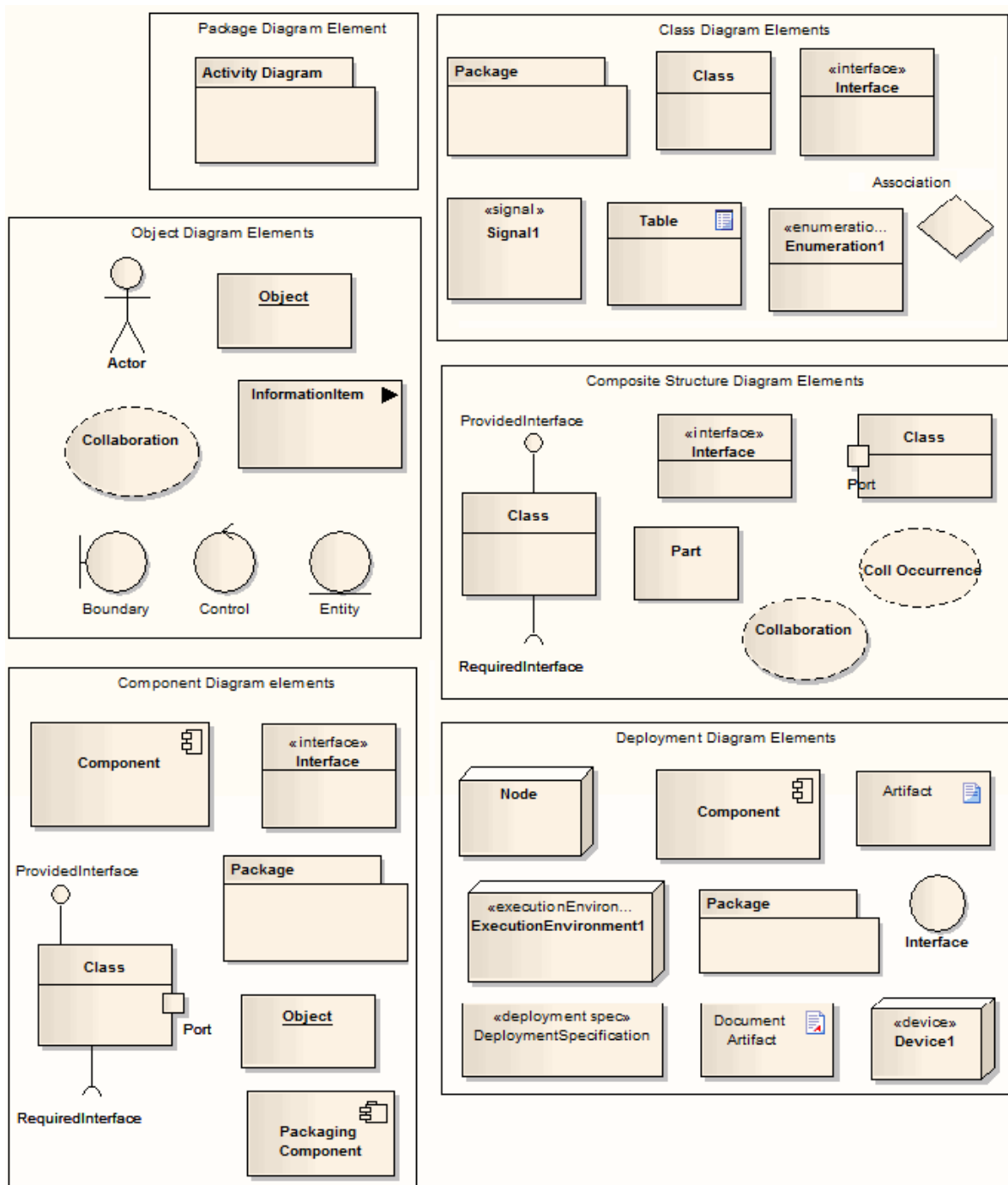
## 2.2 Structural Diagram Elements

The following figure illustrates the main [UML elements](#)<sup>[78]</sup> that are used in [Structural Diagrams](#)<sup>[54]</sup>. For more information on using each element, click on the element name in this list:

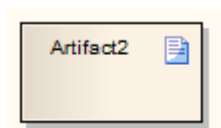
- [Actor](#)<sup>[94]</sup>, [Artifact](#)<sup>[151]</sup>
- [Class](#)<sup>[152]</sup>, [Collaboration](#)<sup>[156]</sup>, [Collaboration Occurrence](#)<sup>[157]</sup>, [Component](#)<sup>[158]</sup>
- [Data Type](#)<sup>[159]</sup>, [Deployment Specification](#)<sup>[160]</sup>, [Document Artifact](#)<sup>[161]</sup>
- [Enumeration](#)<sup>[162]</sup>, [Execution Environment](#)<sup>[162]</sup>, [Expose Interface](#)<sup>[163]</sup>
- [Information Item](#)<sup>[163]</sup>, [Interface](#)<sup>[164]</sup>
- [Node](#)<sup>[165]</sup>, [Note](#)<sup>[126]</sup>
- [Object](#)<sup>[165]</sup>
- [Package](#)<sup>[168]</sup>, [Part](#)<sup>[168]</sup>, [Port](#)<sup>[169]</sup>, [Primitive](#)<sup>[173]</sup>
- [Qualifiers](#)<sup>[173]</sup>
- [Signal](#)<sup>[178]</sup>

### Note:

Actor, Collaboration, Note, Object and Package elements are used in both Behavioral diagrams and Structural diagrams.



### 2.2.1 Artifact



An *Artifact* is any physical piece of information used or produced by a system, represented in a [Deployment Diagram](#)<sup>[62]</sup>. Artifacts can have associated properties or operations, and can be instantiated or associated with other Artifacts. Examples of Artifacts include model files, source files, database tables, development

deliverables or support documents. The files represented by the Artifact are listed on the **Files** tab of the element **Properties** dialog.

To open the files represented by the Artifact, click on the element on the diagram and press **[Ctrl]+[E]**. Each file is opened either on a separate tab in the **Diagram View** workspace (if the file can be opened within Enterprise Architect) or in the default Windows viewer/editor for the file type (if the file cannot be opened within Enterprise Architect).

Files can also be launched individually from the **Files** tab (opening in the Windows default editor), as for elements of any other type that have associated files.

## Toolbox Icon



## Create Artifact For External File

You can also create an Artifact element on a diagram *from* an external file, by clicking on the file in a file list (such as Windows Explorer) or on your Desktop and dragging it onto the diagram. A short context menu displays with two options - **Hyperlink** and **Artifact**.

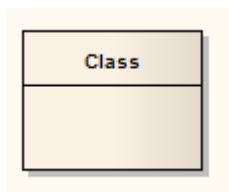
Click on the **Artifact** option to create the element on the diagram. The **Properties** dialog displays, and you can define the name or other properties as required. Click on the **OK** button, and then open the **Properties** dialog again and click on the **Files** tab. The file pathname is listed in the **Files** panel.

## OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 201*) states:

*An Artifact defined by the user represents a concrete element in the physical world. A particular instance (or 'copy') of an artifact is deployed to a node instance. Artifacts may have composition associations to other artifacts that are nested within it. For instance, a deployment descriptor artifact for a component may be contained within the artifact that implements that component. In that way, the component and its descriptor are deployed to a node instance as one artifact instance.*

## 2.2.2 Class



A *Class* is a representation of objects that reflects their structure and behavior within the system. It is a template from which actual running instances are created, although a Class can be defined either to [control its own execution](#)<sup>[153]</sup> or as a *template* or [parameterized Class](#)<sup>[154]</sup> that specifies parameters that must be defined by any binding Class.

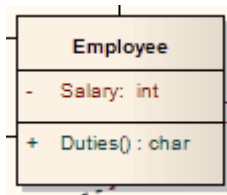
A Class can have *attributes* (data) and *methods* (operations or behavior) - see *UML Modeling with Enterprise Architect – UML Modeling Tool*. Classes can inherit characteristics from parent Classes and delegate behavior to other Classes. Class models usually describe the logical structure of the system and are the building blocks from which components are built.

The top section of a Class, as illustrated below, shows the attributes (or data elements) associated with the Class. These hold the 'state' of an object at run-time. If the information is saved to a data store and can be reloaded, it is termed 'persistent'. The lower section contains the Class operations (or methods at run-time). Operations describe the behavior a Class offers to other Classes, and the internal behavior it has (private methods).

Class elements are generally used in [Class diagrams](#)<sup>[56]</sup> and [Composite Structure diagrams](#)<sup>[59]</sup>.

Enterprise Architect also supports a number of stereotyped Class elements to represent various entities in

[web-page modeling](#)<sup>[194]</sup>. A Class can also be integrated with an [Associate](#)<sup>[199]</sup> connector to form an [Association Class](#)<sup>[200]</sup>, to allow the Associate connector to have operations and attributes that define certain types of UML relationship.



## Toolbox Icon



## OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, pp. 52-53*) states:

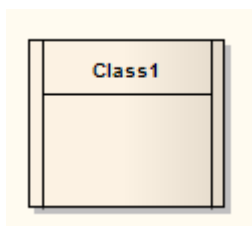
*The purpose of a class is to specify a classification of objects and to specify the features that characterize the structure and behavior of those objects.*

*Objects of a class must contain values for each attribute that is a member of that class, in accordance with the characteristics of the attribute, for example its type and multiplicity.*

*When an object is instantiated in a class, for every attribute of the class that has a specified default, if an initial value of the attribute is not specified explicitly for the instantiation, then the default value specification is evaluated to set the initial value of the attribute for the object.*

*Operations of a class can be invoked on an object, given a particular set of substitutions for the parameters of the operation. An operation invocation may cause changes to the values of the attributes of that object. It may also return a value as a result, where a result type for the operation has been defined. Operation invocations may also cause changes in value to the attributes of other objects that can be navigated to, directly or indirectly, from the object on which the operation is invoked, to its output parameters, to objects navigable from its parameters, or to other objects in the scope of the operation's execution. Operation invocations may also cause the creation and deletion of objects.*

### 2.2.2.1 Active Classes



An *Active Class* indicates that, when instantiated, the [Class](#)<sup>[152]</sup> controls its own execution. Rather than being invoked or activated by other objects, it can operate standalone and define its own thread of behavior.

To define an Active Class in Enterprise Architect, follow the steps below:

1. Highlight a Class, and display its **Properties** dialog.
2. Click on the **Advanced** button.
3. Select the **Is Active** checkbox.
4. Click on the **OK** button to save the details.



## OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 438*) states:

*An active object is an object that, as a direct consequence of its creation, commences to execute its classifier behavior, and does not cease until either the complete behavior is executed or the object is terminated by some external object. (This is sometimes referred to as "the object having its own thread of control.") The points at which an active object responds to communications from other objects is determined solely by the behavior of the active object and not by the invoking object. If the classifier behavior of an active object completes, the object is terminated.*

### 2.2.2.2 Parameterized Classes (Templates)

Enterprise Architect supports *template* or *parameterized Classes*, which specify parameters that must be defined by any binding [Class](#)<sup>[152]</sup>. A template Class enables its functionality to be reused by any bound Class. If a default value is specified for a parameter, and a binding Class doesn't provide a value for that parameter, the default is used. Parameterized Classes are commonly implemented in C++.

Enterprise Architect imports and generates templated Classes for C++. Template Classes are shown with the parameters in a dashed outline box in the upper right corner of a Class.

To create a parameterized Class, follow the steps below:

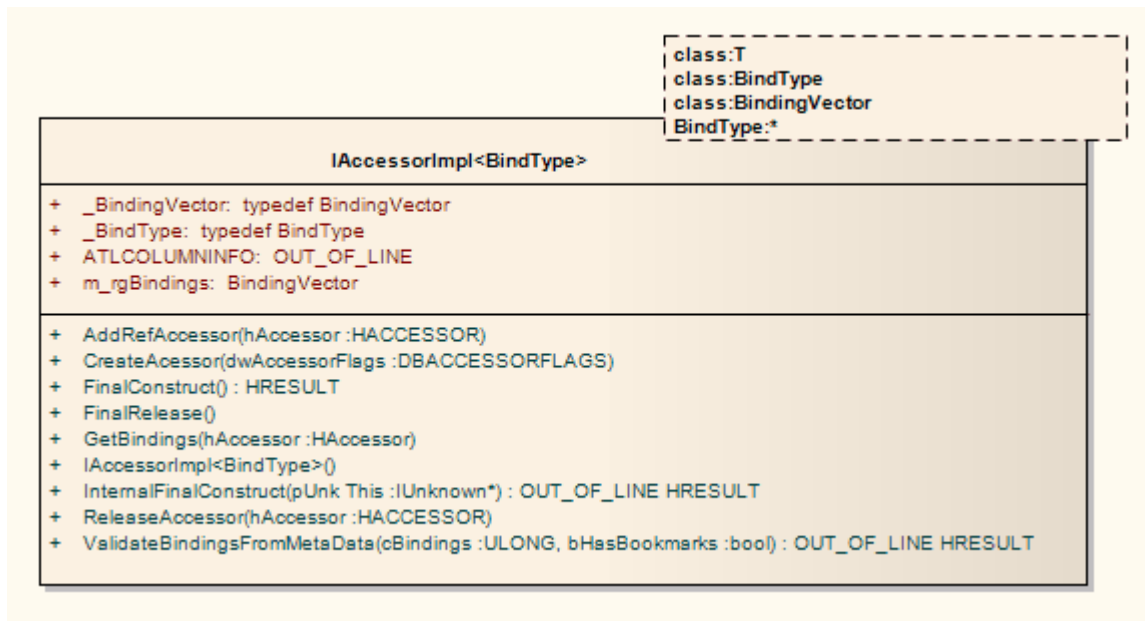
1. Display the **Properties** dialog for a Class.
2. Select the **Details** tab.

The screenshot shows the 'Details' tab of a UML Class Properties dialog box. The 'Cardinality' is set to '0..\*' and 'Visibility' is set to 'Private'. There are buttons for 'Attributes...', 'Operations...', and 'Collection Classes...'. The 'Concurrency' section has radio buttons for 'Sequential', 'Guarded', 'Active', and 'Synchronous'. The 'Templates' section has a 'Type' dropdown set to 'Parameterised', with 'Add', 'Edit', and 'Delete' buttons. Below this is a table with columns 'Parameter', 'Type', and 'Default'. The table contains one row: 'bind type' with 'Class' in the 'Type' column. At the bottom, there is an 'Arguments' text field and 'OK', 'Cancel', 'Apply', and 'Help' buttons.

Parameter	Type	Default
bind type	Class	

3. In the **Type** field, click on the drop-down arrow and select **Parameterized**.  
For an instantiated template, select **Instantiated** and add the arguments in the **Arguments** field.
4. Click on the **Add** button and define the required parameters in the **Class Parameter** dialog.

## Notation Example



## OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 622*) states:

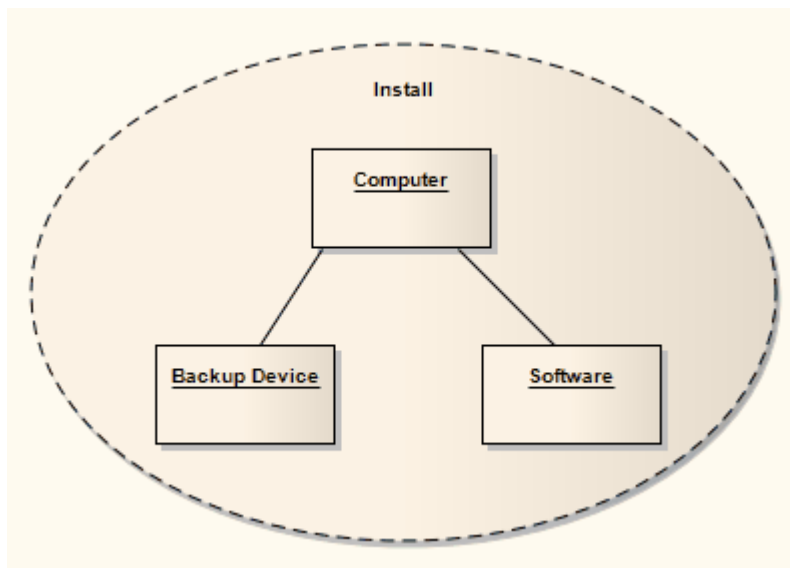
*A template is a parameterized element that can be used to generate other model elements using TemplateBinding relationships. The template parameters for the template signature specify the formal parameters that will be substituted by actual parameters (or the default) in a binding.*

### 2.2.3 Collaboration



A *Collaboration* defines a set of cooperating roles and their connectors. These are used to collectively illustrate a specific functionality, in a [Composite Structure diagram](#)<sup>[59]</sup>. A Collaboration should specify only the roles and attributes required to accomplish a specific task or function. Although in practice a behavior and its roles could involve many tangential attributes and properties, isolating the primary roles and their requisites simplifies and clarifies the behavior, as well as providing for reuse. A Collaboration often implements a pattern to apply to various situations.

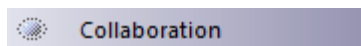
The following example illustrates an *Install* Collaboration, with three roles ([Objects](#)<sup>[165]</sup>) connected as shown. The process for this Collaboration can be demonstrated by attaching an Interaction diagram ([Sequence](#)<sup>[41]</sup>, [Timing](#)<sup>[22]</sup>, [Communication](#)<sup>[49]</sup> or [Interaction Overview](#)<sup>[52]</sup>).



To understand referencing a Collaboration in a specific situation, see the [Collaboration Occurrence](#)<sup>[157]</sup> topic.

Enterprise Architect supports a stereotyped Collaboration to represent a [Business Use Case Realization](#)<sup>[75]</sup> in business modeling.

### Toolbox Icon

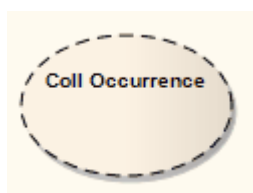


### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 171*) states:

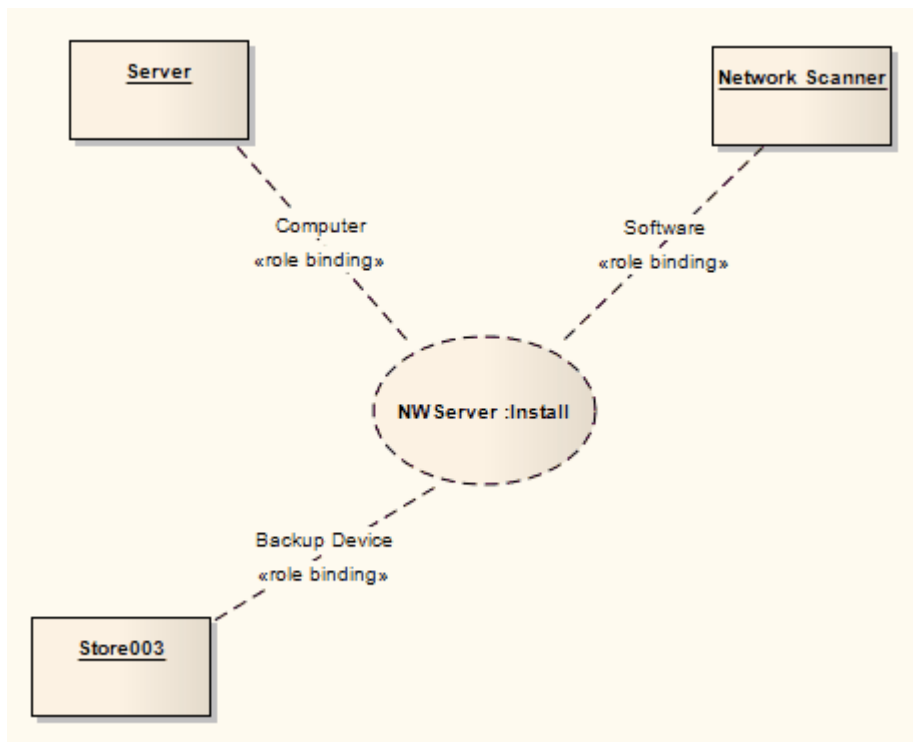
*A collaboration describes a structure of collaborating elements (roles), each performing a specialized function, which collectively accomplish some desired functionality. Its primary purpose is to explain how a system works and, therefore, it typically only incorporates those aspects of reality that are deemed relevant to the explanation.*

### 2.2.4 Collaboration Occurrence



Use a *Collaboration Occurrence* to apply a pattern defined by a Collaboration to a specific situation, in a [Composite Structure diagram](#)<sup>[59]</sup>.

The following example uses an occurrence, *NWServer*, of the Collaboration *Install*, to define the installation process of a network scanner. This process can be defined by an interaction attached to the Collaboration. (See the [Collaboration](#)<sup>[156]</sup> topic for a representation of the *Install* Collaboration.)



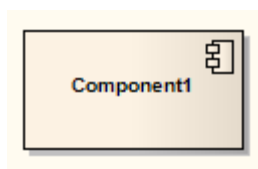
To create a Collaboration Occurrence, drag the required Collaboration from the **Project Browser** onto the diagram and, on the **Paste Element** dialog, select the **Paste as Link** radio button.

## OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 173*) refers to a Collaboration Occurrence as a **Collaboration Use**, and states:

*A collaboration use represents one particular use of a collaboration to explain the relationships between the properties of a classifier. A collaboration use shows how the pattern described by a collaboration is applied in a given context, by binding specific entities from that context to the roles of the collaboration. Depending on the context, these entities could be structural features of a classifier, instance specifications, or even roles in some containing collaboration. There may be multiple occurrences of a given collaboration within a classifier, each involving a different set of roles and connectors. A given role or connector may be involved in multiple occurrences of the same or different collaborations.*

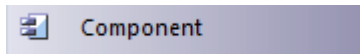
### 2.2.5 Component



A *Component* is a modular part of a system, whose behavior is defined by its provided and required interfaces; the internal workings of the Component should be invisible and its usage environment-independent. Source code files, DLLs, Java beans and other artifacts defining the system can be manifested in Components.

A Component can be composed of multiple [Classes](#)<sup>[152]</sup>, or Components pieced together. As smaller Components come together to create bigger Components, the eventual system can be modeled, building-block style, in [Component diagrams](#)<sup>[65]</sup>. By building the system in discrete Components, localization of data and behavior enables decreased dependency between Classes and [Objects](#)<sup>[165]</sup>, providing a more robust and maintainable design.

## Toolbox Icon



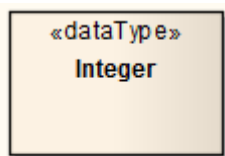
## OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 148*) states:

*A component represents a modular part of a system that encapsulates its contents and whose manifestation is replaceable within its environment.*

*A component defines its behavior in terms of provided and required interfaces. As such, a component serves as a type whose conformance is defined by these provided and required interfaces (encompassing both their static as well as dynamic semantics).*

### 2.2.6 Data Type



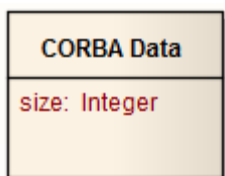
A *Data Type* is a specific kind of classifier, similar to a [Class](#)<sup>1521</sup> except that a Data Type cannot own sub Data Types, and instances of a Data Type are identified only by their value. For example, an instance of a *Person* Class is a *Helen* object, but an instance of an *Integer* Data Type is *12*.

All copies of an instance of a Data Type, and any instances of that Data Type with the same value, are considered to be the same instance. That is, instances of *Helen* are not necessarily the same *Helen*, but all *12* s are the same *12*. For example, the *12* on a watch face is exactly the same integer as the number of months in a year.

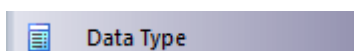
Instances of a Data Type that have attributes (that is, are instances of a structured Data Type) are considered to be the same if the structure is the same and the values of the corresponding attributes are the same. If a Data Type has attributes, instances of that Data Type contain attribute values matching the attributes.

A typical use of Data Types would be to represent programming language primitive types or CORBA basic types. For example, integer and string types are often treated as Data Types.

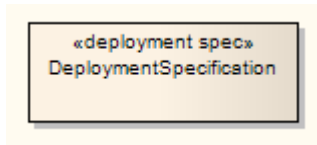
A Data Type is denoted by a rectangle with the keyword «*dataType*», as above or, when it is referenced by (for example) an attribute, by a string containing the name of the Data Type, as below:



## Toolbox Icon



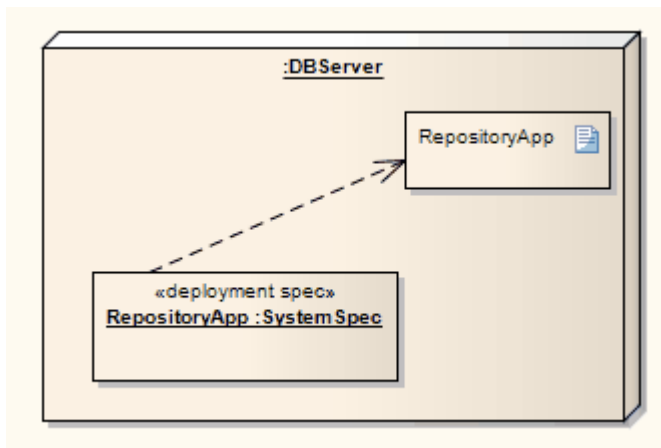
## 2.2.7 Deployment Spec



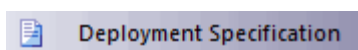
A *Deployment Specification (spec)* specifies parameters guiding deployment of an artifact, as is necessary with most hardware and software technologies. A specification lists those properties that must be defined for deployment to occur, as represented in a [Deployment diagram](#) <sup>62</sup>. An instance of this specification specifies the values for the parameters; a single specification can be instantiated for multiple artifacts.

These specifications can be extended by certain component profiles. Examples of standard Tagged Values that a profile might add to a Deployment Specification are «*concurrencyMode*» with Tagged Values {*thread, process, none*} or «*transactionMode*» with Tagged Values {*transaction, nestedTransaction, none*}.

The following example depicts the artifact *RepositoryApp* deployed on the server node, as per the specifications of *RepositoryApp*, instantiated from the Deployment Specification *SystemSpec*.



### Toolbox Icon

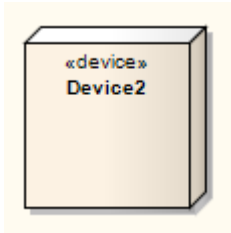


### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 206*) states:

*A deployment specification specifies a set of properties that determine execution parameters of a component artifact that is deployed on a node. A deployment specification can be aimed at a specific type of container. An artifact that reifies or implements deployment specification properties is a deployment descriptor.*

### 2.2.8 Device



A *Device* is a physical electronic resource with processing capability upon which [Artifacts](#)<sup>[15]</sup> can be deployed for execution, as represented in a [Deployment diagram](#)<sup>[62]</sup>. Complex Devices can consist of other devices; that is, a Device can be a nested element, where a physical machine is decomposed into its elements either through namespace ownership or through attributes that are typed by Devices.

#### Toolbox Icon



#### OMG UML Specification

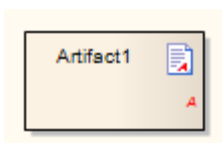
The OMG UML specification (*UML Superstructure Specification, 10.3.7, v2.1.1, p. 207*) states:

*In the metamodel, a Device is a subclass of Node.*

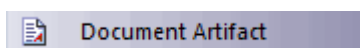
### 2.2.9 Document Artifact



A *Document Artifact* is an [artifact](#)<sup>[15]</sup> having a *stereotype* of «document». You create the Document Artifact on a Component, Documentation or Deployment diagram, and associate it with an RTF document. Double-click on the element to display the [Linked Document Editor](#). See *Linked Documents* in *UML Modeling with Enterprise Architect – UML Modeling Tool*. When you have created the linked document, the Document Artifact element on the diagram shows an **A** symbol in the bottom right corner.

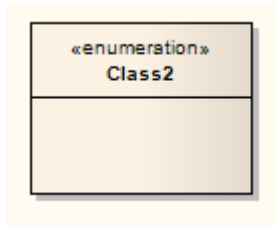


#### Toolbox Icon



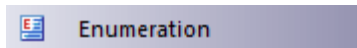


### 2.2.10 Enumeration



An *Enumeration* is a data type, whose instances can be any of a number of user-defined enumeration literals. It is possible to extend the set of applicable enumeration literals in other packages or profiles. You create Enumerations in [Class](#)<sup>[56]</sup> or [Package diagrams](#)<sup>[55]</sup>, and in diagrams developed from the [Metamodel](#) pages of the Enterprise Architect UML [Toolbox](#) (see *Using Enterprise Architect - UML Modeling Tool*).

#### Toolbox Icon

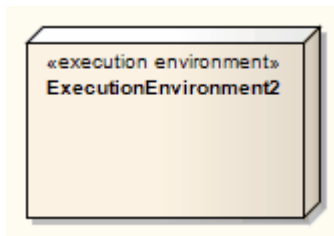


#### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 69*) states:

*An enumeration is a data type whose values are enumerated in the model as enumeration literals.*

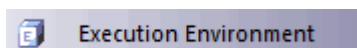
### 2.2.11 Execution Environment



An *Execution Environment* is a [node](#)<sup>[165]</sup> that offers an execution environment for specific types of [components](#)<sup>[158]</sup> that are deployed on it in the form of executable [artifacts](#)<sup>[157]</sup>. This is depicted in a [Deployment diagram](#)<sup>[62]</sup>.

Execution Environments can be nested; for example, a database Execution Environment can be nested in an operating system Execution Environment. Components of the appropriate type are then deployed to specific Execution Environment nodes.

#### Toolbox Icon

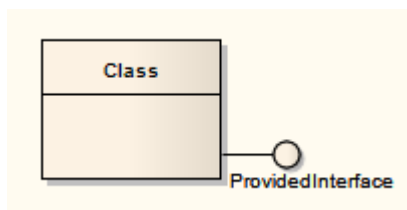
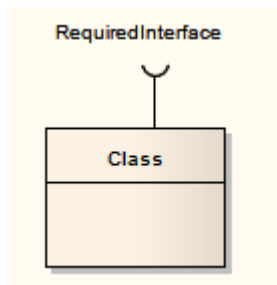


#### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 210*) states:

*... an ExecutionEnvironment is ... usually part of a general Node, representing the physical hardware environment on which the ExecutionEnvironment resides. In that environment, the ExecutionEnvironment implements a standard set of services that Components require at execution time (at the modeling level these services are usually implicit). For each component Deployment, aspects of these services may be determined by properties in a DeploymentSpecification for a particular kind of ExecutionEnvironment.*

### 2.2.12 Expose Interface

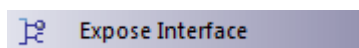


The *Expose Interface* element is a graphical method of depicting the required or supplied [interfaces](#)<sup>[164]</sup> of a [Component](#)<sup>[158]</sup>, [Class](#)<sup>[152]</sup> or [Part](#)<sup>[168]</sup>, in a [Component](#)<sup>[65]</sup> or [Composite Structure](#)<sup>[59]</sup> diagram. It just identifies the fact that the element provides or requires an interface; to depict the fact that the provided interface is used, or the required interface provided, by another element use the [Assembly](#)<sup>[199]</sup> connector.

