

# **Rational Rhapsody Modeler Tutorial**



Before using the information in this manual, be sure to read the “Notices” section of the Help.

This edition applies to IBM® Rational® Rhapsody® 7.5 and to all subsequent releases and modifications until otherwise indicated in new editions.

© Copyright IBM Corporation 1997, 2009.

US Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

# Contents

---

<b>Getting Started</b> .....	<b>1</b>
<b>Modeler Tutorial Overview</b> .....	<b>2</b>
<b>Modeler Tutorial Objectives</b> .....	<b>3</b>
<b>Using Rational Rhapsody Modeler Edition</b> .....	<b>3</b>
<b>UML Diagrams</b> .....	<b>4</b>
<b>Tutorial Setup</b> .....	<b>5</b>
Creating the Handset Project .....	5
Creating Packages .....	6
<b>Summary</b> .....	<b>10</b>
<b>Module 1:</b>	
<b>Creating Use Case Diagrams</b> .....	<b>11</b>
<b>Goals of this Module</b> .....	<b>11</b>
<b>Exercise 1.1: Creating a Functional Overview UCD</b> .....	<b>12</b>
Drawing the Functional Overview UCD .....	14
Drawing a Boundary Box and Actors .....	15
Drawing the Use Cases .....	17
Entering a Use Case Description .....	19
Associating Actors with Use Cases .....	21
Drawing Generalizations .....	23
Adding Remarks to Model Elements and Diagrams .....	25
<b>Exercise 1.2: Creating the Place Call Overview UCD</b> .....	<b>27</b>
Drawing the Use Cases .....	29
Entering a Use Case Description .....	31
Drawing Generalizations .....	33
Adding Requirement Elements .....	34
<b>Exercise 1.3: Drawing the Data Call Requirements Diagram</b> .....	<b>44</b>
Creating the Data Call Requirements UCD .....	45
Adding Requirements .....	46
Drawing and Defining the Dependencies .....	48
<b>Summary</b> .....	<b>49</b>

<b>Module 2:</b>	
<b>Creating Structure Diagrams</b>	<b>51</b>
<b>Goals of this Module</b>	<b>51</b>
<b>Exercise 2.1: Creating a Block Diagram</b>	<b>52</b>
Drawing the Block Diagram	53
Drawing Objects	54
Drawing More Objects	57
Setting the Object Stereotype and Type	57
Drawing Ports	59
Specifying Port Attributes	61
Drawing Flows	61
Allocating the Functions Among Subsystems	74
<b>Exercise 2.2: Drawing the Connection Management Structure Diagram</b>	<b>77</b>
Creating the Connection Management Structure Diagram	78
<b>Exercise 2.3: Drawing the Data Link Diagram</b>	<b>84</b>
Creating the Data Link Diagram	84
<b>Exercise 2.4: Drawing the MM Architecture Diagram</b>	<b>89</b>
Creating the MM Architecture Diagram	89
<b>Summary</b>	<b>95</b>
<b>Module 3:</b>	
<b>Creating Object Model Diagrams</b>	<b>97</b>
<b>Goals of this Module</b>	<b>97</b>
<b>Exercise 3.1: Creating the Subsystem Architecture</b>	<b>97</b>
Drawing the Subsystem Architecture OMD	98
<b>Summary</b>	<b>102</b>
<b>Module 4:</b>	
<b>Creating Sequence Diagrams</b>	<b>103</b>
<b>Goals for this Module</b>	<b>103</b>
<b>Exercise 4.1: Creating the Place Call Request Successful SD</b>	<b>104</b>
Drawing the Place Call Request Successful Diagram	105
<b>Exercise 4.2: Drawing the NetworkConnect SD</b>	<b>113</b>
Moving Events	117
<b>Exercise 4.3: Drawing the Connection Management Place Call Request Success SD</b>	<b>118</b>
Creating the Connection Management Place Call Request Success SD	119
<b>Summary</b>	<b>123</b>

<b>Module 5:</b>	
<b>Creating Activity Diagrams</b>	<b>125</b>
Goals for this Module	125
Exercise 5.1: Creating the MCallControl Activity Diagram	126
Drawing the MCallControl Activity Diagram	127
Exercise 5.2: Drawing the InCall Subactivity Diagram	137
Opening the InCallSubactivity Diagram	138
Drawing Action Elements	138
Drawing a Default Connector	139
Drawing Transitions	139
Drawing a Timeout Transition	139
Exercise 5.3: Drawing the RegistrationMonitor Activity Diagram	140
Creating the RegistrationMonitor Activity Diagram	141
Summary	143
 <b>Module 6:</b>	
<b>Creating a Statechart</b>	<b>145</b>
Goals for this Module	145
Exercise 6.1: Creating the CallControl Statechart	146
Drawing the CallControl Statechart	147
Summary	150
 <b>Module 7:</b>	
<b>Generating a Report</b>	<b>151</b>
Producing an Internal Report	151
Using the Internal Report Output	153



# Getting Started

---

Welcome to the *IBM® Rational® Rhapsody® Modeler Edition* and *IBM® Rational® Rhapsody® Modeler Corporate Edition Tutorial*.

Rational Rhapsody is the visual programming environment (VPE) of choice for real-time, embedded software developers. Rational Rhapsody goes beyond defining requirements and designing a solution. It implements your solution from design diagrams, automatically generating ANSI-compliant code that is optimized for the most widely used real-time, embedded target environments. Rational Rhapsody is the industry leading Model-Driven Development environment based on UML 2.0 and SysML for systems, software, and test, and has the unique ability to extend its modeling environment to allow both functional and object oriented design methodologies in one environment.

Rational Rhapsody generates full production C++, C, Java, or Ada code for a variety of target platforms based on UML 2.0 behavioral and structural diagrams as well as providing reverse engineering of code for reuse of your intellectual property within a Model-Driven environment.

Rational Rhapsody Modeler Edition helps users develop easy to understand systems and software designs, including structure and intended behavior of a system. It helps users increase productivity and shorten design cycles.

## Modeler Tutorial Overview

This tutorial shows you how to use the Rational Rhapsody Modeler to analyze, design, and build a model of a wireless telephone using a file-based modeling approach. Before you begin creating this model, you need to consider the functions of the wireless telephone. Wireless telephony provides voice and data services to users placing and receiving calls. To deliver services, the wireless network must receive, set up, and direct incoming and outgoing call requests, track and maintain the location of users, and facilitate uninterrupted service when users move within and outside the network.

When the wireless user initiates a call, the network receives the request, and validates and registers the user; once registered, the network monitors the user's location. In order for the network to receive the call, the wireless telephone must send the minimum acceptable signal strength to the network. When the network receives a call, it directs it to the appropriate registered user.

For this tutorial, you create a project called Handset. Modeler contains a sample handset model that you can use to review the modules in this tutorial. The sample model is located in the directory:

```
<Rational Rhapsody installation folder>\Samples\Handset (Tutorial)
```

### Note

---

To minimize the complexity of the tutorial, the operations have been simplified to focus on the function of placing a call.



## Modeler Tutorial Objectives

At the end of this tutorial, you should be able to:

- ♦ Create a project
- ♦ Create a Use Case Diagram
- ♦ Create a Structure Diagram
- ♦ Create an Object Model Diagram
- ♦ Create a Sequence Diagram
- ♦ Create an Activity Diagram
- ♦ Create a Statechart

## Using Rational Rhapsody Modeler Edition

Rational Rhapsody Modeler is a visual design tool for developing object-oriented embedded software and performing structural and systems modeling. It enables you to perform the following tasks:

- ♦ **Analyze** - define, analyze, and validate the system requirements.
- ♦ **Design** - specify and design the architecture.
- ♦ **Implement** - automatically generate code, then build and run it within Rational Rhapsody. (Rational Rhapsody Developer edition only)
- ♦ **Model Execution** - animate the model on the local host or a remote target to perform design-level debugging within animated views. (Rational Rhapsody Developer edition only)
- ♦ **Documentation Generation** - create documentation reports containing diagrams and model data.
- ♦ **Domain Specific Modeling** - create domain specific language elements based on your model.

## UML Diagrams

The UML specification includes the following diagrams:

- ♦ **Use case diagrams** show the main functions of the system (use cases) and the entities (actors) outside the system.
- ♦ **Structure diagrams** show the system structure and identify the organizational pieces of the system.
- ♦ **Object model diagrams** show the structure of the system in terms of classes, objects, and blocks, and the relationships between these structural elements.
- ♦ **Sequence diagrams** show sequences of steps and messages passed between structural elements when executing a particular instance of a use case.
- ♦ **Activity diagrams** specify a flow for classifiers (classes, actors, use cases), objects, blocks, and operations.
- ♦ **Statecharts** show the behavior of a particular classifier (class, actor, use case), object, or block over its entire life cycle.
- ♦ **Collaboration diagrams** provide the same information as sequence diagrams, emphasizing structure rather than time.
- ♦ **Component diagrams** describe the organization of the software units and the dependencies among units.
- ♦ **Deployment diagrams** show the nodes in the final system architecture and the connections between them.

# Tutorial Setup

## Creating the Handset Project

This section describes how to start Rational Rhapsody Modeler and how to create the handset project and the packages.

### Starting Modeler

To start Modeler:

1. From the **Start** menu, click **Programs**, click **IBM Rational** > click **IBM Rational Rhapsody Modeler <version number>**, and then click **Rhapsody**.


**Note:** Modeler opens with the **Tip of the Day** dialog box. If you want to see another tip, click **Next Tip**. If you prefer not to see tips on startup, clear the **Show Tips on StartUp** check box.

2. Click **Close** to close the **Tip of the Day** dialog box.

### Creating a New Project Called Handset

A Modeler project includes the UML diagrams, packages, and code generation configurations that define the model. When you create a new project, Modeler creates a directory containing the *project files* in the specified location. The name you choose for your new project is used to name project files and directories, and appears at the top level of the project hierarchy in the Modeler browser. Modeler provides several default elements in the new project, including a default package, component, and configuration.

To create a new project:

1. From the **File** menu, click **New** or click the  icon. The **New Project** dialog box appears.
2. In the **Project Name** box, type “Handset.”
3. In the **In folder** box, type the path to the directory, or browse to find an existing directory, where the project files are placed.

**Note:** To avoid overwriting the handset sample project provided with Modeler, make sure you do not create your project in  
<Rational Rhapsody installation folder>\Samples\Handset  
(Tutorial)

4. In the **Product Type** box, click **Default**.
5. Click **OK**. The Modeler dialog box appears asking if you want to create a Handset directory. Click **Yes**.

Modeler creates a new project in the **Handset** subdirectory and opens the **Handset.rpy** project.

**Note:** If the **Browser Window** does not display, from the **View** menu, click **Browser**.

## Creating Packages

Packages are used to divide the model into functional domains or subsystems, which consist of objects, object types, functions, variables, and other logical artifacts. They are organized into hierarchies to provide a high level of partitioning.

The handset model has the following main packages:

- ♦ **RequirementsPkg** contains the functional requirements of the system
- ♦ **AnalysisPkg** contains the use case diagrams, which identify the requirements of the system
- ♦ **ArchitecturePkg** contains the sequence diagrams, identifying the high-level architecture and the structure diagram, detailing the design of the system model and the flow of information
- ♦ **SubsystemsPkg** contains the components of the system

### Note

To establish traceability between analysis and implementation, the **RequirementsPkg**, **AnalysisPkg**, and **ArchitecturePkg** packages can be referenced from the software implementation model (even if it is a different Modeler project) to establish traceability from design to analysis.

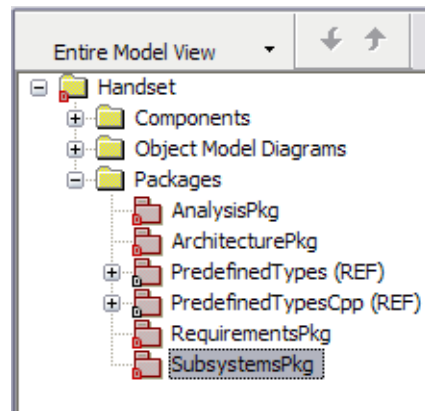
To create and organize the model into packages:

1. In the **Browser Window**, expand the **Packages** category.
2. Select the **Default** package and rename it **RequirementsPkg**.
  - a. To rename it, click on **Default** once and let the cursor hover over it until the cursor blinks.
  - b. Type the new name.
3. In the **Browser Window**, right-click **Packages** and click **Add New Package**. A package, with the default name **package\_n**, where **n** is equal to or greater than 0, is created.
4. Rename **package\_n** to **AnalysisPkg**.

5. Repeat Steps 3 and 4 to create the following packages:

- ♦ **ArchitecturePkg**
- ♦ **SubsystemsPkg**

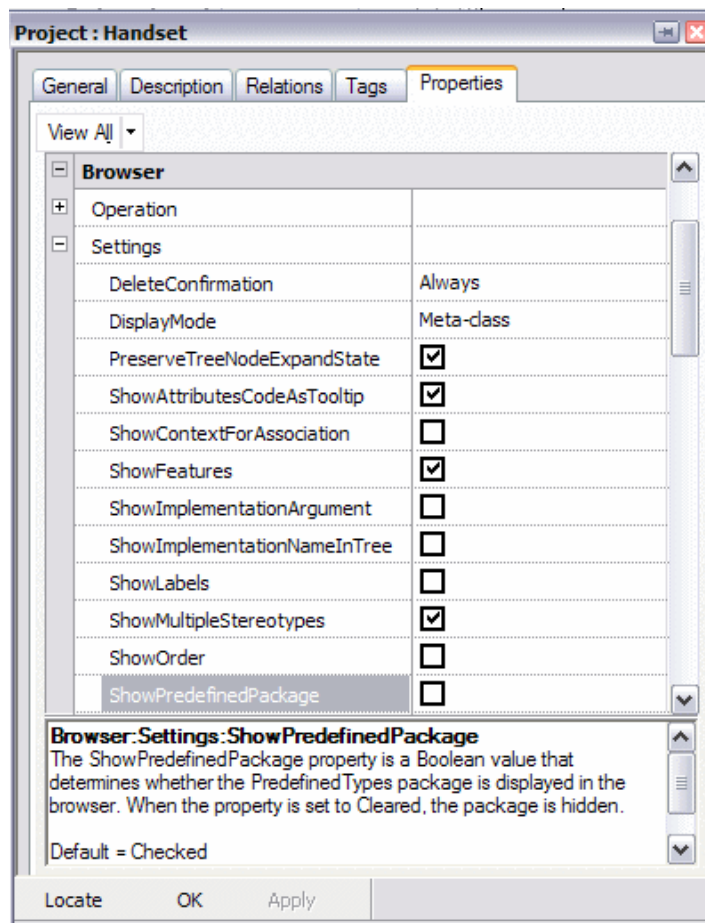
The **Packages** category should resemble this example:



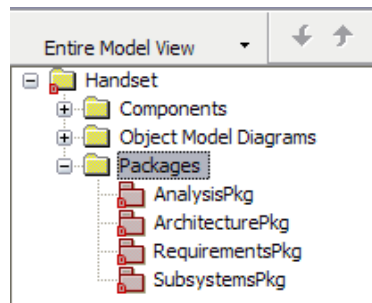
## Hiding Predefined Packages

To hide the predefined packages:

1. From the **File** menu, click **Project Properties** and the **Features** dialog box displays.
2. Click the **Properties** tab, click the **View Common** arrow, and then click **All**.
3. Scroll down to the **Browser** properties and click **+** to expand it.
4. Click the **Settings** **+** to expand it.
5. Clear the **ShowPredefinedPackage** check box, as shown in this example.



6. Click **OK** to hide the predefined packages, as shown below.



## Summary

In this section, you have performed the following:

- ♦ Created and saved the handset project
- ♦ Created and organized the handset model using packages
- ♦ Used the Features dialog box to set a project property

You are now ready to proceed to the modules where you create the handset model. In the next section, you model the requirements of the wireless telephone and the functions of placing a call using use case diagrams.

For ease of presentation, this guide organizes the sections by diagram type and general workflow. However, when modeling systems, diagrams are often created in parallel or may require elements in one diagram to be planned or designed before another diagram can be finalized. For example, you might identify the communication scenarios using sequence diagrams before defining the flows, flow items, and port contracts in the structure diagrams. In addition, you might perform black-box analysis using activity diagrams, sequence diagrams, and statecharts, and white-box analysis using sequence diagrams before decomposing the functions for the system into subsystem components.



# Module 1:

## Creating Use Case Diagrams

---

**Use Case Diagrams** (UCDs) model relationships between one or more users (actors) and a system or class (classifier). They enable you to specify requirements for system behavior.

### Goals of this Module

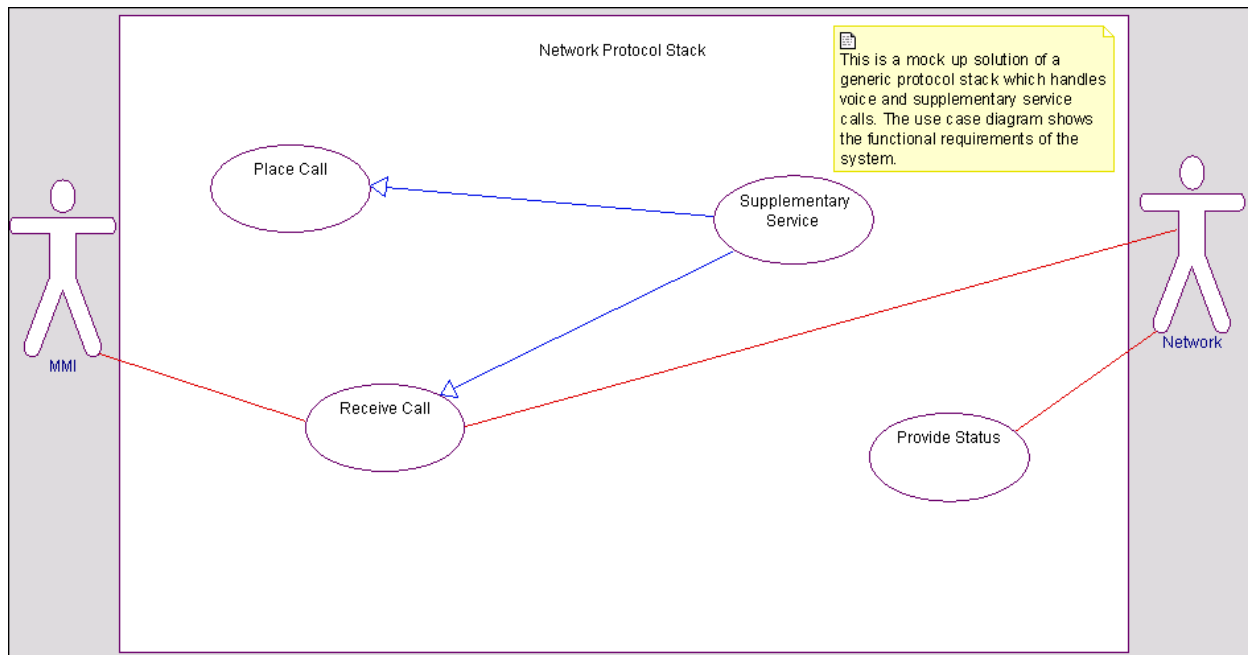
In this module, you create the following UCDs:

- ♦ **Functional Overview** shows the requirements and functions of the handset
- ♦ **Place Call Overview** shows the functions of placing a call
- ♦ **Data Call Requirements** shows the relations among requirement elements

## Exercise 1.1: Creating a Functional Overview UCD

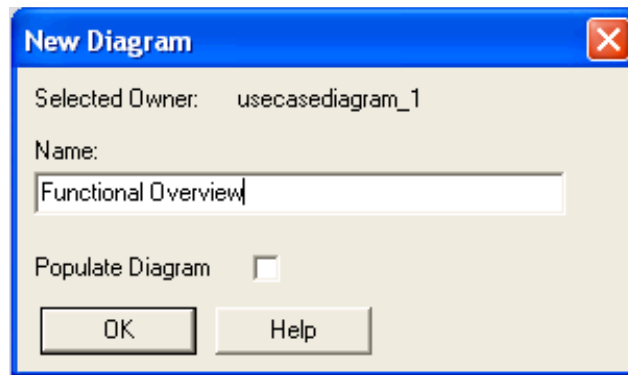
The Functional Overview UCD shows the system requirements, including the actors, the major function points of the system, and the relationships between them.

In this exercise, you develop a Functional Overview UCD, as shown in the following illustration.

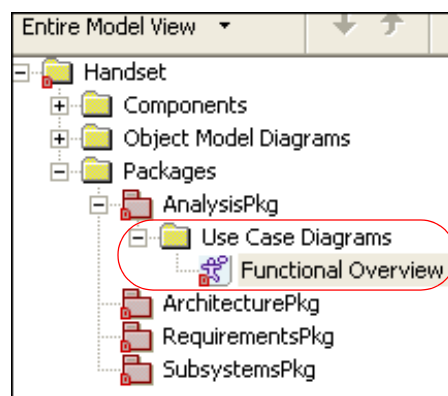


To create the Functional Overview UCD:

1. In the **Browser Window**, right-click **AnalysisPkg**, click **Add New > Diagrams**, and then click **Use Case Diagram**. The **New Diagram** dialog box appears.



2. Type "Functional Overview" and then click **OK**.



Modeler automatically adds the **Use Case Diagrams** category and the **Functional Overview** UCD to the **Browser Window**.

## Drawing the Functional Overview UCD

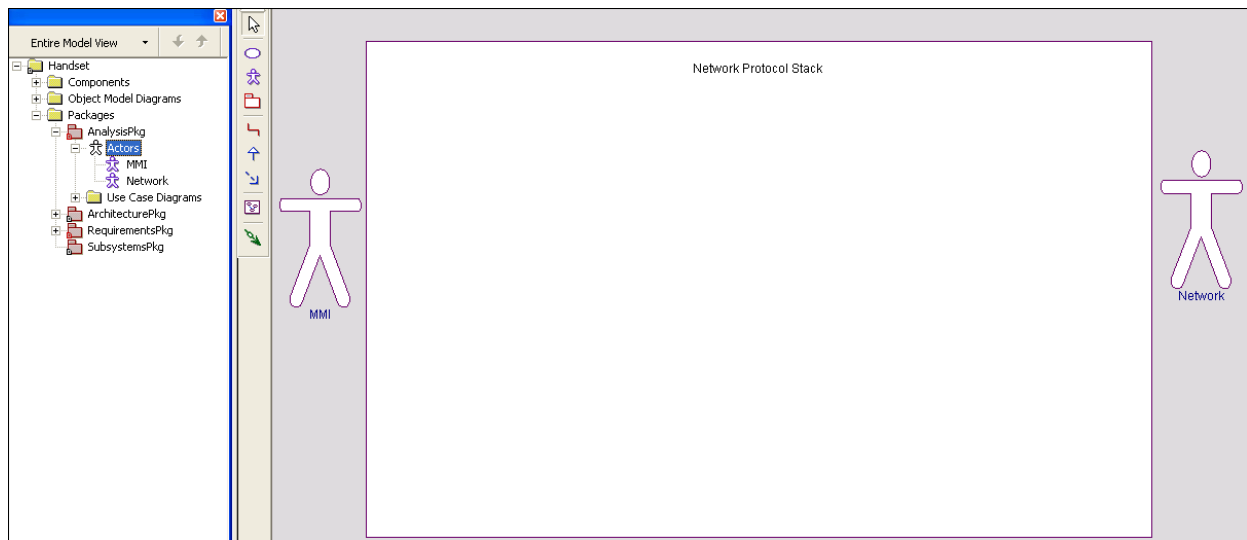
The following are the system requirements, including the actors, the major function points of the system, and the relationships between them, for this procedure:

- ♦ The actors that interact with the system are:
  - **MMI** - The handset user interface including the keypad and display
  - **Network** - The system network or infrastructure of the signalling technology
- ♦ The system function points are:
  - The handset enables users to place and receive calls
  - The network receives incoming and outgoing call requests and tracks users
- ♦ The actors interact with the system in the following ways:
  - The MMI places and receives calls
  - The network tracks users, monitors signal strength, and provides network status and location registration



## Drawing a Boundary Box and Actors

The boundary box defines the system under design from the external actors. Use cases are inside the boundary box and actors are outside the boundary box.

In this procedure, you draw a boundary box and actors, as shown in the following illustration.



To draw a boundary box and actors:

1. Click the **Create Boundary box** icon  in the Diagram Tools.
2. Click in the upper, left corner of the drawing area and drag to the lower right. A boundary box called **System Boundary Box** is created.
3. Rename the boundary box to **Handset Protocol Stack** by typing “Handset Protocol Stack” and pressing **Enter**.
4. Click the **Create Actor** icon  in the Diagram Tools.
5. Click on the left side (outside) of the **Handset Protocol Stack** boundary box. An actor with a default name of **actor<sub>n</sub>**, where **n** is equal to or greater than 0, is created.

6. Rename the actor to **MMI** by typing “MMI” and pressing **Enter**.

**Note:** Do not include spaces in the names of the actors. Use an underscore instead.

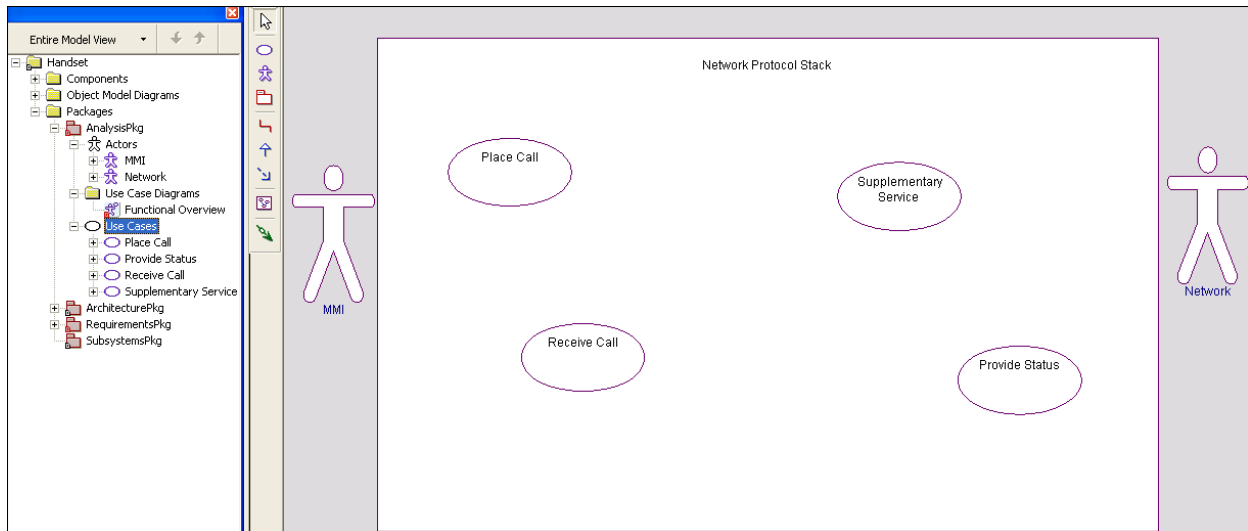
7. Draw another actor called **Network** on the right side of the **Handset Protocol Stack** boundary box.
8. In the **Browser Window**, expand the **AnalysisPkg** package and the **Actors** category to view the newly created actors.