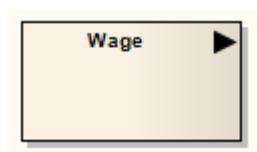
The Expose Interface element must be attached to the Class or Component element, and it becomes a child element of that Class or Component; it cannot exist independently. You can attach more than one Expose Element to another element.

When you create the Expose Interface element, a dialog displays in which you enter a name for the element and specify whether it represents a required interface or a provided interface.

#### Toolbox Icon

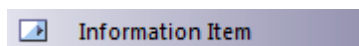


### 2.2.13 Information Item



An *Information Item* represents an abstraction of data. It is used in [Activity](#)<sup>[54]</sup>, [Analysis](#)<sup>[67]</sup> and [Object](#)<sup>[58]</sup> diagrams. An Information Item is also represented by an [Information Flow](#)<sup>[208]</sup> connector.

#### Toolbox Icon



#### OMG UML Specification

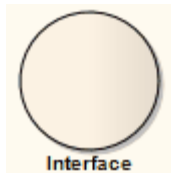
The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 608*) states:

*An information item is an abstraction of all kinds of information that can be exchanged between objects. It is a*

kind of classifier intended for representing information at a very abstract way, one which cannot be instantiated.

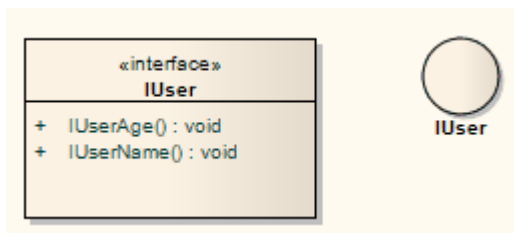
One purpose of information items is to be able to define preliminary models, before having made detailed modeling decisions on types or structures. One other purpose of information items and information flows is to abstract complex models by a less precise but more general representation of the information exchanged between entities of a system.

### 2.2.14 Interface



An *Interface* is a specification of behavior (or contract) that implementers agree to meet. By implementing an Interface, [Classes](#)<sup>[56]</sup> are guaranteed to support a required behavior, which enables the system to treat non-related elements in the same way; that is, through the common interface. You also use Interfaces in a [Composite Structure](#)<sup>[59]</sup> diagram.

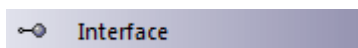
Interfaces are drawn in a similar way to a [Class](#)<sup>[152]</sup>, with operations specified, as shown below. They can also be drawn as a circle with no explicit operations detailed. Right-click on the element and select the **Use Circle Notation** context menu option to switch between styles. [Realization](#)<sup>[233]</sup> connectors to an Interface drawn as a circle are drawn as a solid line without target arrows.



#### Note:

An Interface cannot be instantiated (that is, you cannot create an object from an Interface). You must create a Class that 'implements' the Interface specification, and in the Class body place operations for each of the Interface operations. You can then instantiate the Class.

#### Toolbox Icon



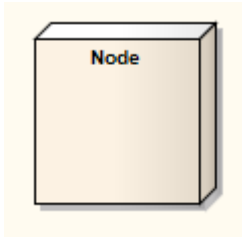
#### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 88*) states:

*An interface is a kind of classifier that represents a declaration of a set of coherent public features and obligations. An interface specifies a contract; any instance of a classifier that realizes the interface must fulfill that contract. The obligations that may be associated with an interface are in the form of various kinds of constraints (such as pre- and post-conditions) or protocol specifications, which may impose ordering restrictions on interactions through the interface.*

*Since interfaces are declarations, they are not instantiable. Instead, an interface specification is implemented by an instance of an instantiable classifier, which means that the instantiable classifier presents a public facade that conforms to the interface specification. Note that a given classifier may implement more than one interface and that an interface may be implemented by a number of different classifiers.*

### 2.2.15 Node



A *Node* is a physical piece of equipment on which the system is deployed, such as a workgroup server or workstation. A Node usually hosts components and other executable pieces of code, which again can be connected to particular processes or execution spaces. Typical Nodes are client workstations, application servers, mainframes, routers and terminal servers.

Nodes are used in [Deployment diagrams](#)<sup>[62]</sup> to model the deployment of a system, and to illustrate the physical allocation of implemented artifacts. They are also used in web modeling, from dedicated web modeling pages in the [Enterprise Architect UML Toolbox](#) (see *Using Enterprise Architect - UML Modeling Tool*).

#### Toolbox Icon

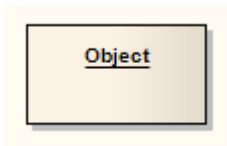


#### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 213*) states:

*In the metamodel, a Node is a subclass of Class. It is associated with a Deployment of an Artifact. It is also associated with a set of Elements that are deployed on it. This is a derived association in that these PackageableElements are involved in a Manifestation of an Artifact that is deployed on the Node. Nodes may have an internal structure defined in terms of parts and connectors associated with them for advanced modeling applications.*

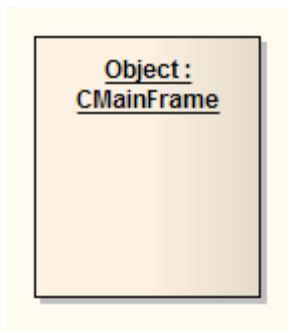
### 2.2.16 Object



An *Object* is a particular *instance* of a [Class](#)<sup>[152]</sup> at run time. For example a car with the license plate **AAA-001** is an instance of the general class of cars with a license plate number attribute. Objects are often used in analysis to represent the numerous artifacts and items that exist in any business, such as pieces of paper, faxes and information. To model the varying behavior of Objects at run-time, use [run-time states](#)<sup>[166]</sup>.

Early in analysis, Objects can be used to quickly capture all the things that are of relevance within the system domain, in an [Object](#)<sup>[58]</sup>, [Composite Structure](#)<sup>[59]</sup> or [Communication](#)<sup>[49]</sup> diagram. As the model progresses these analysis Objects are refined into generic Classes from which instances can be derived to represent common business items. Once Classes are defined, Objects can be typed; that is they can have a classifier set that indicates their base type. See the *Object Classifiers* topic in *UML Modeling with Enterprise Architect – UML Modeling Tool*.

Enterprise Architect also supports a number of [stereotyped Object](#)<sup>[75]</sup> elements to represent various entities in business modeling.



### Toolbox Icon



#### 2.2.16.1 Run-time State

At run-time, an [Object](#)<sup>[165]</sup> instance can have specific values for its attributes, or exist in a particular state. To model the varying behavior of Objects at run-time, use instance values selected from the **Select <Item>** dialog (see *UML Modeling with Enterprise Architect – UML Modeling Tool*) and *run-time states* or *run-states*.

Typically there is interest in the run-time behavior of Objects that already have a classifier set. You can select from the classifier's attribute list and apply specific values for your Object instance. If the classifier has a child [State Machine](#)<sup>[9]</sup>, its [States](#)<sup>[129]</sup> propagate to a list where the run-time state for the Object can be defined. To do this, see the following topics:

- [Define a Run-Time Variable](#)<sup>[166]</sup>
- [Remove a Defined Variable](#)<sup>[167]</sup>
- [Object State](#)<sup>[167]</sup>

The following example defines run-time values for the listed variables, which are attributes of the instances' classifier *AccountItem*.



##### 2.2.16.1.1 Define a Run-time Variable

To add [run-time state](#)<sup>[166]</sup> instance variables to an Object, follow the steps below:

1. Right-click on the Object. The context menu displays.
2. If Instance Variables are supported, select the **Advanced | Set Run State** menu option (or press **[Ctrl]+[Shift]+[R]**). The **Set Run State** dialog displays.

3. In the **Variable** field, click on the drop-down arrow and select the variable, or type in the new variable name.
4. Set the **Operator**, the **Value** and optionally type in a **Note**.
5. Click on the **OK** button to save the variable.

### 2.2.16.1.2 Remove a Defined Variable

To delete a [run-time state](#)<sup>[166]</sup> variable for an Object:

1. Right-click on the required Object. The context menu displays.
2. Select the **Set Run State** option. The **Run State** dialog displays.
3. In the **Variable** field, click on the drop-down arrow and select the variable to delete.
4. Clear the **Value** field.
5. Click on the **OK** button.

### 2.2.16.2 Object State

To set the Object state for a Class instance, follow the steps below:

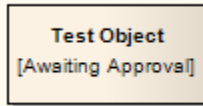
1. Right-click on the required Object and select the **Advanced | Set Object State** context menu option. The **Set Instance State** dialog displays.

2. In the **State** field, either type the required State (such as **Awaiting Approval**) or select a State from the drop-down list.

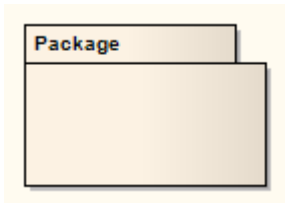
#### Note:

The drop-down list for the **State** field is populated with:

1. Any States owned by the object's classifier.
  2. Any States owned by any superclasses of the object's classifier.
  3. Any States owned by State Machines owned by the object's classifier.
  4. Any States owned by State Machines owned by any superclasses of the object's classifier.
3. Click on the **OK** button to apply the State. The object now shows the run-time state in square brackets below the object name.

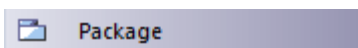


### 2.2.17 Package



A *Package* is a namespace as well as an element that can be contained in other *Package*'s namespaces. A *Package* can own or merge with other *Packages*, and its elements can be imported into a *Package*'s namespace. In addition to using *Packages* in the **Project Browser** to organize your project contents, you can drag these *Packages* onto a diagram workspace (most diagram types, both standard and extended) for structural or relational depictions, including *Package* imports or merges.

#### Toolbox Icon

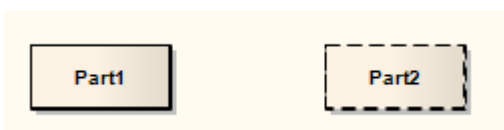


#### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 109*) states:

*A package is a namespace for its members, and may contain other packages. Only packageable elements can be owned members of a package. By virtue of being a namespace, a package can import either individual members of other packages, or all the members of other packages. In addition a package can be merged with other packages.*

### 2.2.18 Part



*Parts* are run-time instances of [Classes](#)<sup>[152]</sup> or [Interfaces](#)<sup>[164]</sup>. Multiplicity can be specified for a *Part*, using the notation:

[x{...}y]

where *x* specifies the initial or set amount of instances when the composite structure is created, and *y* indicates the maximum amount of instances at any time.

*Parts* are used to express [composite structures](#)<sup>[59]</sup>, or modeling patterns that can be invoked by various objects to accomplish a specific purpose. When illustrating the composition of structures, *Parts* can be embedded as properties of other *Parts*. When embedded as [properties](#)<sup>[61]</sup>, *Parts* can be bordered by a solid outline, indicating the surrounding *Part* owns the *Part* by composition. Alternatively, a dashed outline indicates that the property is referenced and used by the surrounding *Part*, but is not composed within it.

You can also set [properties](#)<sup>[172]</sup> and [property values](#)<sup>[169]</sup> for *Parts*.

## Toolbox Icon

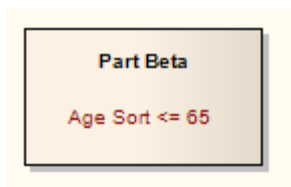


### 2.2.18.1 Add Property Value

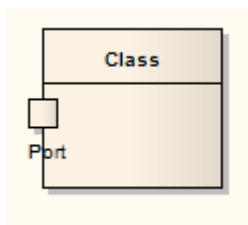
To add property value variables to a Part, follow the steps below:

1. Right-click on the Part. The context menu displays.
2. Select the **Advanced | Set Property Values** menu option (or press **[Ctrl]+[Shift]+[R]**). The **Set Property Values** dialog displays.

3. In the **Variable** field, click on the drop-down arrow and select the variable, or type in the new variable name.
  4. Set the **Operator**, the **Value** and optionally type in a **Note**.
  5. Click on the **OK** button to save the variable.
- A Part with a property value resembles the following figure.



### 2.2.19 Port



*Ports* define the interaction between a classifier and its environment. *Interfaces* controlling this interaction can be depicted using the [Interface element](#)<sup>[164]</sup>. Any connector to a Port must provide the required interface, if defined. Ports can appear on a contained [Part](#)<sup>[168]</sup>, a [Class](#)<sup>[152]</sup>, or the boundary of a [Composite element](#)<sup>[180]</sup>.

A Port is a *typed* structural feature or property of its containing classifier. Ports are typically [created](#)<sup>[170]</sup> in [Class diagrams](#)<sup>[56]</sup>, [Object diagrams](#)<sup>[58]</sup> and [Composite Structure diagrams](#)<sup>[59]</sup>.



You can [expose an inherited Port, or redefine a Port](#)<sup>[170]</sup>. You also define specific [properties](#)<sup>[172]</sup> for a Port element.

## Toolbox Icon



## OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 182*) states:

*A port is a property of a classifier that specifies a distinct interaction point between that classifier and its environment or between the (behavior of the) classifier and its internal parts. Ports are connected to properties of the classifier by connectors through which requests can be made to invoke the behavioral features of a classifier. A Port may specify the services a classifier provides (offers) to its environment as well as the services that a classifier expects (requires) of its environment.*

### 2.2.19.1 Add a Port to an Element

To add a new [Port](#)<sup>[169]</sup> to an element, use one of the following steps:

1. Click on the **Port** symbol in the **Composite Elements** page of the Enterprise Architect UML **Toolbox** and drag it to (or click on) the target host element. This creates an untyped, simple Port on the boundary, near the cursor position.
2. On the context menu of a suitable Class, Part or [Composite element](#)<sup>[180]</sup>, select the **Embedded Elements | Add Port** menu option to add a new Port at the cursor position.
3. Drag a suitable classifier from the **Project Browser** onto a Class or Part. Enterprise Architect prompts you to add a typed Port or Part at the cursor position. The new Port is typed by the original dragged classifier.
4. Use the [Embedded Elements dialog](#)<sup>[170]</sup> to add a new Port to the currently selected element.

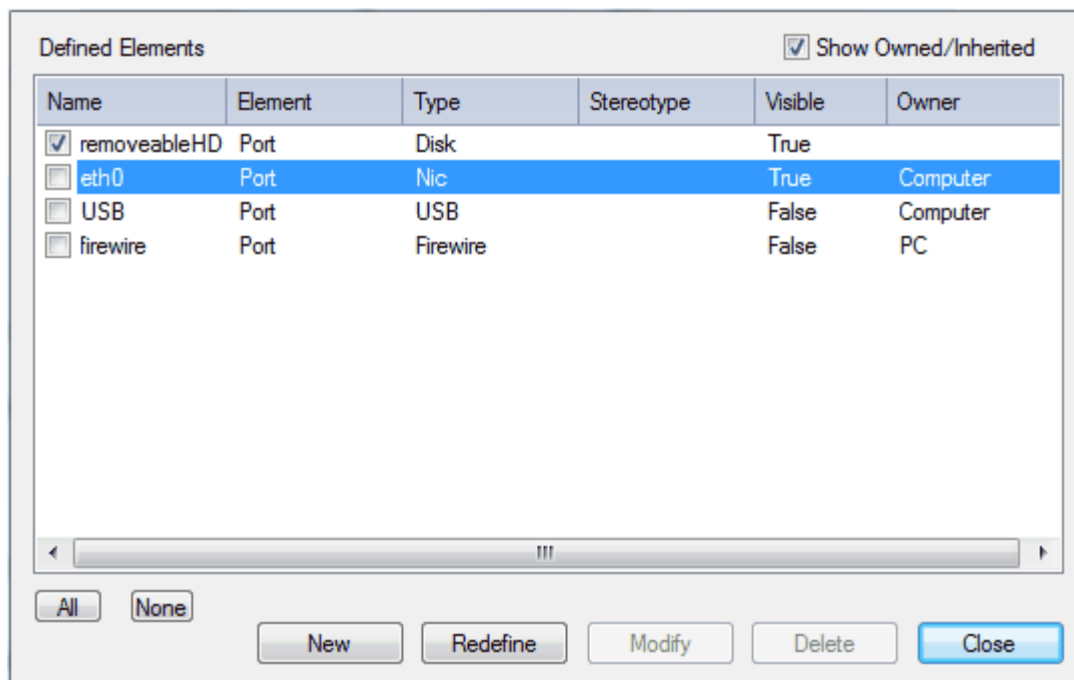
### 2.2.19.2 Inherited and Redefined Ports

A [Port](#)<sup>[169]</sup> is a *redefinable* and *re-useable* property of a composite classifier. So, as for attributes, any Class can inherit Ports from its parent and realized interfaces. If you have an inheritance hierarchy with Ports defined in the parent Classes, when you open the **Embedded Elements** window the inherited Ports and their named owners are listed there.

It is possible to expose, for design purposes, an inherited Port (that is, the child Class is re-using the parent Port). In this case, Enterprise Architect creates a clone of the re-used Port and marks it as read-only in the child Class. This is convenient for modeling Port interactions in child Classes where the Ports are defined in the parent elements.

It is also possible to redefine a Port in a child Class, so that the name is the same but the child is a modifiable clone of the original. This is useful where a child Class places additional restrictions or behavior on the Port. The **Embedded Elements** window enables you to highlight an inherited Port and mark it as *redefined*; this creates a new Port on the child Class, which is editable but still logically related to the initial Port.

The **Embedded Elements** window below illustrates Port inheritance. The Port *removableHD* is owned by the child Class. The Ports *eth0* and *USB* are owned by the *Computer* Class. The Port *firewire* has been added to *PC*. If any of the inherited Ports are made visible, they are considered re-use Ports and appear on the child in read-only format. Using the **Redefine** button, the inherited Port can be copied down and made writeable.



### 2.2.19.3 The Property Tab

The element **Properties** dialog for Ports and Parts has a **Property** tab in place of the Class element **Details** tab.

The screenshot shows the 'Property' tab of a dialog box. The tabs at the top are: General, Property, Requirements, Constraints, Links, Scenarios, Files, and Tagged Values. The 'Property' tab is selected. The dialog contains the following elements:

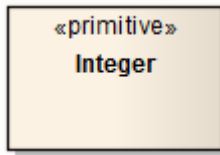
- Type:** A dropdown menu showing 'Class8' and a '...' button.
- Initial:** A text field containing '10' and a '...' button.
- Qualifiers...** button.
- Const** and **Derived** checkboxes.
- Multiplicity** section:
  - Lower bound:** 0
  - Upper bound:** 2
  - Allow Duplicates** and **Multiplicity is Ordered** checkboxes.
- Redefined Property:** A table with one header row 'Property' and one empty data row. Below the table are 'Add...' and 'Delete' buttons.
- Subsetted Property:** A table with one header row 'Property' and one data row containing the text '<input element>Development Model::Class Model::ClassLib.m\_delivery'. Below the table are 'Add...' and 'Delete' buttons.
- At the bottom of the dialog are four buttons: **OK**, **Cancel**, **Apply**, and **Help**.

This tab defines the type, initial value, [Qualifiers](#)<sup>[173]</sup>, multiplicity, and redefined and subsetted properties of the Port or Part.

You set the Qualifiers by clicking on the **Qualifiers** button, to display the [Qualifiers](#)<sup>[173]</sup> dialog.

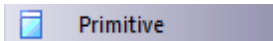
You add **Redefined** and **Subsetted Properties** by clicking on the appropriate **Add** button, to display the **Select Property** dialog.

### 2.2.20 Primitive



A *Primitive* element identifies a predefined data type, without any relevant substructure (that is, it has no parts in the context of UML). It could be regarded as a conceptual [Data Type](#)<sup>[159]</sup>.

#### Toolbox Icon



#### OMG UML Specification

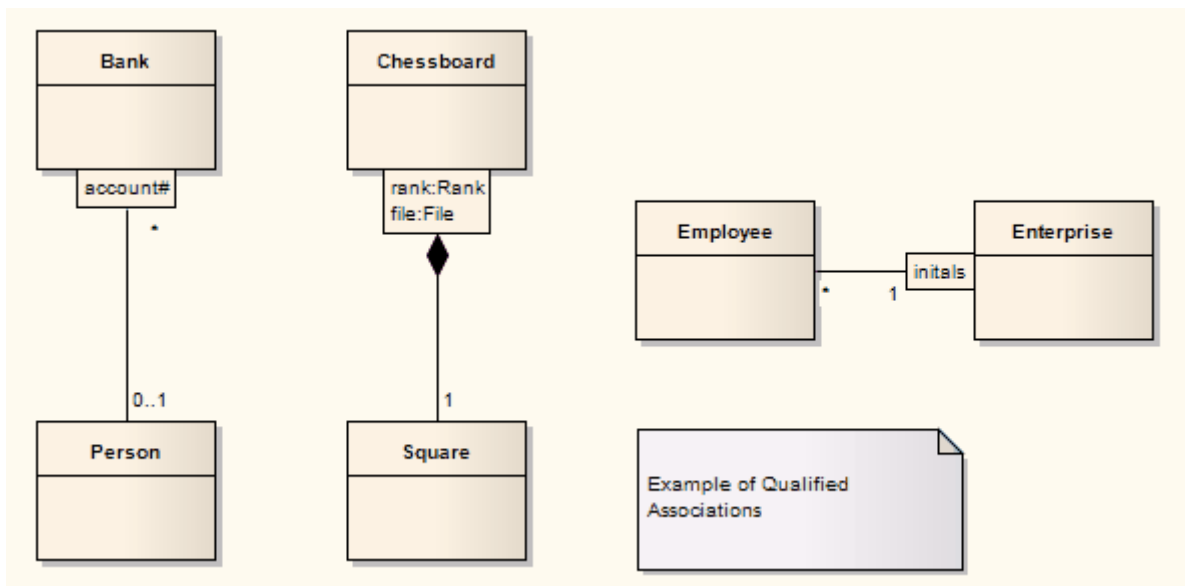
The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 124*) states:

*A primitive data type may have an algebra and operations defined outside of UML, for example, mathematically ... The run-time instances of a primitive type are data values. The values are in many-to-one correspondence to mathematical elements defined outside of UML (for example, the various integers). Instances of primitive types do not have identity. If two instances have the same representation, then they are indistinguishable.*

### 2.2.21 Qualifiers

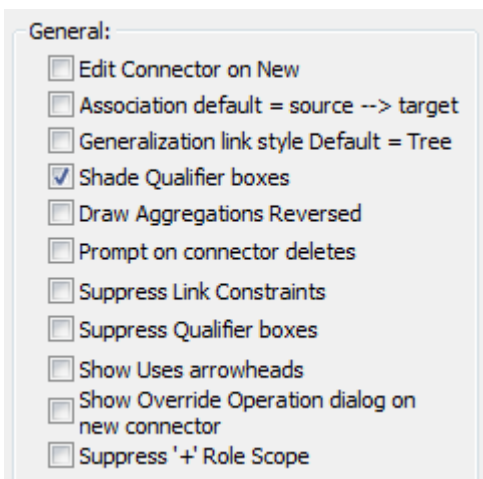
*Qualifiers* are ordered sets of properties of an Association end point, a [Part](#)<sup>[168]</sup>, a [Port](#)<sup>[169]</sup>, or an Attribute, that limit the nature of the relationship between two classifiers or objects. You define a qualifier on the [Qualifiers](#)<sup>[175]</sup> dialog, which you display by clicking on the [ ... ] button at the end of the **Qualifiers** field on the Association, Part, Port or Attribute **Properties** dialog.

Some examples of qualified Associations are shown in the following diagram:



**Notes:**

- When typing multiple Qualifiers into the **Qualifier(s)** field on a **Properties** dialog, separate them with a semi-colon; each Qualifier then displays on a separate line. For example, in the diagram the Qualifier *'rank: Rank;file:File'* has been rendered in two lines, with a line break at the ; character.
- You can enable or disable Qualifier rectangles in the **Diagram** page of the **Options** dialog (select the **Tools | Options | Diagram** menu option). If disabled, the old style text Qualifiers are used. It is not recommended that you disable Qualifiers as they are an integral part of the UML.
- You can enable or disable a mild shading on the Qualifier rectangles in the **Links** page of the **Options** dialog.



## OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 129*) states:

*A qualifier declares a partition of the set of associated instances with respect to an instance at the qualified end (the qualified instance is at the end to which the qualifier is attached). A qualifier instance comprises one value for each qualifier attribute. Given a qualified object and a qualifier instance, the number of objects at the other end of the association is constrained by the declared multiplicity. In the common case in which the multiplicity is 0..1, the qualifier value is unique with respect to the qualified object, and designates at most one associated object. In the general case of multiplicity 0..\*, the set of associated instances is partitioned into subsets, each selected by a given qualifier instance. In the case of multiplicity 1 or 0..1, the qualifier has both semantic and implementation consequences. In the case of multiplicity 0..\*, it has no real semantic consequences but suggests an implementation that facilitates easy access of sets of associated instances linked by a given qualifier value.*

### 2.2.21.1 Qualifiers Dialog

The **Qualifiers** dialog is used to define the [Qualifiers](#) [173] of an Association connector end, [Port](#) [169], [Part](#) [168] or Attribute.

The screenshot shows the 'Qualifiers' dialog box with the 'General' tab selected. The fields are as follows:

- Name: aqua1
- Alias: a\_qualifier
- Type: UnlimitedNatural
- Scope: Public
- Stereotype: (empty)
- Initial: 101
- Notes: Notes on [aqua1](#).

Checkboxes for 'Derived', 'Static', and 'Const' are present and unchecked. Below the fields is a 'Qualifiers' table:

Name	Type	Initial Value
◆ aqua1	UnlimitedNatural	101
◆ aqua2	UnlimitedNatural	10

Buttons for 'New', 'Copy', 'Save', 'Delete', 'OK', 'Cancel', and 'Help' are located at the bottom of the dialog.

#### General Tab

Review, edit or complete the fields as indicated in the following table.

Field	Use to
<b>Name</b>	Display the name of the Qualifier. For a new Qualifier, type the name (with no spaces).
<b>Alias</b>	Display an optional alias for the Qualifier. If necessary, type in a new alias.
<b>Type</b>	<p>Display the Qualifier type.</p> <p>The type can be defined by the code language (data type) or by a classifier element. When you click on the drop-down arrow, the set of values in the list provides the appropriate data types.</p> <p>To select or define possible classifiers, either click on the <b>Select Type</b> option in the list, or click on the [ ... ] (Select) button to display the <b>Select &lt;Item&gt;</b> dialog.</p> <p>To add new code language data types that can be displayed in this list, see the Data Types topic in <i>UML Model Management</i>.</p>

Field	Use to
<b>Scope</b>	Define the Qualifier as <b>Public</b> , <b>Protected</b> , <b>Private</b> or <b>Package</b> . If necessary, click on the drop-down arrow and select a different scope.
<b>Stereotype</b>	Define the optional stereotype of the Qualifier. If necessary, either type a different stereotype name or click on the drop-down arrow and select a stereotype.
<b>Derived</b>	Indicate that the Qualifier is a calculated value. If you select this checkbox, the Qualifier name on the element has the derived symbol ( <i>l</i> ) as a prefix.
<b>Static</b>	Indicate that the Qualifier is a static member.
<b>Const</b>	Indicate that the Qualifier is a constant.
<b>Initial</b>	Display an optional initial value. If necessary, type in a new initial value.
<b>Notes</b>	Enter any free text notes associated with the Qualifier. You can format the notes text using the <b>Rich Text Notes</b> toolbar at the top of the field (see <i>Using Enterprise Architect - UML Modeling Tool</i> ).

To change the position of a Qualifier in the list in the **Qualifiers** panel, click on the **Scroll Up** or **Scroll Down** (hand) buttons.

### Detail Tab

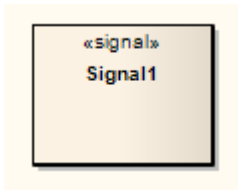
Use the **Detail** tab to model additional properties of a selected Qualifier, such as its multiplicity, redefined properties and subsetted properties.

Select a Qualifier on the **General** tab, then review, edit or complete the **Detail** tab fields as indicated in the following table.

Field	Use to
Multiplicity	
Lower bound	Define a lower limit to the number of elements allowed in the collection.
Upper bound	Define an upper limit to the number of elements allowed in the collection.
Allow Duplicates	Indicate that duplicates are allowed. Maps to the UML property <i>isUnique</i> , value <i>FALSE</i> ).
Multiplicity is Ordered	Indicate that the collection is ordered.
Redefined Property	Review the redefined properties for the Qualifier. Add redefined properties by clicking on the <b>Add</b> button to display the <b>Select Property</b> dialog.
Subsetted Property	Review the subsetted properties for the qualifier. Add subsetted properties by clicking on the <b>Add</b> button to display the <b>Select Property</b> dialog.



## 2.2.22 Signal



A *Signal* is a specification of [Send](#)<sup>[129]</sup> request instances communicated between objects, typically in a [Class](#)<sup>[56]</sup> or [Package](#)<sup>[55]</sup> diagram. The receiving object handles the [Received](#)<sup>[127]</sup> request instances as specified by its *receptions*. The data carried by a Send request is represented as attributes of the Signal. A Signal is defined independently of the classifiers handling the signal occurrence.

To define a reception, create an operation in the receiving object and assign the stereotype <<signal>> to it. The reception has the same name as the signal that the object can receive.

### Toolbox Icon



### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 450*) states:

*A signal triggers a reaction in the receiver in an asynchronous way and without a reply. The sender of a signal will not block waiting for a reply but continue execution immediately. By declaring a reception associated to a given signal, a classifier specifies that its instances will be able to receive that signal, or a subtype thereof, and will respond to it with the designated behavior.*

And (*UML Superstructure Specification, v2.1.1, p. 447 - 448*):

*A reception is a declaration stating that a classifier is prepared to react to the receipt of a signal. A reception designates a signal and specifies the expected behavioral response. The details of handling a signal are specified by the behavior associated with the reception or the classifier itself. ...Receptions are shown using the same notation as for operations with the keyword <signal>*

## 2.3 Inbuilt and Extension Stereotypes

There are many other UML elements that you can also work with in Enterprise Architect, most of which are basic elements extended by the use of stereotypes. This topic gives a brief introduction to some of these elements.

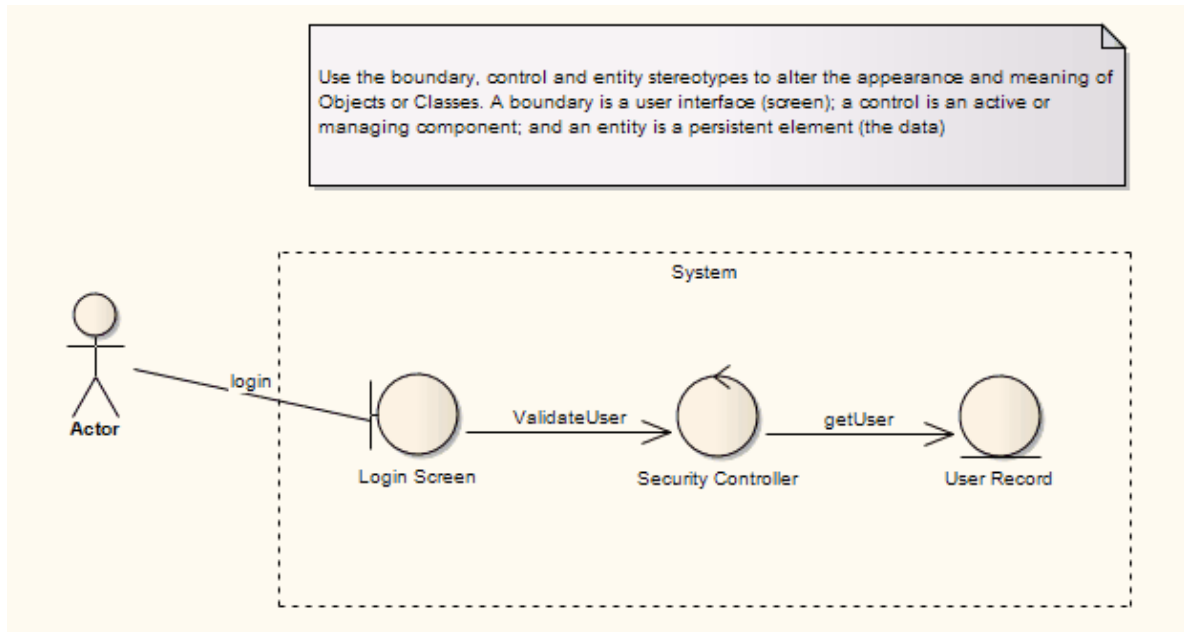
- [Analysis Stereotypes](#)<sup>[179]</sup>
- [Boundary Element](#)<sup>[179]</sup>
- [Composite Elements](#)<sup>[180]</sup>
- [Control Element](#)<sup>[181]</sup>
- [Entity Element](#)<sup>[182]</sup>
- [Event Elements](#)<sup>[183]</sup>
- [Hyperlinks](#)<sup>[184]</sup>
- [N-Ary Association](#)<sup>[187]</sup>
- [Process](#)<sup>[189]</sup>
- [Requirements](#)<sup>[189]</sup>
- [Screen](#)<sup>[190]</sup>
- [Table](#)<sup>[192]</sup>
- [UI Control Element](#)<sup>[192]</sup>
- [Web Stereotypes](#)<sup>[194]</sup>

For more information on the use of stereotypes in Enterprise Architecture, see the *UML Stereotypes* topic in

Extending UML With Enterprise Architect.

### 2.3.1 Analysis Stereotypes

Enterprise Architect has some built in stereotypes that you can assign to an element during analysis. The effect of these stereotypes is to display a different icon from the normal element icon, providing a visual key to the element purpose. The *Robustness* diagram below illustrates the main types of inbuilt icons for elements:



The stereotypes used are:

- [Boundary](#)<sup>[179]</sup> - for a system boundary (for example, a Login screen)
- [Control](#)<sup>[181]</sup> - to specify an element is a controller of some process (as in the Model-View-Controller pattern)
- [Entity](#)<sup>[182]</sup> - the element is a persistent or data element

Also see the [Business Modeling](#)<sup>[75]</sup> elements, used in Business Modeling and Business Interaction diagrams.

### 2.3.2 Boundary



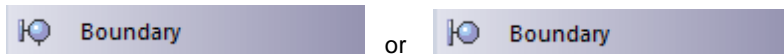
A *Boundary* is a stereotyped [Object](#)<sup>[165]</sup> that models some system boundary, typically a user interface screen. You can also create a Boundary as a stereotyped [Class](#)<sup>[152]</sup>. See the [Create a Boundary](#)<sup>[180]</sup> topic.

A Boundary is used in the conceptual phase to capture users interacting with the system at a screen level (or some other boundary interface type). It is often used in [Sequence](#)<sup>[39]</sup> and *Robustness* ([Analysis](#)<sup>[67]</sup>) diagrams. It is the *View* in the [Model-View-Controller](#)<sup>[180]</sup> pattern.

#### Tip:

Use Boundary elements in analysis to capture user interactions, screen flows and element interactions (or 'collaborations').

## Toolbox Icon



### 2.3.2.1 Create a Boundary

#### Using the Toolbox

To create a [Boundary](#)<sup>[179]</sup> element on a diagram as an Object, follow the steps below:

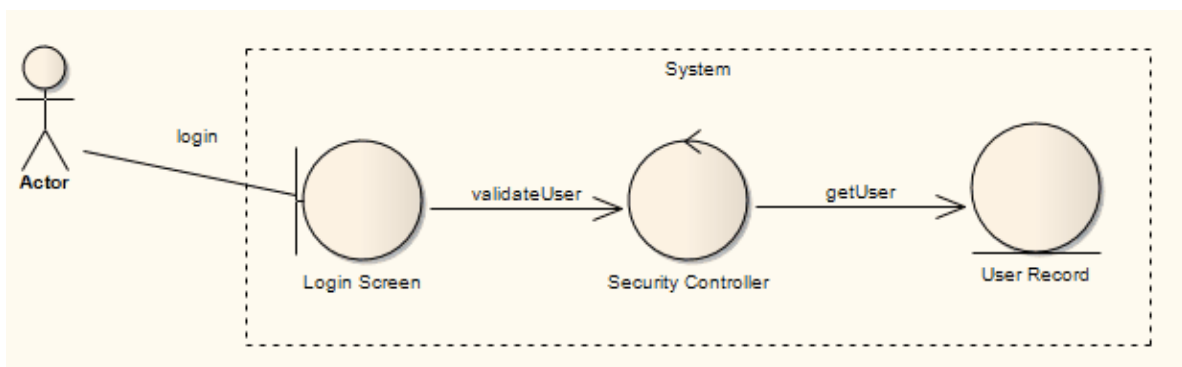
1. In the Enterprise Architect UML **Toolbox**, select the **More Tools | Analysis** menu option.
2. From the **Analysis Elements** page, drag the *Boundary* element onto the diagram.

#### Using the Properties Dialog

To create a Boundary element as a stereotyped Class, using the Class **Properties** dialog, follow the steps below:

1. Insert a new Class.
2. Right-click on the element and select the **Properties** context menu option; the **Properties** dialog displays.
3. In the **Stereotype** field, type the value **boundary**.
4. Click on the **Apply** and **OK** buttons.
5. Save the diagram (**[Ctrl]+[S]**).

The following illustration shows an [Actor](#)<sup>[94]</sup> interacting with a Boundary (in this case, a Login screen).



#### Note:

The Model-View-Controller (MVC) pattern is a design pattern for building a wide range of applications that have a user interface, business or application logic and persistent data.

### 2.3.3 Composite Elements

Enterprise Architect supports *Composite elements* for Classes, Objects, Use Cases and such. A Composite element is a pointer to a child diagram.

#### Create a Composite Element

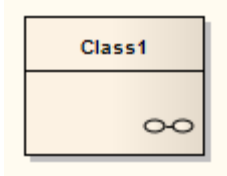
To set Composite elements from the element context menu, follow the steps below:

1. Create the element to set as a Composite element.
2. Right-click on the element in the diagram and select the **Advanced | Make Composite** context menu option.

**Note:**

If the **Make Composite** option is not listed in the context menu, the option is not available for the type of element you have selected.

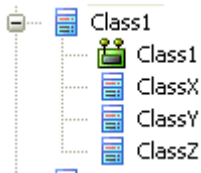
The element displays as follows:



Note the small icon in the bottom right hand corner indicating that this is now a Composite element.

3. Double-click on the Composite element to access the child diagram that it points to.

The Composite element and its child diagram are represented in the **Project Browser** as follows:



Note that *ClassX*, *ClassY* and *ClassZ* are elements in the child diagram.

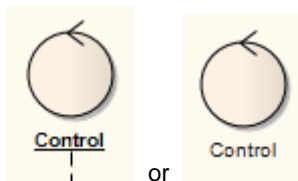
### Alternative Notation

Composite elements can show their contents instead of their usual notation. To enable this, right-click on the element to open the context menu, then select the **Advanced | Show Composite Diagram** option.

### The Automation Interface

Automation support is available for Composite elements. *Element* has an *Elements* collection and a *Diagrams* collection. See *SDK for Enterprise Architect*.

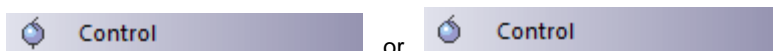
### 2.3.4 Control



A *Control* is a stereotyped [Object](#)<sup>[165]</sup> that models a controlling entity or manager. A Control organizes and schedules other activities and elements, typically in [Analysis](#)<sup>[67]</sup> (including Robustness), [Sequence](#)<sup>[39]</sup> and [Communication](#)<sup>[49]</sup> diagrams. It is the *controller* of the [Model-View-Controller](#)<sup>[182]</sup> pattern.

You can also create a Control as a stereotyped [Class](#)<sup>[152]</sup>. See the [Create a Control Element](#)<sup>[182]</sup> topic.

### Toolbox Icon



### 2.3.4.1 Create a Control Element

#### Using the Toolbox

To create a [Control](#)<sup>[187]</sup> element on a diagram as an Object, follow the steps below:

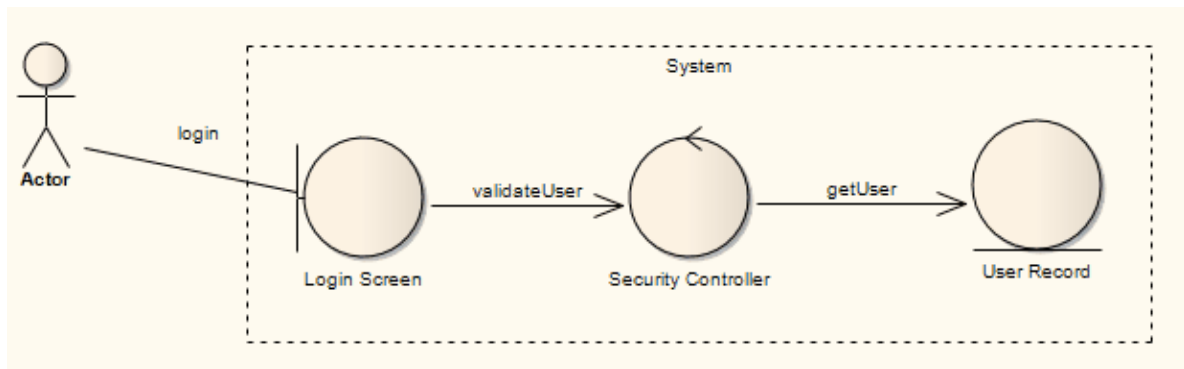
1. In the Enterprise Architect UML **Toolbox**, select the **More Tools | Analysis** menu option.
2. From the **Analysis Elements** page, drag the *Control* element onto the diagram.

#### Using the Properties Dialog

To create a Control element as a stereotyped Class, using the Class **Properties** dialog, follow the steps below:

1. Insert a new Class.
2. Right-click on the element and select the **Properties** context menu option; the **Properties** dialog displays.
3. In the **Stereotype** field, type the value **control**.
4. Click on the **Apply** and **OK** buttons.
5. Save the diagram (**[Ctrl]+[S]**).

The appearance changes as illustrated in the following diagram (for the *Security Controller* element):



#### Note:

The *Model-View-Controller (MVC)* pattern is a design pattern for building a wide range of applications that have a user interface, business or application logic and persistent data.

### 2.3.5 Entity



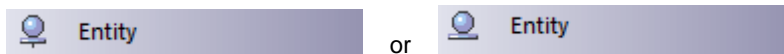
From: [Sequence Diagram](#)<sup>[397]</sup>

[Communication](#)<sup>[497]</sup>, [Object](#)<sup>[587]</sup>, [Analysis](#)<sup>[677]</sup>  
(including Robustness) Diagrams

An *Entity* is a stereotyped [Object](#)<sup>[165]</sup> that models a store or persistence mechanism that captures the information or knowledge in a system. It is the *Model* in the [Model-View-Controller](#)<sup>[182]</sup> pattern.

You can also create an Entity as a stereotyped [Class](#)<sup>[152]</sup>. See the [Create an Entity](#)<sup>[183]</sup> topic.

## Toolbox Icon



### 2.3.5.1 Create an Entity

#### Using the Toolbox

To create an [Entity](#)<sup>[182]</sup> element on a diagram as an Object, follow the steps below:

1. In the Enterprise Architect UML **Toolbox**, select the **More Tools | Analysis** menu option.
2. From the **Analysis Elements** page, drag the *Entity* element onto the diagram.

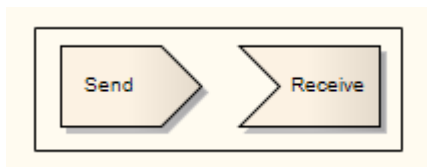
#### Using the Properties Dialog

To create an Entity element as a stereotyped Class, using the Class **Properties** dialog, follow the steps below:

1. Insert a new Class.
2. Right-click on the element and select the **Properties** context menu option; the **Properties** dialog displays.
3. In the **Stereotype** field, type the value **entity**.
4. Click on the **Apply** and **OK** buttons.
5. Save the diagram (**[Ctrl]+[S]**).

### 2.3.6 Event

The UML includes two elements that are used to model *Events*. The first element is the *Send Event*. This element models the generation of a stimulus in the system and the passing of that stimulus to other elements, either within the system or external to the system.



The second element is the *Receive Event*, which is depicted as a rectangle with a recessed 'V' on the left side. This element indicates that an event occurs in the system due to some external or internal stimulus. Typically this invokes further activities and processing.

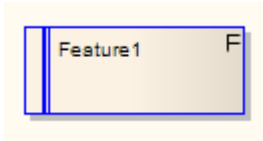
Send and Receive Events can be added from the **Analysis** and **Activity Element** pages of the Enterprise Architect UML **Toolbox** (see *Using Enterprise Architect - UML Modeling Tool*).

If you should select the wrong type of event, or otherwise want to change the type, right-click on the Event and select the **Advanced | Make Sender** or **Advanced | Make Receiver** context menu option, as appropriate.

#### Toolbox Icons



### 2.3.7 Feature



A *Feature* is a small, granular function or characteristic expressed in client-valued terms as a satisfaction of a requirement; for example: 'context-sensitive Help', or 'ability to reverse-engineer VB.Net'.

Features are the primary requirements-gathering artifact of the [Feature-Driven Design \(FDD\) methodology](#). They define the product feature that satisfies what a [Requirement](#)<sup>[189]</sup> element has formalized as a contractual, testable, expected deliverable (for example: requirement - 'every element must provide context-sensitive Help'; feature - 'every element provides context-sensitive Help'). One Feature might realize one or more Requirements, and one Requirement might be realized by more than one Feature.

Features also have relationships with [Use Cases](#)<sup>[146]</sup>. A Use Case defines the interaction a user has with the system in order to satisfy one or more Requirements. The Feature identifies the facility that provides the means for that interaction.

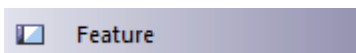
Feature elements are non-UML and are not related to UML-defined features, which are either *BehavioralFeatures* (operations, or methods) or *StructuralFeatures* ([Ports](#)<sup>[169]</sup>, [Parts](#)<sup>[168]</sup> and attributes) (see *Using Enterprise Architect - UML Modeling Tool*).

Feature elements are available from the [Requirements](#) page of the Enterprise Architect UML [Toolbox](#).

#### Note:

Feature elements can be created with or without an identifying **F** in the top right corner of the element. To toggle the display of this letter, select or deselect the **Show stereotype icon for requirements** checkbox on the [Options](#) dialog, [Objects](#) page. (See *Using Enterprise Architect - UML Modeling Tool*.)

#### Toolbox Icon



### 2.3.8 Hyperlinks



You can place a *Hyperlink* element onto a diagram. This element is a type of text element, but one that can contain a pointer to a range of objects such as associated document files, web pages, Help, model features and even other Enterprise Architect model files. When you double-click on the element, Enterprise Architect executes the link. To add a Hyperlink element, drag the *Hyperlink* icon from the [Common](#) page of the Enterprise Architect UML [Toolbox](#) onto the diagram.

(Alternatively, click on the **Hyperlink** icon in the [UML Elements](#) toolbar and then click on the diagram.)



#### Configure the Hyperlink

When you add the Hyperlink to the diagram, the [Hyperlink Details](#) dialog displays. If you want to display the information in a more readable layout, you can resize the dialog.

You first select the type of object to link to, by clicking on the drop-down arrow in the **Hyperlink Type** field. The **Hyperlink Details** dialog displays the appropriate fields, prompts or dialog to enable you to specify the object to link to. For example, if you intend to hyperlink to:

- an attribute, the **Set Attribute dialog** displays to enable you to select that attribute.
- a file, the **Action** field displays to enable you to specify whether to **Open** the file in read only mode, or **Edit** the file; in either case the file is opened within Enterprise Architect if possible, or, if not possible, with the Windows default viewer/editor for the file type. For example, if you hyperlink to a .rtf file, you can *view* it in whichever viewer is appropriate; however, you cannot *edit* .rtf files in Enterprise Architect, so the file always opens in the *Windows* default .rtf editor.
- a diagram, the **Select a Diagram** dialog displays, which enables you to select the diagram from anywhere in the project; you can filter the selection to diagrams of certain types.

If you select **EA Command**, the **Hyperlink Address** field changes to a drop-down list of Enterprise Architect commands. You can select **LocalPath** and click on the [ ... ] (Browse) button to display the **Local Paths** dialog (see *Code Engineering Using UML Models*), which you complete as required. Subsequently, when you click on the hyperlink the **Local Paths** dialog immediately displays and you can apply, switch, expand or update the current path.

Once you have defined the object and its location, you can change the location either by overtyping the **Hyperlink Address** field or by clicking on the [ ... ] (Browse) button.

In the **Alias** field, type the text to display in the hyperlink. If you do not provide an alias, either the text defaults to the link itself, or (for certain link targets such as a matrix profile) the dialog generates a simple text instruction.

If you prefer to display only the hyperlink text, without the icon, select the **Hide Icon** checkbox.

#### Notes:

- If required, you can create a number of empty hyperlinks to complete later. If you then double-click on an empty hyperlink, the **Hyperlink Details** dialog displays and you can enter the details.
- Once you have created the hyperlink, you can also edit the hyperlink text by clicking once on the field and once on the text, then right-clicking and selecting the **Edit Selected** context menu option.

Note that you can add notes to the hyperlink, which display in the **Hyperlink Details** dialog when you right-click on the hyperlink and select the **Properties** context menu option. You can format these notes using the **Rich Text Notes** toolbar (see *Using Enterprise Architect - UML Modeling Tool*).

There are three alternative methods of creating a hyperlink, as explained in the following sections.



### Create Hyperlink To File

You can create a hyperlink on a diagram to an external file simply by clicking on the file in a file list (such as Windows Explorer) or on your Desktop and dragging it onto the diagram. A short context menu displays with two options - **Hyperlink** and **Artifact**. Click on the **Hyperlink** option to create the hyperlink on the diagram. The link is effective immediately, and you can right-click on it to add or change properties as described above.

### Create Action As Hyperlink

You can create an [Action](#) element to represent a wide range of behaviors and actions, including a hyperlink. To do this, follow the steps below:

1. Drag an Action element from the **Activity** page of the **Toolbox** onto the diagram. A context menu immediately displays.
2. Select the **Other** menu option. The **New Action** dialog displays, with the **Other** radio button selected.
3. Click on the drop down arrow on the field in the **Select kind** panel, and click on **Hyperlink**.
4. Click on the **OK** button. The *Hyperlink Action* element displays on the diagram.
5. Right click on the element and select the **Advanced | Set Hyperlink** menu option. The **Hyperlink Details** dialog displays.
6. Set the hyperlinks properties as described above.

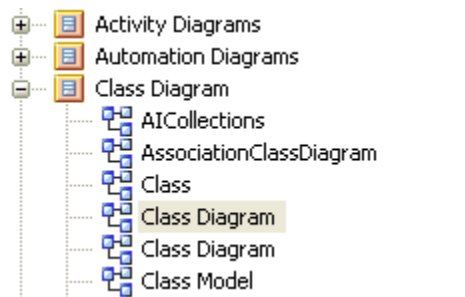
### Create Hyperlink Between Diagrams

To create a hyperlink between diagrams, follow the steps below.

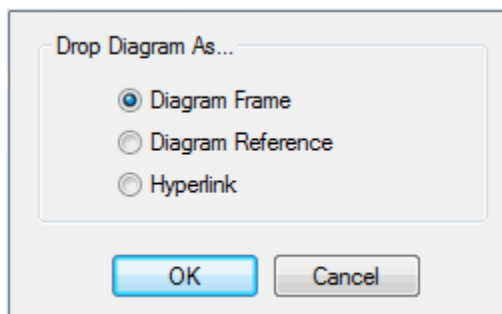
#### Note:

If the hyperlink appears as a Sub Activity, select the **Tools | Options | Diagram | Behavior** menu option and deselect the **Use Automatic SubActivities** checkbox.

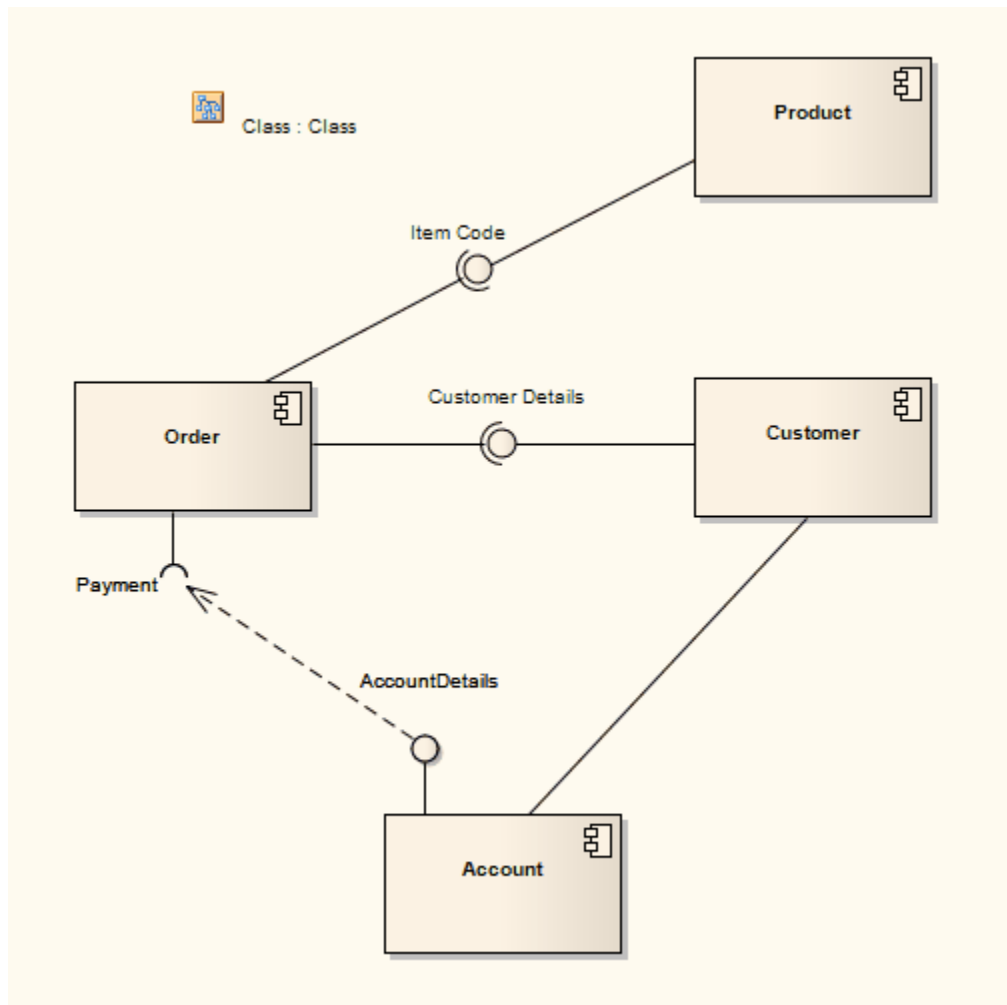
1. Open the diagram in which to display the hyperlink to another diagram. From the **Project Browser** select the diagram you want to create a hyperlink to.



2. Drag the diagram on to the current diagram. The **Select Type** dialog displays.



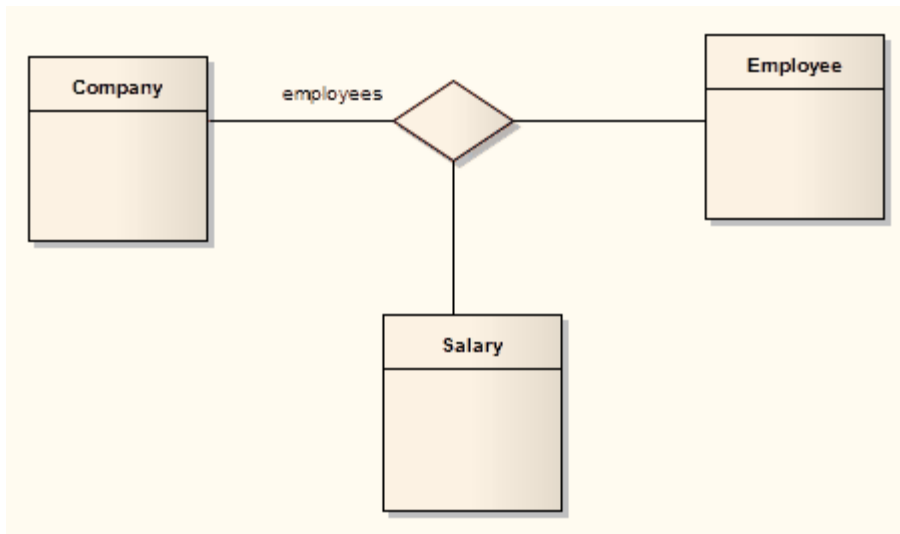
3. Select the **Hyperlink** option and click on the **OK** button. The final hyperlinked diagram should resemble the diagram below, where the *Class* diagram is the diagram to which the *Product Order* diagram hyperlinks (notice that the hyperlink icon is different).



### 2.3.9 N-Ary Association

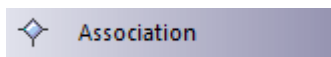


An *n-Ary Association* element is used to model complex relationships between three or more elements, typically in a [Class diagram](#)<sup>[56]</sup>. It is not a commonly-employed device, but can be used to good effect where there is a dependant relationship between several elements. It is generally used with the [Associate](#)<sup>[199]</sup> connector, but the relationships can include other types of connector.

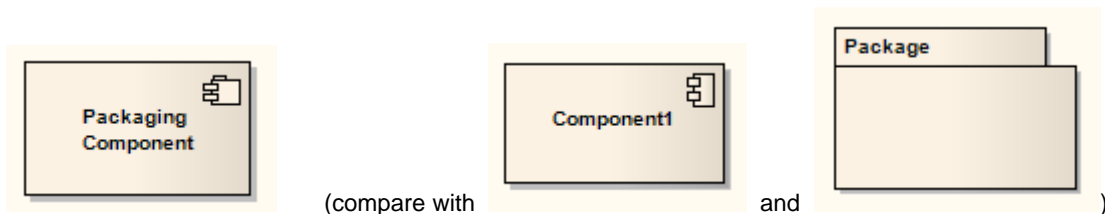


In the example above there is a relationship between a *Company*, an *Employee* and a *Salary*.

### Toolbox Icon

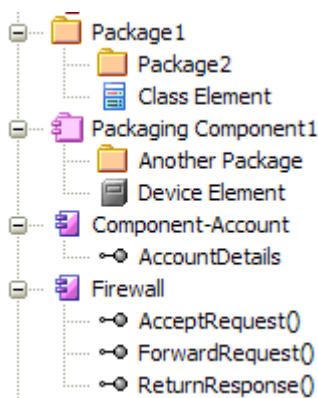


### 2.3.10 Packaging Component



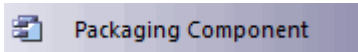
A *Packaging Component* is an element that appears very similar to a [Component](#)<sup>[158]</sup> in a diagram but behaves as a [Package](#)<sup>[168]</sup> in the **Project Browser** (that is, it can be version controlled and can contain other Packages and elements). It is typically used in [Component diagrams](#)<sup>[65]</sup>.

In the **Project Browser**, the three elements display as shown below:

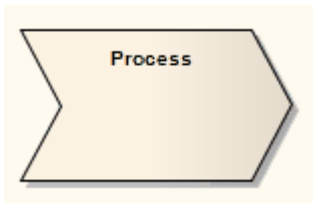


The Component element cannot contain child Packages or Packaging Components.

### Toolbox Icon



#### 2.3.11 Process



A *Process* is an [Activity](#)<sup>[90]</sup> element with the stereotype **process**, which expresses the concept of a business process. Typically this involves inputs, outputs, work flows, goals and connections with other Processes. The Process element is typically used in [Analysis diagrams](#)<sup>[67]</sup>.

Business processes typically range across many parts of the organization and span one or more systems.