## Drawing the Use Cases


A **Use Case** represents a particular function of the system.



In this procedure, you draw the following use cases, as shown in the following illustration:

- ◆ **Place Call** - The user can place various types of calls.
- ◆ **Supplementary Service** - The system can provide services, such as messaging, call forwarding, call holding, call barring, and conference calling.
- ◆ **Receive Call** - The system can receive various types of calls.
- ◆ **Provide Status** - The system can provide network status, user location, and signal strength.



To draw the use cases:

1. Click the **Create Use Case** icon  in the Diagram Tools.
2. Click in the boundary box. A use case with a default name of **usecase\_***n*, where *n* is equal to or greater than 0, is created.
3. Rename the use case to **Place Call** by typing “Place Call” and pressing **Enter**.

**Note:** To resize the use case, click **Select** , click the use case element, place the cursor over one of the small boxes , and then drag the element to the desired size.

4. Repeat Steps 2 and 3 to create the following use cases:
  - ♦ **Supplementary Service**
  - ♦ **Receive Call**
  - ♦ **Provide Status**
5. In the **Browser Window**, expand the **AnalysisPkg** package and the **Use Cases** category to view the newly created use cases.

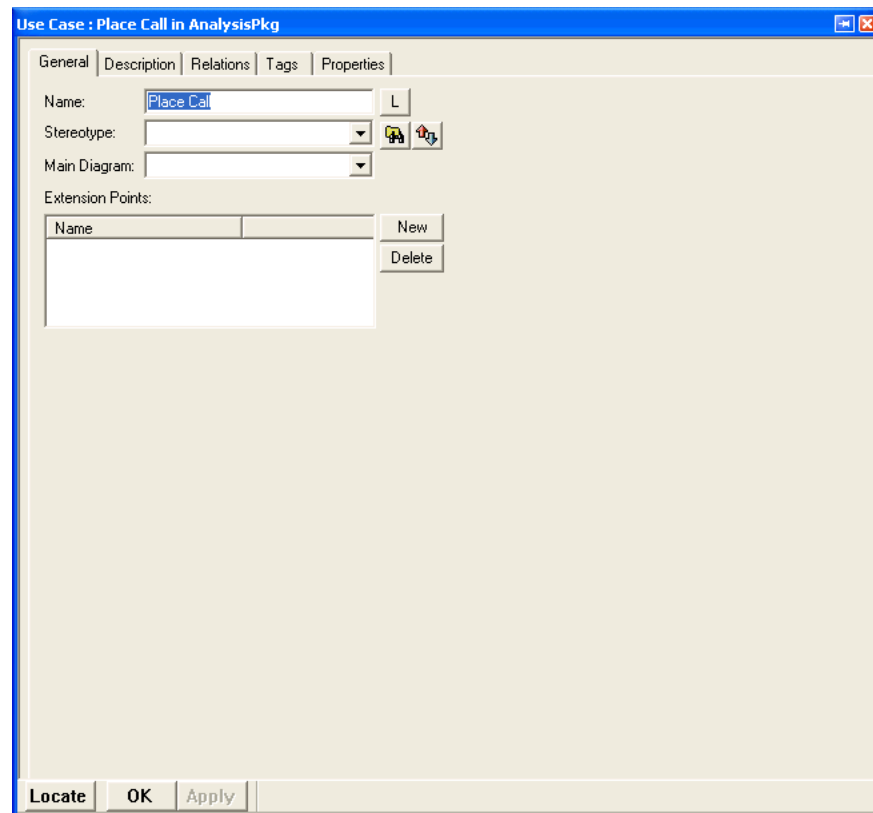


## Entering a Use Case Description


In this procedure, you enter a description for each use case.

To enter a use case description:

1. In the **Browser Window**, expand the **AnalysisPkg** package and **Use Cases** category, right-click **Place Call**, and click **Features**. The **Features** dialog box appears.



2. Click the **Description** tab and type the following text: “General function of the system is that it must be able to place various types of calls.”

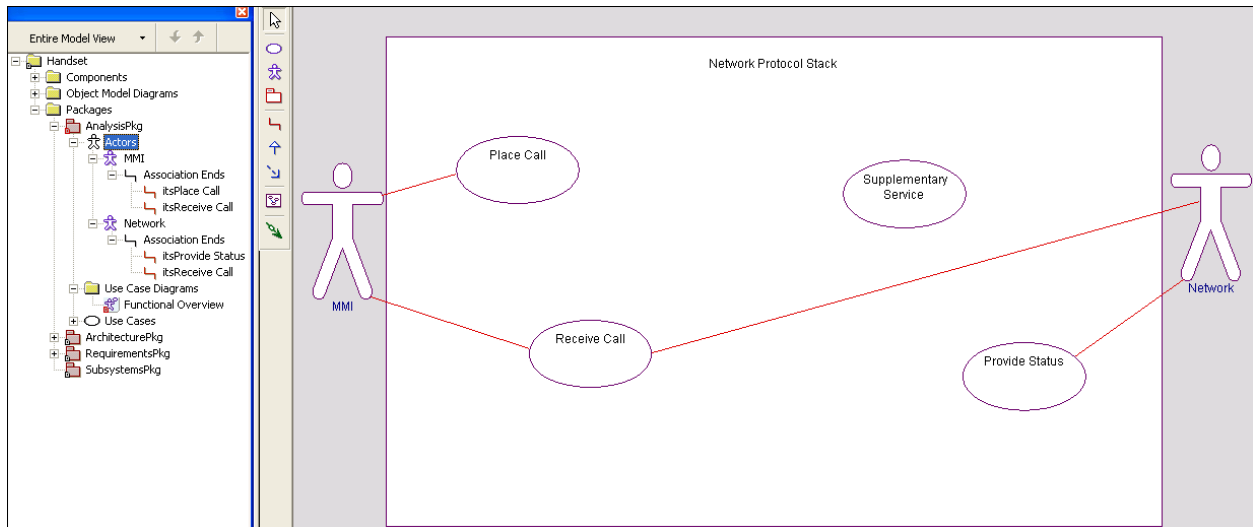
**Note:** Click the ellipsis icon  next to the **Description** box to use the internal text editor.

3. Click **OK** to apply the changes and to close the **Features** dialog box.
4. Repeat Steps 1 through 3 for the rest of the use cases:
  - ♦ **Supplementary Service Description** – “A supplementary service is a short message, call forwarding, call holding, call barring, or conference calling.”
  - ♦ **Receive Call Description** - “General function of the system is that it must be able to receive and terminate calls.”
  - ♦ **Provide Status Description** - “The stack must be able to communicate with the network in order to show the user the visual status such as signal strength and current registered network. It must also handle user requests for network status and location registration.”
5. Click **OK** to apply the changes and to close the **Features** dialog box.


## Associating Actors with Use Cases

The **MMI** actor places calls and receives calls. The **Network** actor notifies the system of incoming calls and provides status. An **Association** represents a connection between objects or users.

In this procedure, you show the associations between actors and the relevant use cases, as shown in the following illustration.



To draw association lines:

1. Click the **Create Association** icon  in the Diagram Tools.
2. Click the edge of the **MMI** actor and then click the edge of the **Place Call** use case. An association line with the name label highlighted is created. Press **Enter**.
3. Repeat Steps 1 and 2 to create association lines between:
  - ♦ The **MMI** actor and the **Receive Call** use case.
  - ♦ The **Network** actor and the **Receive Call** use case.
  - ♦ The **Network** actor and the **Provide Status** use case.
4. In the **Browser Window**, expand the **AnalysisPkg** package and the **Actors** category to view the newly created associations.

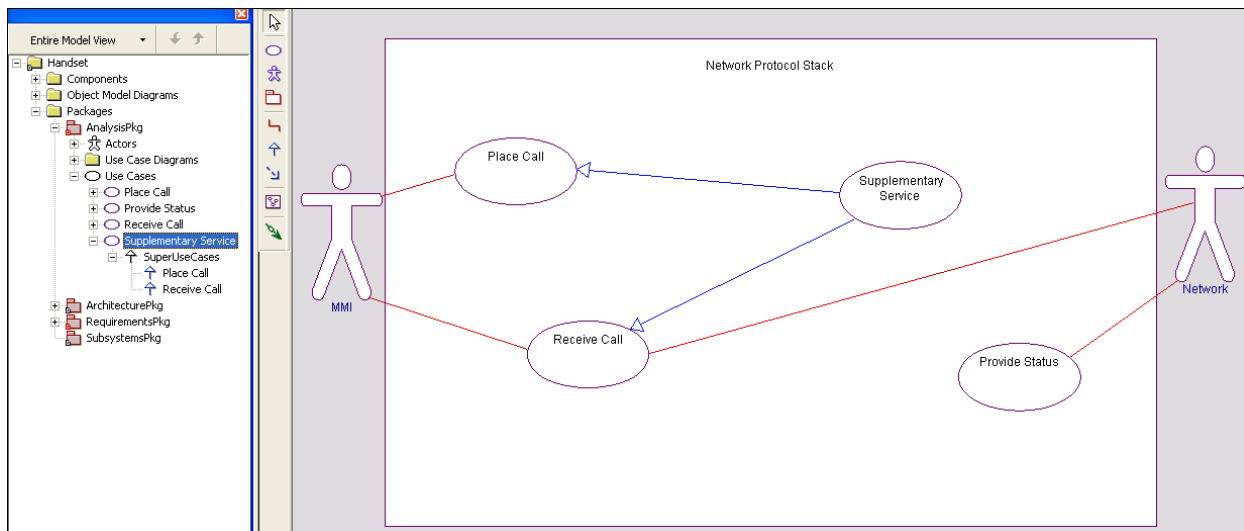
The **MMI** actor has two new relations:

- ♦ **itsPlace Call** - The role played by the **Place Call** use case in relation to this actor
- ♦ **itsReceive Call** - The role played by the **Receive Call** use case in relation to this actor
- ♦ The **Network** actor has two new relations:
  - ♦ **itsProvide Status** - The role played by the **Provide Status** use case in relation to this actor
  - ♦ **itsReceive Call** - The role played by the **Receive Call** use case in relation to this actor



## Drawing Generalizations

A **Generalization** is a relationship between a general element and a more specific element. The more specific element inherits the properties of the general element and is substitutable for the general element. A generalization lets you derive one use case from another.

In this procedure, you draw generalizations indicating that **Supplementary Service** use case (a more specific case of placing and receiving a call) is derived from the **Place Call** use case and the **Receive Call** use case, as shown in the following illustration.



To draw a generalization:

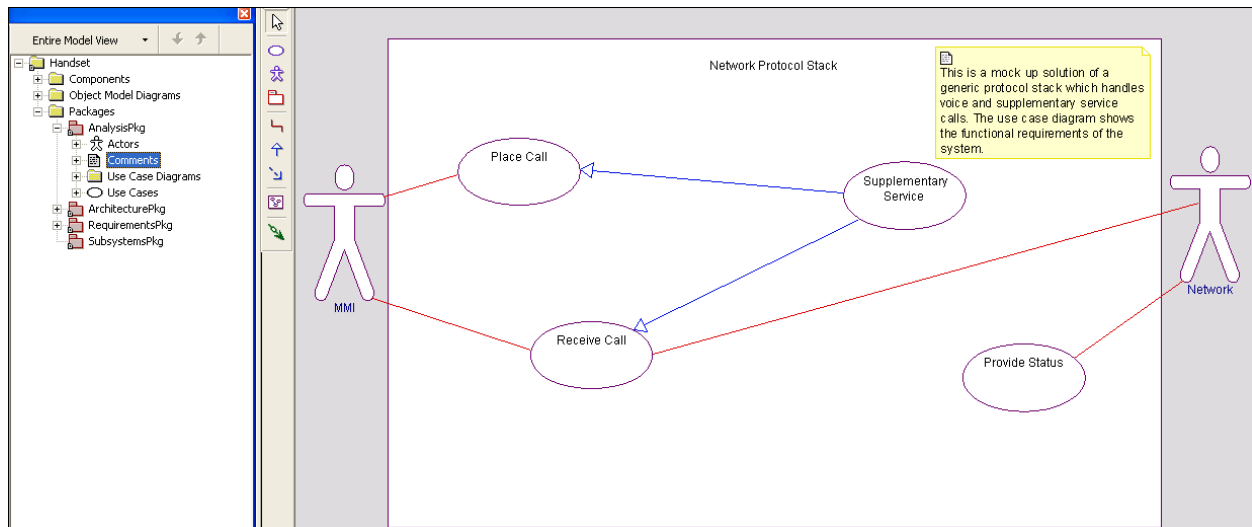
1. Click the **Create Generalization** icon  in the Diagram Tools.
2. Click the **Supplementary Service** use case and draw a line to the **Place Call** use case.
3. Click the **Create Generalization** icon  in the Diagram Tools.
4. Click the **Supplementary Service** use case and draw a line to the **Receive Call** use case.
5. In the **Browser Window**, expand the **AnalysisPkg** package, the **Use Cases** category, and then the **Supplementary Service** use case to view the newly created generalizations.

**Note:** **Place Call** and **Receive Call** are **SuperUseCases** for the **Supplementary Service** use case.

## Adding Remarks to Model Elements and Diagrams

You can add remarks to specify additional information about a model element or diagram.

In this procedure, you add a comment to the Functional Overview UCD, as shown in the following illustration.



To add a remark:

1. Click the **Comment** icon  in the Common Diagram Tools.

**Note:** If the Common Diagram Tools is not open, from the **View** menu, click **Toolbars**, and then click **Common Drawing**.

2. Click in the top section of the diagram.
3. Type the following text:

“This is a mock up solution of a generic protocol stack which handles voice and supplementary service calls. The use case diagram shows the functional requirements of the system.”

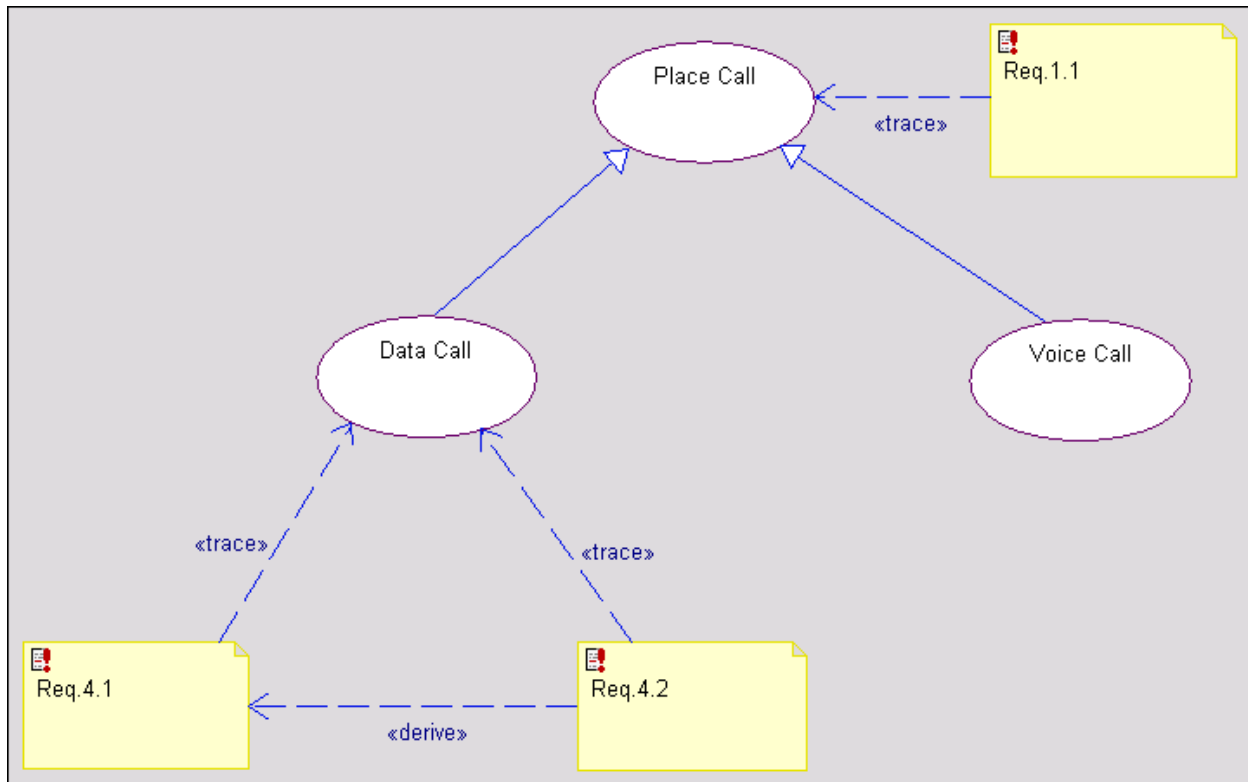
4. In the **Browser Window**, expand the **AnalysisPkg** package and the **Comments** category to view the newly added comment.



## Exercise 1.2: Creating the Place Call Overview UCD

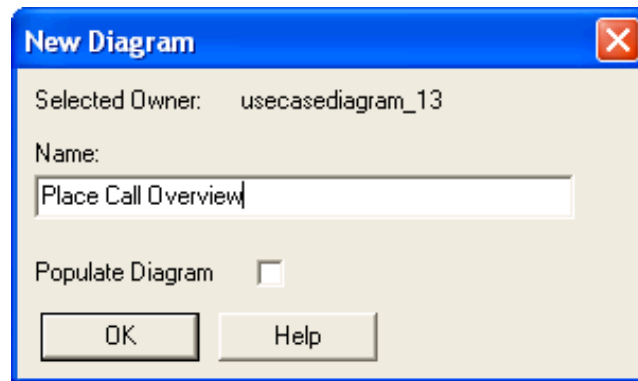
The Place Call Overview UCD breaks down the Place Call use case and identifies the different types of calls that can be placed as use cases.

In this exercise, you develop the Place Call Overview UCD, as shown in the following illustration.

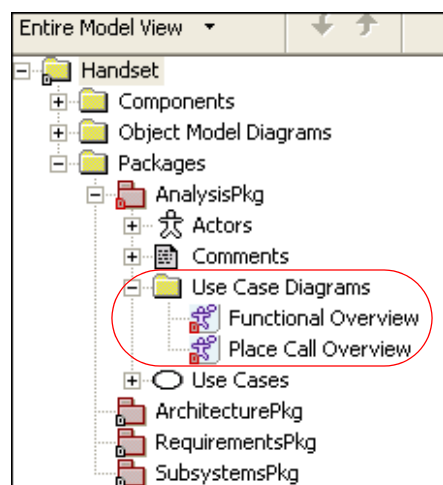


To create the Place Call Overview UCD:

1. In the **Browser Window**, expand **AnalysisPkg** package, right-click **Use Case Diagrams**, and then click **Add New Use Diagram**. The **New Diagram** dialog box appears.



2. Type "Place Call Overview" and then click **OK**.

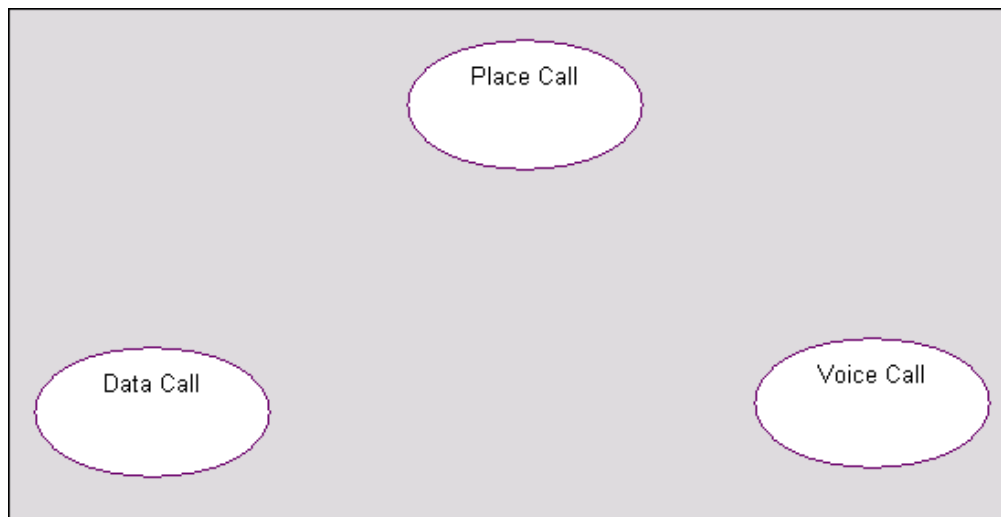


Modeler automatically adds the **Place Call Overview** UCD to **Use Case Diagrams** category in the **Browser Window**.


## Drawing the Use Cases



In this procedure, you draw the following uses cases, as shown in the following illustration:

- ♦ **Place Call** - The user can place various types of calls.  
**Note:** You defined the Place Call use case in the Functional Overview UCD.
- ♦ **Data Call** - The user can originate and receive data requests. It is a more specific case of placing a call.
- ♦ **Voice Call** - The user can place and receive voice calls, either while transmitting or receiving data, or standalone. It is a more specific case of placing a call.



To draw the use cases:

1. In the **Browser Window**, expand the **AnalysisPkg** and **Use Cases** category.
2. Click the **Place Call** use case and drag it to the top center of the drawing area.
3. Click the **Create Use Case** icon  in the Diagram Tools.
4. Click in the drawing area. A use case with a default name of **usecase\_***n*, where *n* is equal to or greater than 0, is created.
5. Rename the use case to **Data Call** by typing “Data Call” and pressing **Enter**.

**Note:** To resize the use case, click **Select** , click the use case element, place the cursor over one of the small boxes , and then drag the element to the desired size.

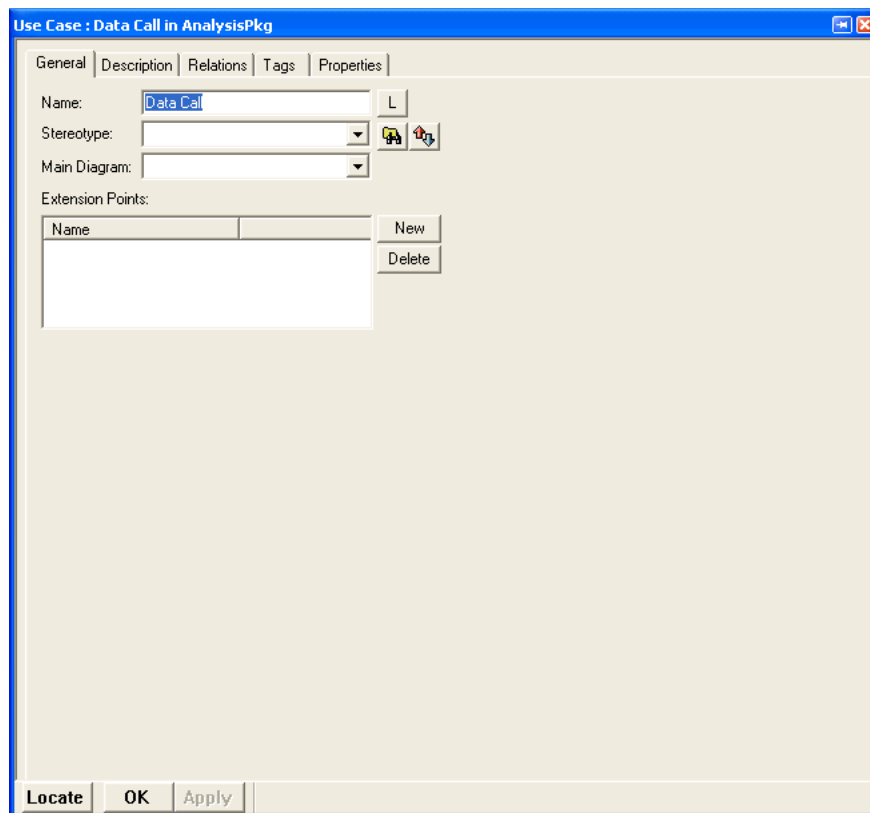
6. Repeat Steps 3 through 5 to create a **Voice Call** use case.
7. In the **Browser Window**, expand the **AnalysisPkg** package and the **Use Cases** category to view the newly created use cases.

## Entering a Use Case Description

In this procedure, you enter a description for each use case.


To enter a use case description:

1. In the **Browser Window**, expand the **AnalysisPkg** package and **Use Cases** category, right-click **Data Call**, and click **Features**. The **Features** dialog box appears.



2. Click the **Description** tab and type the following text:

“The stack must be able to originate and receive data requests of up to 384 kbps. Data calls can be originated or terminated while active voice calls are in progress.”

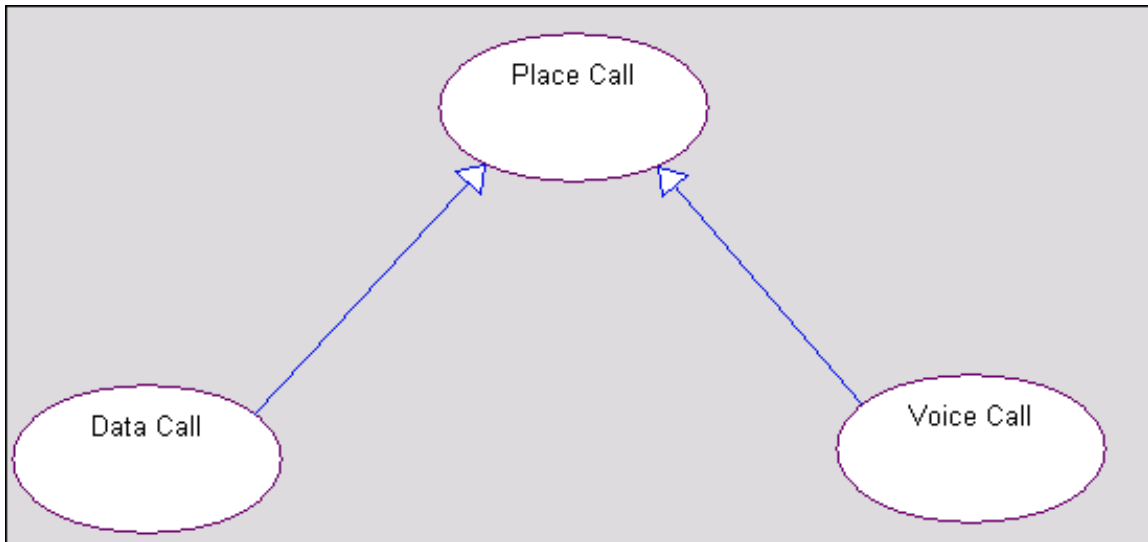
**Note:** Click the ellipsis icon  next to the **Description** box to use the internal text editor.

3. Click **OK** to apply the changes and to close the **Features** dialog box.
4. Repeat Steps 1 through 3 for the **Voice Call** use cases. Type the following text:



“The user must be able to place or receive voice calls, either while transmitting or receiving data, or standalone. The limit of the voice calls a user can engage in at once is dictated by the conference call supplementary service.”
5. Click **OK** to apply the changes and to close the **Features** dialog box.