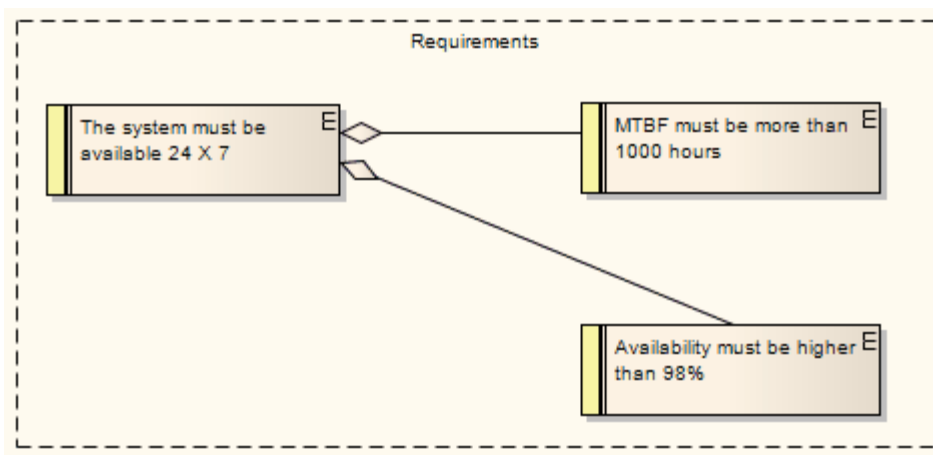
### Toolbox Icon



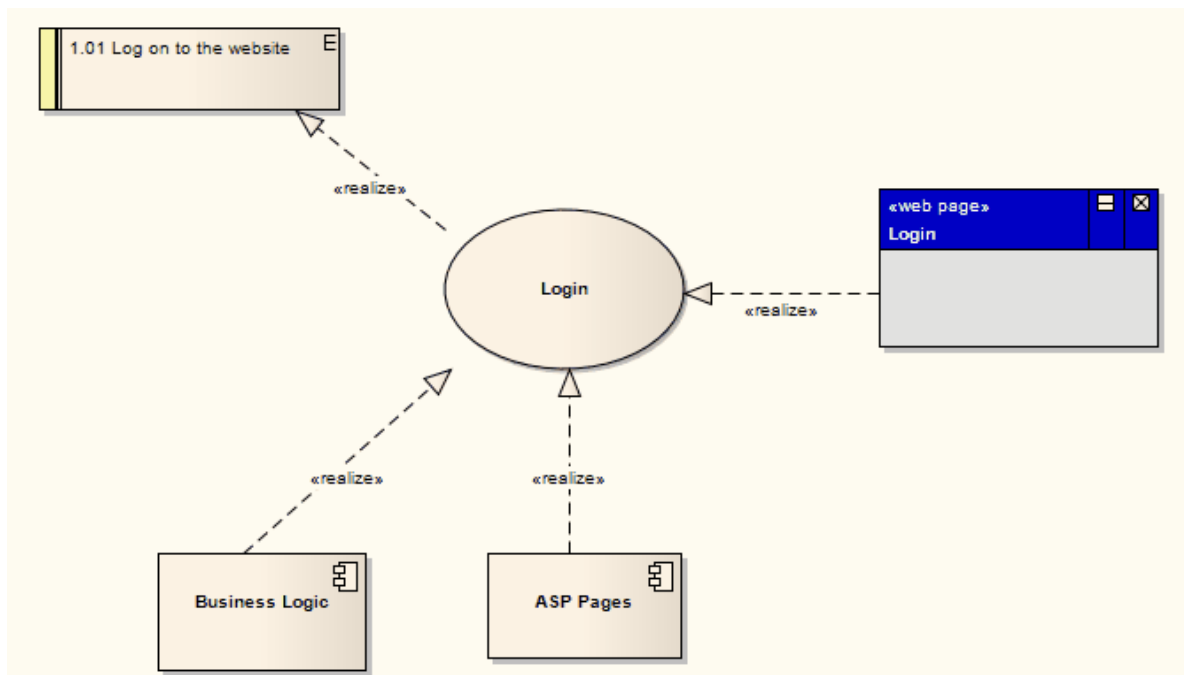
#### 2.3.12 Requirements

As an analysis step, often it is desirable to capture simple *system requirements*. These are eventually realized by [Use Cases](#)<sup>[146]</sup>.

In the initial requirement gathering phase, cataloging requirements can be achieved using the *Requirement* extension on a [Custom diagram](#)<sup>[69]</sup>.



Requirements can also be aggregated to create a hierarchy. The diagram below illustrates how this might be done.

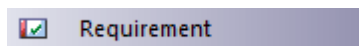


A requirement that a user can log into a website is implemented by the *Login* Use Case, which in turn is implemented by the *Business Logic*, *ASP Pages* and *Login Web Page*. Using this approach, you can easily model quite detailed and complex dependencies and implementation relationships.

#### Notes:

- External requirements can be created with or without an identifying **E** in the top right corner of the element. To toggle the display of this letter, select or deselect the **Show stereotype icon for requirements** checkbox on the **Options** dialog, **Objects** page. (See *Using Enterprise Architect - UML Modeling Tool*.)
- The colors on Requirement elements identify the status of the requirement. You change the status - and hence color - on the element **Properties** dialog. You set the color for each status on the **Status Types** dialog. (See *Requirements Management*.)

### Toolbox Icon

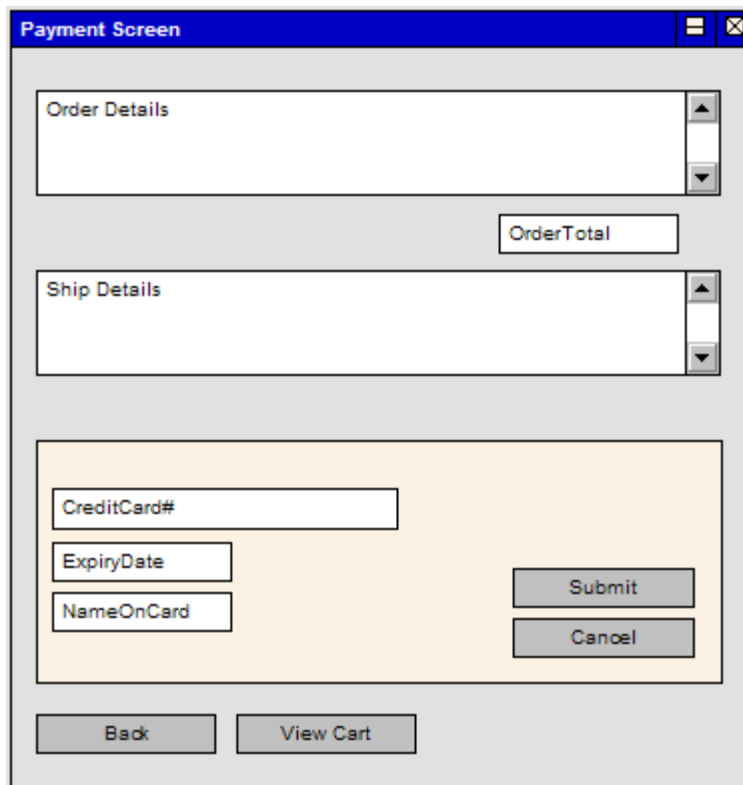


### 2.3.13 Screen

A *Screen* is used to prototype User Interface screen flow. By using UML features such as requirements, constraints and scenarios against [User Interface](#) <sup>[73]</sup> diagram elements, you can build up a solid and detailed understanding of user interface behavior without having to use code. This becomes an excellent means of establishing the precise behavior the system has from a user perspective, and in conjunction with the Use Case model (see *Using Enterprise Architect - UML Modeling Tool*), defines exactly how a user gets work done.

Web pages can also be prototyped and specified rigorously using Enterprise Architect's custom interface extensions.

The example diagram below illustrates some features of Enterprise Architect's screen modeling extensions that support web page prototyping. By adding requirements, rules, scenarios and notes to each element, a detailed model is built up of the form or web page, without having to resort to GUI builders or HTML.

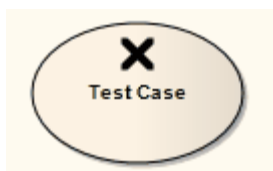
**Note:**

Enterprise Architect displays [UI Controls](#) <sup>1921</sup> as a range of special icons, depending on the stereotype used; for example, a Control stereotyped as a «list» displays with a vertical scroll bar.

### Toolbox Icon



### 2.3.14 Test Case



A *Test Case* is a stereotyped [Use Case](#) <sup>1461</sup> element. You might use it to extend the facilities of the **Testing** window (see *Project Management With Enterprise Architect*), by applying element properties and capabilities to the tests of a feature represented by another element or - more appropriately - set of elements. That is, you can define in one go - in the **Testing** window for the Test Case element - the details of the tests that apply to each of several elements, instead of recording the details separately in each element.

Within the Test Case element properties you can define test requirements and constraints, and associate the test with test files. You can also link the element to Document Artifacts or (in the Corporate, Business and Software Engineering, System Engineering and Ultimate editions) directly to linked documents, such as a Test Plan.

The Test Case element enables you to give greater visibility to tests, in the **Project Browser**, **Element List**,

Model Search, Relationship Matrix, Traceability window and reports.

The Test Case element is available through the **Use Case** and **Maintenance** pages of the Enterprise Architect UML **Toolbox**.

### 2.3.15 Table



A *Table* is a stereotyped [Class](#)<sup>[152]</sup>. It is drawn with a small table icon in the upper right corner. You typically use this element in [Data Modeling](#)<sup>[75]</sup> and [Class](#)<sup>[56]</sup> diagrams.

A Table element has a special **Properties** dialog, with settings for database type and the ability to set column information and data-related operations such as triggers and indexes. When setting up a Table, make sure you set the default database type for that Table, otherwise you do not have any data types to choose from when creating columns. (See *Code Engineering Using UML Models*.)

### Toolbox Icon



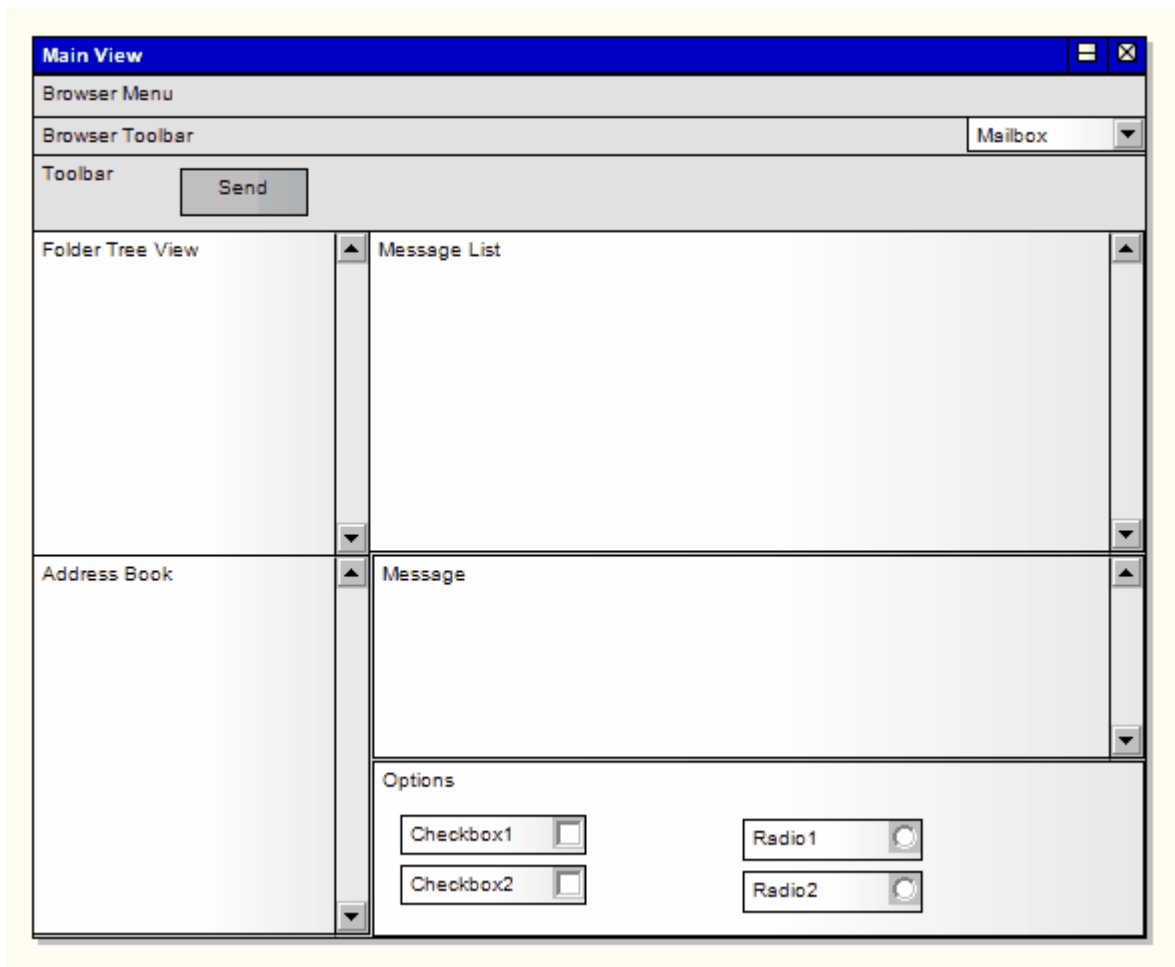
### 2.3.16 UI Control Element

A *UI Control element* represents a user interface control element (such as an edit box). It is used for capturing the components of a [screen](#)<sup>[190]</sup> layout and requirements in a [Custom](#)<sup>[69]</sup> or [User Interface](#)<sup>[73]</sup> diagram.

There are a number of UI Control elements available in the **User Interface** page of the Enterprise Architect UML **Toolbox** (see *Using Enterprise Architect - UML Modeling Tool*). These include:

- List
- Table
- Text Box
- Label
- Form
- Panel
- Button
- Combobox
- Checkbox
- Checkbox (left hand side)
- Radio button
- Radio button (left hand side)
- Vertical Line
- Horizontal Line.

The icons can be combined on a Screen icon to represent the appearance of a user interface screen, as shown:



You can also extend the available icons by selecting other stereotypes in the UI Control Element **Properties** dialog. The full set of available stereotypes is shown below; type or select the text in the **Stereotype** field to create the corresponding icon.





).

### Set a Web Icon

To set a web icon, follow the steps below:

1. Create a new Class element in a diagram.
2. Display the Class **Properties** dialog.
3. In the **Stereotype** field, either type in the required stereotype name or click on the drop-down arrow and select the required stereotype (as named above).
4. Click on the **OK** button. The Class displays as in one of the examples above.

### 3 UML Connectors



#### What is a Connector?






















































A *connector* is a logical or functional relationship between model elements. There are several different connector types, each having a particular purpose and syntax. Enterprise Architect supports all of the UML connectors as well as some custom ones of its own. Together with the [UML Elements](#)<sup>[78]</sup>, these form the basis of UML models.

For more information on using these connectors, consult the appropriate topic by clicking on the required connector icon in the table below.

#### Notes:

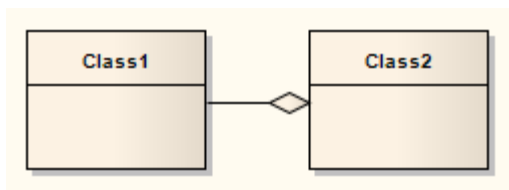
- *Invokes* and *Precedes* relationships are defined by the Open Modeling Language (OML). They are stereotyped *Dependency* relationships; *Invokes* indicates that Use Case A, at some point, causes Use Case B to happen, whilst *Precedes* indicates that Use Case C must complete before Use Case D can begin.
- An *Extension* relationship shows that a stereotype extends one or more metaclasses. All stereotypes must extend either one or more Metaclasses, or another stereotype that extends a stereotype (that itself extends a stereotype, and so on).
- A *Tagged Value* relationship defines a reference-type (that is, RefGUID) Tagged Value owned by the source stereotype. The Tagged Value is named for the target role of this association, and is limited to referencing elements with the stereotype by the association target element.
- The *Application* and *Redefinition* relationships are **deprecated**.

Behavioral Diagram Connectors	Structural Diagram Connectors	Inbuilt and Extended Connectors
<b>Activity Diagrams</b> <ul style="list-style-type: none"> <li>Control Flow</li> <li>Object Flow</li> <li>Interrupt Flow</li> </ul>	<b>Composite Structure Diagrams</b> <ul style="list-style-type: none"> <li>Connector</li> <li>Assembly</li> <li>Delegate</li> <li>Role Binding</li> <li>Represents</li> <li>Occurrence</li> </ul>	<b>Analysis Diagrams</b> <ul style="list-style-type: none"> <li>Information Flow</li> <li>Object Flow</li> <li>Associate</li> <li>Realize</li> <li>Representation</li> </ul>
<b>Use Case Diagrams</b> <ul style="list-style-type: none"> <li>Use</li> <li>Associate</li> <li>Generalize</li> </ul>	<b>Package and Class Diagrams</b>	<b>Common Connectors</b> <ul style="list-style-type: none"> <li>Dependency</li> </ul>

Behavioral Diagram Connectors	Structural Diagram Connectors	Inbuilt and Extended Connectors
 Include	 Associate	 Realize
 Extend	 Generalize	 Trace
 Realize	 Compose	 Information Flow
 Invokes	 Aggregate	 Note Link
 Precedes	 Association Class	
	 Assembly	<b>Profile</b>
<b>State Diagrams</b>	 Realize	 Extension
 Transition	 Nesting	 Generalize
 Object Flow	 Package Merge	 Application
	 Package Import	 Tagged Value
<b>Timing Diagrams</b>		 Redefinition
 Message	<b>Component Diagrams</b>	
	 Assembly	<b>Metamodel</b>
<b>Sequence Diagrams</b>	 Delegate	 Generalize
 Message	 Associate	 Associate
 Self-Message	 Realize	 Compose
 Recursion	 Generalize	 Aggregate
 Call		
<b>Communication Diagrams</b>	<b>Deployment Diagrams</b>	<b>Custom</b>
 Associate	 Associate	 Associate
 Realize	 Communication Path	 Aggregate
 Nesting	 Association Class	 Generalize
	 Generalize	 Realize
	 Realize	 Nesting

Behavioral Diagram Connectors	Structural Diagram Connectors	Inbuilt and Extended Connectors
<b>Interaction Overview Diagrams</b>	Deployment	
Control Flow	Manifest	<b>Requirements</b>
Object Flow	Nesting	Aggregate
Interrupt Flow		Inheritance
	<b>User Interface</b>	Associate
<b>Maintenance</b>	Associate	Implements
Aggregate	Aggregate	
	Generalize	<b>WSDL</b>
<b>XML Schema</b>	Realize	No special connectors
Generalize		
Associate	<b>Object</b>	<b>Documentation</b>
	Information Flow	No special connectors
<b>Data Modeling</b>	Associate	
Association	Dependency	

### 3.1 Aggregate

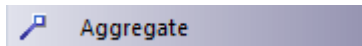


An *Aggregation* connector is a type of association that shows that an element contains or is composed of other elements. It is used in [Class models](#)<sup>[56]</sup>, [Package models](#)<sup>[55]</sup> and [Object models](#)<sup>[58]</sup> to show how more complex elements (aggregates) are built from a collection of simpler elements (component parts; for example, a car from wheels, tires, motor and so on).

A stronger form of aggregation, known as Composite Aggregation, is used to indicate ownership of the whole over its parts. The part can belong to only one Composite Aggregation at a time. If the composite is deleted, all of its parts are deleted with it.

After drawing an Aggregation association, its [form can be changed](#)<sup>[199]</sup>.

## Toolbox Icon



### 3.1.1 Change Aggregation Connector Form

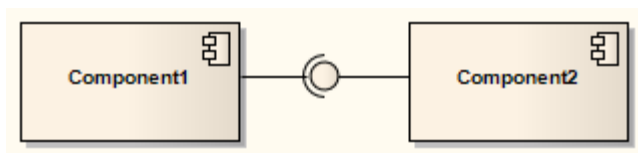
In Enterprise Architect, the default [Aggregation relationship](#)<sup>[198]</sup> is the weak form of the relationship, represented by a hollow diamond. To change the form of an Aggregation connector from weak to strong, follow the steps below.

1. Right-click on an Aggregation connector to display the context menu.
2. Select **Set Aggregation to Composite**. The diamond is shown as filled.

#### Note:

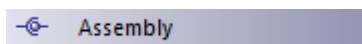
If the connector is already a Strong (Composition) connector, the context menu option changes to **Set Aggregation to Shared**.

## 3.2 Assembly



An *Assembly* connector bridges a component's required [interface](#)<sup>[163]</sup> (Component1) with the provided interface of another component (Component2), typically in a [Component diagram](#)<sup>[65]</sup>.

## Toolbox Icon

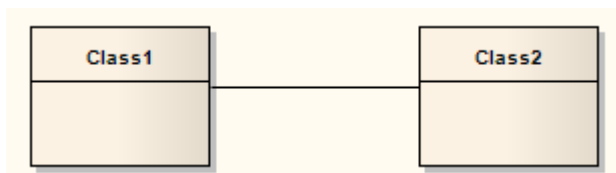


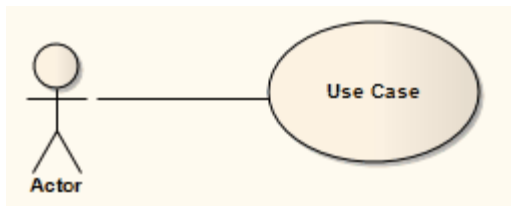
## OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 156*) states:

*An assembly connector is a connector between two components that defines that one component provides the services that another component requires. An assembly connector is a connector that is defined from a required interface or port to a provided interface or port.*

### 3.3 Associate



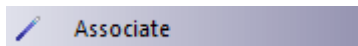


An *Association* implies two model elements have a relationship, usually implemented as an instance variable in one [Class](#)<sup>[152]</sup>. This connector can include named roles at each end, multiplicity, direction and constraints. Association is the general relationship type between elements. To connect more than two elements in an association, you can use the [N-Ary Association](#)<sup>[187]</sup> element.

When code is generated for [Class diagrams](#)<sup>[56]</sup>, Associations become instance variables in the target Class. The relationship is also used in [Package](#)<sup>[55]</sup>, [Object](#)<sup>[58]</sup>, [Communication](#)<sup>[49]</sup>, [Data Modeling](#)<sup>[75]</sup> and [Deployment](#)<sup>[62]</sup> diagrams.

An Associate connector can also be integrated with a Class element to form an [Association Class](#)<sup>[200]</sup>, to allow the Associate connector to have operations and attributes that define certain types of UML relationship.

## Toolbox Icon



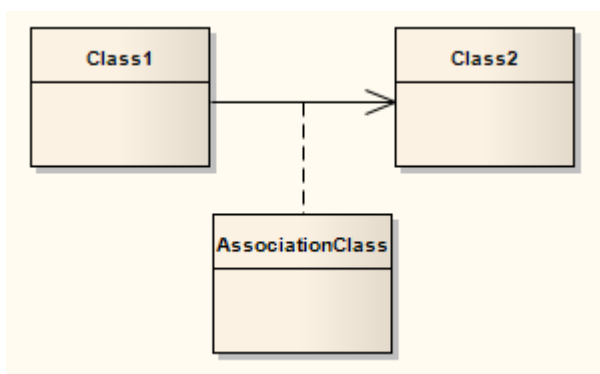
## OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 41*) states:

*An association specifies a semantic relationship that can occur between typed instances. It has at least two ends represented by properties, each of which is connected to the type of the end. More than one end of the association may have the same type.*

*An end property of an association that is owned by an end class or that is a navigable owned end of the association indicates that the association is navigable from the opposite ends; otherwise, the association is not navigable from the opposite ends.*

## 3.4 Association Class



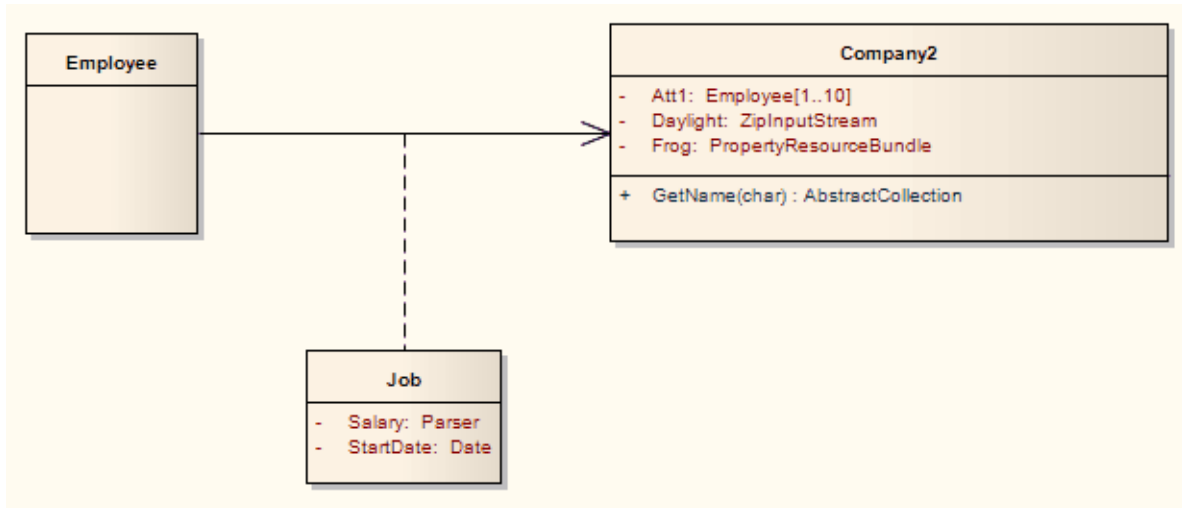
An *Association Class* connector is a UML construct that enables an [Associate](#)<sup>[199]</sup> connector to have *attributes* and *operations* (*features* - see *UML Modeling With Enterprise Architect - UML Modeling Tool*). This results in a hybrid relation with the characteristics of a connection and a [Class](#)<sup>[152]</sup>. It is used to model particular types of connections in UML (see the [OMG UML Specification](#)<sup>[201]</sup> for more details).

When you add an Association Class connection, Enterprise Architect also creates a Class that is automatically connected to the Association. When you hide or delete the Association, the Class is also hidden or deleted.

To add an Association Class to a [Class](#)<sup>[56]</sup> or [Deployment](#)<sup>[62]</sup> diagram, click on the *Association Class* icon in

the Enterprise Architect UML **Toolbox**. Click and hold on the source object in the diagram while you drag the line to the target element, then release the mouse button. Enterprise Architect draws the connector and adds the Class, then prompts you to add the Class name. Note that the names of the Class and the connector are the same. You can also [connect a new Class to an existing Association](#) <sup>[201]</sup>.

The following diagram illustrates an Association Class between model elements. Note the dotted line from the Class to the Association. You cannot move or delete this line.



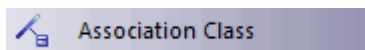
#### Note:

If you are applying a stereotype with a Shape Script to an Association Class (see *SDK for Enterprise Architect*), be aware that the Shape Script is applied to both the Class part and the Association part. Therefore, you might have to include logic in the *shape main* that tests the type of the element so that you can give separate drawing instructions for Class and for Association. Such logic is not necessary in the:

- *shape source* or *shape target*, which are ignored by Classes, or the
- *decoration* shapes, which are ignored by Associate connectors.

If you dis-associate the Class from the Associate connector, both parts keep their Shape Scripts until the stereotypes are removed.

### Toolbox Icon



### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 49*) states:

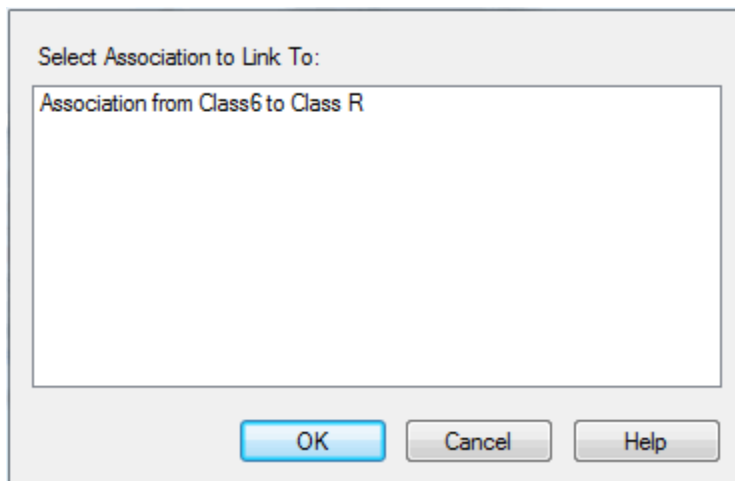
*A model element that has both association and class properties. An AssociationClass can be seen as an association that also has class properties, or as a class that also has association properties. It not only connects a set of classifiers but also defines a set of features that belong to the relationship itself and not to any of the classifiers.*

#### 3.4.1 Connect New Class to Association

To connect a new Class to an existing Association, follow the steps below:

1. Create a Class in the diagram containing the Association to connect.
2. Right-click on the new Class. The context menu displays.
3. Select the **Advanced** | **Association Class** <sup>[200]</sup> menu option. The **Create Association Class** dialog displays.





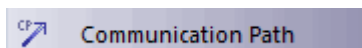
4. Select the connector to connect to.
5. Click on the **OK** button.

### 3.5 Communication Path

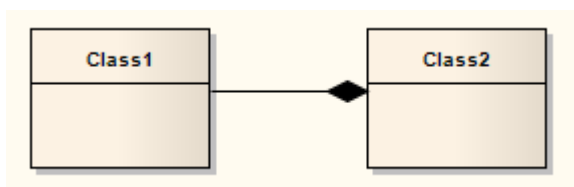


A *Communication Path* defines the path through which two *DeploymentTargets* are able to exchange signals and messages. Communication Path is a specialization of [Association](#)<sup>[199]</sup>. A *DeploymentTarget* is the target for a deployed [Artifact](#)<sup>[157]</sup> and can be a [Node](#)<sup>[165]</sup>, Property or [InstanceSpecification](#)<sup>[160]</sup> in a [Deployment diagram](#)<sup>[62]</sup>.

#### Toolbox Icon

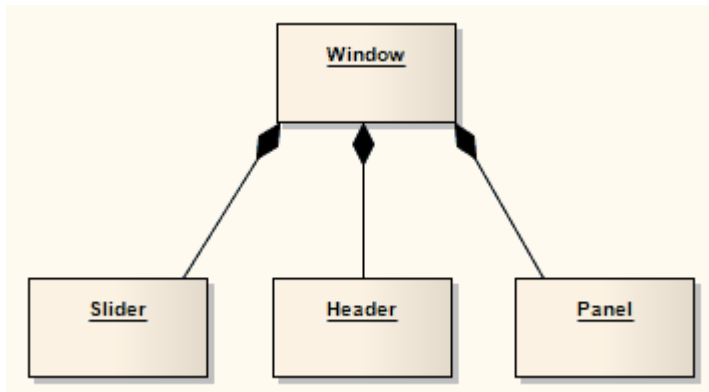


### 3.6 Compose

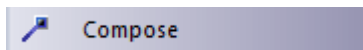


A *Composite Aggregation*<sup>[198]</sup> is used to depict an element that is made up of smaller components, typically in a [Class](#)<sup>[56]</sup> or [Package](#)<sup>[55]</sup> diagram. A component - or part instance - can be included in a maximum of one composition at a time. If a composition is deleted, usually all of its parts are deleted with it; however, a part can be individually removed from a composition without having to delete the entire composition. Compositions are transitive, asymmetric relationships and can be recursive.

See the example below.



### Toolbox Icon

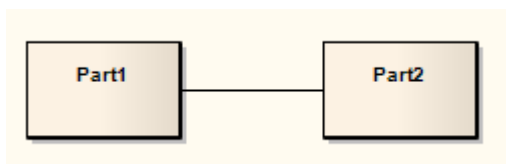


### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 43*) states:

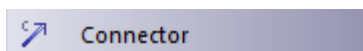
*Composite aggregation is a strong form of aggregation that requires a part instance be included in at most one composite at a time. If a composite is deleted, all of its parts are normally deleted with it.*

## 3.7 Connector



Connectors illustrate communication links between parts to fulfill the structure's purpose, typically in a [Composite Structure](#) diagram. Each Connector end is distinct, controlling the communication pertaining to its connecting element. These elements can define constraints specifying this behavior. Connectors can have multiplicity.