## Drawing Generalizations

In this procedure, you draw generalizations to show that the **Data Call** and the **Voice Call** use cases derive from the **Place Call**, as shown in the following illustration.



To draw a generalization:

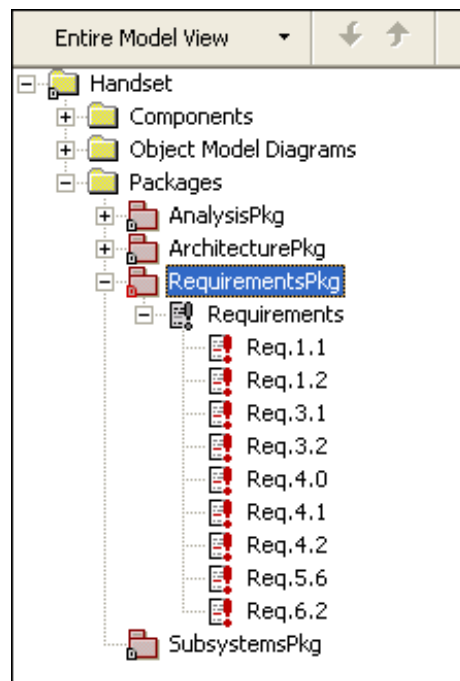
1. Click the **Create Generalization** icon  in the Diagram Tools.
2. Click the **Data Call** use case and draw a line to the **Place Call** use case.
3. Click the **Create Generalization** icon  in the Diagram Tools.
4. Click the **Voice Call** use case and draw a line to the **Place Call** use case.

## Adding Requirement Elements

You can represent requirements in the browser window and diagrams as requirement elements. Requirement elements are textual annotations that describe the intent of the element.

### Adding Requirement Elements to the Model

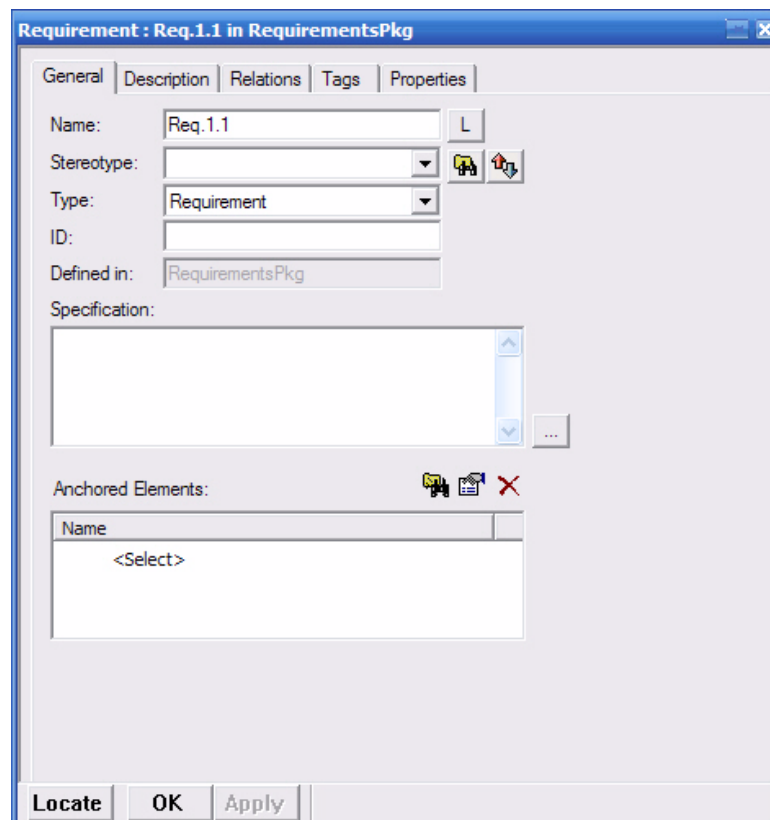
In this procedure, you add the handset model requirements to the **RequirementsPkg** package in the **Browser Window**, as shown in the following illustration.





To add requirements elements:

1. In the **Browser Window**, right-click the **RequirementsPkg** package, click **Add New**, and then click **Requirement**. A requirement with a default name of **requirement\_*n***, where *n* is equal to or greater than 0, is created.
2. Rename the use case to **Req.1.1** by typing “Req.1.1” and pressing **Enter**.
3. Right-click **Req.1.1** and then click **Features**. The **Features** dialog box appears.



4. Click the **Description** tab and type the following text: “The mobile shall be fully registered before a place call sequence can begin.”
5. Click **OK** to apply the changes and to close the **Features** dialog box.
6. Repeat Steps 1 through 5 to create the following requirements and descriptions:
  - ♦ **Req.1.2** - “The mobile shall have a signal strength within +/- 1 of the minimum acceptable signal.”
  - ♦ **Req.3.1** - “The mobile shall be able to place short messages while registered.”

- ♦ **Req.3.2** - “The mobile shall be able to receive short messages while registered.”
  - ♦ **Req.4.0** - “The mobile shall be able to receive data calls at the rate of 128 kbps.”
  - ♦ **Req.4.1** - “The mobile shall be able to send data at the rate of 384 kbps.”
  - ♦ **Req.4.2** - “The mobile shall be able to receive streaming video at 384 kbps.”
  - ♦ **Req.5.6** - “The mobile shall be able to receive a maximum of 356 characters in a short message.”
  - ♦ **Req.6.2** - “The optimal size of messages the mobile can send in a text message is 356 characters.”
7. In the **Browser Window**, expand the **RequirementsPkg** and the **Requirements** category to view the newly created requirements elements.

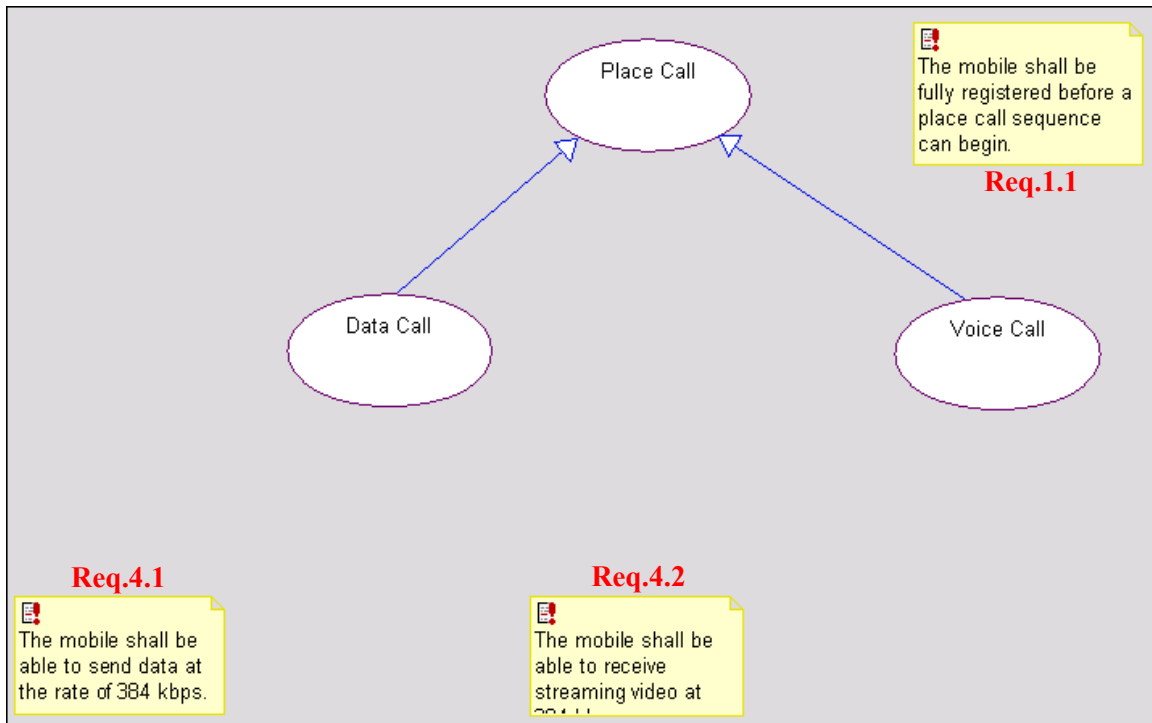
### Note

---

Modeling requirement elements in Rational Rhapsody Developer enables you to provide requirements traceability without a Requirements Management (RM) tool. Requirements traceability is the ability to describe and follow the life of a requirement, in both a forward and backward direction. It supports requirements verification and validation, prevents the introduction of unspecified features, and provides visibility to derived requirements that need to be specified and tested.

## Adding Requirement Elements to a Diagram

In this procedure, you add requirement elements to the Place Call Overview UCD to show the requirements trace to the use cases.

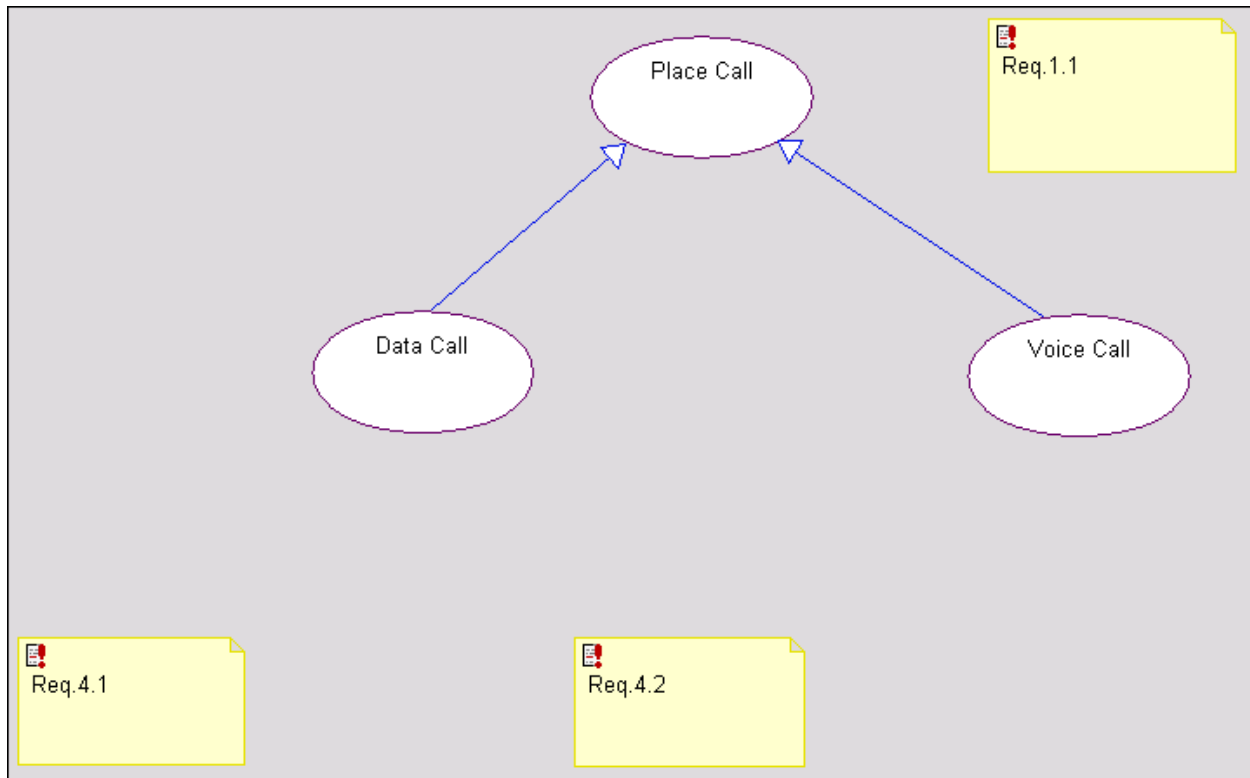


To add the requirement elements to the Place Call Overview UCD:

1. In the **Browser Window**, expand the **RequirementsPkg** package and the **Requirements** category.
2. Click **Req.1.1** and drag it to the right of the **Place Call** use case.
3. Select **Req.4.1** and drag it to the lower left of the **Data Call** use case.
4. Select **Req.4.2** and drag it to the lower right of the **Data Call** use case.

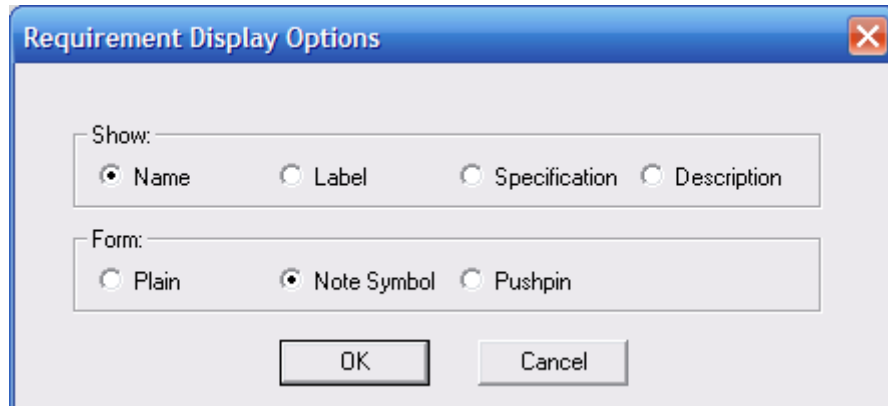
### Setting the Display Option to Show the Requirement the Element Name

In this procedure, you set the display option to show the requirement the names of the elements, as shown in the following illustration.



To set the display option to show the requirement element name:

1. Right-click **Req.1.1** in the diagram and click **Display Options**. The **Requirement Display Options** dialog box appears.

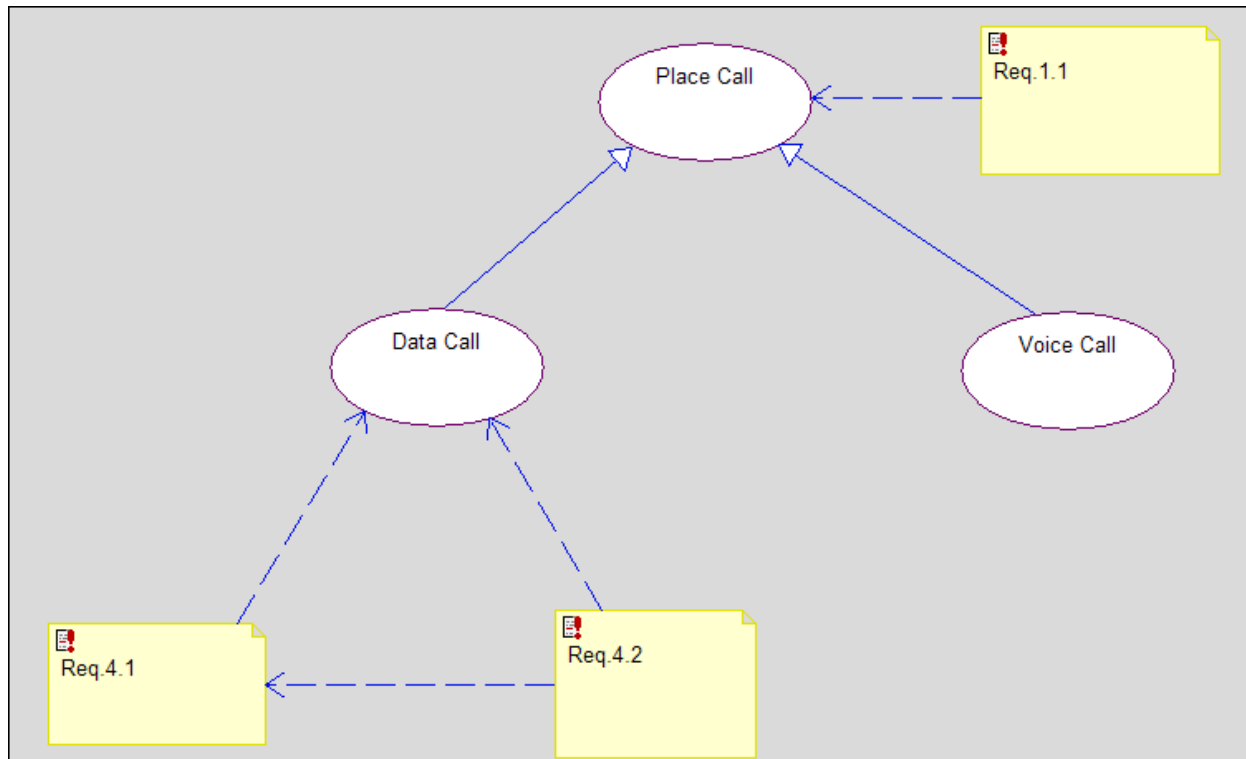


2. Under **Show**, click **Name** to display the name of the requirement.
3. Click **OK**.
4. Repeat Steps 1 through 3 for **Req.4.1** and **Req.4.2** to **Name**.


### Drawing Dependencies

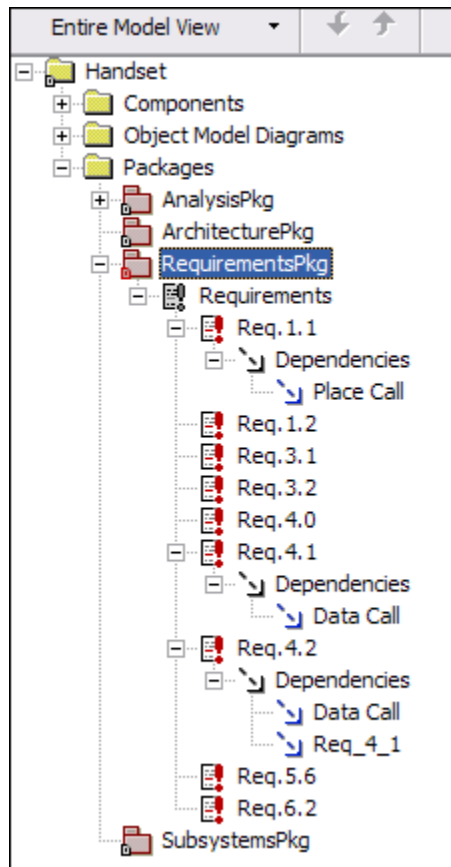
A dependency is a direct relationship in which the function of an element requires the presence of and may change another element. You can show the relationship between requirements, and between requirements and model elements using dependencies.

In this procedure, you draw dependencies between the requirement elements and the use cases, as shown in the following illustration.



To draw dependencies:

1. Click the **Dependency** icon  in the Diagram Tools.
2. Click the **Req.1.1** requirement and draw a line to the **Place Call** use case.
3. Click the **Req.4.1** requirement and draw a line to the **Data Call** use case.
4. Click the **Req.4.2** requirement and draw a line to the **Data Call** use case.
5. Click the **Req.4.2** requirement and draw a line to **Req.4.1**.
6. In the **Browser Window**, expand the **RequirementsPkg** package and the **Requirements** category to view the dependency relationships.

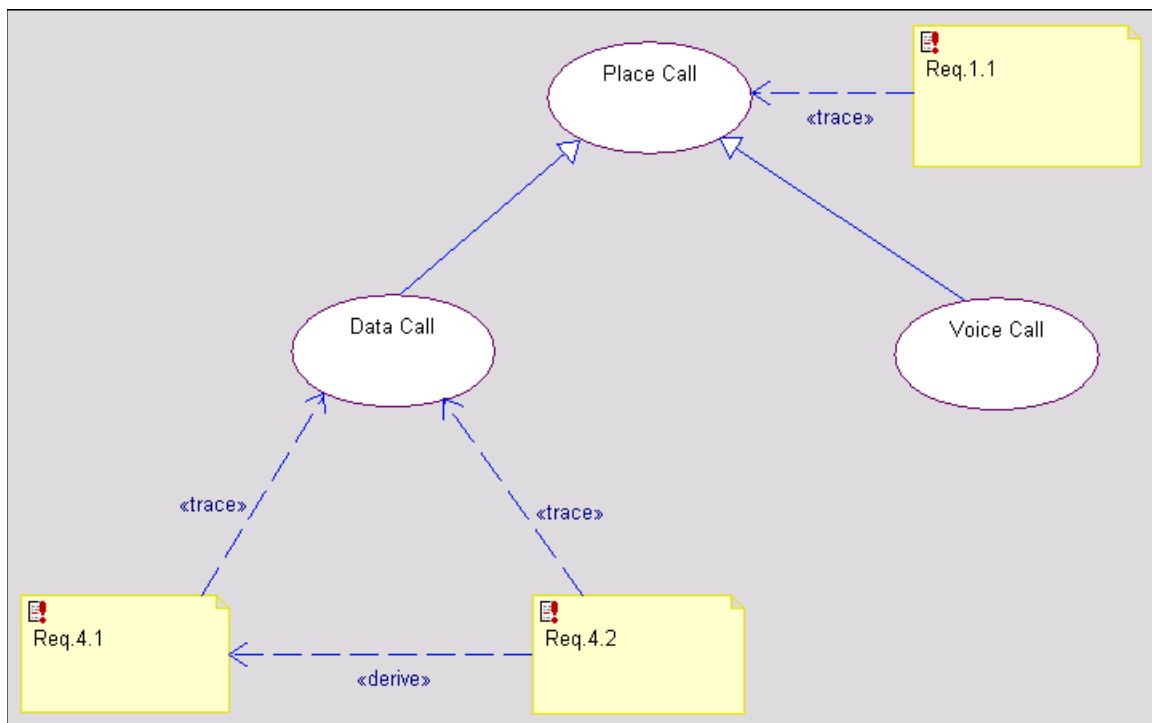


### Defining the Stereotype of a Dependency

You can specify the ways in which requirements relate to other requirements and model elements using stereotypes. A stereotype is a modeling element that extends the semantics of the UML metamodel by typing UML entities. Modeler includes predefined stereotypes and you can define your own stereotypes. Stereotypes are enclosed in guillemets on diagrams, for example, «derive».

In this procedure, you set the following types of dependency stereotypes, as shown in the following illustration:

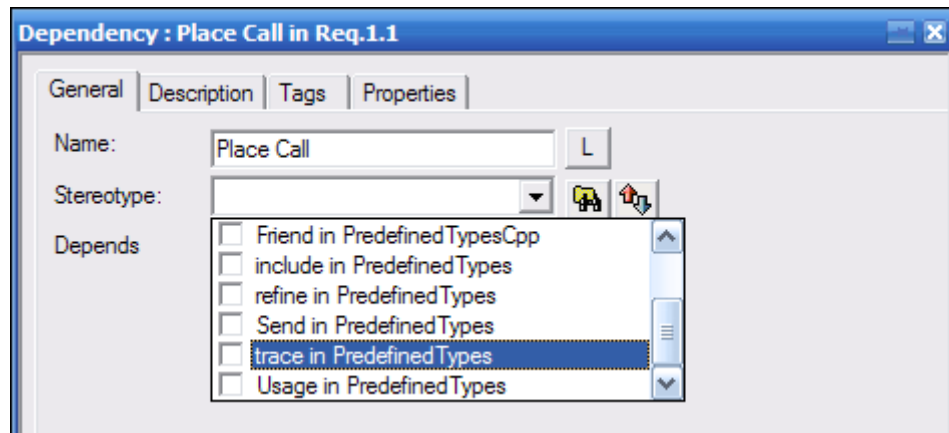
- ◆ **Derive** - A requirement is a consequence of another requirement.
- ◆ **Trace** - A requirement traces to an element that realizes it.



To define the stereotype of a dependency:

1. Double-click the dependency between **Req.1.1** and **Place Call**. The **Features** dialog box appears.
2. In the **Stereotype** list, select the **trace in PredefinedTypes** check box.



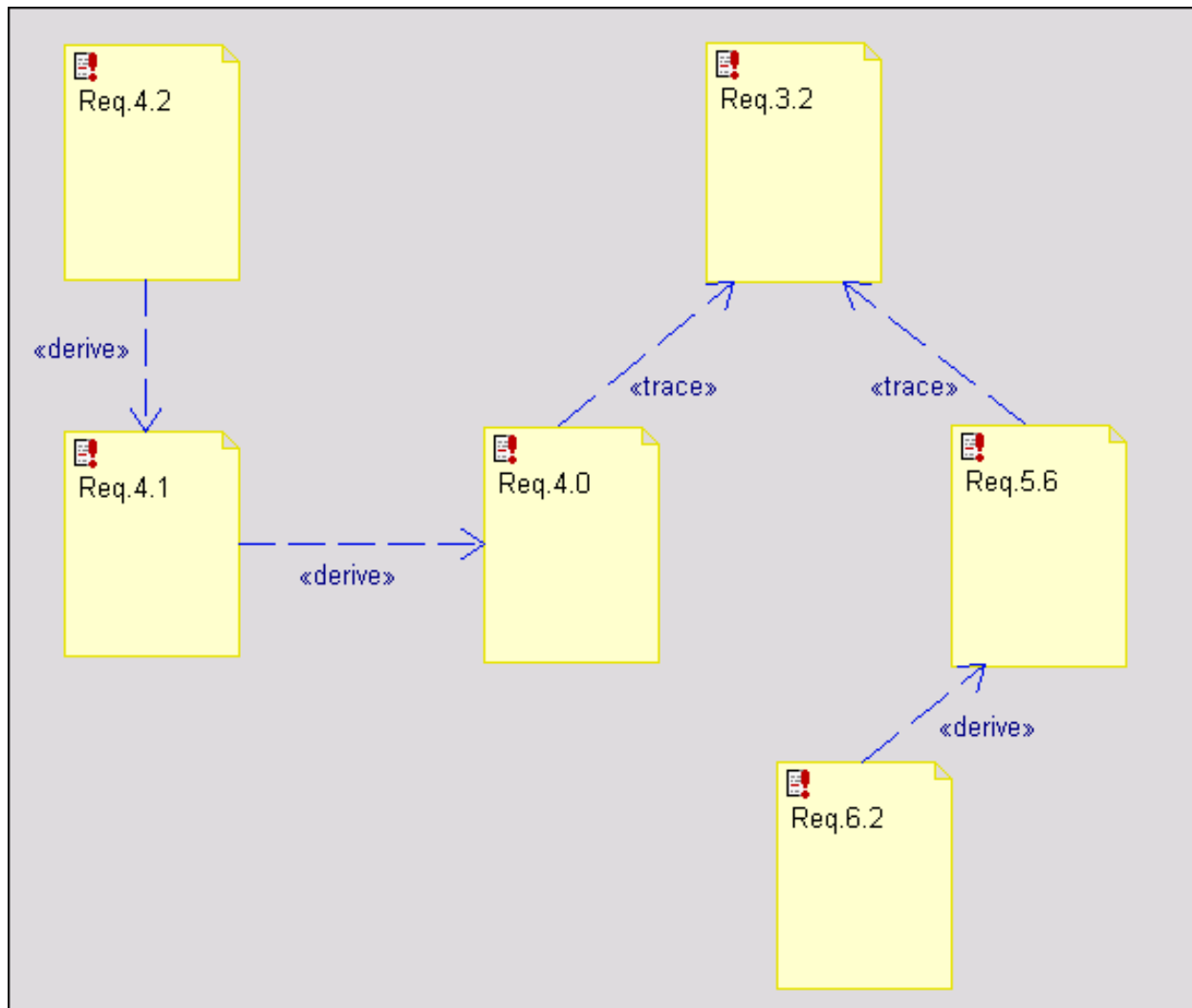


3. Click **OK** to apply the changes and to close the **Features** dialog box.
4. Repeat Steps 1 through 3 for:
  - ♦ The dependency between **Req.4.1** and **Data Call** - Set to **trace in PredefinedTypes**.
  - ♦ The dependency between **Req.4.2** and **Data Call** - Set to **trace in PredefinedTypes**.
  - ♦ The dependency between **Req.4.1** and **Req.4.2** - Set to **drive in PredefinedTypes**.