### Toolbox Icon

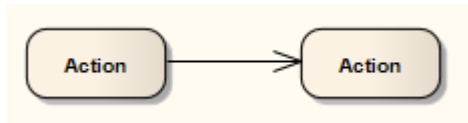


### OMG UML Specification

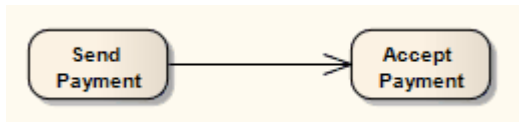
The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 177*) states:

*Specifies a link that enables communication between two or more instances. This link may be an instance of an association, or it may represent the possibility of the instances being able to communicate because their identities are known by virtue of being passed in as parameters, held in variables or slots, or because the communicating instances are the same instance. The link may be realized by something as simple as a pointer or by something as complex as a network connection. In contrast to associations, which specify links between any instance of the associated classifiers, connectors specify links between instances playing the connected parts only.*

### 3.8 Control Flow



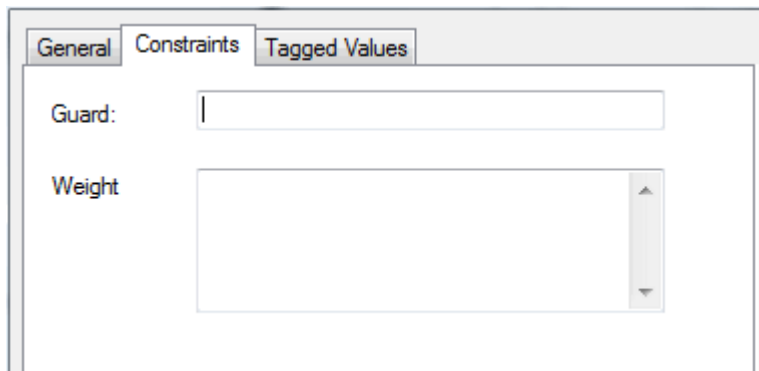
The *Control Flow* is a connector connecting two nodes in an [Activity diagram](#)<sup>[54]</sup>. Control Flow connectors bridge the flow between Activity nodes, by directing the flow to the target node once the source node's activity is completed.



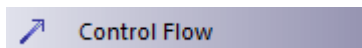
Control Flows and *Object Flows* can define a *Guard* and a *Weight* condition.

A *Guard* defines a condition that must be true before control passes along that activity edge. A practical example of this is where two or more activity edges (Control Flows) exit from a [Decision](#)<sup>[102]</sup> element. Each flow should have a Guard condition that is exclusive of the other and defines which edge is taken under what conditions. The Control Flow **Properties** dialog enables you to set up Guard conditions on Control Flows and on Object Flows.

A *Weight* defines the number of tokens that can flow along a Control or Object Flow connection when that edge is traversed. Weight can also be defined on the Control Flow and Object Flow **Properties** dialogs.



#### Toolbox Icon

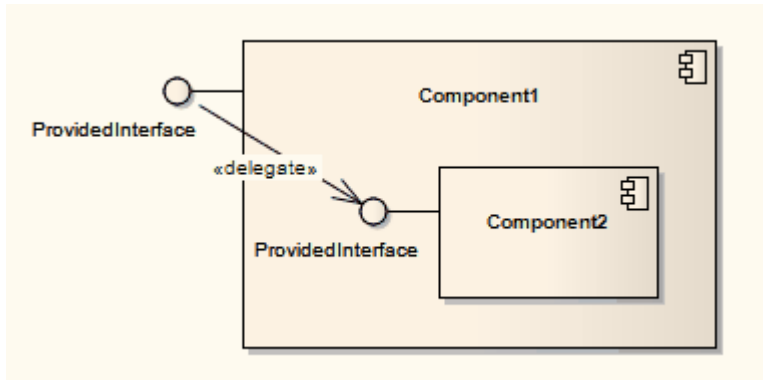


#### OMG UML specification:

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 356*) states:

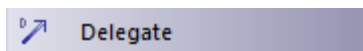
*A control flow is an edge that starts an activity node after the previous one is finished.*

### 3.9 Delegate



A *Delegate* connector defines the internal assembly of a component's external [Ports](#)<sup>[163]</sup> and [Interfaces](#)<sup>[164]</sup>, on a [Component diagram](#)<sup>[65]</sup>. Using a Delegate connector wires the internal workings of the system to the outside world, by a delegation of the external interfaces' connections.

#### Toolbox Icon

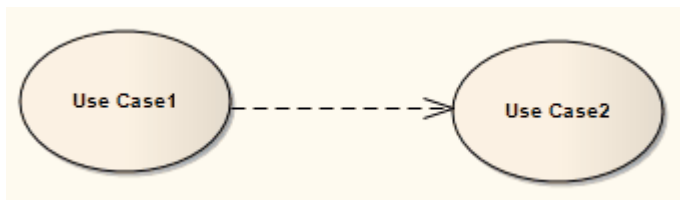
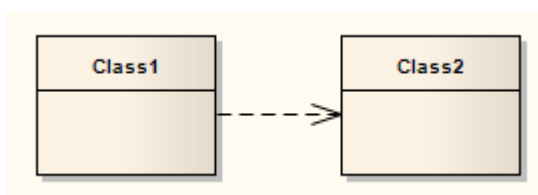


#### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 156*) states:

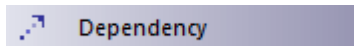
*A delegation connector is a connector that links the external contract of a component (as specified by its ports) to the internal realization of that behavior by the component's parts. It represents the forwarding of signals (operation requests and events): a signal that arrives at a port that has a delegation connector to a part or to another port will be passed on to that target for handling.*

### 3.10 Dependency



*Dependency relationships* are used to model a wide range of dependent relationships between model elements in [Use Case](#)<sup>[7]</sup>, [Activity](#)<sup>[90]</sup> and [Structural](#)<sup>[54]</sup> diagrams, and even between models themselves. You can create the Dependency from the [Common](#) page of the Enterprise Architect UML [Toolbox](#). The Dependencies package as defined in UML 2.1 has many derivatives, such as [Realization](#)<sup>[233]</sup>, [Deployment](#)<sup>[206]</sup> and [Use](#)<sup>[238]</sup>. Once you create a Dependency you can further refine its meaning by [applying a specialized stereotype](#)<sup>[206]</sup>.

## Toolbox Icon



## OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 64*) states:

*A dependency is a relationship that signifies that a single or a set of model elements requires other model elements for their specification or implementation. This means that the complete semantics of the depending elements is either semantically or structurally dependent on the definition of the supplier element(s).*

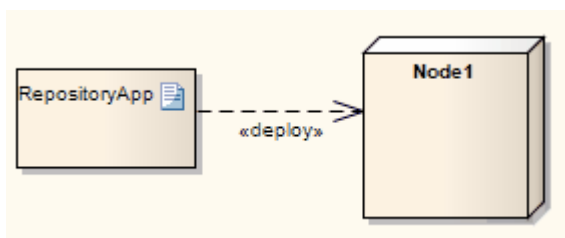
### 3.10.1 Apply a Stereotype

To apply a stereotype to a [Dependency](#)<sup>[205]</sup> relationship, follow the steps below:

1. Select the Dependency relationship to change.
2. Right-click on the connector and, from the context menu, select the **Dependency Properties** option. The **Dependency Properties** dialog displays.
3. In the **Stereotype** field, either type in the required stereotype name or click on the drop-down arrow and select the stereotype from the list.
4. Click on the **OK** button.

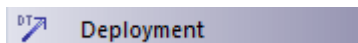
Alternatively, you can right-click on the Dependency relationship and select the **Advanced | Dependency Stereotypes** context menu option, then select from a shorter list of standard stereotypes.

## 3.11 Deployment

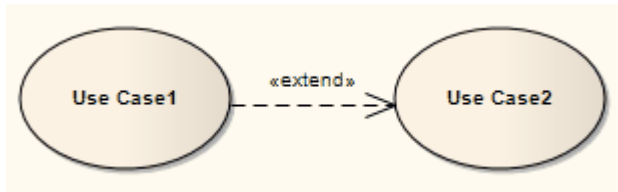


A *Deployment* is a type of [Dependency](#)<sup>[205]</sup> relationship that indicates the deployment of an artifact onto a node or executable target, typically in a [Deployment diagram](#)<sup>[62]</sup>. A Deployment can be made at type and instance levels. At the type level, a Deployment would be made for every instance of the node. Deployment can also be specified for an instance of a node, so that a node's instances can have varied deployed artifacts. With composite structures modeled with nodes defined as [Parts](#)<sup>[168]</sup>, Parts can also serve as targets of a Deployment relationship.

## Toolbox Icon



### 3.12 Extend



An *Extend* connection is used to indicate that an element extends the behavior of another. Extensions are used in [Use Case models](#) <sup>[71]</sup> to indicate that one [Use Case](#) <sup>[146]</sup> (optionally) extends the behavior of another. An extending Use Case often expresses alternative flows.

#### Toolbox Icon



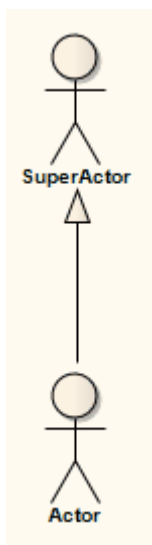
#### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 587*) states:

*This relationship specifies that the behavior of a Use Case may be extended by the behavior of another (usually supplementary) Use Case. The extension takes place at one or more specific extension points defined in the extended Use Case. Note, however, that the extended Use Case is defined independently of the extending Use Case and is meaningful independently of the extending Use Case. On the other hand, the extending Use Case typically defines behavior that may not necessarily be meaningful by itself. Instead, the extending Use Case defines a set of modular behavior increments that augment an execution of the extended Use Case under specific conditions.*

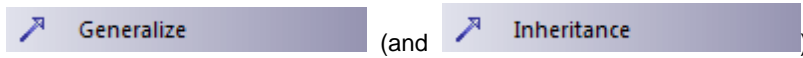
*Note that the same extending Use Case can extend more than one Use Case. Furthermore, an extending Use Case may itself be extended.*

### 3.13 Generalize



A *Generalization* is used to indicate inheritance. Drawn from the specific classifier to a general classifier, the generalize implication is that the source inherits the target's characteristics. It is used typically in [Class](#) <sup>[56]</sup>, [Component](#) <sup>[65]</sup>, [Object](#) <sup>[58]</sup>, [Package](#) <sup>[55]</sup>, [Use Case](#) <sup>[71]</sup> and [Requirements](#) <sup>[71]</sup> diagrams.

## Toolbox Icon

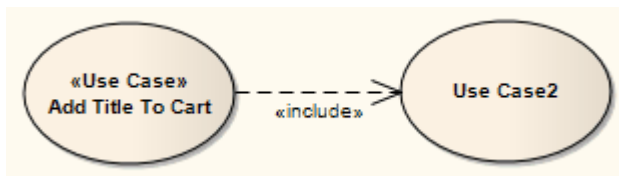


## OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 73*) states:

*A generalization is a taxonomic relationship between a more general classifier and a more specific classifier. Each instance of the specific classifier is also an indirect instance of the general classifier. Thus, the specific classifier inherits the features of the more general classifier.*

### 3.14 Include



An *Include* connection indicates that the source element includes the functionality of the target element. Include connections are used in [Use Case models](#)<sup>[7]</sup> to reflect that one [Use Case](#)<sup>[146]</sup> includes the behavior of another. Use an Include relationship to avoid having the same subset of behavior in many Use Cases; this is similar to [delegation](#)<sup>[205]</sup> used in [Class models](#)<sup>[56]</sup>.

## Toolbox Icon

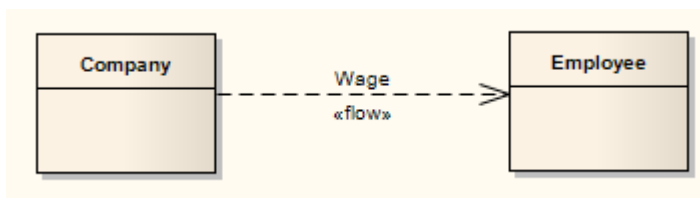


## OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 591*) states:

*Include is a DirectedRelationship between two Use Cases, implying that the behavior of the included Use Case is inserted into the behavior of the including Use Case. It is also a kind of NamedElement so that it can have a name in the context of its owning Use Case. The including Use Case may only depend on the result (value) of the included Use Case. This value is obtained as a result of the execution of the included Use Case.*

### 3.15 Information Flow



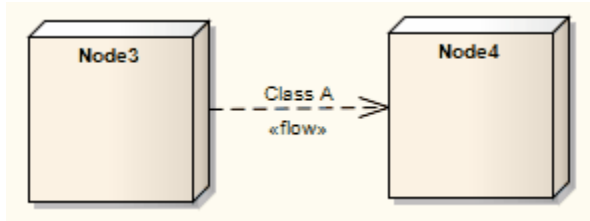
An *Information Flow* represents [information items](#)<sup>[163]</sup> or classifiers flowing between two elements in any diagram. The connector is available from the **Common** toolbox page and from every Quick Link menu.

You can have more than one Information Flow connector between the same two elements, identifying which items flow between the two under differing conditions.

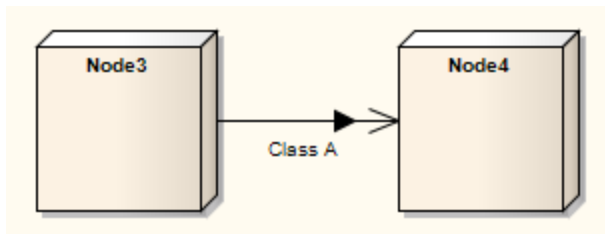
## Example of Use

1. Open a diagram and add two elements (for example, Nodes on a Deployment diagram).

- Click on the *Information Flow* connector in the **Common** toolbox page and drag between the two elements. The [Information Items Conveyed](#) [210] dialog displays.
- Add the classifier or information item element(s) to the Information Flow. The diagram now resembles the following.



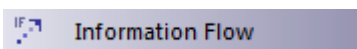
- Add another connector between the same two elements (for example, a *Communication Path* connector).
- Right-click the connector and select the **Advanced | Information Flows Realized** context menu option. The [Information Flows Realized](#) [210] dialog displays.
- Tick the checkbox against the required classifier element and click on the **OK** button. The combined connector now resembles the following:



#### Notes:

- Once the connectors are combined, you cannot access the [Information Items Conveyed](#) dialog directly. You add or delete information items on the connector using the [Information Items Realized](#) [210] dialog. If you have more than one Information Flow connector between the elements, they form part of the same combined connector; you can again work on them separately through the [Information Items Realized](#) dialog.
- If you have information flows in a diagram that you use as the source for a Pattern, the Information Items Conveyed and Information Flows Realized data is not copied into the Pattern.

#### Toolbox Icon



#### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 606*) states:

*An InformationFlow specifies that one or more information items circulates from its sources to its targets.*

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 607*) also states:

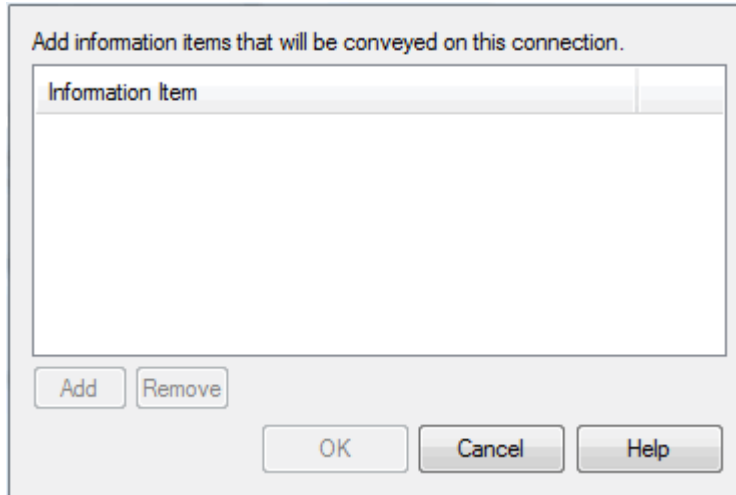
*An information flow is an abstraction of the communication of an information item from its sources to its targets. It is used to abstract the communication of information between entities of a system. Sources or targets of an information flow designate sets of objects that can send or receive the conveyed information item.*



### 3.15.1 Convey Information on a Flow

As you create an [Information Flow](#) connector between two elements, you can specify which [Information Items](#) or classifiers are conveyed on this flow.

To specify these Information Items or classifiers, right-click on the connection and select the **Advanced | Information Item Conveyed** context menu option. The **Information Items Conveyed** dialog displays.

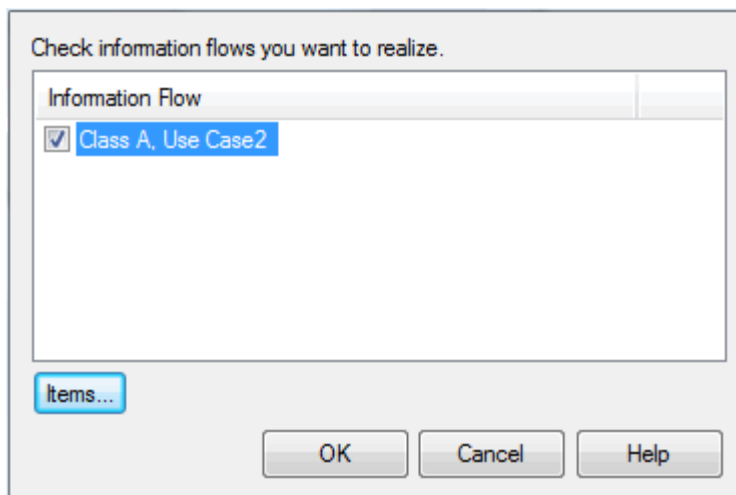


Button	Use to
<b>Add</b>	Display the <b>Select &lt;Item&gt;</b> dialog, from which you select the required Information or Classifier element(s) (see <i>UML Modeling With Enterprise Architect - UML Modeling Tool</i> ).
<b>Remove</b>	Remove the selected item.

**Note:**

If you select more than one element, they are listed in one entry for the Information Flow connector.

### 3.15.2 Realize an Information Flow

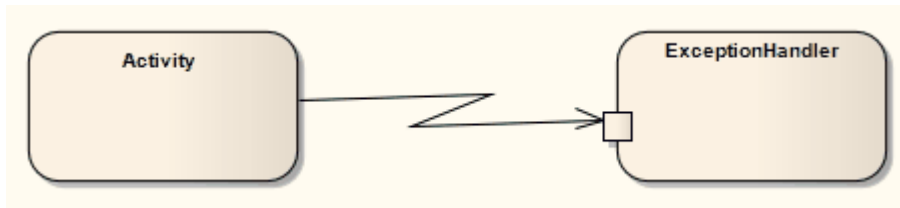


The **Information Flows Realized** dialog displays all flows that can be realized on the selected connector. To

realize an [Information Flow](#) [208] on this connector, select the corresponding checkbox and click on the **OK** button.

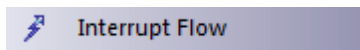
If you want to change the information items conveyed on an information flow, click on the flow *text* and click on the **Items** button. The [Information Items Conveyed](#) [210] dialog displays, and you can add or remove items as required. When you click on the **OK** button, the [Information Items Realized](#) dialog redisplay and you can realize the selected flow or flows as above.

### 3.16 Interrupt Flow



The *Interrupt Flow* is a connection used to define the two UML concepts of connectors for [Exception Handler](#) [107] and [Interruptible Activity Region](#) [127]. An Interrupt Flow is also known as an *activity edge*. It is typically used in an [Activity diagram](#) [57].

#### Toolbox Icon

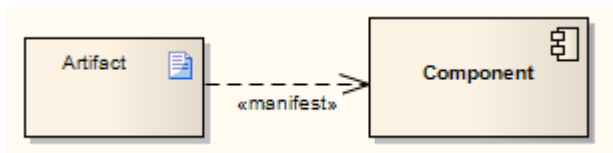


#### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 327*) states:

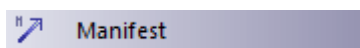
*An activity edge is an abstract class for directed connections between two activity nodes.*

### 3.17 Manifest



A *Manifest* relationship indicates that the [Artifact](#) [157] source embodies the target model element, typically in [Component](#) [65] and [Deployment](#) [62] diagrams. Stereotypes can be added to Enterprise Architect to classify the type of manifestation of the model element (see *Extending UML With Enterprise Architect*).

#### Toolbox Icon



#### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 212*) states:

*An artifact embodies or manifests a number of model elements. The artifact owns the manifestations, each representing the utilization of a packageable element.*

*Specific profiles are expected to stereotype the manifestation relationship to indicate particular forms of manifestation, e.g. «tool generated» and «custom code» might be two manifestations for different classes*

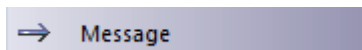
embodied in an artifact.

### 3.18 Message

Messages indicate a flow of information or transition of control between elements. Messages can be used by [Timing Diagrams](#) [226], [Sequence Diagrams](#) [212] and [Communication Diagrams](#) [212] (but not [Interaction Overview](#) [52] diagrams) to reflect system behavior. If between [Classes](#) [152] or classifier instances, the associated list of operations is available to specify the event.

Moving a Message can disrupt the organization of other features on the diagram. To avoid this, and move *only* the Message, press **[Alt]** while you move the Message.

#### Toolbox Icon



#### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 491*) states:

*A Message defines a particular communication between Lifelines of an Interaction.*

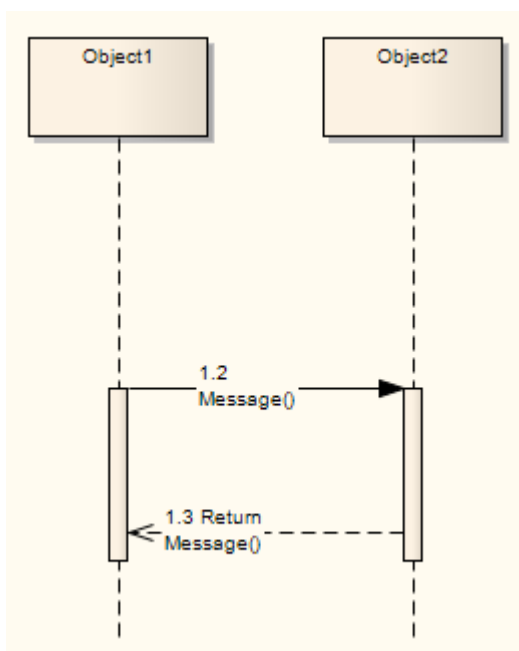
*A Message is a NamedElement that defines one specific kind of communication in an Interaction. A communication can be, for example, raising a signal, invoking an Operation, creating or destroying an Instance. The Message specifies not only the kind of communication given by the dispatching ExecutionSpecification, but also the sender and the receiver.*

*A Message associates normally two OccurrenceSpecifications - one sending OccurrenceSpecification and one receiving OccurrenceSpecification.*

#### Note:

Communication diagrams were known as Collaboration diagrams in UML 1.4.

#### 3.18.1 Message (Sequence Diagram)



[Sequence diagrams](#) <sup>[39]</sup> depict work flow or activity over time using Messages passed from element to element. These Messages correspond in the software model to Class operations and behavior. They are semantically similar to the Messages passed between elements in a Communication diagram, and can be of many different [types](#) <sup>[217]</sup>.

To create a Message on a Sequence diagram, follow the steps below:

1. Access the Sequence diagram. The **Interaction** pages of the Enterprise Architect UML **Toolbox** display.
2. In the **Interaction Relationships** page, click on the **Message** icon, click on the source object and drag the cursor to the destination (target) object. The **Message Properties** dialog displays (if not, right-click on the Message and select the **Message Properties** context menu option).

The image shows the 'Message Properties' dialog box in Enterprise Architect. It is organized into four main sections:

- Signature:** Contains fields for 'Message' (set to 'Message'), 'Parameters', 'Argument(s)', 'Return Value' (set to 'void'), 'Assign To', 'Stereotype', and 'Alias'. A 'Show Inherited Methods' checkbox is checked. An 'Operations' button is located to the right of the Message field.
- Sequence Expression:** Contains fields for 'Condition' and 'Constraint', and an 'Is Iteration' checkbox.
- Control Flow Type:** Contains dropdowns for 'Synchron' (set to 'Synchronous'), 'Lifecycle', and 'Kind' (set to 'Call'). An 'Is Return' checkbox is also present.
- Notes:** A rich text editor with a toolbar (including Bold, Italic, Underline, Font Color, Bulleted List, Numbered List, Indent, Decrease Indent, Text Color, Text Background Color, and Undo) and a text area.

At the bottom of the dialog are three buttons: 'OK', 'Cancel', and 'Help'.

3. In the **Message** field, type the Message name.

**Notes:**

- If the Message flow is *towards* a [Class](#) <sup>[152]</sup> element (dropped in from a Class diagram) or a [Lifeline](#) <sup>[123]</sup> element having a classifier, and the *destination* Class has defined *operations*, you can click on the drop-down arrow and select an appropriate operation name. The Message then reflects the destination Class operations.
  - If the available operations are not appropriate, you can click on the **Operations** button and define a new operation in the target element, using the **Operations** dialog.
  - If you create a Message without making reference to the target Class operations, no new operation is added to the target Class.
4. In the **Parameters** field, type any parameters that the Message has, as a comma-separated list. If required, in the **Parameter Values** field type the actual value for each parameter, again as a comma-separated list.
  5. If the Message is a return message, in the **Return Value** field enter the returned value or type.

**Note:**

It is possible to depict returns from a [Self Message](#) <sup>[215]</sup>. Simply create a second Self Message at the end of execution and select the **Is Return** checkbox in the **Control Flow Type** panel.

6. If the Message flow is *from* a Class element or Lifeline element with classifier that has defined *attributes*, click on the drop-down arrow in the **Assign to** field and select an appropriate attribute name. The Message reflects the attributes from the *source* Class. You cannot add further attributes to the source Class here - if no appropriate attribute is listed, open the element **Properties** dialog and add the required attribute.

Otherwise, if required, type the name of the object to assign the message flow to.

7. In the **Stereotype** field, type or select an optional stereotype for the connector (this is displayed on the diagram, if entered).
8. If required, in the **Alias** field type an alias for the name of the Message.

**Note:**

On the diagram, the alias displays if the **Use Alias if Available** checkbox is selected on the **Diagram** tab of the **Diagram Properties** dialog. The Alias displays instead of or as well as the Message name, depending on the setting selected in the **Alias Usage** panel of the **Diagram Behavior** page of the **Options** dialog.

9. In the **Condition** field, type any conditions that must be true in order for the message to be sent.
10. In the **Synch:** field in the **Control Flow Type** panel, select **Synchronous** or [Asynchronous](#) <sup>[221]</sup> as appropriate.
11. In the **Lifecycle** field, select **New** to create a new element at the end of the Message, or **Delete** to terminate the message flow at the end of the Message. If neither case applies, leave the field at the default of **<none>**.
12. If required, in the **Notes** field type any explanatory notes. You can format the notes using the **Rich Text Notes** toolbar at the top of the field.
13. Click on the **OK** button to save the Message definition.

**Notes:**

- You can [change the timing details](#) <sup>[218]</sup> of a message on the **Timing Details** dialog, and emphasize the sequence of closely-ordered messages using [General Ordering](#) <sup>[220]</sup>.
- To toggle the numbering of messages on a Sequence diagram, select or deselect the **Show Sequence Numbering** checkbox on the **Options** dialog.

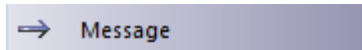
## Co-Region Notation

*Co-Region notation* can be used as a short hand for parallel combined fragments. To access the **Co-Region** submenu, right-click on a connector in a Sequence diagram and select the **Co-Region** context menu option. There are four sub-options available:

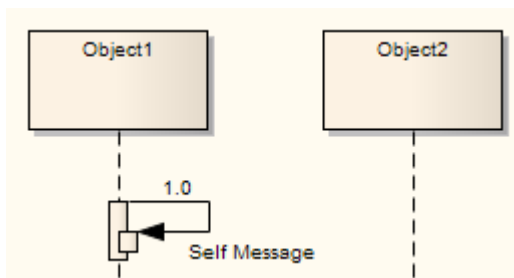
- **Start at head**

- End at head
- Start at tail
- End at tail.

### Toolbox Icon



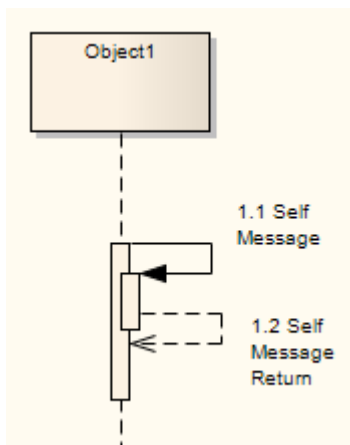
#### 3.18.1.1 Self-Message



A *Self-Message* reflects a new process or method invoked within the calling lifeline's operation. It is a specification of a [Message](#) <sup>[212]</sup>, typically in a [Sequence diagram](#) <sup>[39]</sup>.

### Self-Message as Return

It is possible to depict a return from a Self Message call.

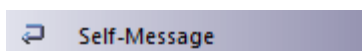


To create a Self Message return:

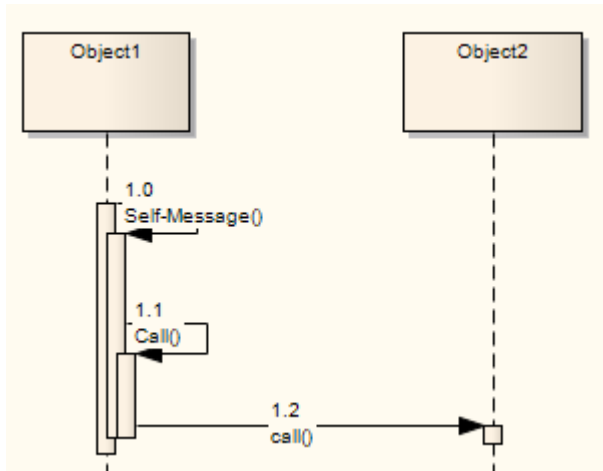
1. Create a second Self Message at the end of execution.
2. Double-click on the Message name to open the **Message Properties** dialog.
3. Select the **Is Return** checkbox.
4. [Raise the Activation level](#) <sup>[46]</sup> of the return.

Self-Message [Calls](#) <sup>[216]</sup> indicate a nested invocation; new activation levels are added with each Call.