## Exercise 1.3: Drawing the Data Call Requirements Diagram

The Data Call Requirements UCD graphically shows the relationship among textual requirement elements for sending and receiving data calls.

In this exercise, you develop a Data Call Requirements UCD, as shown in the following illustration.



## Creating the Data Call Requirements UCD

The Data Call Requirements UCD contains only requirements. You create the requirements in the **RequirementsPkg** package.

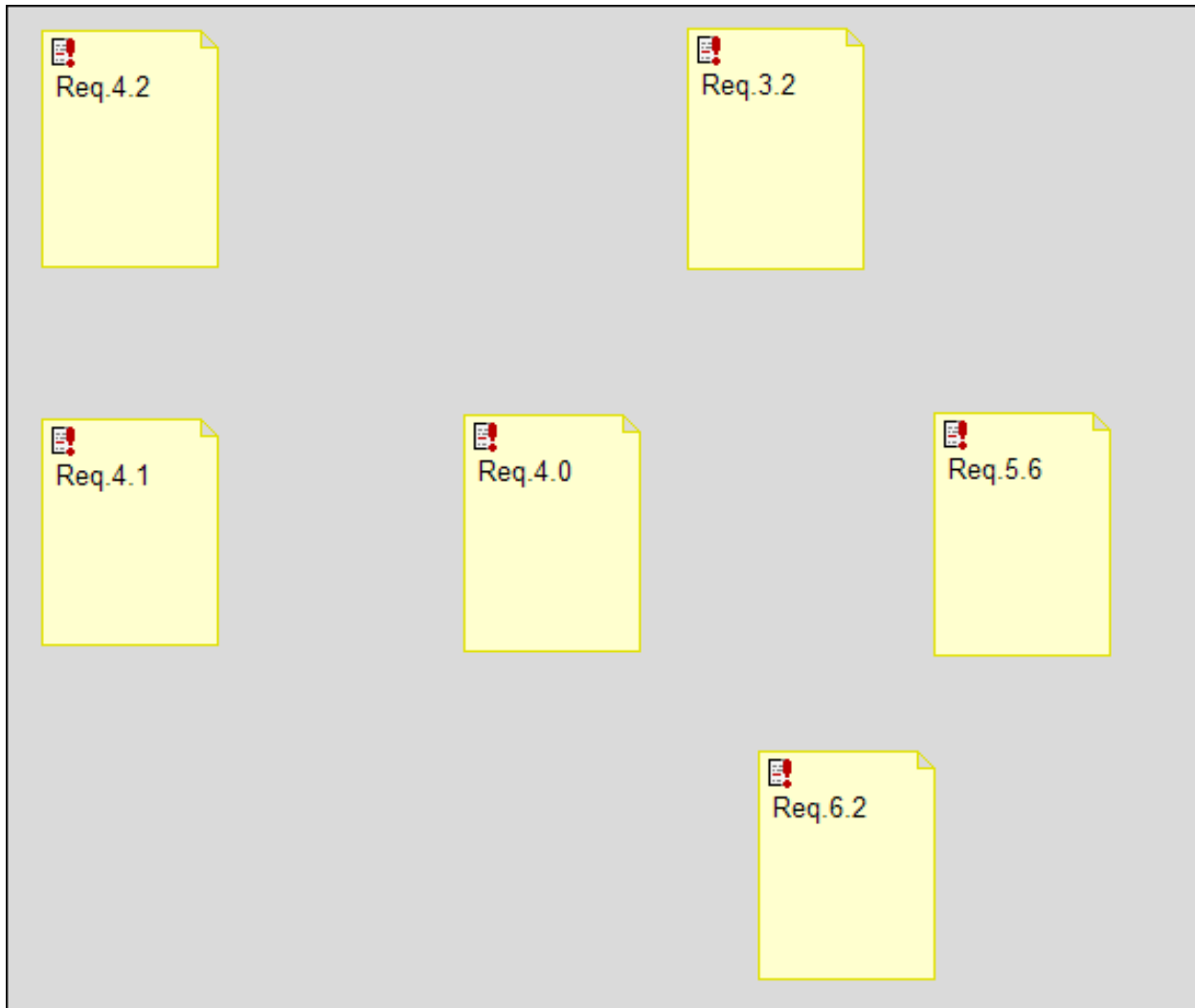
To create the Data Call Requirements UCD:

1. Right-click the **RequirementsPkg** package, click **Add New**, and then click **Use Case Diagram**. The **New Diagram** dialog box appears.
2. Type “Data Call Requirements” and then click **OK**.

Modeler automatically adds the **Use Case Diagrams** category and the new **Data Call UCD** to the **RequirementsPkg** package in the **Browser Window**. A new drawing area opens.

## Adding Requirements

In this procedure, you add requirements to the Data Call UCD, as shown in the following illustration.



To add the requirements:

1. In the **Browser Window**, expand the **RequirementsPkg** package and the **Requirements** category.
2. Click **Req.4.2** and drag it to the top left of the drawing area.
3. Click **Req.4.1** and drag it below **Req.4.2**.
4. Click **Req.3.2** and drag it to the top center of the drawing area.
5. Click **Req.4.0** and drag it to the lower left side of **Req.3.2**.
6. Click **Req.5.6** and drag it to the lower right side of **Req.3.2**.
7. Click **Req.6.2** and drag it below **Req.5.6**.

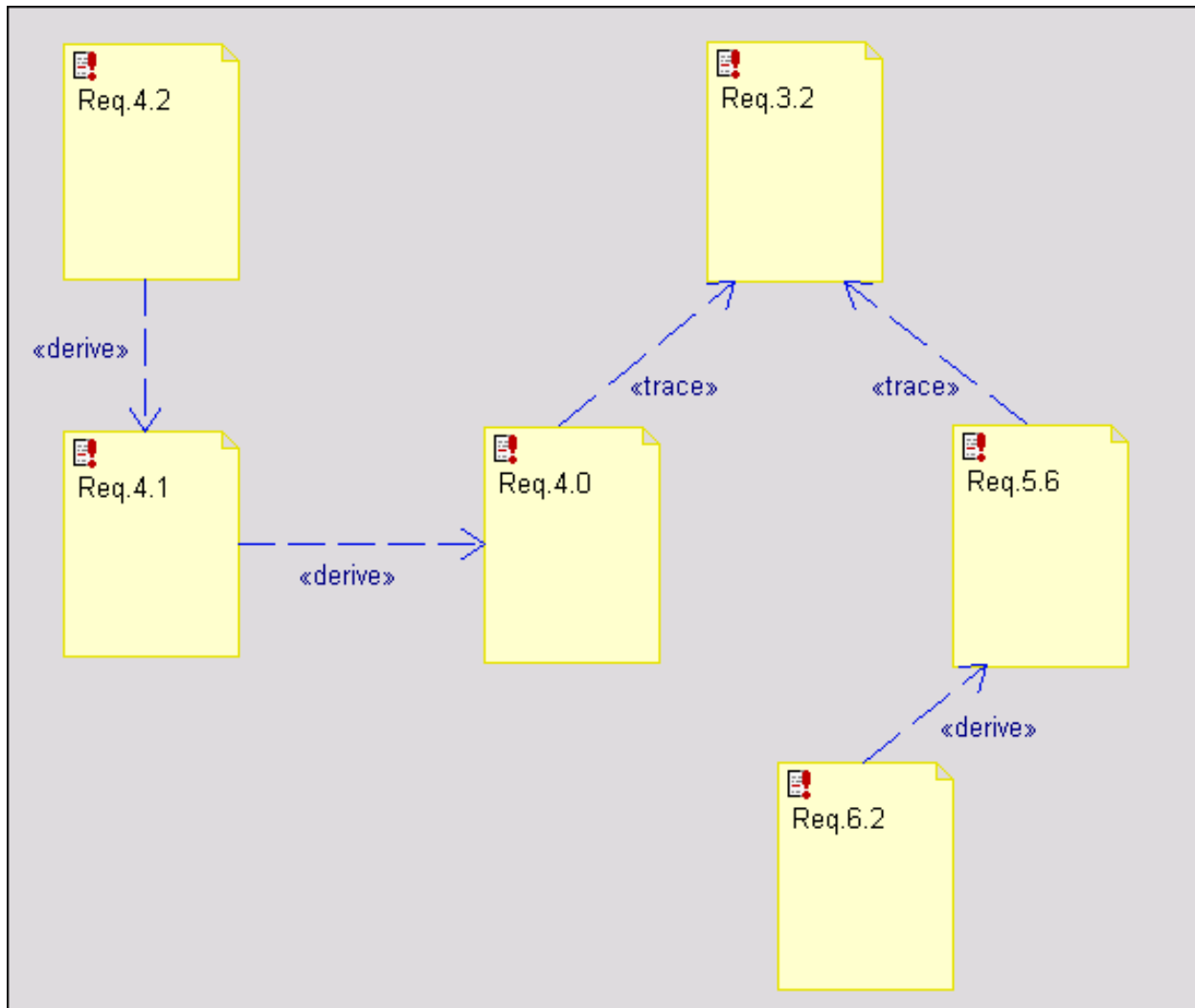
### Note

---


See the [Setting the Display Option to Show the Requirement the Element Name](#) procedure to set the display option of the requirements to **Name**.

## Drawing and Defining the Dependencies

In this procedure, you show the relationship between requirements by drawing dependencies and then setting the dependency stereotype, as shown in the following illustration.



To draw and define the dependencies:

1. Click the **Dependency** icon .
2. Draw a dependency line from **Req.4.2** to **Req.4.1**, and then set the **Stereotype** to **derive**.
3. Draw a dependency line from **Req.4.1** to **Req.4.0**, and then set the **Stereotype** to **derive**.
4. Draw a dependency line from **Req.4.0** to **Req.3.2**, and then set the **Stereotype** to **trace**.
5. Draw a dependency line from **Req.5.6** to **Req.3.2**, and then set the **Stereotype** to **trace**.
6. Draw a dependency line from **Req.6.2** to **Req.5.6**, and then set the **Stereotype** to **derive**.

Modeler automatically adds the dependency relationships to the **Browser Window**.

## Summary

In this module, you created UCDs that show the functions and requirements of the wireless telephone and placing a call. You became familiar with the parts of a UCD and created the following:

- ◆ System boundary box
- ◆ Actors
- ◆ Use cases
- ◆ Association lines
- ◆ Dependencies
- ◆ Generalizations
- ◆ Requirements

In the next module, you define the components of the system and the flow of information using structure diagrams.

Requirements traceability, coverage, and impact analysis can be performed using the using the RG in conjunction with these Rational Rhapsody editions:

- ◆ Rational Rhapsody Developer
- ◆ Rational Rhapsody Designer for Systems Engineers
- ◆ Rational Rhapsody Architect for Systems Engineers





# Module 2:

## Creating Structure Diagrams

---

**Structure diagrams** define the system structure and identify the large-scale organizational pieces of the system. They can show the flow of information between system components, and the interface definition through ports. In large systems, the components are often decomposed into functions or subsystems.

### Goals of this Module

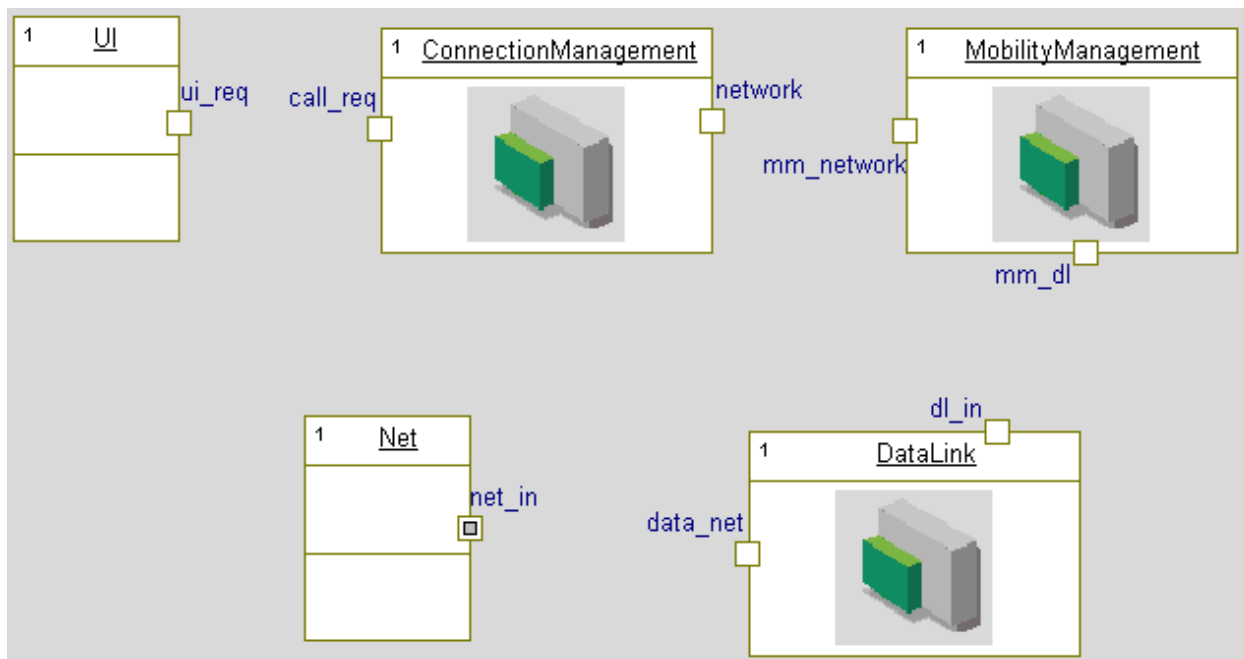
In this module, you create the following structure diagrams:

- ♦ **Block Diagram** - Identifies the system-level components and flow of information
- ♦ **Connection Management** - Identifies the ConnectionManagement functions
- ♦ **Data Link** - Identifies the DataLink functions
- ♦ **MM Architecture Structure** - Identifies the MobilityManagement functions

For ease of presentation, this module includes both the system and subsystem structure diagrams. Depending on your workflow, you might identify the communication scenarios using sequence diagrams before defining the flows, flow items, and port contracts. In addition, you might perform black-box analysis using activity diagrams, sequence diagrams, and statecharts, and white-box analysis using sequence diagrams before decomposing the functions of the system into subsystem components.

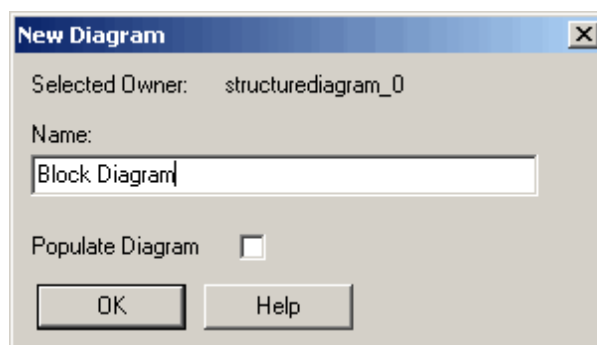
## Exercise 2.1: Creating a Block Diagram

In this exercise, you develop a Block Diagram, as shown in the following illustration.



To create a block diagram:

1. In the **Browser Window**, right-click the **ArchitecturePkg** package, click **Add New**, and then click **Structure Diagram**. The **New Diagram** dialog box appears.



2. Type “Block Diagram” and click **OK**.

Modeler automatically creates the **Structure Diagrams** category and the new block diagram to the **ArchitecturePkg** package in the **Browser Window**. A new drawing area opens.

### Drawing the Block Diagram

The **Block Diagram** identifies the system components (blocks and objects) and describes the flow of data between the components from a black-box perspective. In the subsequent procedures, you break down the system components (blocks) to show the subfunctions and flow of data. In the Object Model Diagrams section, you decompose the system components (blocks) to show the modules and flow of data.

Draw structure diagrams using the following general steps:

1. Draw blocks.
2. Draw objects.
3. Draw ports.
4. Draw flows.

### Drawing Objects




An object is an entity with a well-defined boundary and identity that encapsulates state and behavior. *State* is represented by attributes and relationships, whereas *behavior* is represented by operations, methods, and state machines.

An object is an instance of a class. In object-oriented languages such as C++, a class is a template for the creation of instances (objects) that share the same attributes, operations, methods, relationships, and semantics.

The handset model contains the following three system components or functions:

- ♦ **ConnectionManagement** to handle the reception, setup, and transmission of incoming and outgoing call requests.
- ♦ **MobilityManagement** to handle the registration and location of users.
- ♦ **DataLink** to monitor registration.

To draw the objects:

1. Click the Object button  in the Diagram Tools.
2. Click the top center of the drawing area. (You can also use click-and-drag.) Rational Rhapsody creates an object with a default name of **object\_n**, where *n* is equal to or greater than 0.
3. Rename the object `ConnectionManagement` press **Enter**.
4. Click the Object button  in the Diagram Tools, but this time click the upper right of the drawing area and rename the object `MobilityManagement`.
5. Click the Object button , but this time click the bottom right of the drawing area and rename the object `DataLink`.

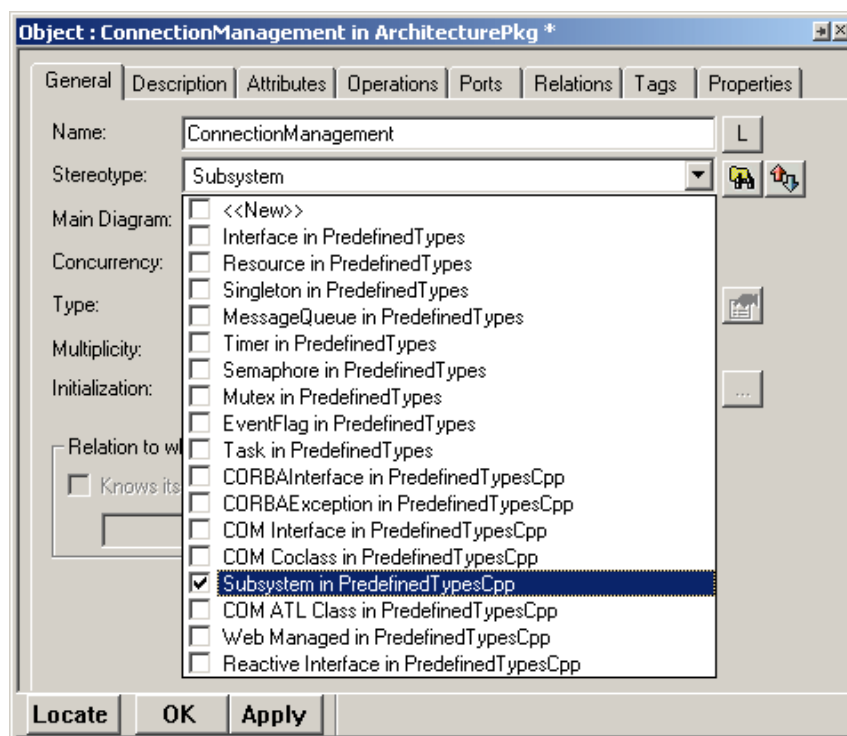
**Note:** You can use the tools on the **Layout** toolbar to help you with the layout of selected elements in your diagram. Keep in mind that the last element selected is used as the default. Plus, if you want to move a drawn element (including labels) on a drawing more precisely, click one or more elements, press the **Ctrl** key and use the standalone directional arrow keys. You can also use the directional arrows on the numeric keypad with NumLock not active.

## Defining the Object Stereotype

To indicate that the **ConnectionManagement**, **MobilityManagement**, and **DataLink** objects are subsystems that are to be further decomposed, you must set the stereotype to **Subsystem**.

To define the stereotype:

1. Double-click the **ConnectionManagement** object, or right-click and select **Features**. The Features dialog box opens.
2. On the **General** tab, in the **Stereotype** box, select the **Subsystem in PredefinedTypes Cpp** check box, as shown in the following figure:



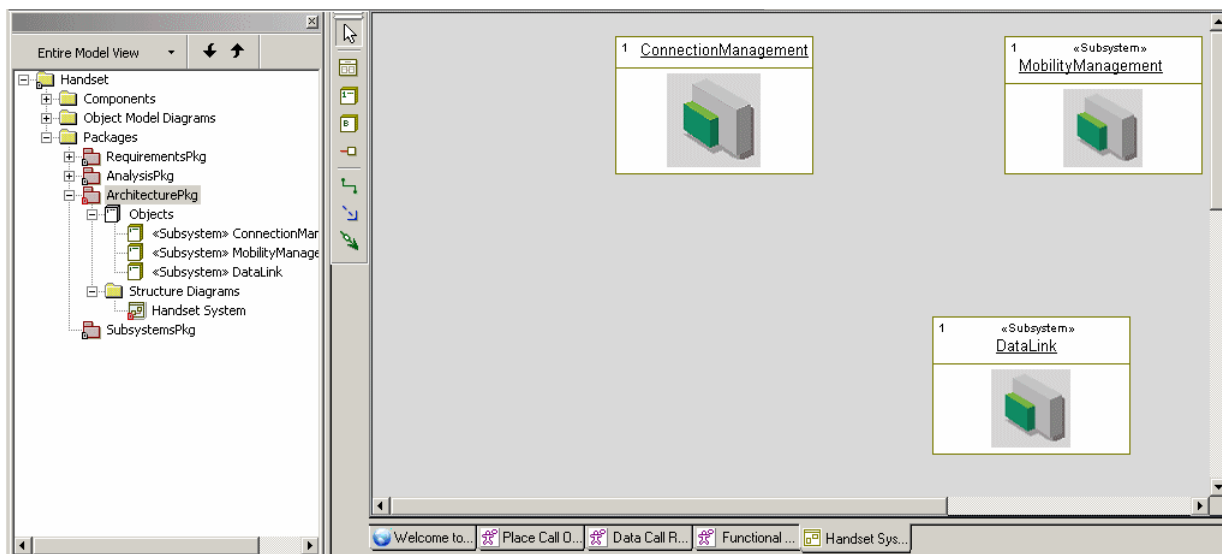
**Note:** After you make your selection, **Subsystem** appears in the box.

3. Click **Apply** to apply your changes.
4. With the Features dialog box still open, set the stereotype to **Subsystem** for the **MobilityManagement** and **DataLink** objects.
5. When done setting stereotypes, click **OK** to close the Features dialog box.
6. Right-click one of the objects (for example, **ConnectionManagement**) and select **Display Options**. The Display Options dialog box opens.

7. Make the following settings for the object:
  - a. In the **Display Name** group, select the **Label** option button.
  - b. Clear the **Show Stereotype Label** check box.
  - c. In the Image View group, select the **Enable Image View** check box select the **Use Associated Image** option button; click the **Advanced** button to open the Advanced Image View Options dialog box and select the **Structured** option button click **OK** to close the dialog box.
  - d. Click **OK** to close the Display Options dialog box.

The object appears on your drawing with its label underlined. This is the default style for the appearance of the label. In addition, the object shows the image associated with it. The number in the upper left corner shows the multiplicity for the object.


8. Set the same display options for the other objects (for example, **MobilityManagement** and **DataLink** if you already did **ConnectionManagement**).
9. Save your model. Your Handset System structure diagram and browser should resemble the following figure:



## Drawing More Objects

In this task, you are going to draw the two objects that interact with the system: **UI** (user interface) and **Net** (network).

To draw these objects:

1. Click the Object button  in the Diagram Tools.
2. Click the upper, left corner of the drawing window. (You can also use click-and-drag.) Rational Rhapsody creates an object with a default name of **object\_n**, where **n** is equal to or greater than 0.
3. Rename the object **UI** press **Enter**.
4. Create another object, but this time click the bottom center of the drawing area and rename the object **Net**.

## Setting the Object Stereotype and Type

You can define the features of an object, including the stereotype and type, using the Features dialog box. The type specifies the class of which the object is an instance; that is, it provides a unique instance for each object.

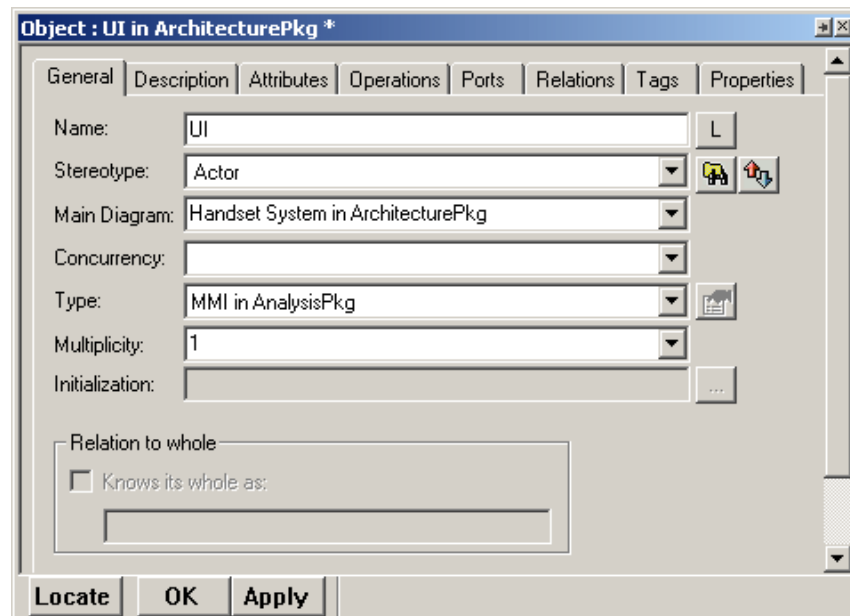
In this task, you are going to define and set the stereotype for the **UI** and **Net** objects to **Actor** to indicate that the objects are actors. You also going to set the **UI** object type to **MMI in AnalysisPkg** and the **Net** object type to **Network in AnalysisPkg**.

To set the stereotype and type:

1. Double-click the **UI** object, or right-click and select **Features**. The Features dialog box opens.
2. On the **General** tab, set the following options:
  - a. In the **Stereotype** box, select **<<New>>**.
  - b. Type **Actor** in the **Name** box of the dialog box that appears click **OK**. After you make your selection, **Actor** appears in the **Stereotype** box, as shown in the following diagram.

**Note:** This step is necessary if you are creating the handset model from scratch because there are no stereotypes created for the **ArchitecturePkg** package yet. If you are using the handset model provided with the Rational Rhapsody product you will see an **Actor in Architecture** check box in the list for the **Stereotype** box, because this stereotype would have already been created for the package.

- c. In the **Type** box, select **MMI in AnalysisPkg**.



3. Click **Apply** to apply the changes.
4. Rational Rhapsody displays a message stating that turning object to be of a specific type will cause the loss of current object features. Click **Yes** to continue.
5. Click **OK** to close the dialog box.
6. Open the Features dialog box for the **Net** object and set the following options:
  - a. In the **Stereotype** box, select the **Actor in Architecture** check box.
  - b. In the **Type** box, select **Network in AnalysisPkg**.
7. Click **Apply** to apply the changes.
8. Rational Rhapsody displays a message stating that turning object to be of a specific type will cause the loss of current object features. Click **Yes** to continue.
9. Click **OK** to close the dialog box.
10. Right-click one of the objects (for example, **UI**) and select **Display Options**. The Display Options dialog box opens.
11. Make the following settings for the object:
  - a. In the Display Name group, select the **Label** option button.

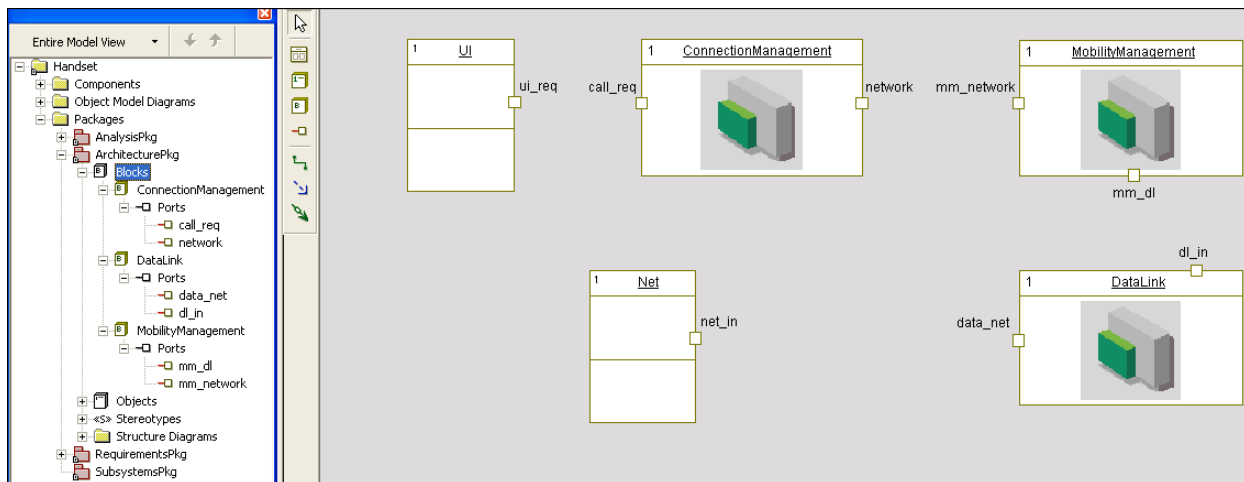


- b. Clear the **Show Stereotype Label** check box.
  - c. Click **OK** to close the Display Options dialog box.
12. Repeat the previous step to set the same display options for the **Net** object.


## Drawing Ports

A **Port** is a distinct interaction point between a class or object and its environment. Ports enable you to capture the architecture of the system by specifying the interfaces between the system components and the relationships between the subsystems. A port appears as a small square on the boundary of a class, object, or block.

In this procedure, you draw several ports, as shown in the following illustration.



To draw ports:

1. Click the **Create Port** icon  in the Diagram Tools.
2. Click on the right edge of the **UI** object to place the port. A port with a default name of **port\_n**, where **n** is equal to or greater than 0, is created.
3. Rename the port to **ui\_req** by typing “ui\_req” and pressing **Enter**.

This port represents the access point where user interface messages flow in and out.

4. Repeat Steps 1 through 3 to create the following ports:
  - ♦ Click on the left edge of **ConnectionManagement** and create a port called **call\_req**. This port sends and relays messages to and from the user interface.

- ♦ Click on the right edge of **ConnectionManagement** and create a port called **network**. This port sends and relays messages from **MobilityManagement**.
  - ♦ Click on the left edge of **MobilityManagement** and create a port called **mm\_network**. This port sends and relays messages from **ConnectionManagement**.
  - ♦ Click on the bottom edge of **MobilityManagement** and create a port called **mm\_dl**. This port relays registration information to **DataLink**.
  - ♦ Click on the top edge of **Datalink** and create a port called **dl\_in**. This port relays information between **DataLink** and **MobilityManagement**.
  - ♦ Click on the left edge of **Datalink** and create a port called **data\_net**. This port relays information between **DataLink** and the network.
  - ♦ Click on the right edge of **Net** and create a port called **net\_in**. This port represents the access point where network data flows in and out.
5. In the **Browser Window**, expand the **Blocks** category to view the newly created objects.

## Specifying Port Attributes

You can specify ports as **Behavioral ports**, which indicate that the messages sent are relayed to the owner class. A behavioral port terminates an object or part that provides the service.

In this procedure, you set the **ui\_req** port as a behavioral port.

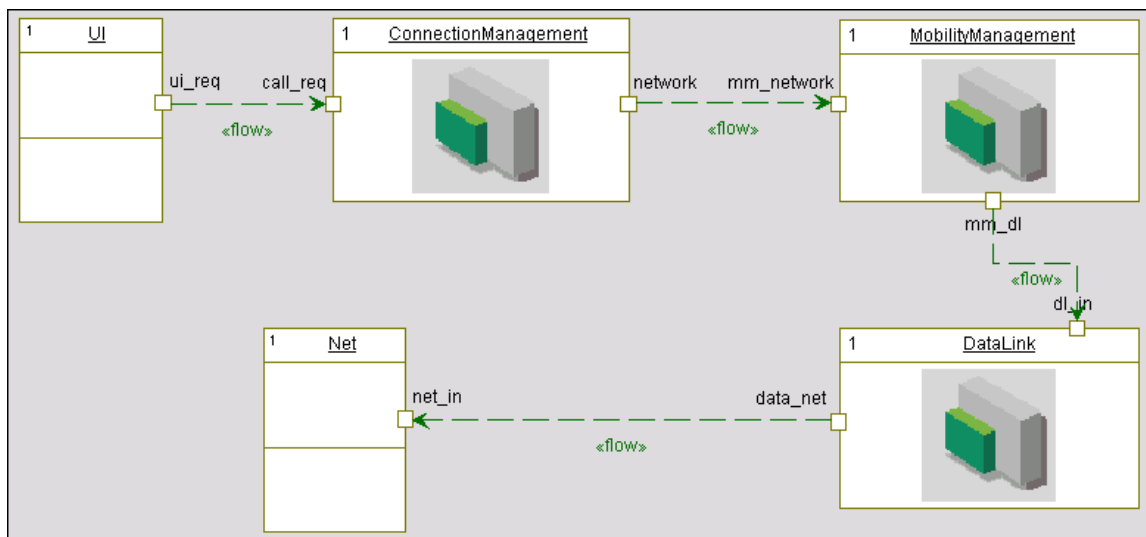
To specify port attributes:

1. Double-click the **ui\_req** port. The **Features** dialog box appears.
2. Click the **General** tab.
3. Under **Attributes**, select the **Behavior** check box.
4. Click **OK** to save the changes and to close the dialog box.

## Drawing Flows

**Flows** specify the exchange of information between system elements. They enable you to describe the flow of data and commands within a system at a very early stage, before committing to a specific design.

In this procedure, you draw several flows between the objects and block, as shown in the following illustration.



To draw a flow:

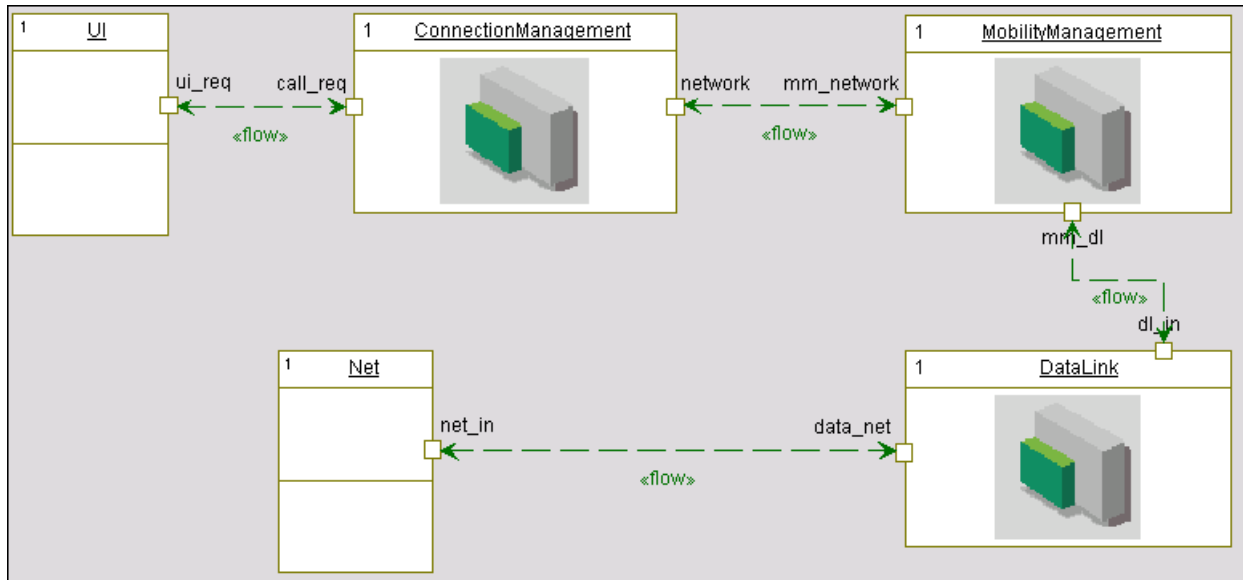
1. Click the **Flow** icon  in the Diagram Tools.

2. Click the **ui\_req** port, click the **call\_req** port, and then press **Enter**. A flow line appears between **ui\_req** object and **call\_req** block.
3. Repeat Steps 1 and 2 to create the flow lines:
  - ♦ Between the **network** port and the **mm\_network** port.
  - ♦ Between the **mm\_dl** port and the **dl\_in** port.
  - ♦ Between the **data\_net** port and the **net\_in** port.

## Changing the Direction of the Flow

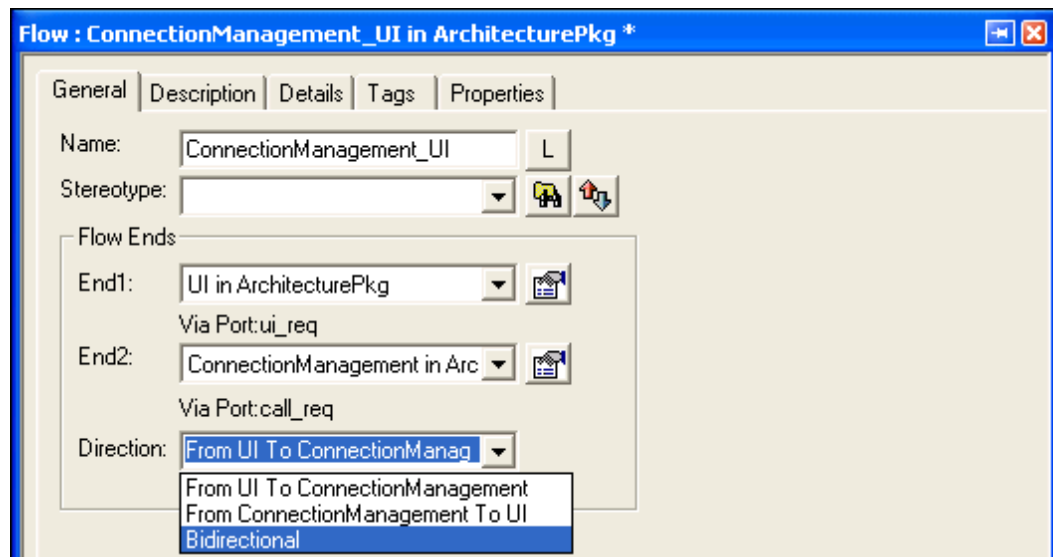
Information can flow from one element to another or between elements in either direction. You can change the direction of the flow or make the flow bidirectional using the features dialog box.

In this procedure, you change all flows to bidirectional to indicate that information can flow in either direction between system elements, as shown in the following illustration.



To change the direction of the flow:

1. Double-click the flow between **UI** and **ConnectionManagement**. The **Features** dialog box appears.
2. Click the **General** tab.
3. In the **Direction** list, click **Bidirectional**.



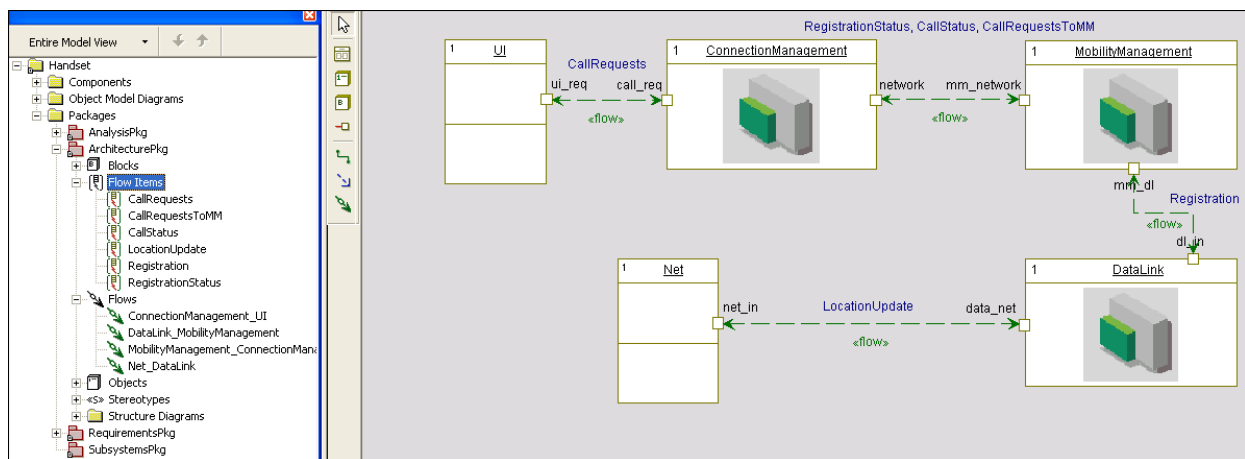
4. Click **OK** to save the changes and to close the dialog box.
5. Repeat Steps 1 through 4 to set the flow between:
  - ♦ **ConnectionManagement** and **MobilityManagement** to **Bidirectional**.
  - ♦ **MobilityManagement** and **DataLink** to **Bidirectional**.
  - ♦ **Datalink** and **Net** to **Bidirectional**.

## Specifying the Flow Items

Once you have determined how communication occurs through flows, you can specify the information that passes over a flow using a **Flow Item**. A flow item can represent either pure data, data instantiation, or commands (events).

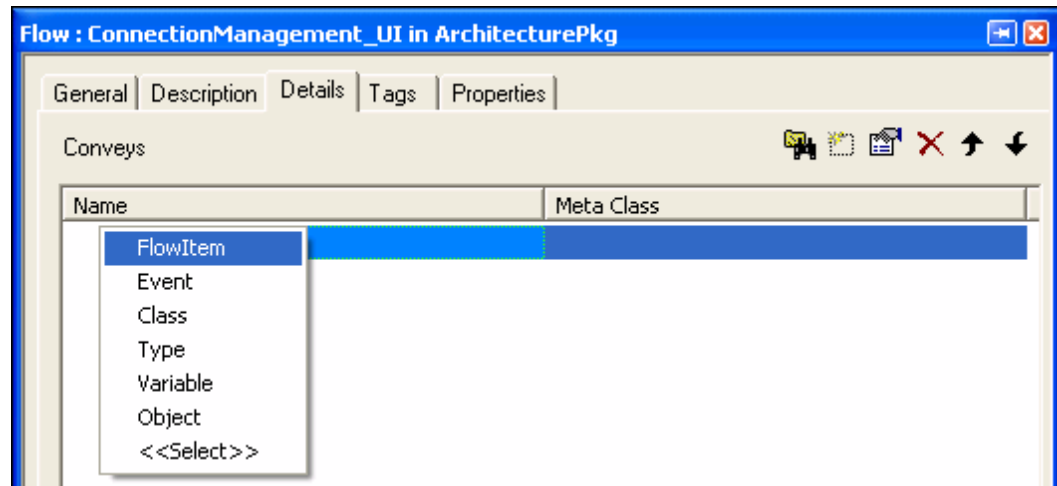
As the system specification evolves, such as by defining the communication scenarios using sequence diagrams, you can refine the flow items to relate to the concrete implementation and elements. See [Module 4: Creating Sequence Diagrams](#) for more information on defining scenarios.

In this procedure, you specify the flow items for the block and object flows, as shown in the following illustration.

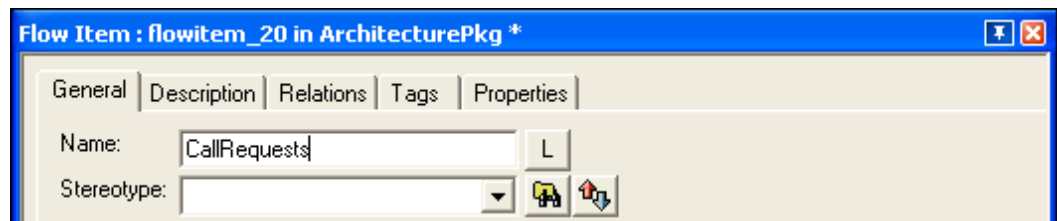


To specify the flow items:

1. Double-click the flow between the **ui\_req** port and the **call\_req** port. The **Features** dialog box appears.
2. Click the **Details** tab.
3. Click **<Add>** and then click **FlowItem**.



The **Flow Item** dialog box appears.



4. In the **Name** box, type “CallRequests” and click **OK** to save the changes and close the **Flow Item** dialog box.
5. Click **OK** to save the changes and to close the **Features** dialog box.



6. Repeat Steps 1 through 5 to specify the flow items for the flows between:

- ♦ The **network** port and the **mm\_network** port. Create three flow items called:
  - **RegistrationStatus**
  - **CallStatus**
  - **CallRequestsToMM**

These flow items represents the relay of information between the main call control logic (**ConnectionManagement**), and user location (**MobilityManagement**).

- ♦ The **mm\_dl** port and the **dl\_in** port called **Registration**.

This flow item represents network registration status information.

The **data\_net** port and the **net\_in** port called **LocationUpdate**.

This flow item represents all network information into and out of the system.

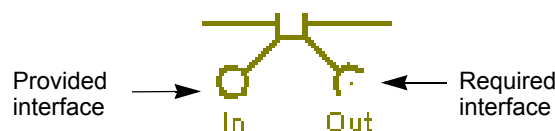
7. In the **Browser Window**, expand the **ArchitecturePkg** package and the **Flows** and the **Flow Items** categories to view the newly created flows and flow items.

## Specifying the Port Contract

Modeler provides contract-based ports and noncontract-based ports.

- ♦ **Contract-based ports** - Define a contract that specifies the precise allowable inputs and outputs of a component. A contract-based port can have the following interfaces:
  - **Provided Interfaces** - Characterize the requests that can be made from the environment. A provided interface is denoted by a lollipop notation.
  - **Required Interfaces** - Characterize the requests that can be made from the port to its environment (external objects). A required interface is denoted by a socket notation.

**Provided and Required Interfaces** enable you to encapsulate model elements by defining the access through the port. The following figure shows an example of the port interfaces.

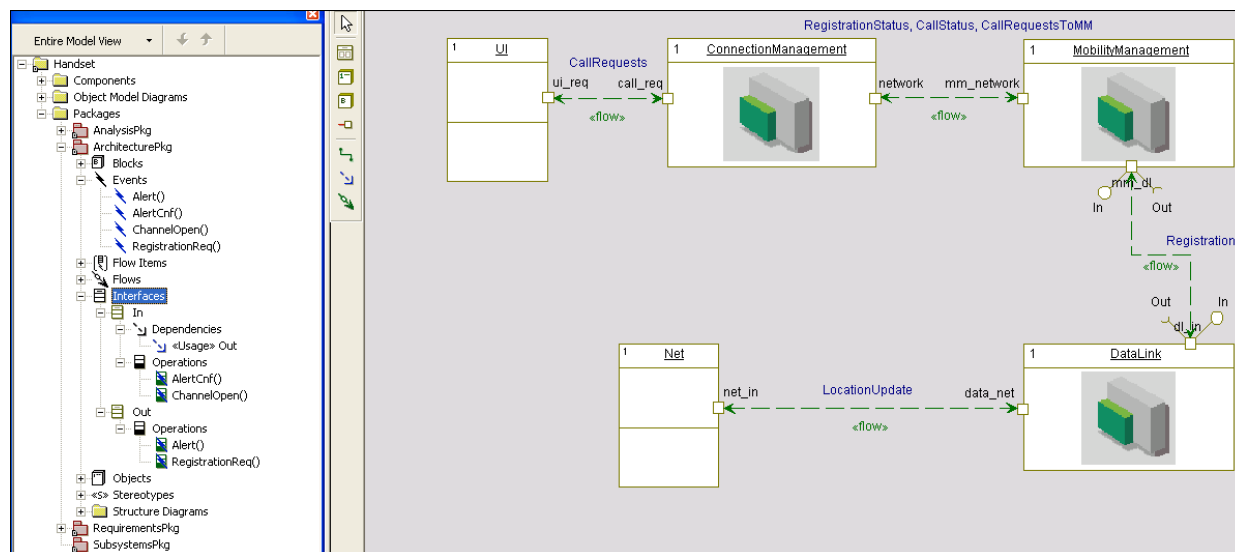


- ♦ **Noncontract-based ports** - Called **rapid ports**, enable you to relay messages to the appropriate part of the structured class through a connector. They do not require a contract to be established initially, but allow the routing of incoming data through a port to the appropriate part.

## Module 2: Creating Structure Diagrams

Rational Rhapsody Developer and Designer for Systems Engineers are able to execute models with rapid port, allowing testing up front, before the design is finalized.

In this procedure, you specify the provided and required interfaces for the **mm\_dl** and **dl\_in** ports, as shown in the following illustration.

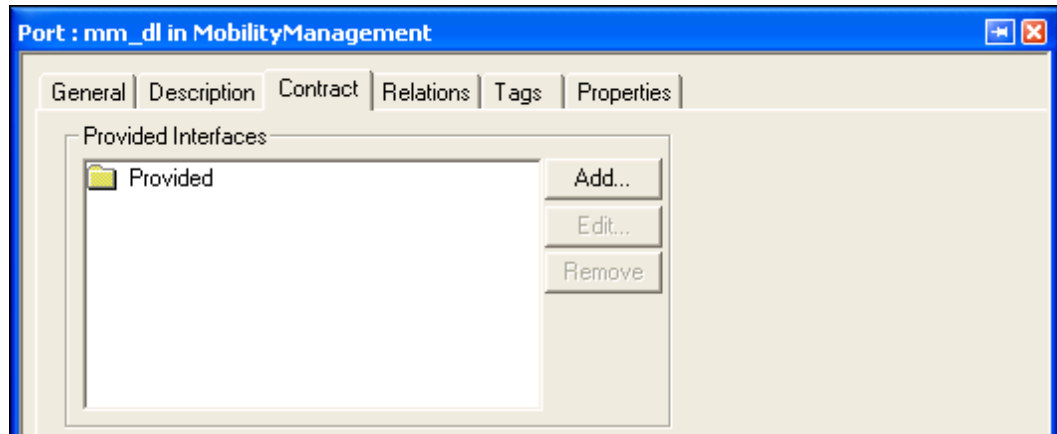


### Note

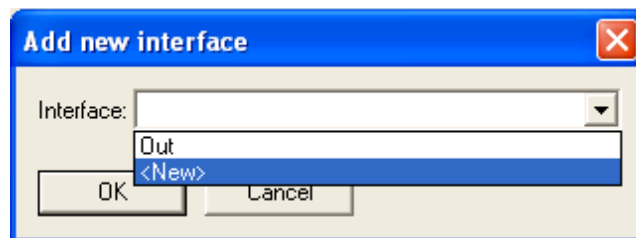
Depending on your workflow, you might identify the communication scenarios using sequence diagrams before defining the port contracts. See [Module 4: Creating Sequence Diagrams](#) for more information.

To specify the port contract:

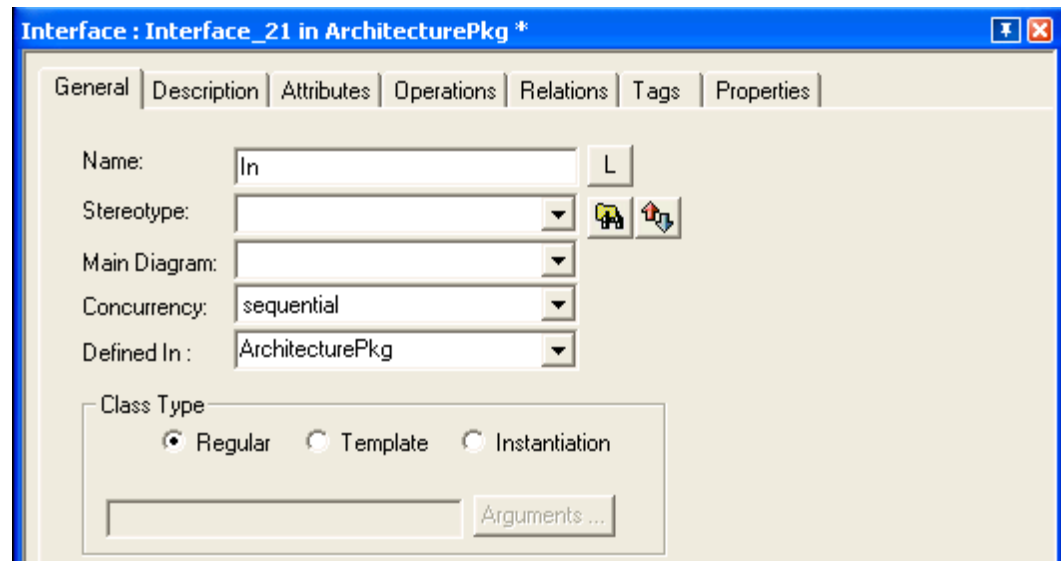
1. Double-click the **mm\_dl** port. The **Features** dialog box appears.
2. Click the **Contract** tab. The **Contract** window appears.