### Toolbox Icon



### 3.18.1.2 Call



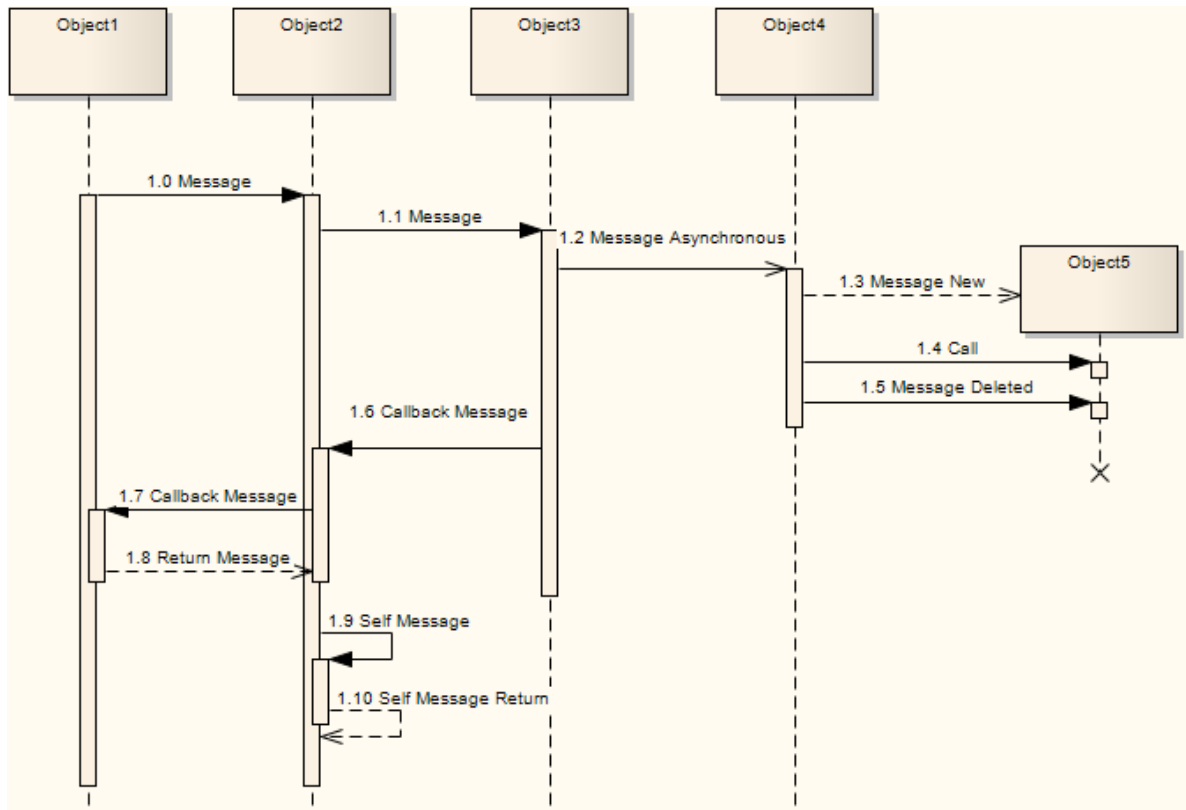
A *Call* is a type of [Message](#) <sup>[212]</sup> connector that extends the level of activation from the previous Message. All [Self-Messages](#) <sup>[215]</sup> create a new activation level, but this focus of control usually ends with the next Message (unless [activation levels](#) <sup>[45]</sup> are manually adjusted). Self-Message Calls, as depicted above by the first Call, indicate a nested invocation; new activation levels are added with each Call. Unlike a regular Message between elements, a Call between elements continues the existing activation in the source element, implying that the Call was initiated within the previous Message's activation scope.

#### Toolbox Icon



### 3.18.1.3 Message Examples

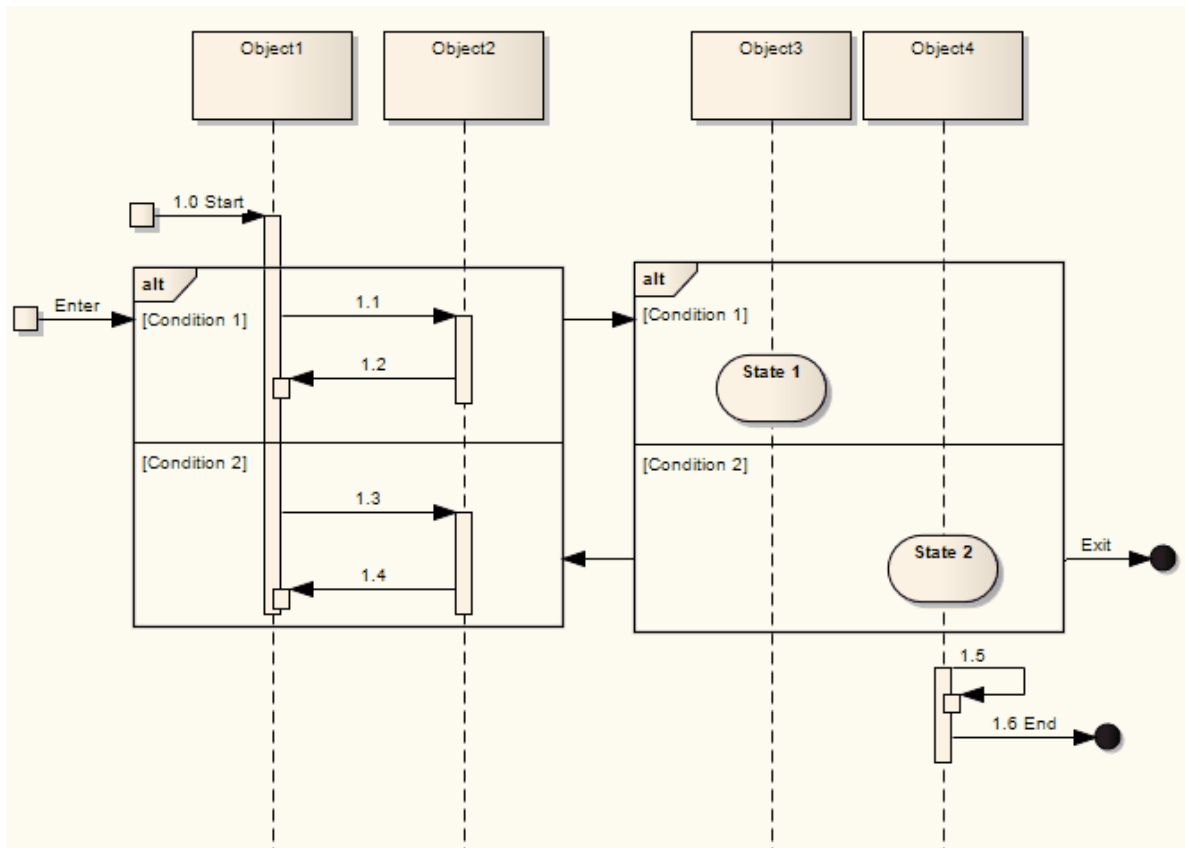
The following are different types of [Messages](#)<sup>[212]</sup> available on [Sequence Diagrams](#)<sup>[39]</sup>. Note that Messages on Sequence diagrams can also be modified with Shape Scripts (see *SDK for Enterprise Architect*).



### Other Sequence Messages

The following are examples of Messages that are not part of the sequence described by the diagram.





**3.18.1.4 Change the Timing Details**

It is possible to change the timing details of a Message in a [Sequence diagram](#) <sup>[39]</sup> by right-clicking on the [Message](#) <sup>[21]</sup> connector and selecting the **Timing Details** context menu option. The **Timing Details** dialog displays.

Duration Constraint:

Duration Constraint Between Messages:

Duration Observation:

Timing Constraint:

Timing Observation:

Complete the fields on this dialog as follows:

Option	Use to
<b>Duration Constraint</b>	Indicate the minimum and maximum limits on how long a message can last.

Option	Use to
<b>Duration Constraint Between Messages</b>	Indicate the minimum and maximum interval between sending or receipt of the previous message at the current message's source Lifeline, and sending the current message.
<b>Duration Observation</b>	Capture the duration of a message.
<b>Timing Constraint</b>	Indicate the minimum and maximum time at which the message should arrive at the target.
<b>Timing Observation</b>	Capture the point at which the message was sent.

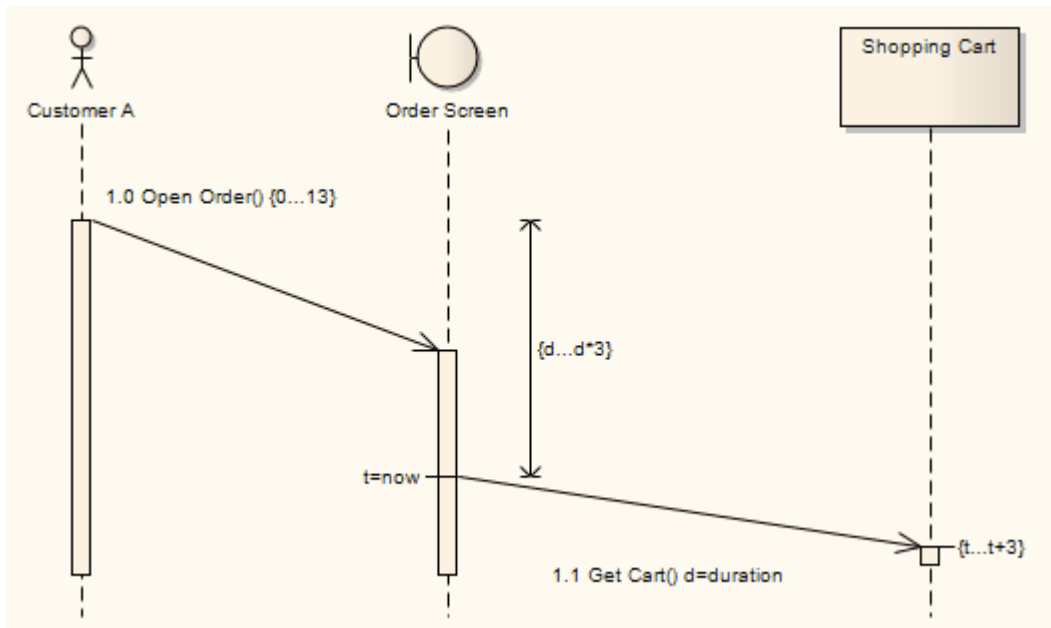
See the OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 511*).

In the diagram below, on the *Open Order* Message:

- **Duration Constraint** has been set to **0...13**.

On the *Get Cart* Message:

- **Duration Constraint Between Messages** has been set to **d...d\*3**
- **Duration Observation** has been set to **d=duration**
- **Timing Constraint** has been set to **t...t+3**
- **Timing Observation** has been set to **t=now**.



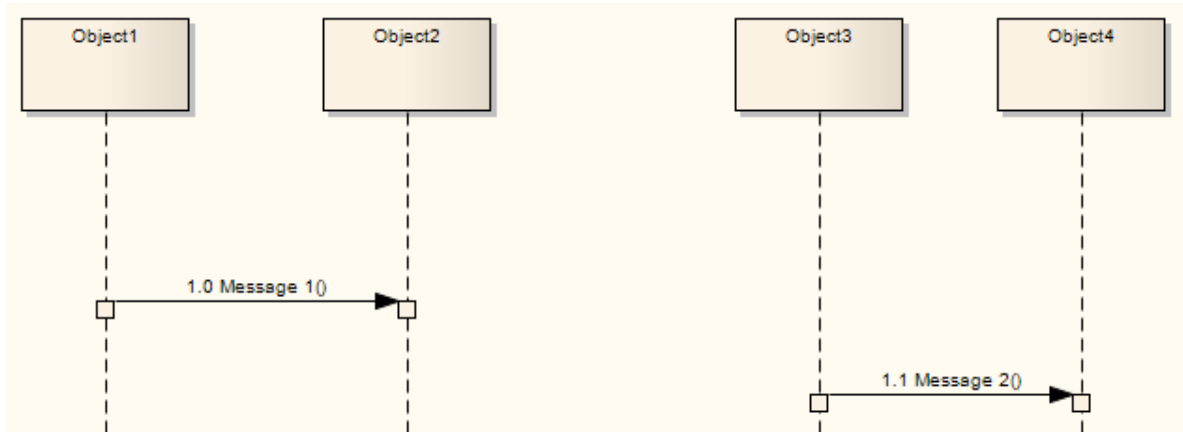
By typing a value in the **Duration Constraint** field, you enable the Message angle to be adjusted. After clicking on the **OK** button on the **Timing Details** dialog, click on the head of the Message connector and drag the connector up or down to change the angle. You cannot extend the angle beyond the life line of the connecting sequence object or create an angle of less than 5 degrees.

You can also create the **Duration Constraint Between Messages** line by dragging the [General Ordering](#) <sup>[220]</sup> arrow up to the point at which the previous message joins the source Lifeline for the current message. A dialog displays on which you enter the value for the constraint. Having created the line, you can move it to any point within half way along the current message and half way along the previous message, to avoid overlap with other message timing details. You can edit or delete the value either through the **Timing Details** dialog or by right-clicking on the line itself and selecting the appropriate context menu option.

### 3.18.1.5 General Ordering

In a [Sequence diagram](#)<sup>[39]</sup>, the workflow is represented by the sequence of Messages down the diagram. Messages near the top of the diagram are passed before Messages lower down the diagram.

Consider the following diagram.

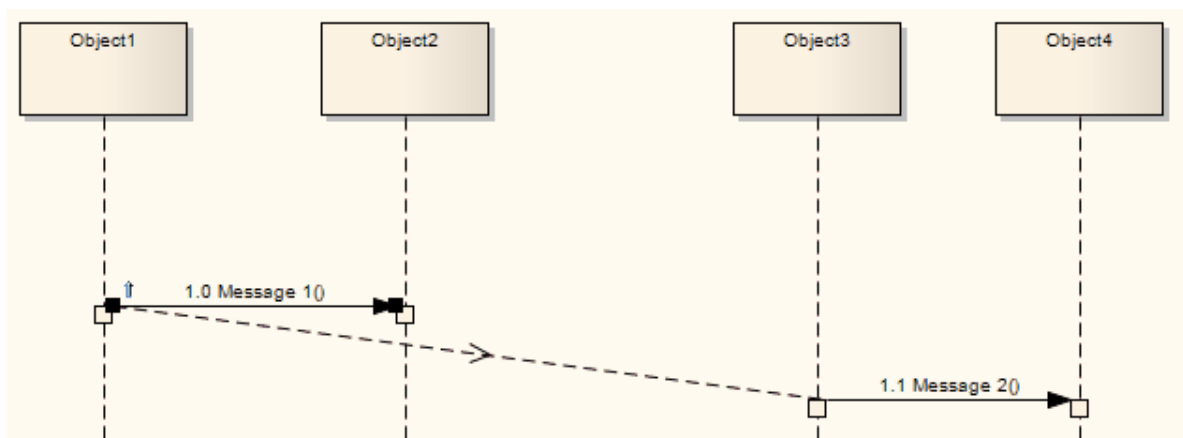


Message 1 is earlier than Message 2. However, in a complex diagram, or when representing finely timed operations or parallel processing, this might not be apparent. You can reinforce the sequence using a *General Ordering* arrow.

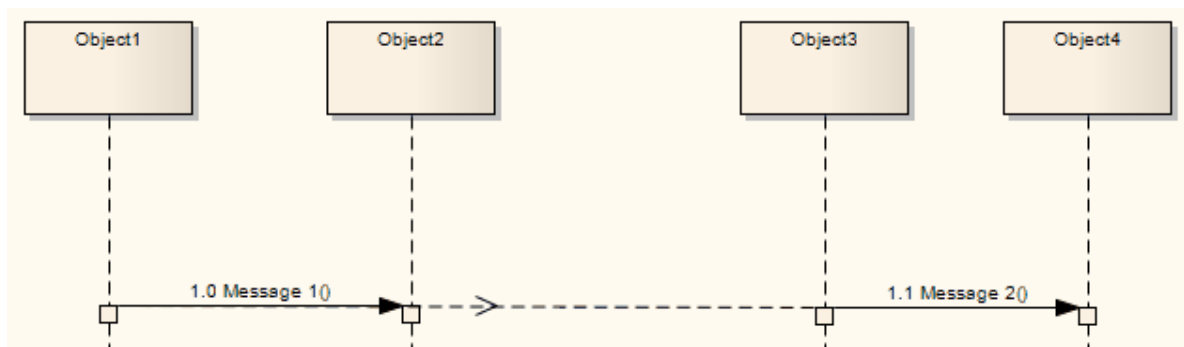
Click on the Message arrow. A small arrow displays at the source anchor point.



Click on this arrow and drag it to the start of the next Message in sequence (Message 2 in the example). The General Ordering arrow displays, indicating that the second Message follows the first.



The General Ordering arrow is exaggerated in the above figure. You would normally have the arrow running almost horizontal across the diagram.



You can have more than one General Ordering arrow issuing from or targeting a Message, if necessary.

### 3.18.1.6 Asynchronous Signal Message

You define a Message as an asynchronous signal message by displaying the [Message Properties](#) <sup>[212]</sup> dialog and setting the **Synch** field to **Asynchronous** and the **Kind** field to **Signal**. (A *synchronous* message cannot be used to convey signals, so setting the **Synch** field to **Synchronous** disables the **Kind** field.)

#### Note:

**Return Value**, **Assign To** and the **Operations** button, which are not applicable to asynchronous *signals*, are disabled.

The **Operations** button changes to a **Signal** button, which you click on to associate the asynchronous signal message with a Signal element in the model. You can type the arguments corresponding to the Signal attributes into the **Argument(s)** field.

**Signature**

Message:

Attributes:

Argument(s):

Return Value:   Show Inherited Methods

Assign To:

Stereotype:

Alias:

**Sequence Expression**

Condition:

Constraint:

Is Iteration

**Control Flow Type:**

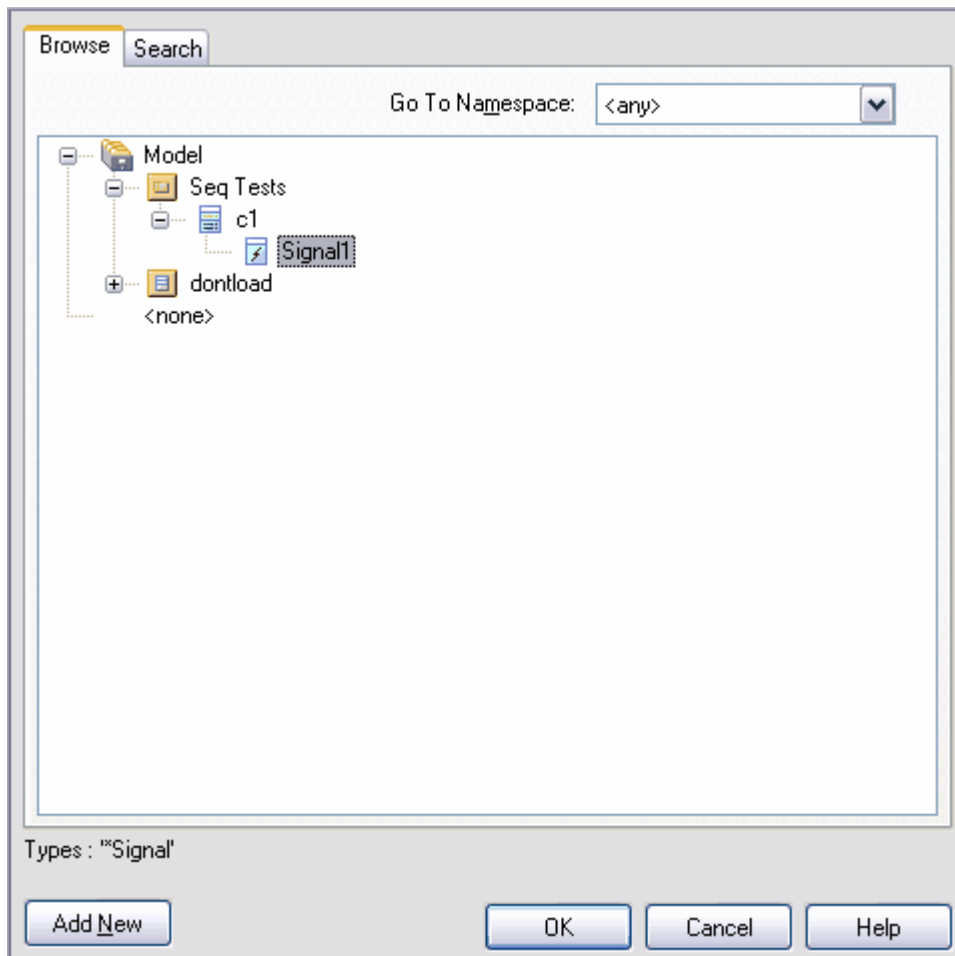
Synch:  Lifecycle:

Kind:   Is Return

**Notes:**

**B I U A** | |  $x^2$   $x_2$

When you click on the **Signal** button, the **Select \*Signal** dialog displays, through which you locate and select the required Signal element. (The **Select \*Signal** dialog is a variation of the **Select <Item>** dialog.)



### 3.18.2 Message (Communication Diagram)

A Message in a [Communication diagram](#)<sup>[49]</sup> is equivalent in meaning to a Message in a Sequence diagram. It implies that one object uses the services of another object, or sends a message to that object. Communication Messages in Enterprise Architect are always associated with an [Association](#)<sup>[199]</sup> connector between object instances. Always create the Association connector first, then [add a Message to the connector](#)<sup>[224]</sup>.

Messages can be dragged into a suitable position by clicking and dragging on the message text.

Communication Messages are ordered to reflect the sequencing of the diagram. The numbering scheme should reflect the nesting of each event. A sequencing scheme could be:

- 1
- 2, 2.1, 2.2, 2.3
- 3.

This would indicate the single sequence of events 2.1, 2.2 and 2.3 occurs within an operation initiated by event 2. This is the default pattern applied by Enterprise architect

Alternatively, the sequence could be:

- 1
- 2, 2.1, 2.1.1, 2.1.1.1  
2.2, 2.2.1, 2.2.1.1
- 3

This would indicate that two sequences of events can be initiated by event 2, and 2.1 and 2.2 are separate sequences, not consecutive events in one sequence. You can set the sequence [pattern and order](#)<sup>[224]</sup> using

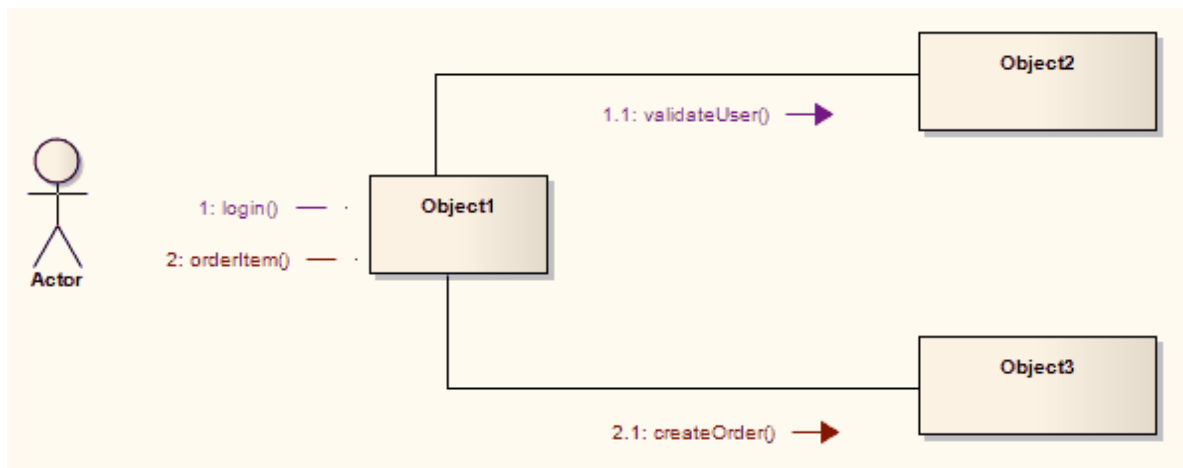
the [Message Properties](#) dialog and the [Sequence Communications](#) dialog.

If the target object is a Class or has its instance classifier set, the drop-down list of possible message names includes the exposed operations for the base type.

### 3.18.2.1 Create a Communication Message

To create a [Communication Message](#)<sup>[223]</sup>, follow the steps below:

1. Open a diagram (one of: Communication, Analysis, Interaction Overview, Object, Activity or State Machine).
2. Add the required objects.
3. Add an [Association](#)<sup>[199]</sup> relationship between each pair of objects that communicate.
4. Right-click on an *Association* to display the context menu.
5. Select the option to add a Message from one object to the other.
6. When the [Message Properties](#)<sup>[212]</sup> dialog displays, type in a name and any other required details.
7. Click on the **OK** button. The Message is added, connected to the Association and Object instances.
8. Move the Message to the required position.



### 3.18.2.2 Re-Order Messages

When constructing your Communication diagram, it is frequently necessary to create or delete Message 'groups' and to re-order the sequence of Messages. There are two dialogs that help you perform these tasks: the [Message Properties](#) dialog and the [Sequence Communications](#) dialog.

#### Organize Message Groups

If you have several [Messages](#)<sup>[223]</sup> in the form 1.1, 1.2, 1.3, 1.4, for example, but would like to start a new numbering group on, say, the third Message (that is, 1.1, 1.2, **2.1**, 2.2, 2.3), you can change a Message in the series to a *Start Group* message.

To reorganize *message groups*, follow the steps below:

1. Double-click on a Message *name*. The [Message Properties](#) dialog displays.

2. To make the selected Message the start of a new group, select the **Start New Group** checkbox.
3. If required, in the **Notes** field, type an explanatory note. You can format the text using the **Rich Text Notes** toolbar at the top of the field.
4. Click on the **OK** button to save changes.

## Sequence Messages

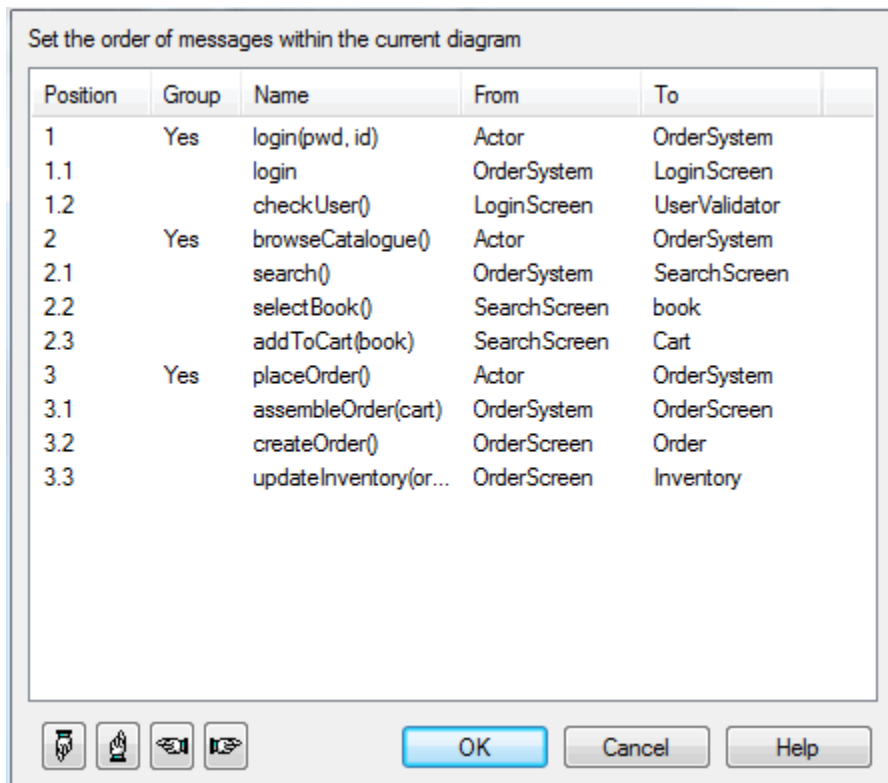
In larger and more complex diagrams, you might have to use deeper levels of Messages in a group; for example, 1, 1.2, 1.2.1, 1.2.1.1. You might also have to change the sequence of Messages, making Message 1.3, for example, into Message 1.1.

To change the sequence or level of *Messages*, follow the steps below:

1. Either:
  - Select the **Diagram | Sequence Messages** menu option
  - Click on the diagram background and select the **Sequence Communication Messages** context menu option or
  - Right-click on a Message and select the **Sequence Communication Messages** context menu option.

The **Communication Messages** dialog displays.



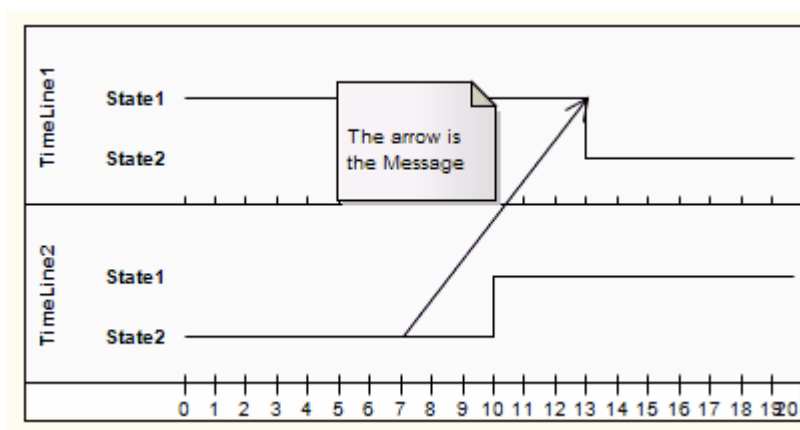


2. Click on the Message to adjust and, at the bottom of the dialog, click on the:
  - 'Up Hand' or 'Down Hand' buttons to move the Message up or down the sequence (e.g. Message 1.2 to Message 1.1 or 1.3)
  - 'Left Hand' or 'Right Hand' buttons to move the Message up or down a level (e.g. Message 1.2.1 to Message 1.2 or Message 1.2.1.1).
3. Repeat step 2 until the Message sequence and levels match your requirements. You might have to adjust other Message numbers (in group, sequence or level) to accommodate the changes you have made.
4. Click on the **OK** button to save changes.