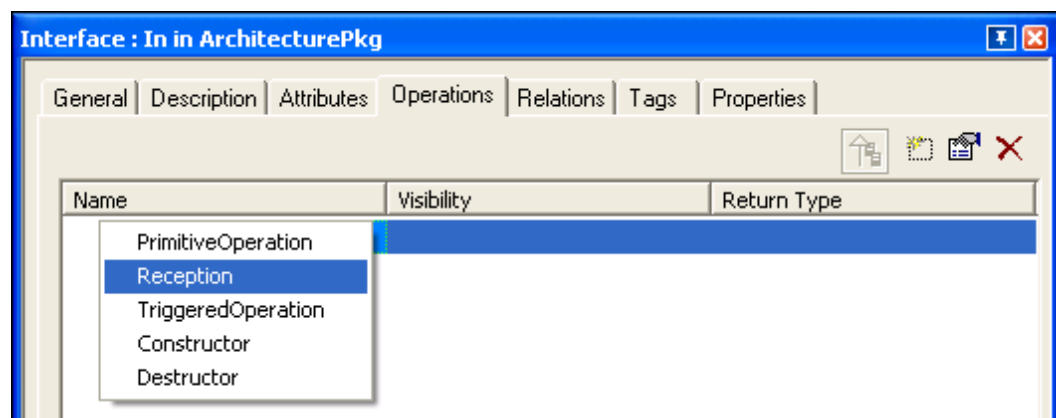
3. Click the **Provided** folder and then click the **Add**. The **Add new interface** dialog box appears.



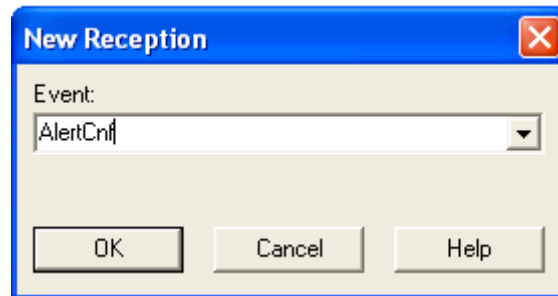
4. In the **Interface** list, click <New> and then click **OK**. The **Interface** dialog box appears.



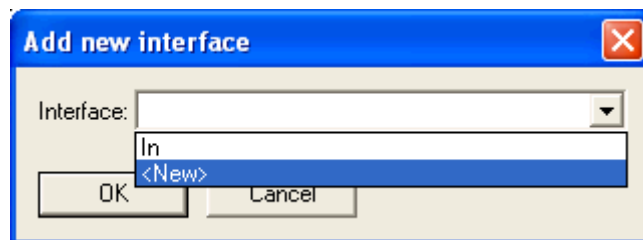
5. Click the **General** tab.
6. In the **Name** box, type "In."
7. Click the **Operations** tab.
8. Click <New> and then click **Reception**.



The **New Reception** dialog box appears.

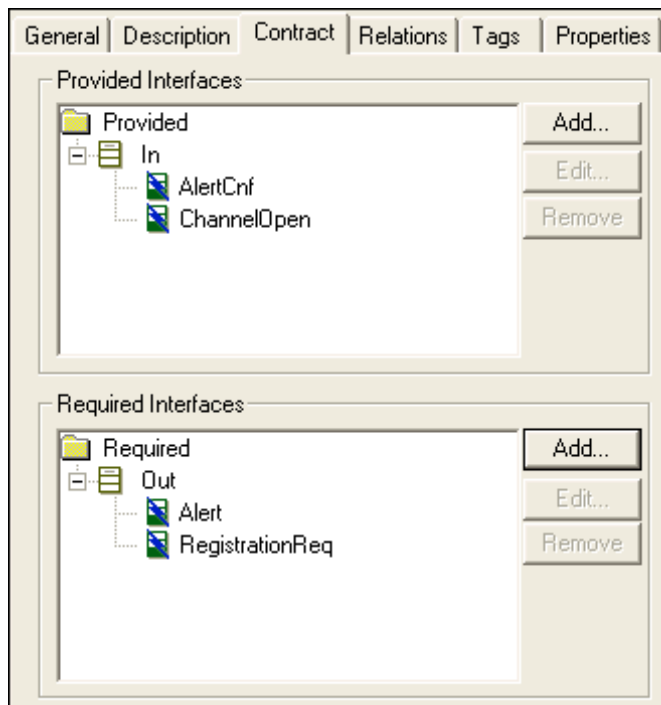


9. Type “AlertCnf” and click **OK**. A message displays that an event with the selected name could not be found. Click **Yes** to create the new event. The reception is added to the **Operations** tab.
10. Repeat Steps 8 and 9 to add a “ChannelOpen” reception.
11. Click **OK** to close the dialog box and return to the **Features** dialog box.
12. Click the **Required** folder and then click the **Add**. The **Add new interface** dialog box appears.



13. In the **Interface** list, click **<New>** and then click **OK**. The **Interface** dialog box appears.
14. Click the **General** tab.
15. In the **Name** box, type “Out.”
16. Click the **Operations** tab, click **<New>**, and then click **Reception**.
17. Type “Alert” and click **OK**. A message displays that an event with the selected name could not be found. Click **Yes** to create the new event. The reception is added to the **Operations** tab.
18. Repeat Steps 16 and 17 to add a “RegistrationReq” reception.

19. Click **OK** to close the dialog box and return to the **Features** dialog box. The **Contract** tab window reappears, listing the interfaces you just created.

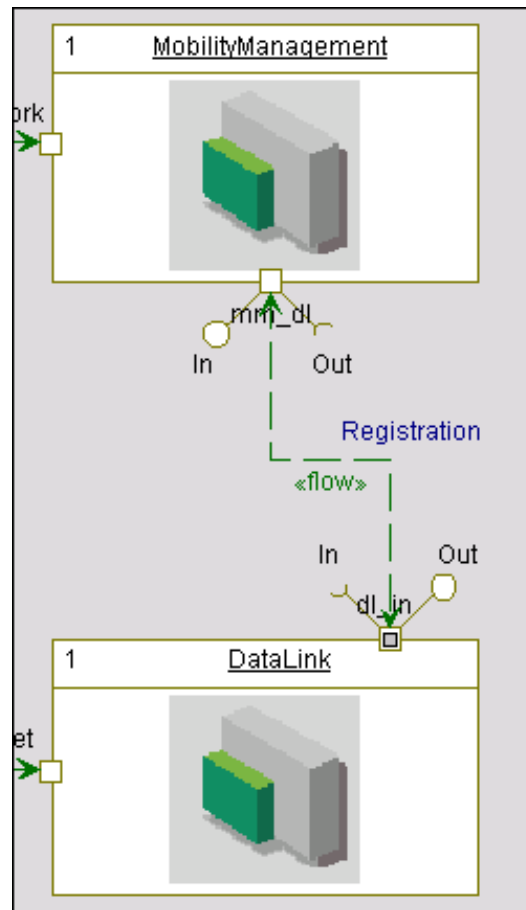


20. Click **OK** to close the **Features** dialog box.
21. In the **Browser Window**, expand the **ArchitecturePkg** package and the **Interfaces** category to view the newly created provided and required interfaces to the **mm\_dl** port. Expand the **Events** category to view the newly created receptions.
22. To specify the port interfaces for the **dl\_in** port, do the following:
  - a. Double-click the **dl\_in** port. The **Features** dialog box appears.
  - b. Click the **General** tab.
  - c. In the **Contract** list, click **In**.
  - d. Click the **Contract** tab.
  - e. Click the **Required** folder icon and then click **Add**.
  - f. In the **Interface** list, click **Out**.
  - g. Click **OK** to apply the changes and to close the **Features** dialog box.

## Reversing a Port

You can reverse ports so that the provided interfaces become the required interfaces, and the required interfaces become the provided interfaces.

In this procedure, you reverse the **dl\_in** port, as shown in the following illustration.



To reverse a port:

1. Double-click the **dl\_in** port. The **Features** dialog box appears.
2. Click the **General** tab.
3. Under **Attributes**, select the **Reversed** check box. A red message appears stating that the contract is reversed.
4. Click **OK** to apply the changes and to close the **Features** dialog box.

## Allocating the Functions Among Subsystems

Now that you have captured the architectural design in the Block Diagram, you need to divide the operations of the system into its functional subsystems and allocate the activities among the subsystems.

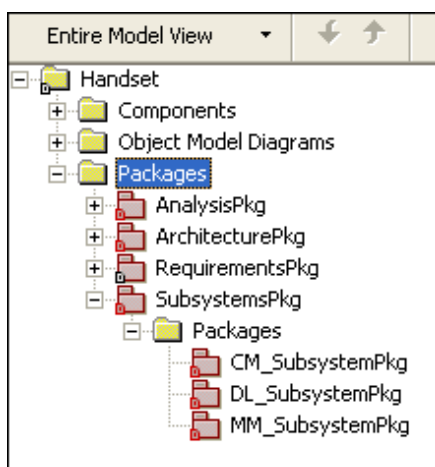
### Note

For ease of presentation, this section includes both the system and subsystem structure diagrams. Depending on your workflow, you might perform further black-box analysis with activity diagrams, sequence diagrams, and statecharts, and white-box analysis using sequence diagrams before decomposing the functions of the system into subsystem components.

## Organizing the Subsystems Package

Packages let you divide the system into functional domains, or subsystems, which consist of objects, object types, functions, variables, and other logical artifacts. They can be organized into hierarchies to provide a high level of partitioning.

In this procedure, you create the following subpackages, which represent the functional subsystems: **CM\_SubsystemPkg** for **ConnectionManagement**, **DL\_SubsystemPkg** for **DataLink**, and **MM\_SubsystemPkg** for **MobilityManagement**, as shown in the following illustration.





To create packages within the subsystems package:

1. In the **Browser Window**, right-click **SubsystemPkg**, click **Add New**, and then click **Package**. A package with the default name **package\_n**, where **n** is equal to or greater than 0, is created within the **SubsystemsPkg** package.
2. Rename the package to **CM\_SubsystemPkg** by typing “CM\_SubsystemsPkg” and pressing **Enter**.
3. Repeat Steps 1 and 2 to create two additional packages called **DL\_SubsystemPkg** and **MM\_SubsystemPkg**.

### Organizing Elements

In this procedure, you allocate the subsystem blocks from the Block Diagram in the **ArchitecturePkg** package to their respective packages in the **SubsystemsPkg** package by moving the following elements:

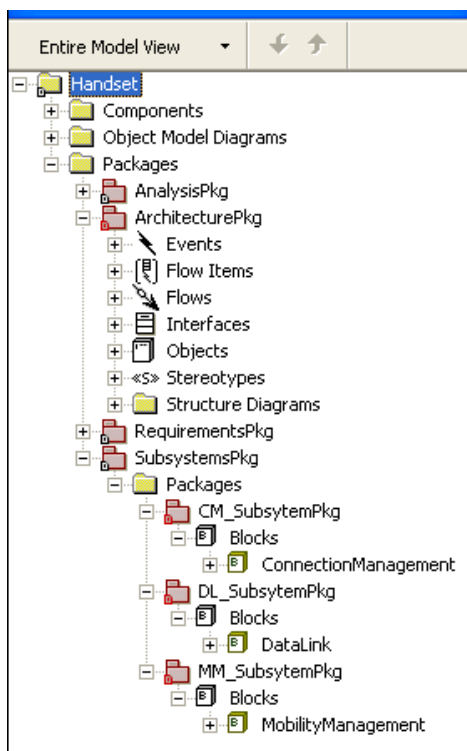
- ♦ The **ConnectionManagement** block to the **CM\_SubsystemPkg** package
- ♦ The **DataLink** block to the **DL\_SubsystemPkg** package
- ♦ The **MobilityManagement** block to the **MM\_SubsystemPkg** package

To organize elements:

1. In the **Browser Window**, expand the **ArchitecturePkg** package and the **Blocks** category.
2. Click the **ConnectionManagement** block and drag it into the **CM\_SubsystemPkg** package.
3. Click the **DataLink** block and drag it into the **DL\_SubsystemPkg** package.

4. Click the **MobilityManagement** block and drag it into the **MM\_SubsystemPkg** package.

The blocks are removed from the **ArchitecturePkg** package and added to the **SubsystemPkg** packages, as shown in the following figure.



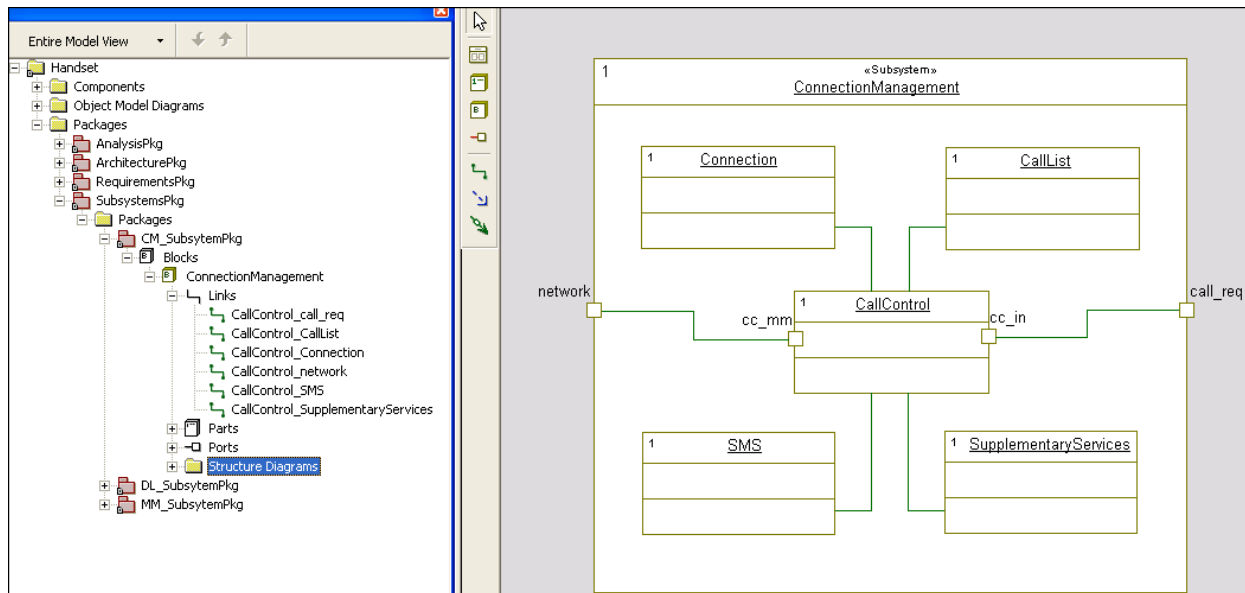
You can decompose the system-level blocks in the Block Diagram into sub-blocks and corresponding structure diagrams to show their decomposition. In the subsequent sections, you create the following subsystem structure diagrams:

- ◆ **Connection Management** from the **ConnectionManagement** block
- ◆ **Data Link** from the **DataLink** block
- ◆ **MM Architecture** from the **Mobility Management** block

## Exercise 2.2: Drawing the Connection Management Structure Diagram

The Connection Management structure diagram decomposes the `ConnectionManagement` block into its subsystems. Connection Management identifies how calls are set up, including the establishment and clearing of calls, short message services, and supplementary services.

In this exercise, you develop a Connection Management Structure diagram, as shown in the following illustration.



## Creating the Connection Management Structure Diagram

To create the Connection Management structure diagram:

1. In the **Browser Window**, expand the **CM\_SubsystemPkg** package and the **Blocks** category.
2. Right-click **ConnectionManagement**, click **Add New**, and then click **Structure Diagram**. The **New Diagram** dialog box appears.
3. Type “Connection Management Structure” and then click **OK**.

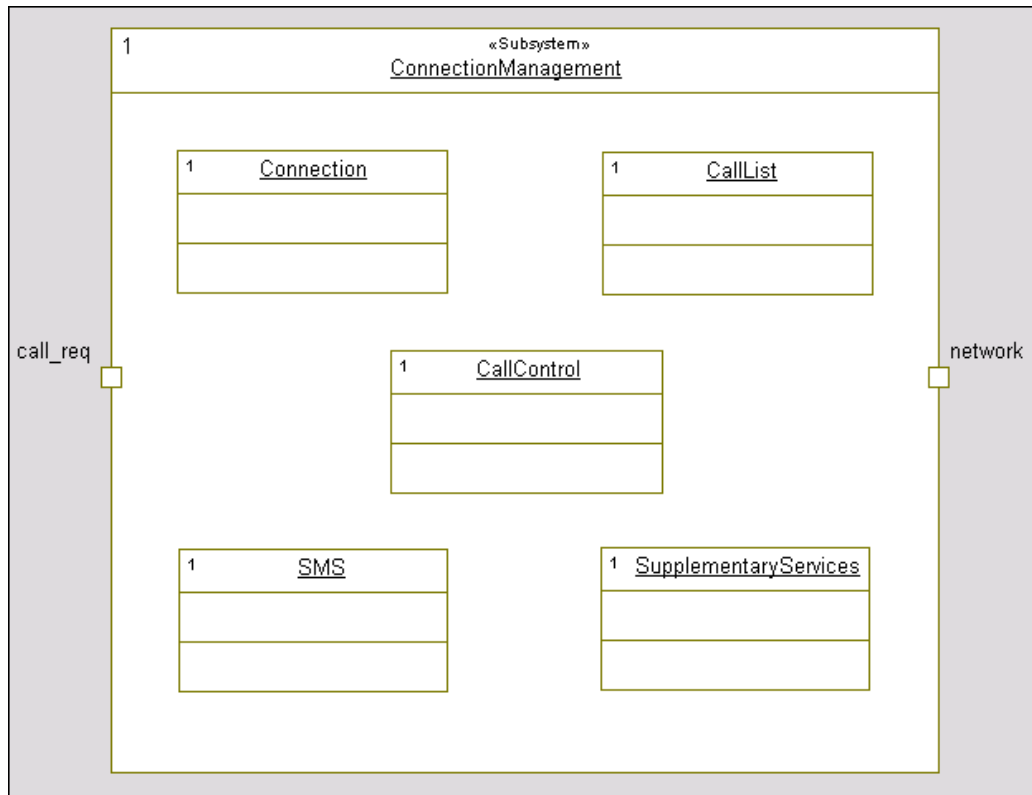
Modeler automatically adds the **Structure Diagrams** category and the **Connection Management Structure** diagram to the **Browser Window**. A new drawing area opens with the **ConnectionManagement** block and its ports, as defined in the Block Diagram.

### Making the Ports Visible


If the ports are not visible, right-click the **ConnectionManagement** block, click **Ports**, and then click **Show All Ports**.

### Drawing Objects

In this procedure, you draw the objects that represent the activities performed by **Connection Management**, as shown in the following illustration.



To draw objects:

1. Click the **Object** icon  in the Diagram Tools.
2. Click in the upper, left corner of **ConnectionManagement** block and drag to the desired size. An object with a default name of **object\_n**, where **n** is equal to or greater than 0, is created.
3. Rename the object to **Connection** by typing “Connection” and pressing **Enter**.

The **Connection** object tracks the number of valid connections.

4. Repeat Steps 1 through 3 to create the following objects:
  - ♦ **CallList** (upper, right corner of **ConnectionManagement** block) - Maintains the list of currently active calls
  - ♦ **CallControl** (middle of **ConnectionManagement** block) - Manages incoming and outgoing calls
  - ♦ **SMS** (lower, left corner of **ConnectionManagement** block) - Manages the short message services
  - ♦ **SupplementaryServices** (lower, right corner of **ConnectionManagement** block) - Manages the supplementary services, including call waiting, holding, and barring

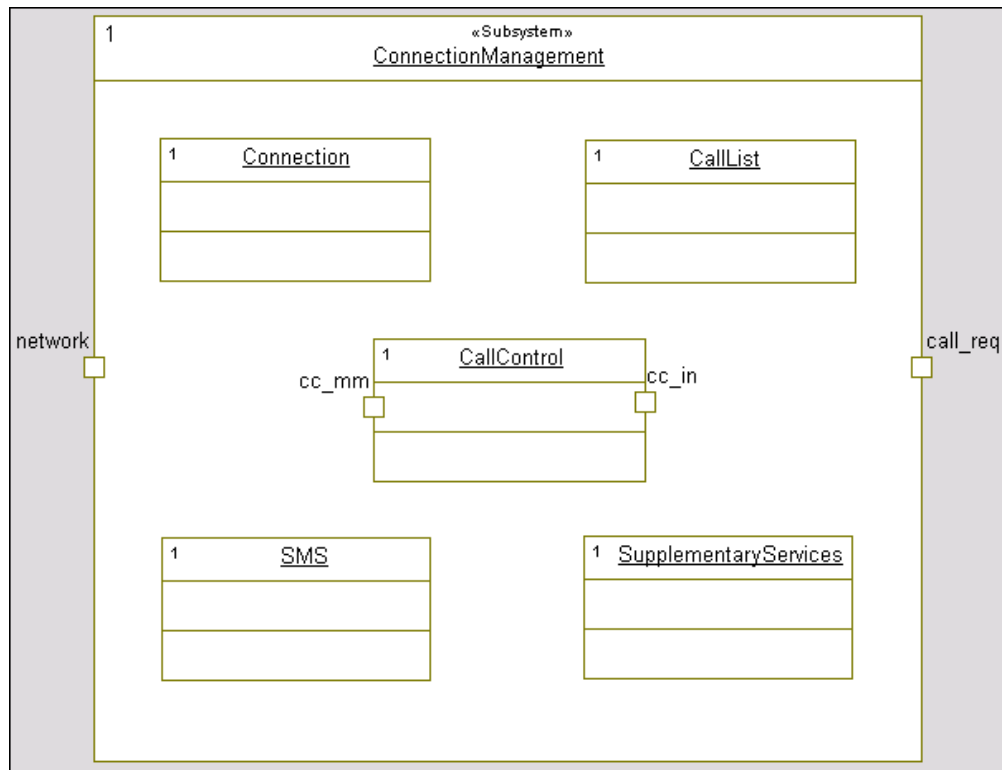
### Moving the Ports

To move the ports:


1. Click and drag the **call\_req** port to the right side of the **ConnectionManagement** block.
2. Click and drag the **network** port to the left side of the **ConnectionManagement** block.

### Drawing Ports

In this procedure, you draw new ports on the **CallControl** object, as shown in the following illustration.



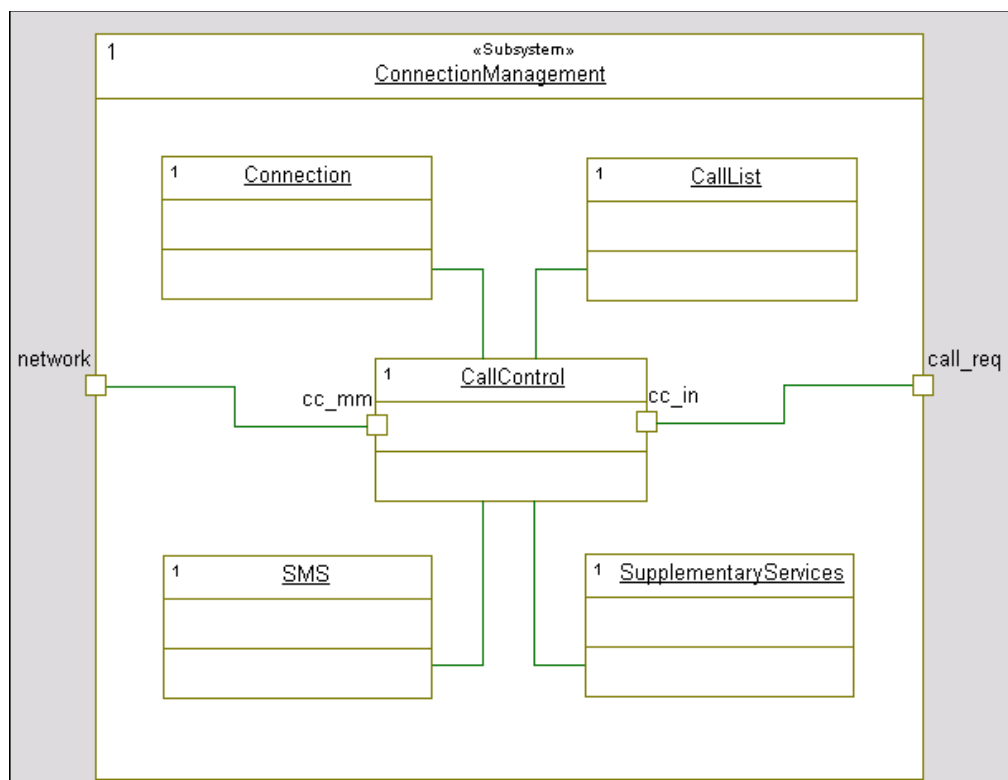
To draw ports:

1. Click the **Create Port** icon  in the Diagram Tools.
2. Click the left edge of the **CallControl** object, type “cc\_mm,” and then press **Enter**. This port relays messages to and from **MobilityManagement**.
3. Click the right edge of the **CallControl** object, type “cc\_in,” and then press **Enter**. This port relays messages from the user interface.


## Drawing Links

A **Link** is an instantiation of an association. You can specify links without having to specify the association being instantiated by the link; you can specify features of links that are not mapped to an association. There must be at least one association that connects one of the base classes of the type of one of the objects to a base class of the type of the second object.

In this procedure, you draw links between the objects and ports, as shown in the following illustration.




To draw links between objects and ports:

1. Click the **Link** icon  in the Diagram Tools.
2. Click the **cc\_mm** port, click the **network** port, and then press **Enter**.
3. Repeat Steps 1 and 2 to create links between:
  - ♦ The **cc\_in** port and the **call\_req** port
  - ♦ The **CallControl** object and the **Connection** object
  - ♦ The **CallControl** object and the **CallList** object



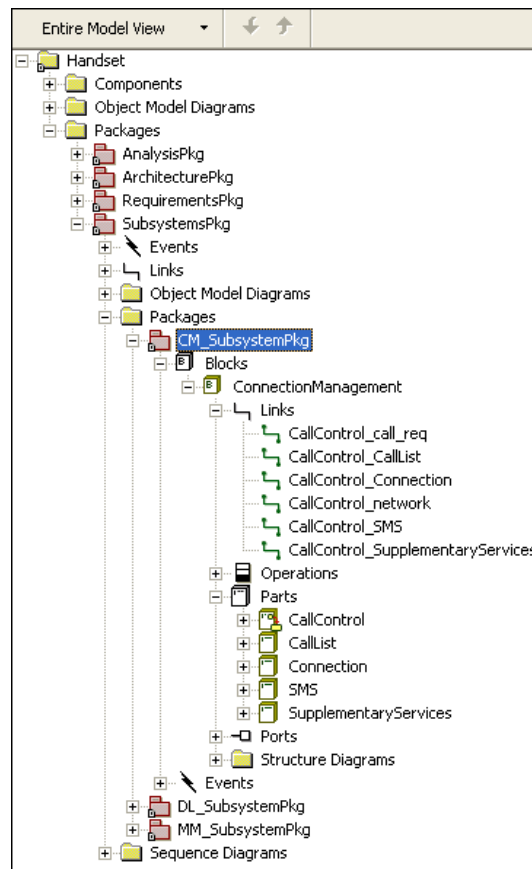
## Exercise 2.2: Drawing the Connection Management Structure Diagram

- ♦ The **CallControl** object and the **SMS** object
- ♦ The **CallControl** object and the **SupplementaryServices** object

**Note:** You can use the **Stamp** icon  for repetitive drawing actions. For example, when creating more than one link, click the **Stamp** icon first before clicking the **Link** icon. Once you have created a link, you can keep creating new links without clicking the **Link** icon each time. Click the **Stamp** icon again when you are done drawing links.

4. In the **Browser Window**, expand the **CM\_SubsystemPkg** package, the **ConnectionManagement** category, and:

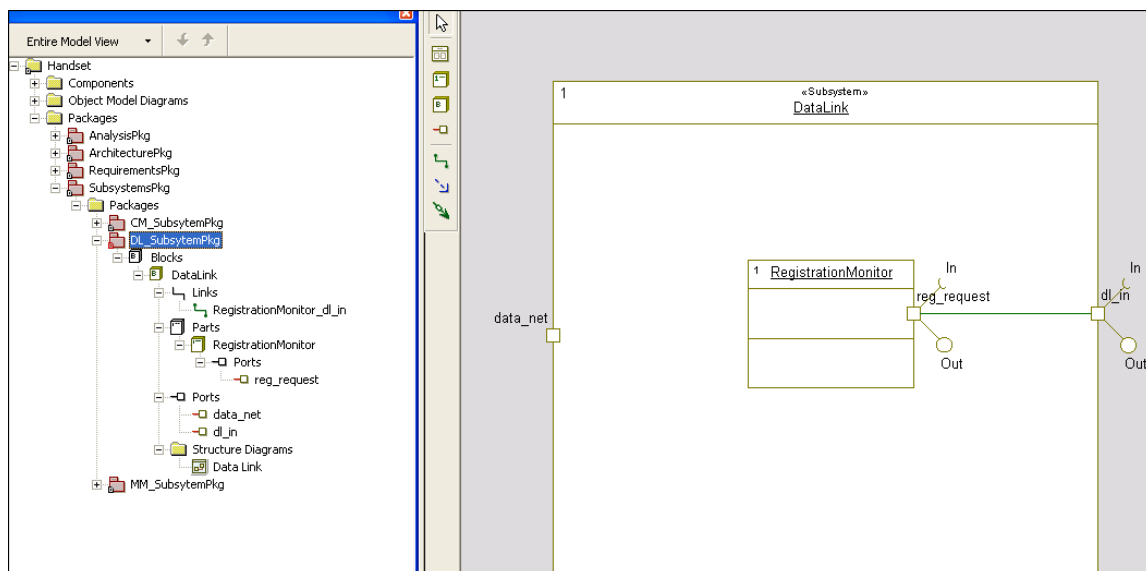
- ♦ The **Parts** category to view the newly created objects
- ♦ The **Links** category to view the newly created links, as shown in the following figure.



## Exercise 2.3: Drawing the Data Link Diagram

The Data Link diagram decomposes the DataLink block into its subsystems. It identifies how the system monitors registration.

In this exercise, you develop a Data Link diagram, as shown in the following illustration.



### Creating the Data Link Diagram

To create the Data Link diagram:

1. In the **Browser Window**, expand the **DL\_SubsystemPkg** package and the **Blocks** category.
2. Right-click **DataLink**, click **Add New**, and then click **Structure Diagram**. The **New Diagram** dialog box appears.
3. Type “Data Link” and then click **OK**.

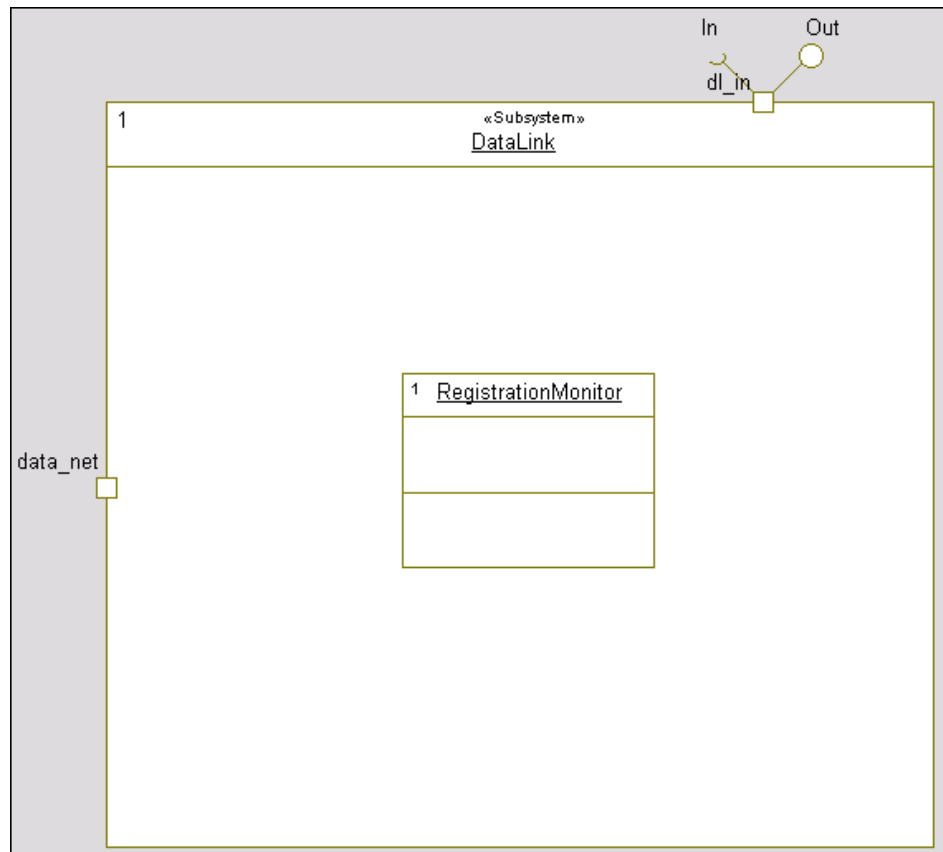
Modeler automatically adds the **Structure Diagrams** category and the **DataLink** diagram to the **Browser Window**. A new drawing area opens with the **DataLink** block and its ports, as defined in the Block Diagram.

### Making the Ports Visible


If the ports are not visible, right-click the **DataLink** block, click **Ports**, and then click **Show All Ports**.

## Drawing Objects

In this procedure, you draw the **RegistrationMonitor** object that represent the activities performed by **DataLink**, as shown in the following illustration.

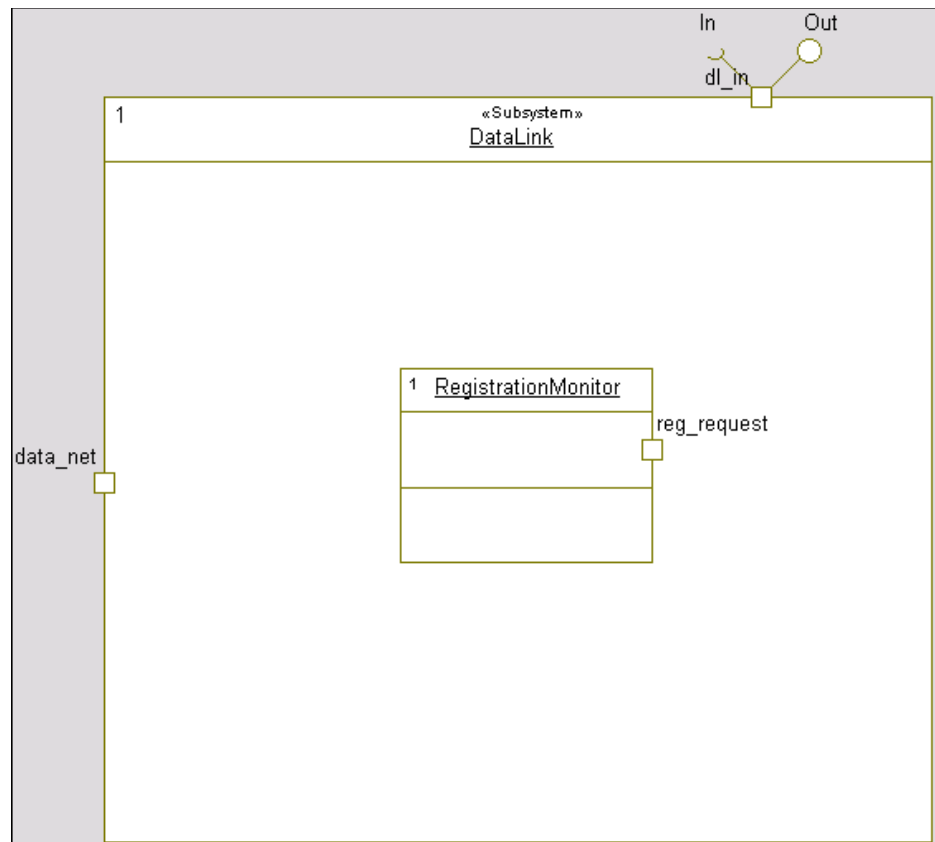


To draw objects:


1. Click the **Object** icon  in the Diagram Tools.
2. Click in the center of **DataLink** block and drag to the desired size. An object with a default name of **object\_n**, where **n** is equal to or greater than 0, is created.
3. Rename the object to **RegistrationMonitor** by typing “RegistrationMonitor” and pressing **Enter**.

### Drawing Ports

In this procedure, you draw a new port on the **RegistrationMonitor** object, as shown in the following illustration.



To draw ports:

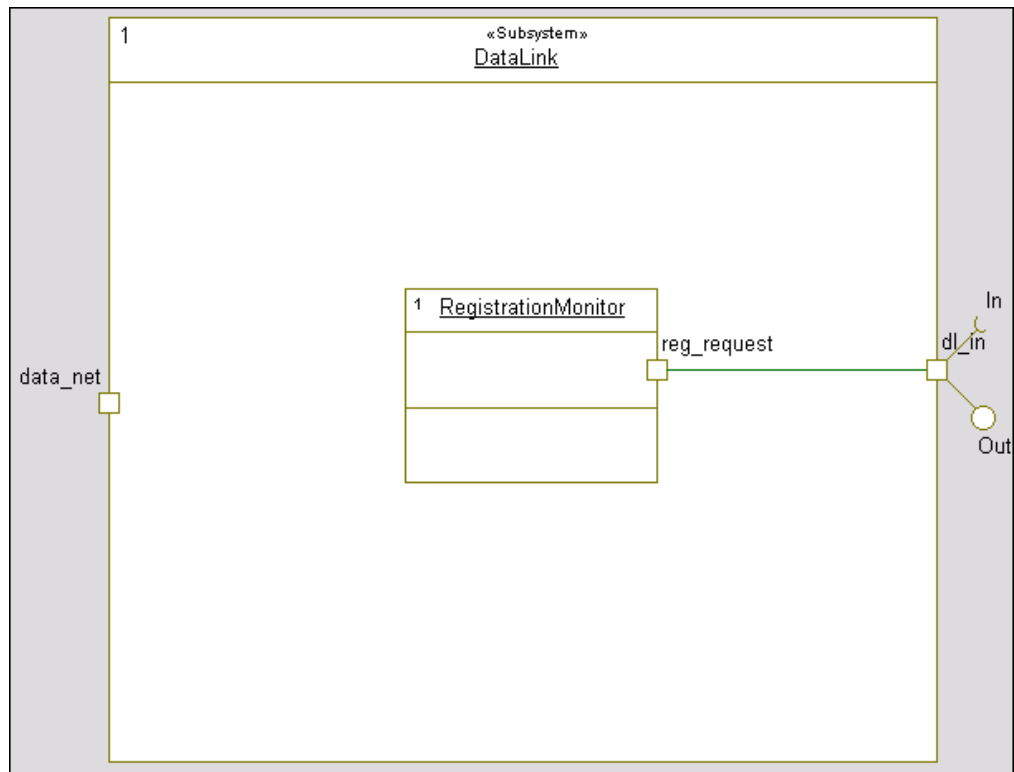
1. Click the **Create Port** icon  in the Diagram Tools.
2. Click the right edge of **RegistrationMonitor** object, type “reg\_request,” and then press **Enter**. This port relays registration requests and results.

### Moving the Ports


To move the **dl\_in** port, click and drag the **dl\_in** port to the right side of the **DataLink** block.

## Drawing Links

In this procedure, you draw a link between an object and a port, as shown in the following illustration.



To draw links between an object and a port:

1. Click the **Link** icon  in the Diagram Tools.
2. Click the **reg\_request** port, click the **dl\_in** port, and then press **Enter**.

### Specifying the Port Contract and Attributes

To specify the port contract and features for a port:

1. Double-click the **reg\_request** port. The **Features** dialog box appears.
2. Click the **General** tab.
3. Under **Attributes**, select the **Behavior** and **Reversed** check boxes.
4. Click the **Contract** tab, click the **Provided** folder icon, and then click **Add**. The **Add new interface** dialog box appears.
5. In the **Interface** list, click **In** and then click **OK**.

Modeler automatically adds the provided and required interfaces.

6. Click the **General** tab.
7. In the **Contract** list, select **In** and then click **Apply**.

Modeler displays a message that the port is not realized. Click **Yes** to add the realization.

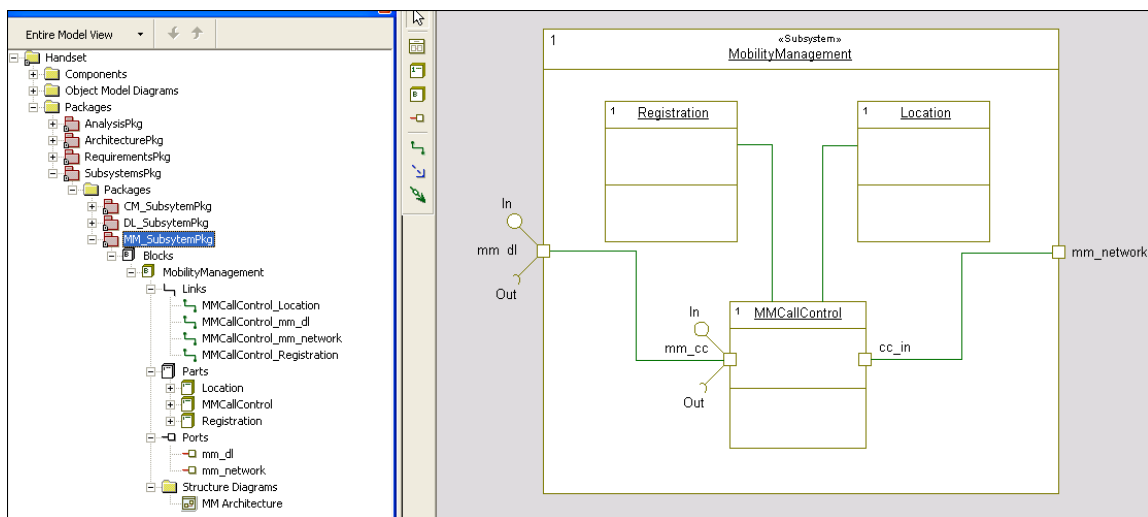
8. Click **OK** to save the changes and to close the **Features** dialog box.

Modeler automatically adds the newly created objects, links, and ports to the **DataLink** block in the **Browser Window**.

## Exercise 2.4: Drawing the MM Architecture Diagram

The MM Architecture diagram decomposes the MobilityManagement block into its subsystems. Mobility Management supports the mobility of users, including registering users on the network and providing their current location. The following figure shows the MM Architecture diagram you create in this module.

In this exercise, you develop a MM Architecture diagram, as shown in the following illustration.



### Creating the MM Architecture Diagram

To create the MM Architecture structure diagram:

1. In the **Browser Window**, expand the **MM\_SubsystemPkg** package and the **Blocks** category.
2. Right-click **MobilityManagement**, click **Add New**, and then click **Structure Diagram**. The **New Diagram** dialog box appears.
3. Type “MM Architecture” and then click **OK**.

Modeler automatically adds the **Structure Diagrams** category and the **MobilityManagement** diagram to the **Browser Window**. A new drawing area opens with the **MobilityManagement** block and its ports, as defined in the Block Diagram.

### Making the Ports Visible

If the ports are not visible, right-click the **MobilityManagement** block, click **Ports**, and then click **Show All Ports**.

### Moving the Ports

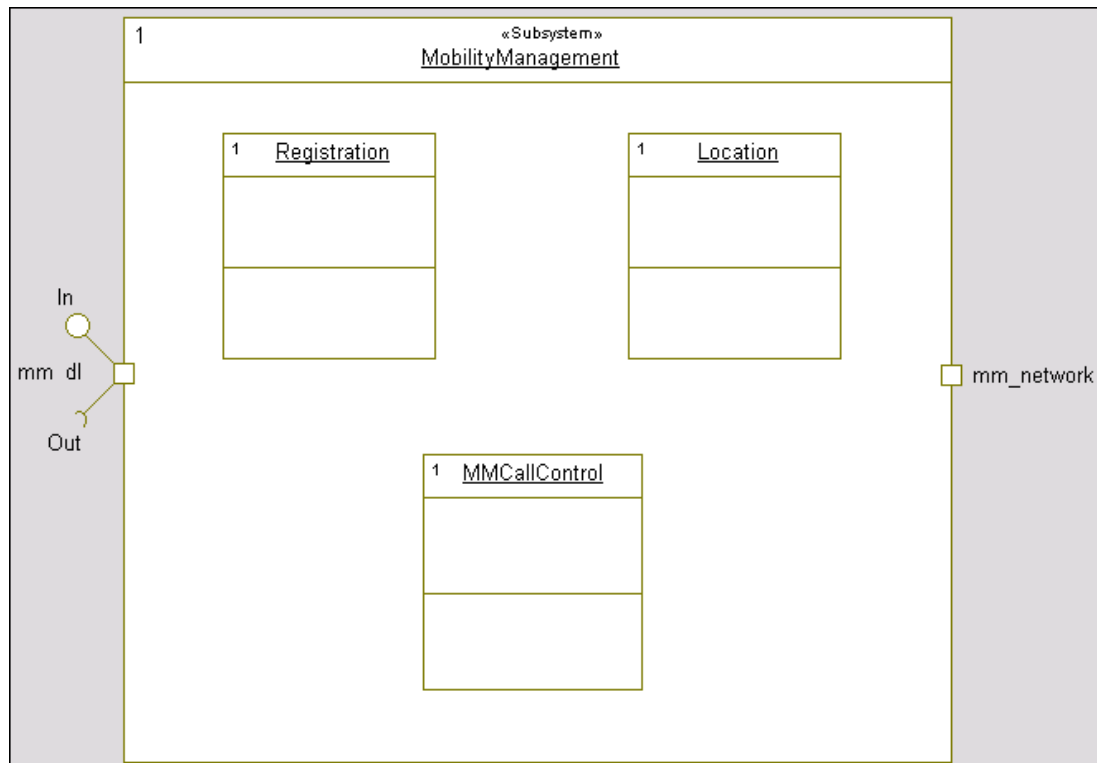
To move the ports:

1. Click and drag the **mm\_network** port to the right side of the **MobilityManagement** block.
2. Click and drag the **mm\_dl** port to the left side of the **MobilityManagement** block.




## Drawing Objects

In this procedure, you draw the objects that represent the activities performed by **MobilityManagement**, as shown in the following illustration.



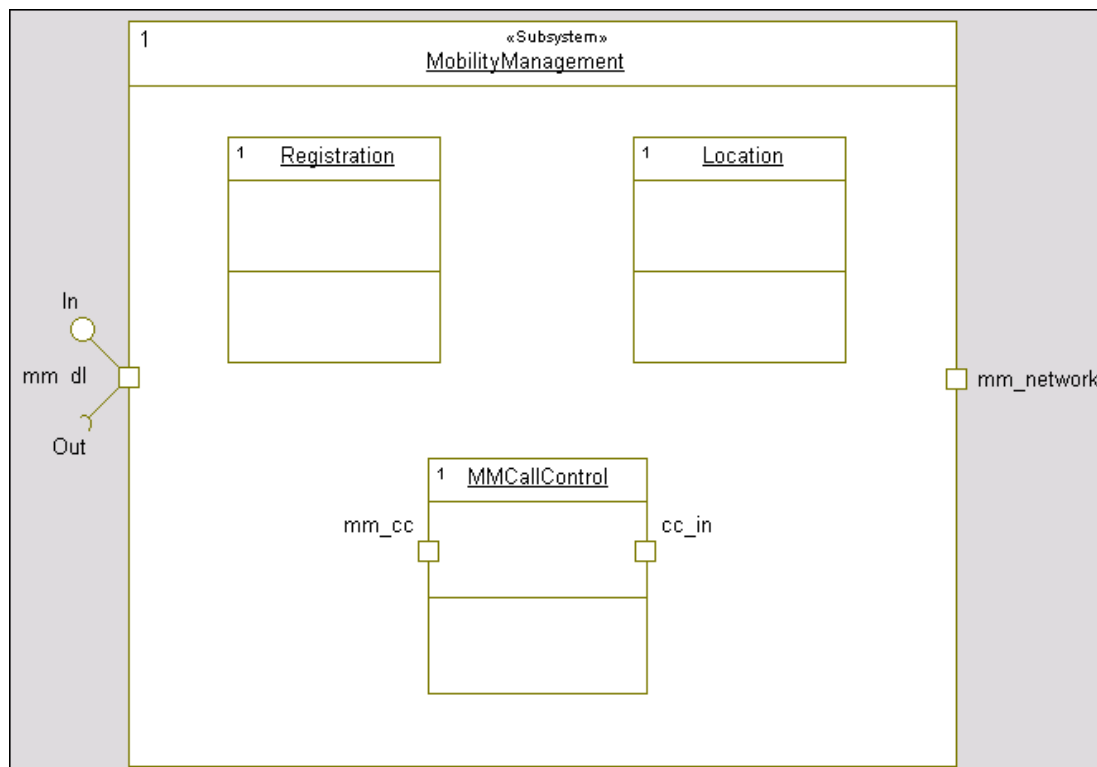
To draw objects:

1. Click the **Object** icon  in the Diagram Tools.
2. Click in the upper, left corner of **MobilityManagement** block and drag to the desired size. An object with a default name of **object\_n**, where **n** is equal to or greater than 0, is created.
3. Rename the object to **Registration** by typing “Registration” and pressing **Enter**.
4. Repeat Steps 1 through 3 to create the following objects:
  - ♦ **Registration** (upper, left corner of **MobilityManagement** block) - maintains the registration status
  - ♦ **Location** (upper, right corner of **MobilityManagement** block) - tracks the location of users


- ♦ **MMCallControl** (lower, center of **MobilityManagement** block) - maintains the logic for **MobilityManagement**

### Drawing Ports

In this procedure, you draw a new port on the RegistrationMonitor object, as shown in the following illustration.

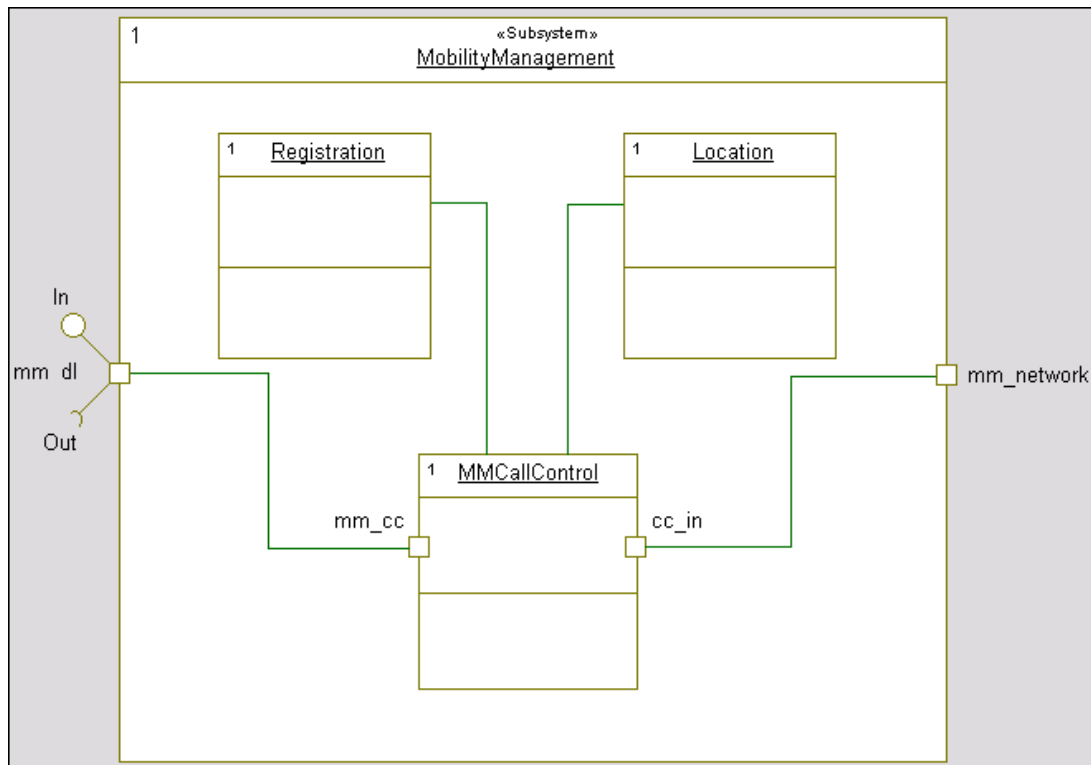


To draw ports:


1. Click the **Create Port** icon  in the Diagram Tools.
2. Click the left edge of **MMCallControl** object, type “mm\_cc,” and then press **Enter**. This port relays information to **ConnectionManagement**.
3. Click the right edge of the **MMCallControl** object, type “cc\_in,” and then press **Enter**. This port sends and receives information from the **DataLink**.


## Drawing Links

In this procedure, you draw links between the objects and ports, as shown in the following illustration.

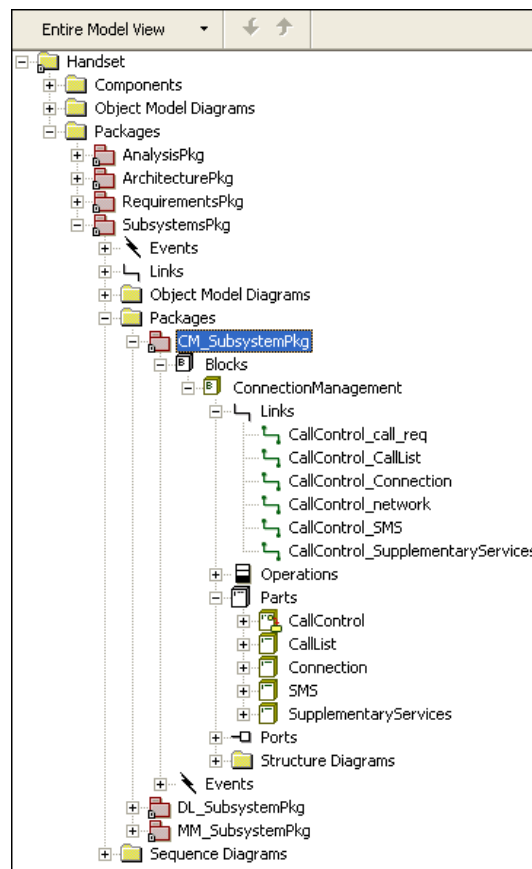


To draw links between objects and ports:

1. Click the **Link** icon  in the Diagram Tools.
2. Click the **mm\_cc** port, click the **mm\_dl** port, and then press **Enter**.
3. Repeat Steps 1 and 2 to create links between:
  - ♦ The **cc\_in** port and the **mm\_network** port
  - ♦ The **MMCallControl** object and the **Registration** object
  - ♦ The **MMCallControl** object and the **Location** object

**Note:** You can use the **Stamp** icon  for repetitive drawing actions. For example, when creating more than one link, click the **Stamp** icon first before clicking the **Link** icon. Once you have created a link, you can keep creating new links without clicking the **Link** icon each time. Click the **Stamp** icon again when you are done drawing links.