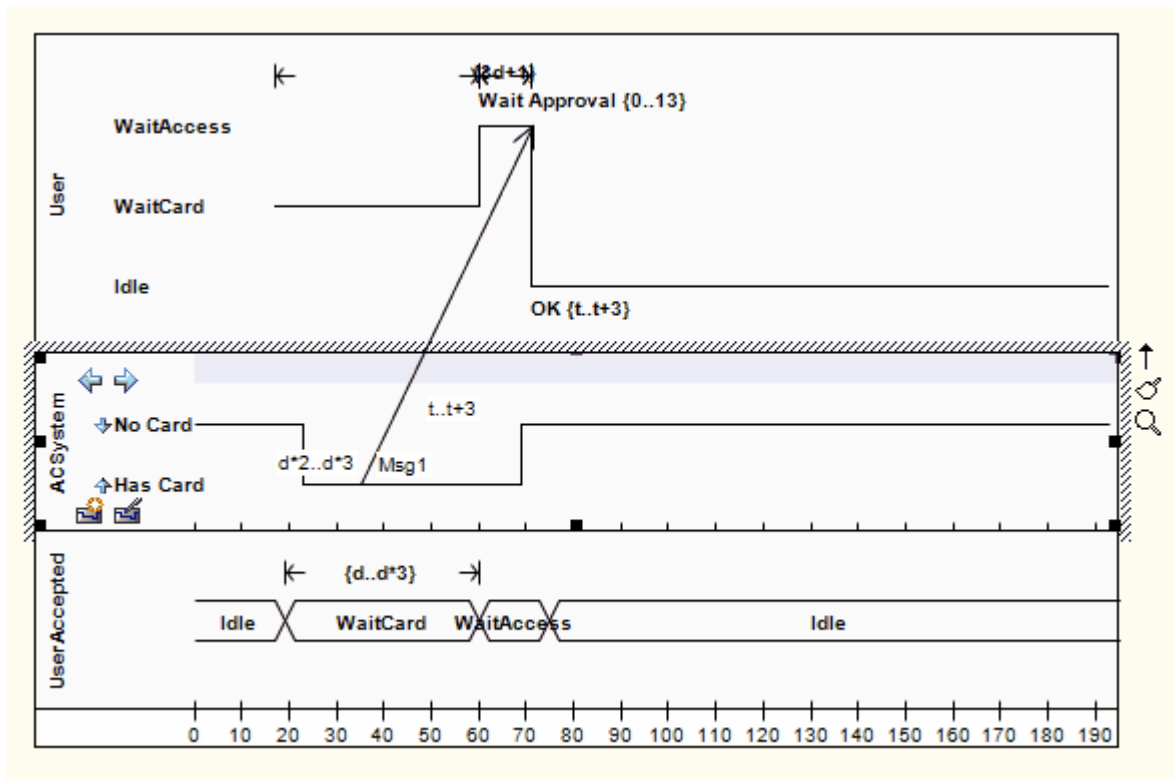
**Note:**

Communication diagrams were known as Collaboration diagrams in UML 1.4.

### 3.18.3 Message (Timing Diagram)

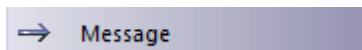


Messages are the communication links between [Lifelines](#) <sup>[135]</sup> in a [Timing diagram](#) <sup>[22]</sup>. In the case of a Timeline, a Message is a connection between two Timeline objects.



See *UML Superstructure Specification, v2.1.1, figures 14.30 and 14.31, p. 520.*

### Toolbox Icon



#### 3.18.3.1 Create a Timing Message

To create a [Message](#) <sup>[226]</sup> in a [Timing diagram](#) <sup>[22]</sup>, at least two Lifeline objects ([State](#) <sup>[129]</sup> or [Value](#) <sup>[149]</sup>) must be created first, each with existing transition points. To create a Message between lifelines, follow the steps below:

1. Click on one of the Lifelines in the Timing diagram.
2. Select the **Message** icon from the **Timing Relationships** page of the Enterprise Architect UML **Toolbox** (**More tools | Timing**).
3. Drag the cursor onto the Lifeline at the point at which the Message originates. The **Timing Message** dialog displays. (If not, double-click on the Message.)

The dialog box is titled 'Message (Timing Diagram)'. It contains the following fields:

- Scope:** Start: ACSystem, End: User
- Message Details:** Start Time: 35, End Time: 71, Name: Msg1, Time Observation: t..t+3, Duration Observation: d\*2..d\*3
- Transition To Detail:** Transition To: Idle, Event: OK, Time Constraint: t..t+3, Duration Constraint: (empty)

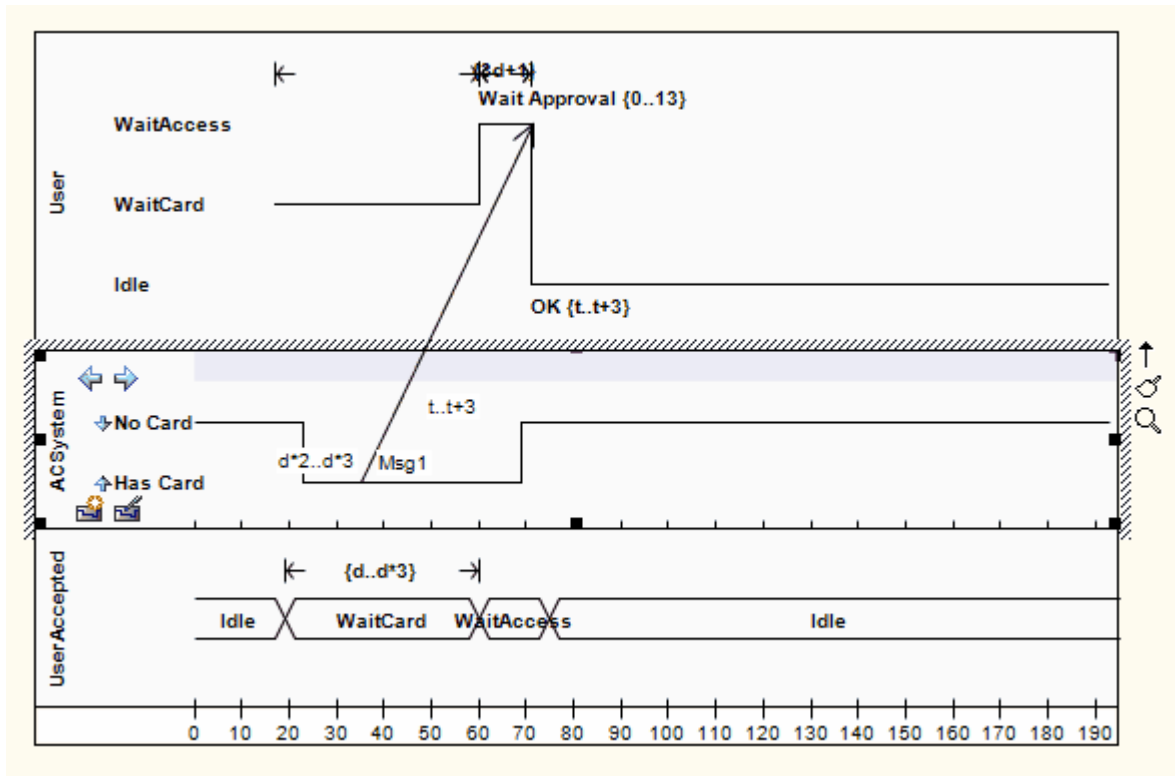
The dialog consists of a set of transition points. Each transition point can be defined with the following properties:

Property	Description
<b>Start</b>	Defines the lifeline where the message originates.
<b>End</b>	Defines the lifeline where the message terminates.

These are set by default when a Message is created by dragging the cursor between two Lifelines.

Property	Description
<b>Start Time</b>	Specifies the start time for a message.
<b>End Time</b>	Specifies the end time for a message.
<b>Name</b>	The name of the message.
<b>Time Observation</b>	Provides information on the time of a sent message.
<b>Duration Observation</b>	Indicates the interval of a Lifeline at a particular state, begun from a message receipt.
<b>Transition To</b>	The state in the target Lifeline that the Message points to.
<b>Event</b>	The occurring event.
<b>Time Constraint</b>	The time taken to transmit a message.
<b>Duration Constraint</b>	Pertains to a lifeline's period at a particular state. The constraint could be instigated by that Lifeline's receipt of a message.

The following diagram shows the Message configured by the above dialog snapshot.

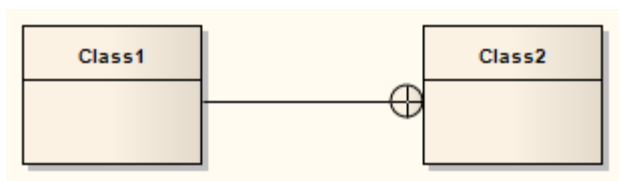


See *UML Superstructure Specification, v2.1.1, figures 14.30 and 14.31, p. 520.*

#### Note:

You can move the source end of the Message freely along the source timeline. However, the target end (arrow head) must attach to a transition. If you create a new Message and do not give it a target transition, it automatically finds and attaches to the nearest transition. If you move the target end, it drags the transition with it.

### 3.19 Nesting

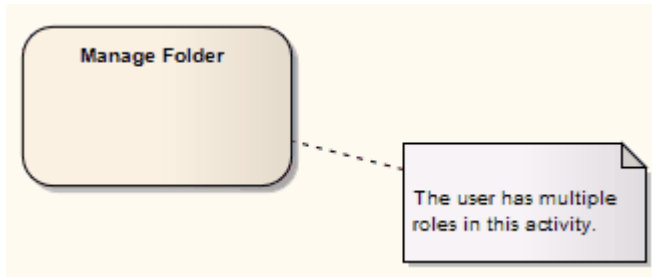


The *Nesting Connector* is an alternative graphical notation for expressing containment or nesting of elements within other elements. It is most appropriately used for displaying [Package](#) <sup>168</sup> nesting in a [Package diagram](#) <sup>55</sup>.

#### Toolbox Icon



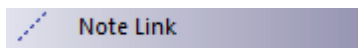
### 3.20 Notelink



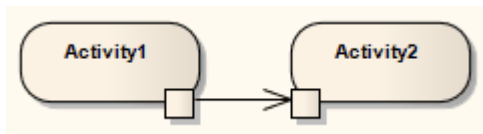
A *Notelink* connector connects a [Note](#)<sup>[126]</sup> to one or more other elements of any other type.

Both Note and Notelink are available in any category of the Enterprise Architect UML **Toolbox**, in the **Common** pages. You can also select them from the **UML Elements** toolbar (see *Using Enterprise Architect - UML Modeling Tool*).

#### Toolbox Icon



### 3.21 Object Flow



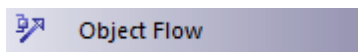
*Object Flows* are used in [Activity diagrams](#)<sup>[5]</sup> and [State Machine diagrams](#)<sup>[9]</sup>. When used in an Activity diagram, an Object Flow connects two elements, with specific data passing through it. To view sample Activity diagrams using Object Flows, see the [Object Flows in Activity Diagrams](#)<sup>[231]</sup> topic.

In State Machine diagrams, an Object Flow is a specification of a state flow or transition. It implies the passing of an [Object](#)<sup>[165]</sup> instance between elements at run-time.

You can insert an Object Flow from the **State** or **Activity** pages of the Enterprise Architect UML **Toolbox**, or from the drop-down list of all relationships located in the header toolbar. You can also modify a transition connection to an Object Flow by selecting the **ObjectFlow** checkbox on the connection **Properties** dialog.

See the [Control Flow](#)<sup>[204]</sup> topic for information on setting up Guards and Weights on Object Flows.

#### Toolbox Icon



#### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 389*) states:

*An object flow is an activity edge that only passes object and data tokens.*

### 3.21.1 Object Flows in Activity Diagrams

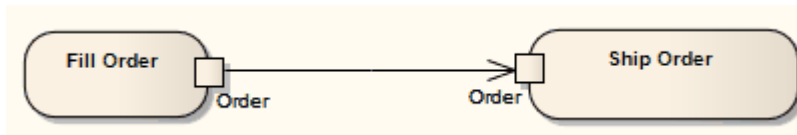
In [Activity diagrams](#)<sup>[5]</sup>, there are several ways to define the flow of data between objects.

The following diagram depicts a simple [Object Flow](#)<sup>[230]</sup> between two actions, *Fill Order* and *Ship Order*, both accessing order information.



See *UML Superstructure Specification, v2.1.1, figure 12.110, p. 391.*

This explicit portrayal of the data object *Order*, connected to the Activities by two Object Flows, can be refined by using the following format. Here, [Action Pins](#)<sup>[87]</sup> are used to reflect the order.



See *UML Superstructure Specification, v2.1.1, figure 12.110, p. 391.*

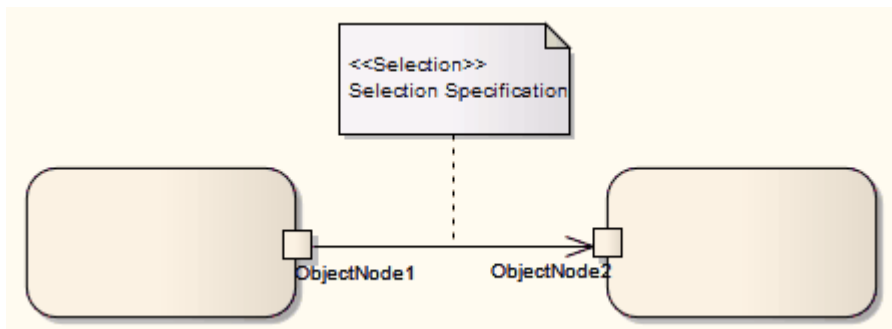
The following diagram is an example of multiple Object Flows exchanging data between two actions.



See *UML Superstructure Specification, v2.1.1, figure 12.111, p. 391.*

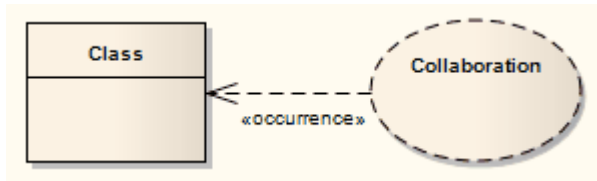
Selection and transformation behavior, together composing a sort of query, can specify the nature of the Object Flow's data access. Selection behavior determines which objects are affected by the connection. Transformation behavior might then further specify the value of an attribute pertaining to a selected object.

Selection and transformation behaviors can be defined by attaching a note to the Object Flow. To do this, right-click on the Object Flow and select the **Attach Note or Constraint** context menu option. A dialog lists other flows in the diagram, to which you can select to attach the note, if the behavior applies to multiple flows. To comply with UML 2, preface the behavior with the notation «selection» or «transformation».



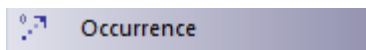
See *UML Superstructure Specification, v2.1.1, figure 12.112, p. 392.*

### 3.22 Occurrence

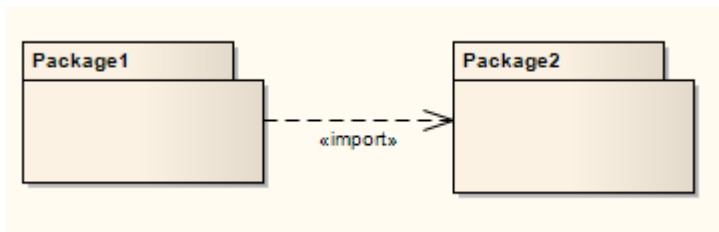


An *Occurrence* relationship indicates that a [Collaboration](#)<sup>[156]</sup> represents a classifier, in a [Composite Structure diagram](#)<sup>[59]</sup>. An Occurrence connector is drawn from the Collaboration to the classifier.

#### Toolbox Icon

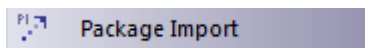


### 3.23 Package Import



A *Package Import* relationship is drawn from a source [Package](#)<sup>[168]</sup> to a Package whose contents are to be imported. Private members of a target Package cannot be imported. The relationship is typically used in a [Package diagram](#)<sup>[55]</sup>.

#### Toolbox Icon

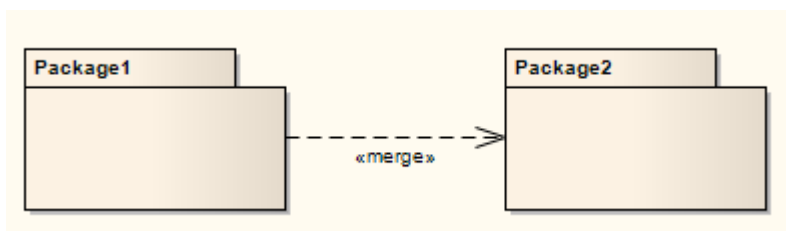


#### OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 112*) states:

*A package import is a relationship between an importing namespace and a package, indicating that the importing namespace adds the names of the members of the package to its own namespace. Conceptually, a package import is equivalent to having an element import to each individual member of the imported namespace, unless there is already a separately-defined element import.*

### 3.24 Package Merge

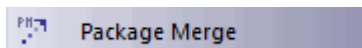


In a [Package diagram](#)<sup>[55]</sup>, a *Package Merge* indicates a relationship between two [Packages](#)<sup>[168]</sup> whereby the contents of the target Package are merged with those of the source Package. Private contents of a target Package are not merged. The applicability of a Package Merge addresses any situation where multiple packages contain identically-named elements, representing the same thing. A Package Merge merges all matching elements across its merged Packages, along with their relationships and behaviors. Note that a Package Merge essentially performs generalizations and redefinitions of all matching elements, but the merged Packages and their independent element representations still exist and are not affected.

The Package Merge serves a graphical purpose in Enterprise Architect, but creates an ordered Package relationship applied to related Packages (which can be seen under the [Link](#) tab in the Package's [Properties](#) dialog). Such relationships can be reflected in XML exports or Enterprise Architect Automation Interface scripts for code generation or other Model Driven Architecture (MDA) interests.

Package Merge relationships are useful to reflect situations where existing architectures contain functionalities involving like elements, which are merged in a developing architecture. Merging doesn't affect the merged objects, and supports the common situation of product progression.

### Toolbox Icon



### OMG UML Specification

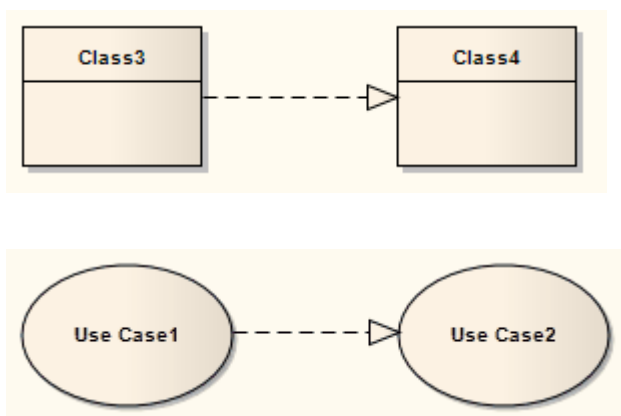
The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 113-114*) states:

*A package merge is a directed relationship between two packages that indicates that the contents of the two packages are to be combined. It is very similar to Generalization in the sense that the source element conceptually adds the characteristics of the target element to its own characteristics resulting in an element that combines the characteristics of both.*

*This mechanism should be used when elements defined in different packages have the same name and are intended to represent the same concept. Most often it is used to provide different definitions of a given concept for different purposes, starting from a common base definition. A given base concept is extended in increments, with each increment defined in a separate merged package. By selecting which increments to merge, it is possible to obtain a custom definition of a concept for a specific end. Package merge is particularly useful in meta-modeling and is extensively used in the definition of the UML metamodel.*

*Conceptually, a package merge can be viewed as an operation that takes the contents of two packages and produces a new package that combines the contents of the packages involved in the merge. In terms of model semantics, there is no difference between a model with explicit package merges, and a model in which all the merges have been performed.*

### 3.25 Realize

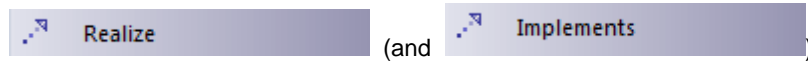


A source object implements or *Realizes* its destination object. Realize connectors are used in a [Use Case](#)<sup>[7]</sup>, [Component](#)<sup>[65]</sup> or [Requirements](#)<sup>[71]</sup> diagram to express traceability and completeness in the model. A business process or [Requirement](#)<sup>[189]</sup> is realized by one or more [Use Cases](#)<sup>[146]</sup>, which in turn are realized by



some [Classes](#)<sup>[152]</sup>, which in turn are realized by a [Component](#)<sup>[158]</sup>, and so on. Mapping Requirements, Classes and such across the design of your system, up through the levels of modeling abstraction, ensures the big picture of your system remembers and reflects all the little pictures and details that constrain and define it. (See *UML Model Management*.)

## Toolbox Icon

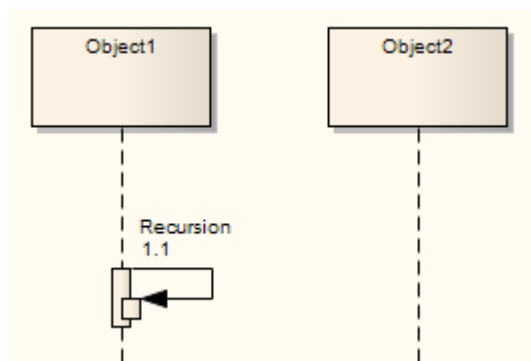


## OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 131*) states:

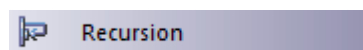
*A Realization signifies that the client set of elements are an implementation of the supplier set, which serves as the specification. The meaning of 'implementation' is not strictly defined, but rather implies a more refined or elaborate form in respect to a certain modeling context. It is possible to specify a mapping between the specification and implementation elements, although it is not necessarily computable.*

### 3.26 Recursion

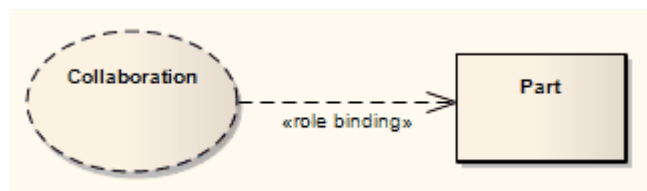


A *Recursion* is a type of [Message](#)<sup>[212]</sup> used in [Sequence diagrams](#)<sup>[39]</sup> to indicate a recursive function.

## Toolbox Icon



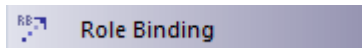
### 3.27 Role Binding



*Role Binding* is the mapping between a [Collaboration Occurrence's](#)<sup>[157]</sup> internal roles and the respective [Parts](#)<sup>[168]</sup> required to implement a specific situation, typically in a [Composite Structure diagram](#)<sup>[59]</sup>. The associated Parts can have properties defined to enable the binding to occur, and the Collaboration to take place.

A Role Binding connector is drawn between a [Collaboration](#)<sup>[156]</sup> and the classifier's fulfilling roles, with the Collaboration's internal binding roles labeled on the classifier end of the connector.

## Toolbox Icon

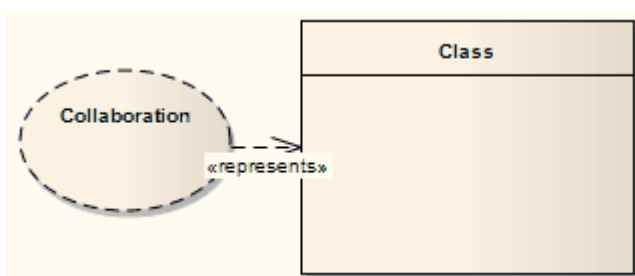


## OMG UML Specification

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 174*) states:

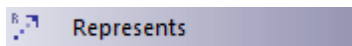
*A mapping between features of the collaboration type and features of the classifier or operation. This mapping indicates which connectable element of the classifier or operation plays which role(s) in the collaboration. A connectable element may be bound to multiple roles in the same collaboration occurrence (that is, it may play multiple roles).*

### 3.28 Represents

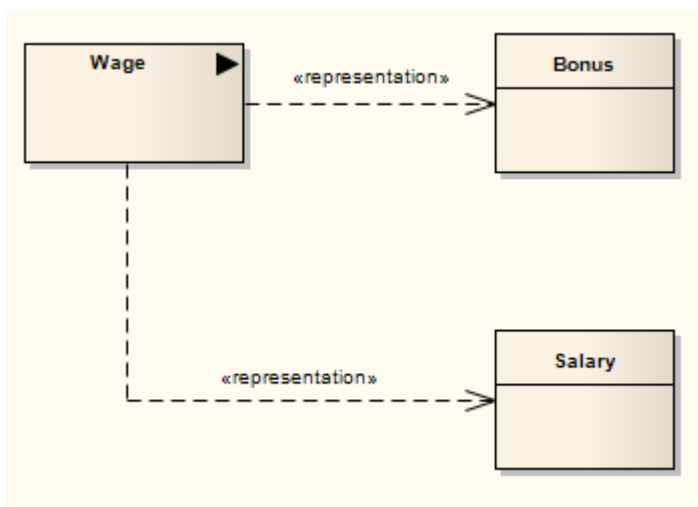


The *Represents* connector indicates that a [Collaboration](#)<sup>[156]</sup> is used in a classifier, typically in a [Composite Structure diagram](#)<sup>[59]</sup>. The connector is drawn from the Collaboration to its owning classifier.

## Toolbox Icon

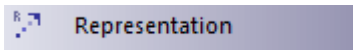


### 3.29 Representation

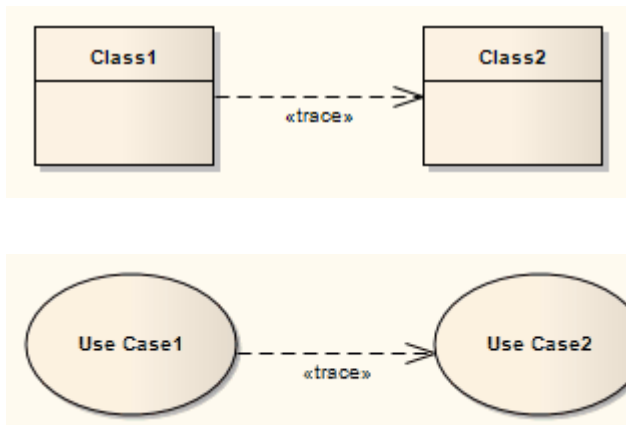


The *Representation* relationship is a specialization of a [Dependency](#)<sup>[205]</sup>, connecting [Information Item](#)<sup>[163]</sup> elements that represent the same idea across models, typically in an [Analysis diagram](#)<sup>[67]</sup>. For example, *Bonus* and *Salary* are both a representation of the Information Item *Wage*.

## Toolbox Icon



### 3.30 Trace



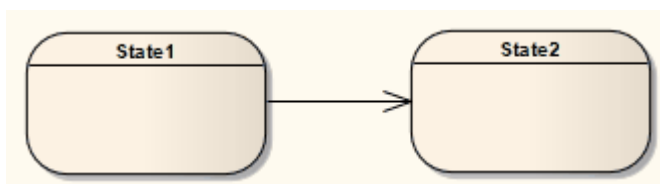
The *Trace* relationship is a specialization of a [Dependency](#)<sup>[208]</sup>, connecting model elements or sets of elements that represent the same concept across models. Traces are often used to track requirements and model changes, typically in a Traceability diagram (see *UML Model Management*), or in a [Class](#)<sup>[56]</sup>, [Use Case](#)<sup>[7]</sup>, [Object](#)<sup>[58]</sup> or [Composite Structure](#)<sup>[59]</sup> diagram.

As changes can occur in both directions, the order of this Dependency is usually ignored. The relationship's properties can specify the trace mapping, but the trace is usually bi-directional, informal and rarely computable.

## Toolbox Icon



### 3.31 Transition



A *Transition* defines the logical movement from one [State](#)<sup>[129]</sup> to another, in a [State Machine diagram](#)<sup>[9]</sup>. The Transition can be controlled through the following connector **Properties** dialog:

The screenshot shows the 'Tagged Values' tab of a UML Transition dialog. It includes the following elements:

- Guard:** A text field containing 'Guard'.
- Effect:** A text field containing 'Effect' with a scroll bar and an ellipsis button.
- Effect is a Behavior:** An unchecked checkbox.
- Trigger:** A section containing:
  - Name:** 'Trigger2'
  - Type:** A dropdown menu set to 'Signal'.
  - Specification:** 'Sig1' with an ellipsis button.
  - Save:** A button to the right of the Specification field.
- Triggers Table:**

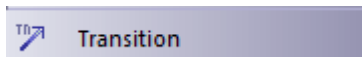
Name	Type	Specification
Trigger2	Signal	Sig1
- Buttons:** 'Add', 'Delete', 'OK', 'Cancel', and 'Help' are located at the bottom of the dialog.

Option	Use to
<b>Guard</b>	Type in an expression that is evaluated after an Event is dispatched, but before the corresponding Transition is triggered. If the guard is true at that time, the Transition is enabled; otherwise, it is disabled.
<b>Effect is a Behavior</b>	Convert the <b>Effect</b> field from a free-text field to the definition of a specific Activity or behavior.  Enterprise Architect displays the <b>Select &lt;Item&gt;</b> dialog to prompt you to select the Activity or behavior element from the model (see <i>UML Modeling With Enterprise Architect - UML Modeling Tool</i> ).
<b>Effect</b>	Either: <ul style="list-style-type: none"> <li>• Type a description of the effect of the Transition, or</li> <li>• If you have selected the <b>Effect is a Behavior</b> check box, select an Activity or behavior to be performed during the Transition (to change this subsequently, click on the [ ... ] button to redisplay the <b>Select &lt;Item&gt;</b> dialog).</li> </ul>
<b>Trigger</b>	
<b>Name</b>	Specify the name of the trigger.
<b>Type</b>	Specify the type of trigger: <b>Call</b> , <b>Change</b> , <b>Signal</b> or <b>Time</b> . <ul style="list-style-type: none"> <li>• <b>Call</b> - specifies that the event is a CallEvent, which sends a message to the associated object by invoking an operation.</li> <li>• <b>Change</b> - specifies that the event is a ChangeEvent, which indicates that the transition is the result of a change in value of an attribute.</li> <li>• <b>Signal</b> - specifies that the event is a SignalEvent, which corresponds to the</li> </ul>

Option	Use to
	<p>receipt of an asynchronous signal instance.</p> <ul style="list-style-type: none"> <li>• <b>Time</b> - corresponds to a TimeEvent; which specifies a moment in time.</li> </ul> <p><b>Note:</b></p> <p>Code generation for State Machines currently supports Change and Time trigger events only, and expects a specification value.</p>
<b>Specification</b>	Specify the event instigating the Transition.
<b>Save</b>	Save the current trigger.
<b>Add</b>	<p>Select triggers from the model using the <b>Select Trigger</b> dialog.</p> <p><b>Note:</b></p> <p>To add multiple triggers, press <b>[Ctrl]</b> while selecting each trigger.</p>
<b>Delete</b>	Remove the selected trigger from the list.
<b>Triggers</b>	List the current triggers for the Transition.

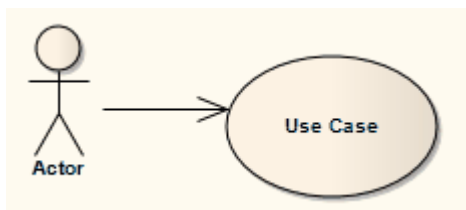
**Note:**

Fork and Join segments can have neither triggers nor guards.

**Toolbox Icon****OMG UML Specification**

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 568*) states:

*A transition is a directed relationship between a source vertex and a target vertex. It may be part of a compound transition, which takes the state machine from one state configuration to another, representing the complete response of the state machine to an occurrence of an event of a particular type.*

**3.32 Use**

A *Use* relationship indicates that one element requires another to perform some interaction. The *Use* (or *Usage*) relationship does not specify how the target supplier is used, other than that the source client uses it in definition or implementation. A *Use* relationship is a sub-typed [Dependency](#)<sup>[205]</sup> relationship.

You typically use the *Use* relationship in [Use Case diagrams](#)<sup>[7]</sup> to model how [Actors](#)<sup>[94]</sup> use system functionality ([Use Cases](#)<sup>[146]</sup>), or to illustrate usage dependencies between [Classes](#)<sup>[152]</sup> or [Components](#)<sup>[158]</sup>.

**Notes:**

- It is more usual (and correct UML) to have an [Association Connector](#)<sup>[199]</sup> between an Actor and a Use Case.
- To depict a usage dependency on a [Class](#)<sup>[56]</sup> or [Component](#)<sup>[65]</sup> diagram, draw a *Dependency* connector. Right-click on the Dependency, and select the **Dependency Stereotypes | Use** context menu option.

**Toolbox Icon****OMG UML Specification**

The OMG UML specification (*UML Superstructure Specification, v2.1.1, p. 138*) states:

*A usage is a relationship in which one element requires another element (or set of elements) for its full implementation or operation. In the metamodel, a Usage is a Dependency in which the client requires the presence of the supplier.*

# Index

## - A -

- Abstract
  - Complex Models 163
- AcceptEvent Action
  - Triggers Tab 81
- Action
  - AcceptEvent, Triggers Tab 81
  - BroadcastSignal, Signal Tab 81
  - Element 79
  - Expansion Node 86
  - Local Pre/Post Conditions 89
  - Notation 81
  - Operations 79
  - SendSignal, Signal Tab 81
  - StructuralFeature 81
  - Trigger, AcceptEvent 81
  - Type, Set 81
  - Update Operation 79
- Action Pin
  - Add To Action 87
  - As Action Property 88
  - As Argument For Call Action 87
  - Assign To Action 88
  - Properties 87
- Activation
  - End 45
  - Extend Down 45
  - Extend Up 45
  - Lower 45
  - Raise 45
  - Sequence Element 45
  - Suppress 45
- Activation Layer
  - Sequence Diagram Lifelines 46
- Activation Levels
  - Sequence Diagram Lifeline Self Messages 46
- Active
  - Classes 153
  - State Configuration 130
- Activity
  - Element 90
  - Instance 236
  - Notation 91
  - Parameter Nodes 91
  - Partition 93, 126
  - Process Element 189
  - Region Element 128
  - Structured 138
  - Structured, Conditional Node 136, 139
  - Structured, Loop Node 136, 139
  - Structured, Sequential 136
- Activity Diagram
  - Description 5
  - Elements And Connectors 5
  - Example 5
  - Object Flows 231
  - Operations 79
- Activity Edge
  - Connector 211
  - Relationship 211
- Activity Final
  - Element 110
- Activity Partition
  - Docking 126
  - Element 126
  - Horizontal 126
  - Vertical 126
- Actor
  - Element 94
- Add
  - Expansion Region 110
  - Instance Variable 166
  - Interruptible Activity Region 122
  - Port To Element 170
  - Property Value to Part 169
- Aggregate
  - Connector 198
  - Relationship 198
- Aggregation Connector
  - Change Form 199
- Analysis
  - Stereotypes 179
- Analysis Diagram
  - Description 67
  - Diagram 67
  - Elements And Connectors 67
  - Example 67
- Apply
  - Stereotype To Dependency Relationship 206
- Artifact
  - Element 151
- Assembly
  - Connector 199
  - Relationship 199
- Associate
  - Connector 199
  - Relationship 199

- Association
  - Class 187
  - Connector 199
  - N-Ary 187
  - Relationship 199
- Association Class
  - Connector 200
  - Link New Class To Association 201
  - Relationship 200
- Association End
  - Qualifiers 173, 175
- Asynchronous Signal Message
  - Associate With Signal 221
  - Connector 221
  - Relationship 221
- Attribute
  - Qualifiers 175
- B -**
- Behavior
  - Instance 236
- Behavioral Diagram
  - Elements 78
  - Overview 4
- Boundary
  - Element 179
  - Element Settings 144
  - Element, Create 180
  - Object Settings 144
  - Properties 144
- BroadcastSignal Action
  - Signal Tab 81
- Business Interaction Diagram
  - Description 75
  - Elements And Connectors 75
  - Example 75
- Business Modeling Diagram
  - Description 75
  - Elements And Connectors 75
  - Example 75
- Business Process Modeling 67
- C -**
- Call
  - Self Message 216
- Call Action
  - Pin As Argument 87
- Central Buffer Node
  - Element 95
- Change
  - Form Of Aggregation Connector 199
- Choice
  - Element 95
- Class
  - Active Classes 153
  - Element 152
  - Make Into Association Class 201
  - Parameterized Classes (Templates) 154
- Class Diagram
  - Description 56
  - Elements And Connectors 56
  - Example 56
- Classifier
  - Behavior 118
  - Item Conveyed 210
  - Properties 61
- Collaboration
  - Element 156
  - Message 224
- Collaboration Diagram
  - Description 49
  - Elements And Connectors 49
  - Example 49
  - Message Colors 51
- Collaboration Occurrence
  - Element 157
- Combined Fragment
  - Create 98
  - Element 96
  - Interaction Operator 99
- Communication
  - Connector 203
  - Message 223
  - Message, Create 224
  - Message, Level 224
  - Message, Properties 224
  - Message, Sequence 224
  - Relationship 203
- Communication Diagram
  - Description 49
  - Elements And Connectors 49
  - Example 49
  - Labelled Associations 49
  - Message Colors 51
  - Numbering In 49
- Communication Path
  - Connector 202
  - Relationship 202
- Component
  - Description 65



- Component
  - Diagram 65
  - Element 158
  - Elements And Connectors 65
  - Example 65
- Compose
  - Connector 202
  - Relationship 202
- Composite
  - Elements 180
  - State 129, 130
  - State Regions 12
- Composite Aggregation
  - Connector 202
  - Relationship 202
- Composite Structure Diagram
  - Description 59
  - Elements And Connectors 59
  - Example 59
- Compress
  - Timeline 36
  - Transition 36
- Concurrent Substate
  - Regions 12
- Conditional Node
  - Structured Activity 136, 139
- Configure Timeline Dialog
  - States Tab 29
  - Transitions Tab 31
- Connector
  - Activity Edge 211
  - Aggregate 198
  - Assembly 199
  - Associate 199
  - Association 199
  - Association Class 200
  - Asynchronous Signal Message 221
  - Communication 203
  - Communication Path 202
  - Compose 202
  - Composite Aggregation 202
  - Connector 203
  - Control Flow 204
  - Delegate 205
  - Dependency 205
  - Dependency, Apply Stereotype 206
  - Deployment 206
  - Extend 207
  - Generalization 207
  - Generalize 207
  - Implements 233
  - Include 208
  - Information Flow 208
  - Inheritance 207
  - Interrupt Flow 211
  - Manifest 211
  - Message 212
  - Nesting 229
  - Notelink 230
  - Object Flow 230
  - Occurrence 232
  - Overview 196
  - Package Import 232
  - Package Merge 232
  - Pkg Import 232
  - Pkg Merge 232
  - Realize 233
  - Recursion 234
  - Relationship 203
  - Representation 235
  - Represents 235
  - Role Binding 234
  - Self-Message 215
  - Trace 236
  - Transition 236
  - Usage 238
  - Use 238
  - What Is A? 196
- Constraint
  - Note, Element 126
  - Post Condition On Actions 89
  - Precondition On Actions 89
- Continuation
  - Element 132
- Control
  - Create 182
  - Element 181
- Control Flow
  - Connector 204
  - Guard 204
  - Relationship 204
  - Weight 204
- Convey
  - Information Item 210
- Co-Region Notation 212
- Create
  - Boundary Element 180
  - Combined Fragment 98
  - Communication Messages 224
  - Control Element 182
  - Entity 183
  - Timing Diagram 23

- Create
  - Timing Message 227
- CSV
  - Export State Machine Table To 21
- Custom
  - Diagram 71, 72, 73
- Custom Diagram
  - Description 69
  - Elements And Connectors 69
  - Example 69
  - Model 69
- D -**
- Data Type
  - Element 159
  - Instance 159
  - Referenced 159
- Database Schema
  - Description 75
  - Diagram 75
  - Elements And Connectors 75
  - Example 75
- Datastore
  - Element, Activity Diagram 102
- Decision
  - Element 102
- Define
  - Run-Time Variable 166
- Delegate
  - Connector 205
  - Relationship 205
- Delete
  - Instance Variable 167
- Dependency
  - Connector 205
  - Relationship 205
  - Relationship, Apply Stereotype 206
- Deployment
  - Connector 206
  - Diagram 62
  - Relationship 206
- Deployment Diagram
  - Description 62
  - Elements And Connectors 62
  - Example 62
- Deployment Spec
  - Element 160
- Device
  - Element 161
- Diagram
  - Activity, Description 5
  - Analysis 67
  - Behavioral, Overview 4
  - Business Interaction 75
  - Business Modeling 75
  - Class 56
  - Collaboration 49
  - Communication 49
  - Component 65
  - Composite Structure 59
  - Custom 69, 71, 72, 73
  - Database Schema 75
  - Deployment 62
  - Diagram 179, 181, 182
  - Extended 4
  - Extended UML 67
  - Frame 104
  - Frame (Border) 104
  - Hyperlink 104
  - Interaction 22, 39, 49, 52
  - Interaction Overview 52
  - Logical 56
  - Maintenance 72
  - MDG Technology 4
  - Move, Impact On Element 102, 110, 111, 117
  - Object 58
  - Overview 4
  - Package 55
  - Reference 104
  - Requirements 71
  - Robustness 39, 49, 67, 179, 181, 182
  - Schema Diagram 75
  - Sequence 39
  - State 9
  - State Machine 9
  - Structural, Overview 54
  - Timing 22
  - Types 4
  - UML 4
  - Use Case 7
  - User Interface 73
  - User Interface Design 69
  - What Is A? 4
- Diagram Frame 104
- Diagram Gate
  - Element 105
- Dialog
  - New Action 81
  - Qualifiers 175
  - Set Attribute 85
  - Set Feature 85

Dialog  
 Set Operation 85  
 Document Artifact  
 Element 161  
 Duration  
 Constraint 218  
 Constraint Between Messages 218  
 Observation 218

## - E -

Element  
 Action 79  
 Activity 90  
 Activity Final 110  
 Activity Partition 126  
 Activity Region 128  
 Actor 94  
 Artifact 151  
 Behavioral Diagram 78  
 Boundary 179  
 Boundary, Settings 144  
 Central Buffer Node 95  
 Choice 95  
 Class 152  
 Collaboration 156  
 Collaboration Occurrence 157  
 Combined Fragment 96  
 Component 158  
 Composite 180  
 Constraint Note 126  
 Continuation 132  
 Continuation 132  
 Control 181, 182  
 Data Type 159  
 Datastore, Activity Diagram 102  
 Decision 102  
 Deployment Spec 160  
 Device 161  
 Diagram Frame 104  
 Diagram Gate 105  
 Document Artifact 161  
 Endpoint 106  
 Entity 182  
 Entry Point 107  
 Enumeration 162  
 Event 183  
 Exception 107  
 Execution Environment 162  
 Exit Point 110  
 Expansion Region 107  
 Expose Interface 163  
 Extended By Stereotype 178  
 Feature 184  
 Flow Final 111  
 Fork 112, 114  
 Fragment 96  
 History 116  
 Hyperlink 184  
 Information Item 163  
 Initial 117  
 Instance 165  
 Interaction 118  
 Interaction Occurrence 119  
 InteractionUse 119  
 Interface 164  
 Interruptible Activity Region 121  
 Join 112, 115  
 Junction 122  
 Lifeline 123  
 Merge 102  
 Merge Node 124  
 N-Ary Association 187  
 Node 165  
 Note (Constraint, Comment) 126  
 Object 165  
 Occurrence 119  
 Package 168  
 Packaging Component 188  
 Part 168  
 Partition 126  
 Port 169  
 Primitive 173  
 Process 189  
 Properties, Edit From Package 55  
 Pseudo-State 13  
 Receive 127  
 Receive Event 183  
 Region 128  
 Region, Expansion 107  
 Region, Interruptible Activity 121  
 Requirement 189  
 Screen 190  
 Send 129  
 Send Event 183  
 Sequence Diagram 43  
 Sequence, Lifecycle 41  
 Signal 178  
 State 129  
 State Invariant 132, 134  
 State Lifeline 135  
 State Machine 136

## Element

State/Continuation 132  
 Structural Diagram 150  
 Structured Activity, Conditional Node 136, 139  
 Structured Activity, Loop Node 136, 139  
 Structured Activity, Sequential Node 136, 138  
 Structured Activity, Structured Node 136, 138  
 Sub-Activity 90  
 Sub-Activity, Conditional Node 136, 139  
 Sub-Activity, Loop Node 136, 139  
 Sub-Activity, Sequential Node 136, 138  
 Sub-Activity, Structured Node 136, 138  
 Submachine State 129, 136  
 Synch 142  
 System Boundary 142  
 Table 192  
 Terminate 144  
 Test Case 191  
 Trigger 145  
 UI Control 192  
 UML 78  
 Use Case 146  
 User Interface 192  
 Value Lifeline 149

## Endpoint

Element 106

## Enterprise Architect

Alignment With UML 2  
 Connectors 196

## Entity

Create 183  
 Element 182

## Entry Point

Element 107

## Enumeration

Element 162  
 Literal 162

## Eriksson-Penker Business Extensions 67

## Event

Element 183  
 Receive 183  
 Send 183

## Exception

Element 107

## Execution Environment

Element 162

## Exit Point

Element 110

## Expansion Node

Action 86

## Expansion Region

Add 110

Element 107

## Export

State Machine Table To CSV 21

## Expose Interface

Element 163

## Extend

Connector 207

Relationship 207

## Extended Elements 178

## Extended UML Diagrams 67

## Extension Points

Use Case 147

## Extension Stereotypes 178

**- F -**

## FDD Methodology 184

## Feature

Element 184

## Feature Driven Design Methodology 184

## File

Hyperlink To 184

## Flow Final

Element 111

## Fork

Element 112, 114

Pseudo-State 112, 114

## Fork/Join

Element 112

## Fragment

Element 96

**- G -**

## General Ordering

Sequence Diagram Messages 220

## Generalize

Connector 207

Relationship 207

**- H -**

## Help

Topic, Hyperlink To 184

## History

Element 116

## Hyperlink

As Sub Activities 184

Diagrams 184

- Hyperlink
  - Element 184
  - To External Files 184
  - To Help Topics 184
  - To Internet Facilities 184
  - To Matrix Profiles 184
  - To Model Search 184
  - To Team Review 184
- I -**
- Implements
  - Connector 233
  - Relationship 233
- Inbuilt Stereotypes 178
- Include
  - Connector 208
  - Relationship 208
- Information Flow
  - And Patterns 208
  - Connector 208
  - In Combination 208
  - Realized 210
  - Relationship 208
- Information Item
  - Conveyed 210
  - Element 163
- Inheritance
  - Connector 207
  - Relationship 207
- Inherited Port
  - Manage 170
- Initial
  - Element 117
- Inline Sequence Elements
  - Part And Port 49
- Insert
  - Boundary Element 144
- Instantiated Template 154
- Interaction
  - Diagram 22, 49, 52
  - Element 118
- Interaction Diagram
  - Description 39
  - Diagram 39
  - Elements And Connectors 39
  - Example 39
- Interaction Occurrence
  - Element 119
- Interaction Operator
  - Combined Fragment 99
- Interaction Overview Diagram
  - Description 52
  - Elements And Connectors 52
  - Example 52
- InteractionUse
  - Element 119
- Interface
  - Element 164
  - Expose Element 163
  - Provided 163
  - Required 163
- Internet Facilities
  - Hyperlink To 184
- Interrupt Flow
  - Connector 211
  - Relationship 211
- Interruptible Activity Region
  - Add 122
  - Element 121
- Interval Bar
  - Context Menu 32
  - Timing Diagram Time Interval 32
- Introduction
  - To UML Objects 78
- J -**
- Join
  - Element 112, 115
  - Pseudo-State 112, 115
- Junction
  - Element 122
- L -**
- Label Visibility
  - On Sequence Messages 48
- Layout
  - Sequence Diagram 42
- Legend
  - Add To State Machine Table 20
  - Remove From State Machine Table 20
- Lifecycle
  - Of A Sequence Element 41
- Lifeline
  - Element 123
  - Objects In Sequence Diagrams 44
  - Sequence Element, Termination 41
- Local
  - Pre/Post Conditions 89
- Logical Diagram

- Logical Diagram
  - Class Diagram 56
- Loop Node
  - Structured Activity 136, 139
- M -**
- Maintenance Diagram
  - Description 72
  - Elements And Connectors 72
  - Example 72
- Manage
  - Inherited Ports 170
  - Redefined Ports 170
- Manifest
  - Connector 211
  - Relationship 211
- Matrix Profile
  - Hyperlink To 184
- Merge
  - Element 102, 124
  - Node 124
- Merge Packages
  - Relationship 232
- Message
  - Asynchronous Signal 221
  - Collaboration 224
  - Colors In Communication Diagrams 51
  - Communication 223
  - Communication, Create 224
  - Connector 212
  - Create On Timing Diagram 227
  - Endpoint 124
  - Group, Start New 224
  - Label 125
  - Level 224
  - Move 212
  - Recursion 234
  - Relationship 212
  - Self Message 215
  - Self Message Call 216
  - Sequence Communication 224
  - Sequence Diagram, Asynchronous Signal 221
  - Sequence Diagram, Examples 217
  - Sequence Diagram, General Ordering 220
  - Sequence Diagram, Self Message 215
  - Sequence, Create 212
  - Sequence, Label Visibility 48
  - Sequencing 224
  - Timing Diagram 226
- Message Angle
  - Adjust With Duration Constraint 218
- Methodology
  - FDD 184
  - Feature-Driven Design 184
- Model Search
  - Hyperlink To 184
- Model-View-Controller Pattern 182
- MVC Pattern 182
- N -**
- N-Ary
  - Association Element 187
- Nesting
  - Connector 229
  - Relationship 229
- New Action Dialog 81
- New Structured Activity Dialog 136
- Node
  - Element 165
- Notation
  - Co-Region 212
- Note
  - Element 126
- Notelink
  - Connector 230
  - Relationship 230
- Numeric Range Generator
  - Timeline Element States 29
- O -**
- Object
  - Element 165
  - Instance 165
  - State, Set 167
- Object Diagram
  - Description 58
  - Elements And Connectors 58
  - Example 58
- Object Flow
  - Connector 230
  - In Activity Diagram 231
  - In State Machine Diagram 230
  - Multiple 231
  - Relationship 230
  - Selection Behavior 231
  - Simple 231
  - Transformation Behavior 231
  - With Action Pins 231
- Occurrence

Occurrence  
 Connector 232  
 Element 119  
 Relationship 232

Operation  
 As Action 79

## - P -

Package  
 Element 168

Package Diagram  
 Description 55  
 Elements And Connectors 55  
 Example 55

Package Import  
 Connector 232  
 Relationship 232

Package Merge  
 Connector 232  
 Relationship 232

Packaging Component  
 Element 188

Parameterized Classes (Templates) 154

Part  
 Add Property Value 169  
 Element 168  
 Property Tab 172  
 Qualifiers 175  
 Represent On Sequence Diagram 49

Partition  
 Activity 126  
 Docking 126  
 Element 126  
 Horizontal 126  
 Vertical 126

Pattern  
 Model-View-Controller 182  
 MVC 182

Pin  
 Action 87  
 Add To Action 87  
 As Action Property 88  
 As Argument For Call Action 87  
 Assign To Action 87, 88  
 Properties 87

Pkg Import  
 Connection 232  
 Relationship 232

Pkg Merge  
 Connector 232

Relationship 232

Port  
 Add To Element 170  
 Element 169  
 For Trigger Element 145  
 Inherited 170  
 Property Tab 172  
 Qualifiers 175  
 Redefined 170  
 Represent On Sequence Diagram 49

Pre/Post Conditions  
 Constraints 89  
 Local 89  
 Notes 89  
 On Actions 89

Primitive  
 Element 173

Process  
 Element 189

Properties  
 Effect Tab 79  
 Element, Trigger Tab 145  
 Of Classifiers, Composite Structure Diagram 61

Properties,  
 Part, Property Tab 172  
 Port, Property Tab 172

Property Value  
 Part, Add To 169

Pseudo-State  
 Elements 13  
 Fork 112, 114  
 In State Machine Diagram 13  
 Join 112, 115

## - Q -

Qualified Association 173

Qualifier  
 Association End 175  
 Association Property 173  
 Attribute 175  
 Attribute Property 173  
 Dialog 175  
 Part 175  
 Part Property 173  
 Port 175  
 Port Property 173  
 Set Properties 175

**- R -**

## Realize

- An Information Flow 210
- Connector 233
- Relationship 233

## Receive

- Element 127
- Event 183

## Reception

- Definition 178
- Of Signal 178

## Rectangle Notation

- For Use Cases 148

## Recursion

- Connector 234
- Message 234
- Relationship 234

## Redefined Port

- Manage 170

## Region

- Composite State 12
- Concurrent Substate 12
- Element 128
- Expansion, Element 107
- Interruptible Activity, Element 121
- On Composite State 130
- State Machine 12

## Relationship

- Activity Edge 211
- Aggregate 198
- Assembly 199
- Associate 199
- Association 199
- Association Class 200
- Asynchronous Signal 221
- Communication 203
- Communication Path 202
- Compose 202
- Composite Aggregation 202
- Connector 203
- Control Flow 204
- Delegate 205
- Dependency 205
- Dependency, Apply Stereotype 206
- Deployment 206
- Extend 207
- Generalization 207
- Generalize 207
- Implements 233

Include 208

Information Flow 208

Inheritance 207

Interrupt Flow 211

Manifest 211

Message 212

Nesting 229

Notelink 230

Object Flow 230

Occurrence 232

Package Import 232

Package Merge 232

Pkg Import 232

Pkg Merge 232

Realize 233

Recursion 234

Representation 235

Represents 235

Role Binding 234

Self Message 215

Trace 236

Transition 236

Usage 238

Use 238

## Re-Order

- Messages 224

## Representation

- Connector 235
- Relationship 235

## Represents

- Connector 235
- Relationship 235

## Requirement

- Element 189
- Hide Stereotype Letter 189
- Show Stereotype Letter 189

## Requirements Diagram

- Description 71
- Elements And Connectors 71
- Example 71

## Robustness Diagram 179

## Role Binding

- Connector 234
- Relationship 234

## Run State 166

- Add Instance Variable 166

## Run-Time

- Variable, Define 166

## Run-Time State

- Add Instance Variable 166
- Delete Instance Variable 167



Run-Time State  
Introduction 166

## - S -

Scenario  
Use Case 43

Schema  
Database 75

Screen  
Element 190

Self Message  
Calls 216

Self-Message  
Connector 215  
Hierarchy, Sequence Diagram 46  
Relationship 215  
Return 215

Send  
Element 129  
Event 183

SendSignal Action  
Signal Tab 81

Sequence  
Communication Messages 224  
Message, Change Timing Details 218  
Message, Create 212  
Message, Timing Details 218

Sequence Diagram  
Activation Levels 46  
And Version Control 44  
Damage To 44  
Description 39  
Element Activation 45  
Elements 43  
Elements And Connectors 39  
Example 39  
Layout 42  
Lifeline Activation Level 46  
Messages, Asynchronous Signal 221  
Messages, Self Message 215  
Self-Message Hierarchy 46  
Top Margin, Change 48

Sequence Diagram Message  
Examples 217  
External To Sequence 217  
General Ordering 220

Sequence Element  
Inline, Part And Port 49

Sequence Message  
Label Visibility 48

Modify Height 42

Sequential Node  
Structured Activity 136, 138

Set  
Object State 167

Set Attribute Dialog 85

Set Feature Dialog 85

Set Operation Dialog 85

Signal  
Element 178  
Reception 178

State  
Add To State Lifeline Element 25  
Chart 9  
Composite 129, 130  
Delete On State Lifeline Element 25  
Diagram 9  
Edit On State Lifeline Element 25  
Element 129  
Entry And Exit Actions 129  
In Timeline Element 29  
Locate In State Machine Diagram 20  
Locate In State Machine Table 20  
Reposition In State Machine Table 20  
Simple 129  
State Machine Table Conventions 21

State Invariant  
Element 132, 134

State Lifeline  
Element 135

State Lifeline Element  
Add State 25  
Add To Timing Diagram 24  
Add Transition 26  
Change Transition Time 26  
Define Name 24  
Delete State 25  
Delete Transition 26  
Edit State 25  
Edit Transition 26  
Merge Transitions 26  
Move Transition 26  
Set Timeline Start Position 24  
Sizing and Scale 24  
Synchronize Transition 26

State Machine  
Element 136  
Regions 12

State Machine Diagram  
Description 9  
Display Format 9

- State Machine Diagram
    - Elements And Connectors 9
    - Example 9
    - Locate State In State Machine Table 20
    - Locate Transition In State Machine Table 20
    - Locate Trigger In State Machine Table 20
  - State Machine Table
    - Add States 18
    - Add Substates 18
    - Add Triggers 19
    - Cell Color 15
    - Cell Enumeration 15
    - Cell Highlights 15
    - Cell Size 15
    - Change Position In Diagram View 18
    - Change Size 18
    - ChangeTransitions 19
    - Conventions 21
    - Description 14
    - Export To CSV 21
    - Format 14
    - Insert Transitions 19
    - Legend, Add 20
    - Legend, Remove 20
    - Locate State In State Machine Diagram 20
    - Locate Transition In State Machine Diagram 20
    - Locate Trigger In State Machine Diagram 20
    - Operations, Overview 17
    - Options 15
    - Remove Substate Parent Relation 18
    - Reposition States 20
    - Reposition Substates 20
    - Reposition Triggers 20
    - State-Next State 14
    - State-Trigger 14
    - Table Format 15
    - Trigger-State 14
  - State Region
    - Composite 12
  - State/Continuation
    - Element 132
  - Stereotype
    - Analysis 179
    - Apply To Dependency Relationship 206
    - Extension 178
    - Inbuilt 178
  - Stereotyped Element
    - Table 192
  - Structural Diagram
    - Elements 150
  - Structural Diagrams
    - Overview 54
  - StructuralFeature Action
    - Set Structural Feature 81
  - Structured Activity
    - Conditional Node 136, 139
    - Element 136, 138, 139
    - Loop Node 136, 139
    - Nested 136
    - Node 136, 138
    - Sequential Node 136, 138
  - Structured Activity Node 138
  - Sub-Activity
    - Conditional Node 136, 139
    - Element 90, 136, 138
    - Loop Node 136, 139
  - Submachine State
    - Element 129, 136
  - Substate
    - Reposition In State Machine Table 20
  - Sub-State 130
  - Synch
    - Element 142
  - System Boundary
    - Element 142
- T -**
- Table
    - Detail 192
    - Element 192
    - Properties 192
    - State Machine 14
  - Team Review
    - Hyperlink To 184
  - Template
    - Instantiated 154
    - Parameterized Classes 154
  - Terminate
    - Element 144
  - Test Case
    - Element 191
  - Time Event 127
  - Time Interval
    - Compress 32, 36
    - Context Menu 32
    - Copy and Paste 36
    - Create 32
    - Delete 32
    - Description 32
    - Move 32
    - Operations 36

- Time Interval
    - Resize 32
    - Select 32
    - Shift Left Or Right 36
    - Transitions 36
  - Time Range
    - Set For Timing Diagram 23
  - Timeline Element States
    - Add Via Configure Timeline Dialog 29
    - Delete via Configure Timeline Dialog 29
    - Edit Via Configure Timeline Dialog 29
    - Maintain 29
    - Numeric Range Generator 29
  - Timeline Range
    - Set For Timing Diagram 23
  - Timeline Start Position
    - Set For State Lifeline Element 24
  - Timing
    - Constraint 218
    - Details, Change 218
    - Message 226
    - Message, Create 227
    - Observation 218
  - Timing Diagram
    - Add Value Lifeline Element 27
    - Create 23
    - Description 22
    - Edit Options 24
    - Edit Value Lifeline Element 27
    - Elements And Connectors 22
    - Example 22
    - Set Time Range 23
  - Top Margin
    - Sequence Diagram, Change 48
  - Trace
    - Connector 236
    - Relationship 236
  - Traceability
    - In Requirements Models 71
  - Transition
    - Add To State Lifeline Elements 26
    - Add Via Configure Timeline Dialog 31
    - Change In State Machine Table 19
    - Change Time, State Lifeline Element 26
    - Connector 236
    - Delete On State Lifeline Element 26
    - Delete Via Configure Timeline Dialog 31
    - Edit In Time Intervals 36
    - Edit On State Lifeline Elements 26
    - Edit Via Configure Timeline Dialog 31
    - Effect 236
    - Guard 236
    - Highlight Associated Trigger or State 19
    - Insert In State Machine Table 19
    - Locate In State Machine Diagram 20
    - Locate In State Machine Table 20
    - Merge On State Lifeline Element 26
    - Move On State Lifeline Elements 26
    - Properties 236
    - Relationship 236
    - State Machine Table Conventions 21
    - Trigger 236
  - Trigger
    - Create 145
    - Create In State Machine Table 19
    - Create In Transition Properties 236
    - Element 145
    - For Transition 236
    - Locate In State Machine Diagram 20
    - Locate In State Machine Table 20
    - Ports 145
    - Properties Tab 145
    - Reposition In State Machine Table 20
    - State Machine Table Conventions 21
    - Type 236
- U -
- UI Control
    - Element 192
  - UML
    - Connectors 196
    - Definition 2
    - Dictionary 2
    - Elements 78
    - Extend 2
    - Recommended Reading 2
  - UML Behavioral Diagram
    - Overview 4
  - UML Diagram
    - Extended 4, 67
    - MDG Technology 4
    - Overview 4
    - Types 4
    - What Is A? 4
  - UML Element
    - Behavioral Diagram Elements 78
    - Structural Diagram Elements 150
  - UML Structural Diagram
    - Overview 54
  - UML Toolbox
    - Business Modeling Group 75

Unified Modeling Language 2

Usage

Connector 238

Relationship 238

Use

Connector 238

Relationship 238

Use Case

Element 146

Extension Points 147

Rectangle Notation 148

Scenarios 43

Use Case Diagram

Description 7

Elements And Connectors 7

Example 7

User Interface

Control Element 192

Element 192

Screen Prototype 190

User Interface Design 69

User Interface Diagram

Description 73

Elements And Connectors 73

Example 73

## - V -

Value Lifeline Element 149

Add States 28

Add To Timing Diagram 27

Add Transitions 28

Change Transition Time 28

Delete Transitions 28

Edit Transitions 28

Sizing and Scale 27

States 27

Transitions 27

Version Control

And Sequence Diagram 44

## - W -

Web

Page Modeling 194

Stereotypes 194

Web Page

Prototype 190

What Is

A Connector? 196

UML Dictionary  
[www.sparxsystems.com](http://www.sparxsystems.com)