4. In the **Browser Window**, expand the **CM\_SubsystemPkg** package, the **ConnectionManagement** category, and:
- ♦ The **Parts** category to view the newly created objects
  - ♦ The **Links** category to view the newly created links, as shown in the following figure.



## Specifying the Port Contract and Attributes

To specify the port contract and features for a port:

1. Double-click the **mm\_cc** port. The **Features** dialog box appears.
2. Click the **General** tab.
3. In the **Contract** list, click **In**. Modeler automatically adds the provided and required interfaces to the **Contract** tab.
4. Under **Attributes**, select the **Behavior** check box. Modeler displays a message that the port is not realized. Click **Yes** to add the realization.
5. Click **OK** to save the changes and to close the **Features** dialog box.

Modeler automatically adds the newly created objects, links, and ports to the **MobilityManagement** block in the **Browser Window**.

## Summary

In this module, you created a system-level structure diagram, and then decomposed that diagram into functions to show how the software systems trace to the system functional blocks. You became familiar with the parts of a structure diagram and created the following:

- ♦ Blocks
- ♦ Objects
- ♦ Ports
- ♦ Flows
- ♦ Links

You are now ready to proceed to the next module, where you define how the system components are interconnected using object model diagrams.

### Note

---

With Rational Rhapsody Developer, you can build and animate the parts of the model you created. This lets you determine whether the model meets the requirements and identify defects early on in the design process.



# Module 3:

## Creating Object Model Diagrams

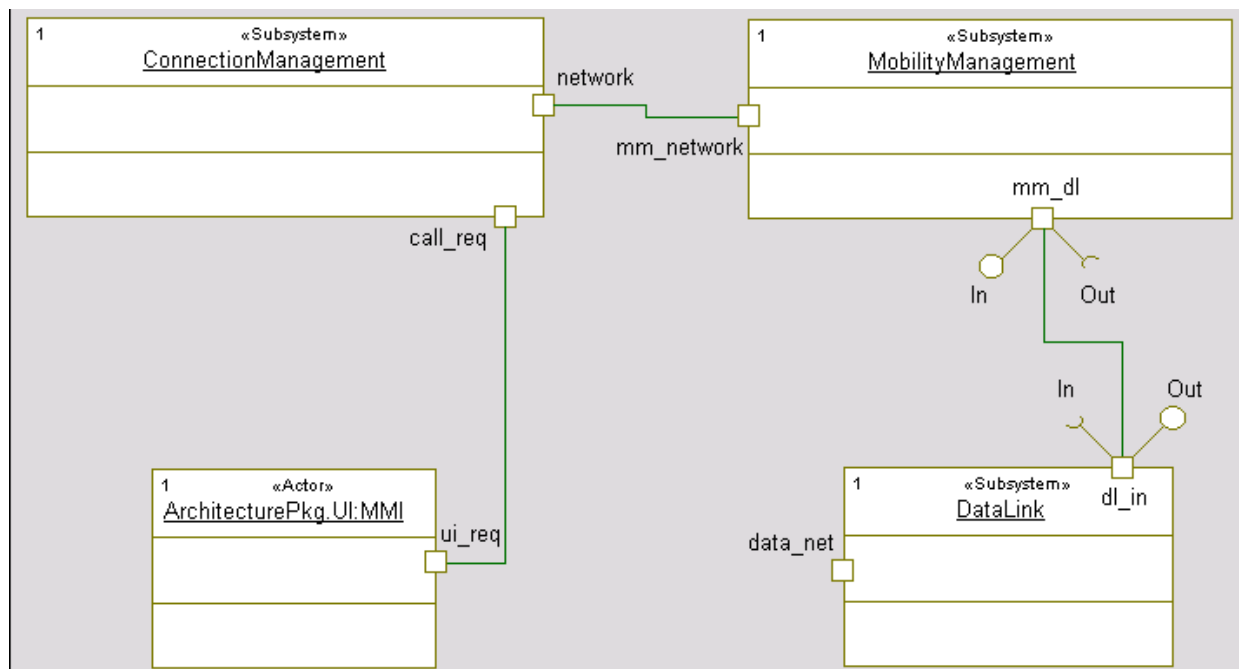
**Object Model Diagrams** (OMDs) specify the structure of the classes, objects, and interfaces in the system and the static relationships that exist among them. Rational Rhapsody Developer code generator directly translates the elements and relationships modeled in OMDs into production quality C++, C, Java, and Ada source code.

### Goals of this Module

You create an OMD, showing the system components interconnected at the subsystem level, and identify the port connections and the information flow between components as links.

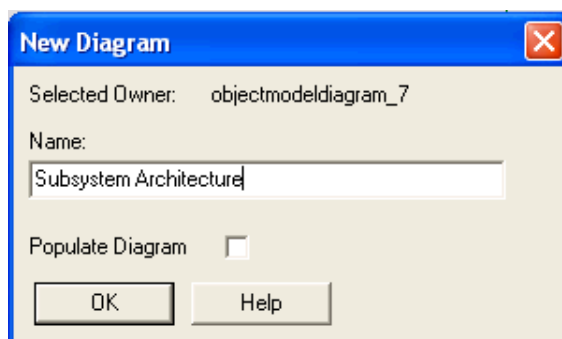
### Exercise 3.1: Creating the Subsystem Architecture

In this exercise, you develop a Subsystem Architecture OMD, as shown in this illustration.



To create the Subsystem Architecture OMD:

1. In the **Browser Window**, right-click the **SubsystemsPkg** package, click **Add New**, and then click **Object Model Diagram**. The **New Diagram** dialog box appears.



2. Type “Subsystem Architecture” and click **OK**.

Modeler automatically creates the **Object Model Diagrams** category and the new OMD to the **SubsystemPkg** package in the **Browser Window**. A new drawing area opens.

## Drawing the Subsystem Architecture OMD

The Subsystem Architecture OMD identifies how the system components are interconnected at the subsystem level. It shows the realization of flows between blocks and objects through links and ports. Flows are used for high level analysis, and links are used for executability (realization of flows).

Draw an object model diagram using the following general steps:

1. Add blocks.
2. Add objects.
3. Draw links.



### Adding Blocks

The Subsystem Architecture OMD contains the subsystems defined in the block diagram.

To add these blocks to the OMD:

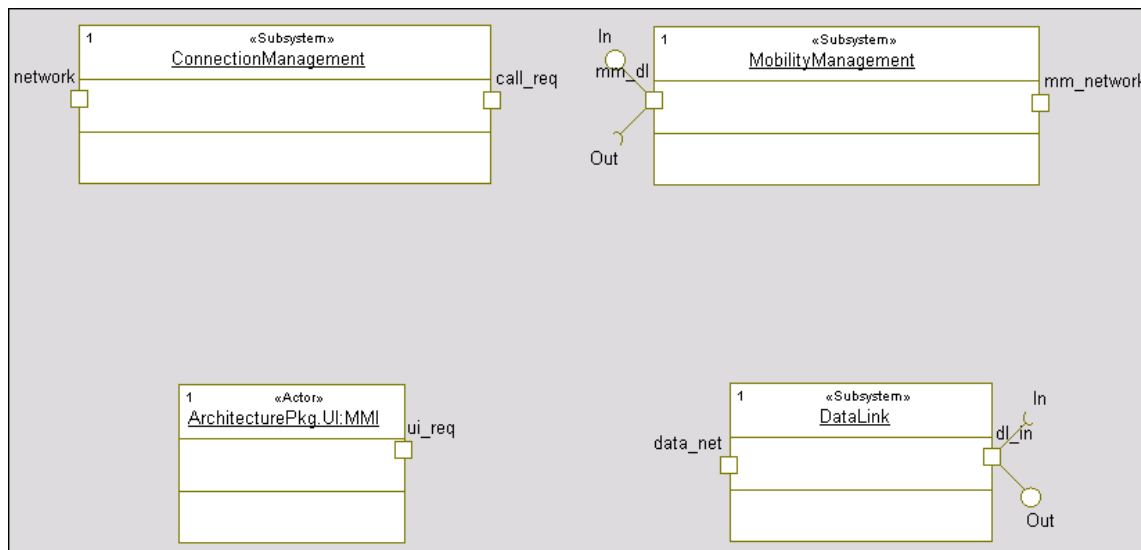
1. In the **Browser Window**, expand the **CM\_SubsystemPkg** package and **Blocks** category.
2. Click the **ConnectionManagement** block and drag it to the upper, left side of the drawing area. The **ConnectionManagement** block and its ports are added to the diagram.
3. In the **Browser Window**, expand the **MM\_SubsystemPkg** package and **Blocks** category.
4. Click the **MobilityManagement** block and drag it to the upper, right side of the drawing area.
5. In the **Browser Window**, expand the **DL\_SubsystemPkg** package and **Blocks** category.
6. Click the **DataLink** block and drag it to the lower, right side of the drawing area.

**Note:** Use the **Display Options** dialog box to set **Name only** as the **Display name** for the blocks.

### Adding Objects

The Subsystem Architecture OMD contains the **UI** object defined in the block diagram. The **UI** object interacts with **ConnectionManagement** to establish and clear calls, and request and receive data services.

In this procedure, you add the **UI** object, as shown in the following illustration.



To draw the object:

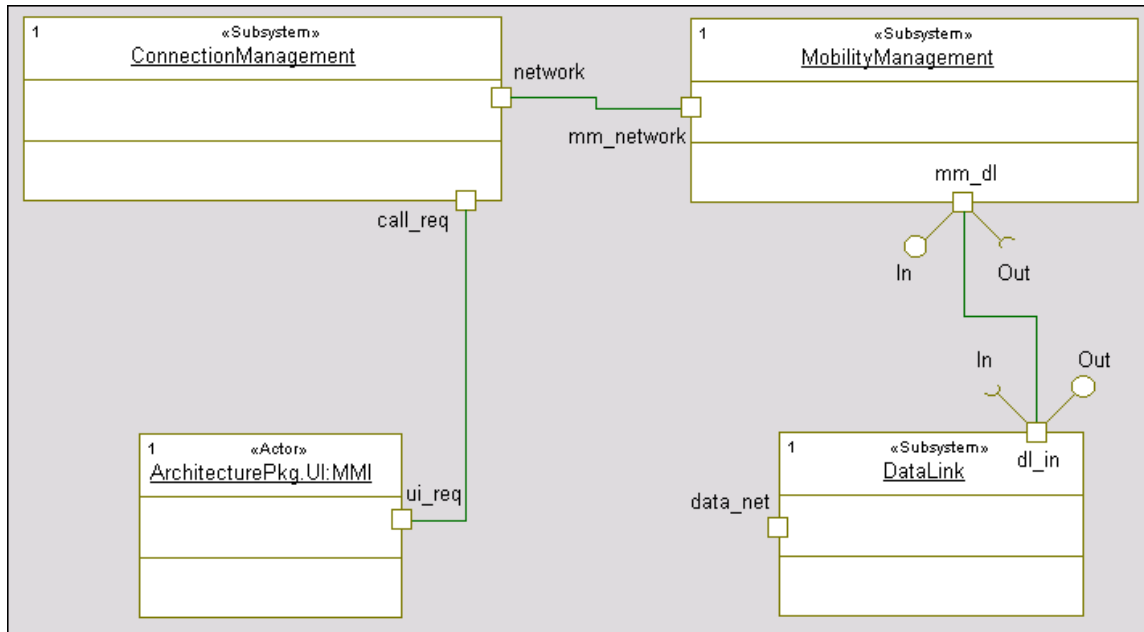
1. In the **Browser Window**, expand the **ArchitecturePkg** package and the **Objects** category.
2. Select the **UI** object and drag it to the bottom, left side of the OMD. The **UI** object and its port are added to the diagram.

**Note:** When Modeler adds the blocks and objects to the diagram, it places the ports on the boundary. Change the placement of ports by selecting a port and dragging it to another location, if necessary.


## Drawing Links

The Subsystem Architecture OMD shows the flow of information, realized as links. A link is an instantiation of an association.

In this procedure, you draw links between the objects and ports, as shown in the following illustration.



To create a link:

1. Click the **Link** icon  in the Diagram Tools.
2. Click the **network** port, click the **mm\_network** port, and then press **Enter**.
3. Repeat Steps 1 and 2 to create links between:
  - ♦ The **call\_req** port and the **ui\_req** port
  - ♦ The **mm\_dl** port and the **dl\_in** port

## Summary

In this module, you created an OMD, which shows how the system components are interconnected OMDs. You became familiar with the parts of an OMD and added the following elements:

- ◆ Blocks
- ◆ Objects
- ◆ Links

You are now ready to proceed to the next module, where you define the message exchange between subsystems and subsystem modules when placing a call using sequence diagrams.

### Note

---

With Rational Rhapsody Developer, you can build and animate the parts of the model you created. This lets you determine whether the model meets the requirements and identify defects early on in the design process.

# Module 4:

## Creating Sequence Diagrams

---

**Sequence Diagrams** (SDs) describe how structural elements communicate with one another over time, and identify the required relationships and messages. SDs can be used at different levels of abstraction. At higher levels of abstractions, SDs show the interactions between actors, use cases, and blocks. At lower levels of abstraction and for implementation, SDs show the communication between classes and objects.

Sequence diagrams have an executable aspect and are a key animation tool in the Rational Rhapsody Developer and Designer for Systems Engineers editions. When you animate a model, Rational Rhapsody Developer dynamically builds SDs that record the object-to-object or block-to-block messaging.

### Goals for this Module

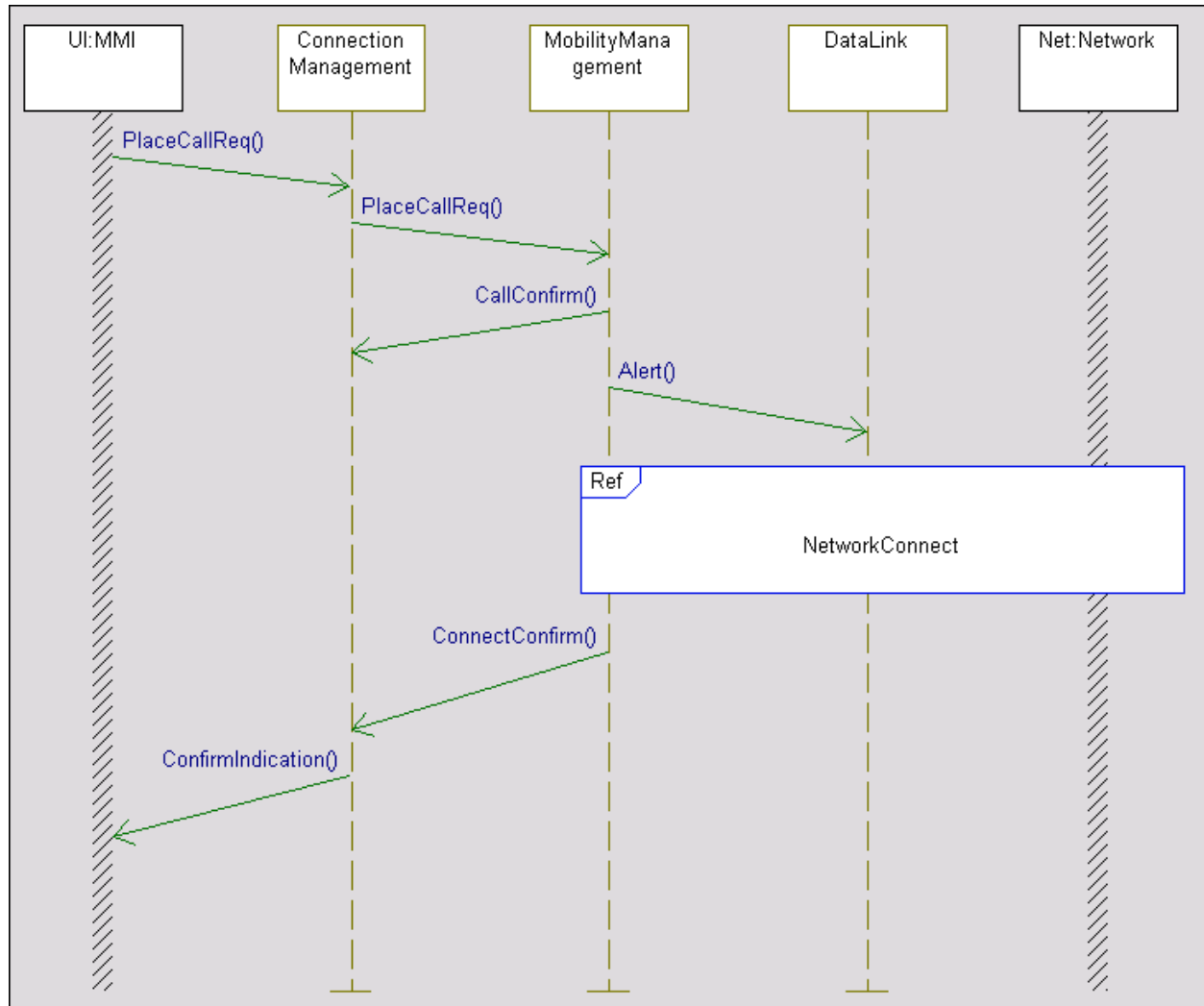
In this module, you create the following sequence diagrams:

- ♦ **Place Call Request Successful** - identifies the message exchange when placing a call
- ♦ **NetworkConnect** - identifies the scenario of connecting to the network
- ♦ **Connection Management Place Call Request Success** - identifies the message exchange between functions when placing a call

For ease of presentation, this section includes all sequence diagrams. Depending on your workflow, you might first identify the high-level communication scenario of placing a call and then refine the high-level structure diagram and OMDs, before defining the communication scenarios of the modules.

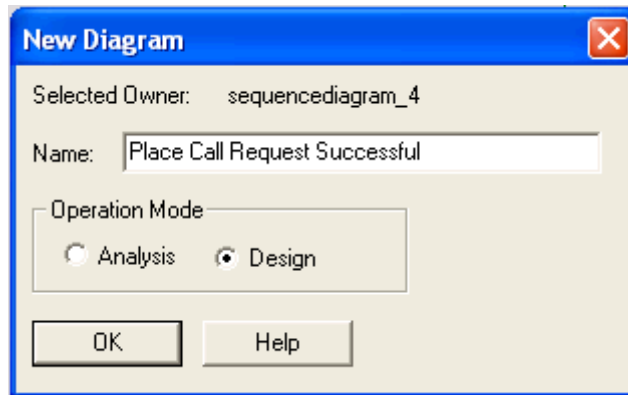
## Exercise 4.1: Creating the Place Call Request Successful SD

In this exercise, you develop a Place Call Request Successful SD, as shown in the following illustration.



To create a new sequence diagram:

1. In the **Browser Window**, right-click the **SubsystemsPkg** package, click **Add New**, and then click **Sequence Diagram**. The **New Diagram** dialog box appears.



2. Type "Place Call Request Successful."
3. Under **Operation Mode**, click **Design** and then click **OK** to close the dialog box. Modeler automatically creates the **Sequence Diagrams** category and the new SD to the **SubsystemPkg** package in the **Browser Window**. A new drawing area opens.

## Drawing the Place Call Request Successful Diagram

The **Place Call Request Successful** diagram shows how subsystems interact during the scenario of successfully requesting to place a call. It identifies the order and exchange of messages between the objects and blocks as represented in the block diagram. By describing the flows through scenarios, you create the logical interfaces of the blocks. For example, if a message is shown going into the **DataLink** block, you can see that the message belongs to the block as an event or operation.

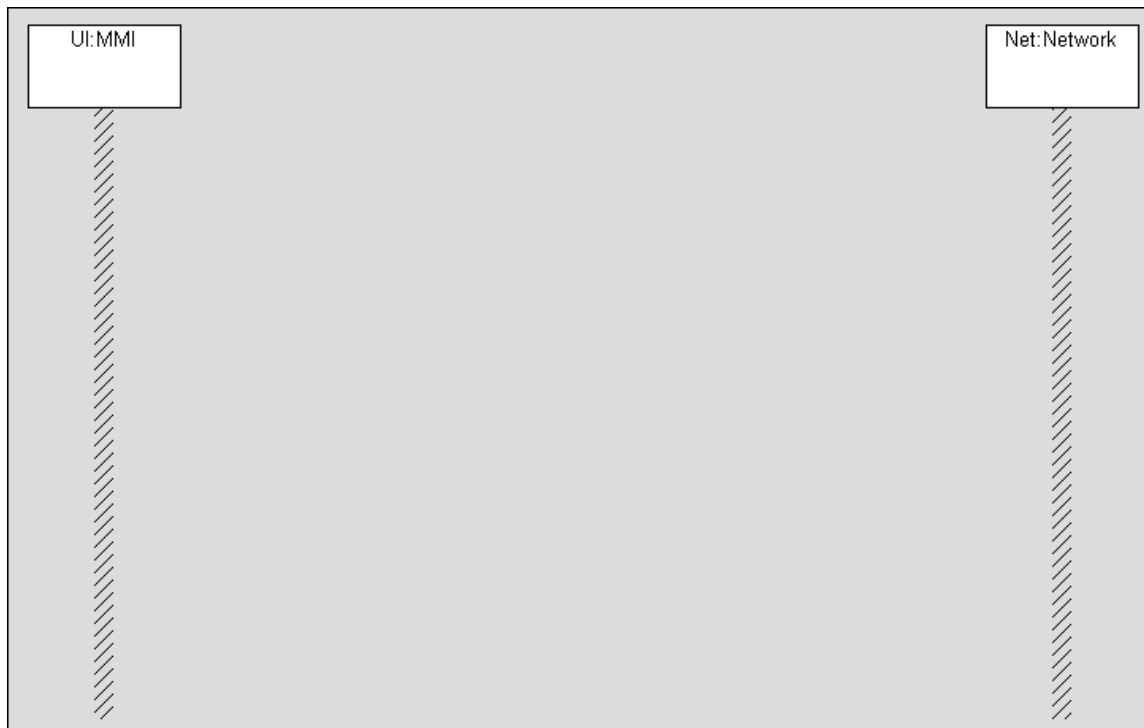
Draw a sequence diagram using the following general steps:

1. Draw the actor lines.
2. Draw classifier roles.
3. Draw messages.
4. Draw interaction occurrences.

### Drawing Actor Lines

**Actor Lines** show how actors participate in the scenario. Actors are represented as instance lines with hatching. In use case diagrams and sequence diagrams, actors describe the external elements with which the system context interacts.

In this procedure, you draw the actor lines that represent the two objects, **MMI** and **Network**, as shown in the following illustration.



To draw actor lines:

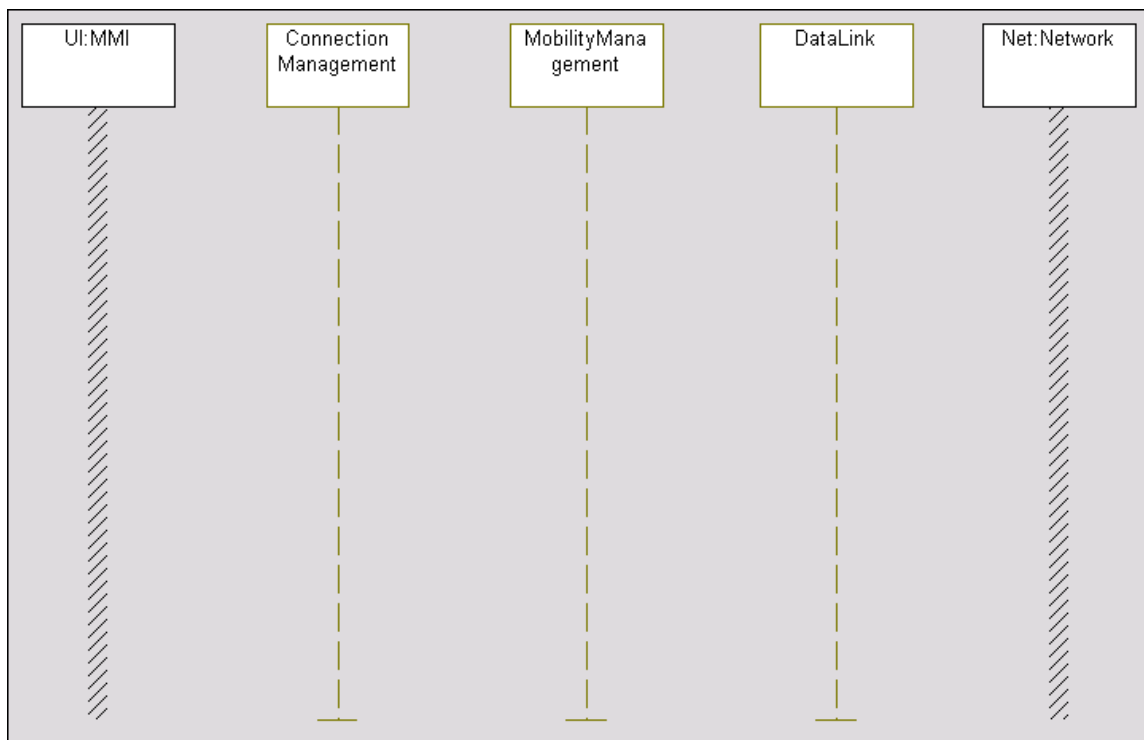
1. In the **Browser Window**, expand the **ArchitecturePkg** package and the **Objects** category.
2. Click the **UI** object and drag it at the beginning of the SD. An actor line is created.
3. Click the **Net** object and drag it at the end of the SD.



## Drawing Classifier Roles

**Classifier Roles** or instance lines are vertical timelines labeled with the name of an instance, which indicate the lifecycle of classifiers or blocks that participate in the scenario. They represent a typical instance in the scenario being described. Classifier roles can receive messages from or send messages to other instance lines. Time proceeds downward on the vertical axis.

In this procedure, you draw the classifier roles that represent the system components, **ConnectionManagement**, **MobilityManagement**, and **DataLink**, as shown in the following illustration.



To draw classifier roles:

1. In the **Browser Window**, expand the **SubsystemsPkg** package, the **CM\_SubsystemPkg** package, and the **Blocks** category.
2. Click **ConnectionManagement** and drag it next to the **UI** object. A classifier role with the name of the function in the names pane is created.
3. In the **Browser Window**, expand the **MM\_SubsystemPkg** package and the **Blocks** category.
4. Click **MobilityManagement** and drag it next to **ConnectionManagement**.

5. In the **Browser Window**, expand the **DL\_SubsystemPkg** package and the **Blocks** category.
6. Click **DataLink** and drag it next to **MobilityManagement**.

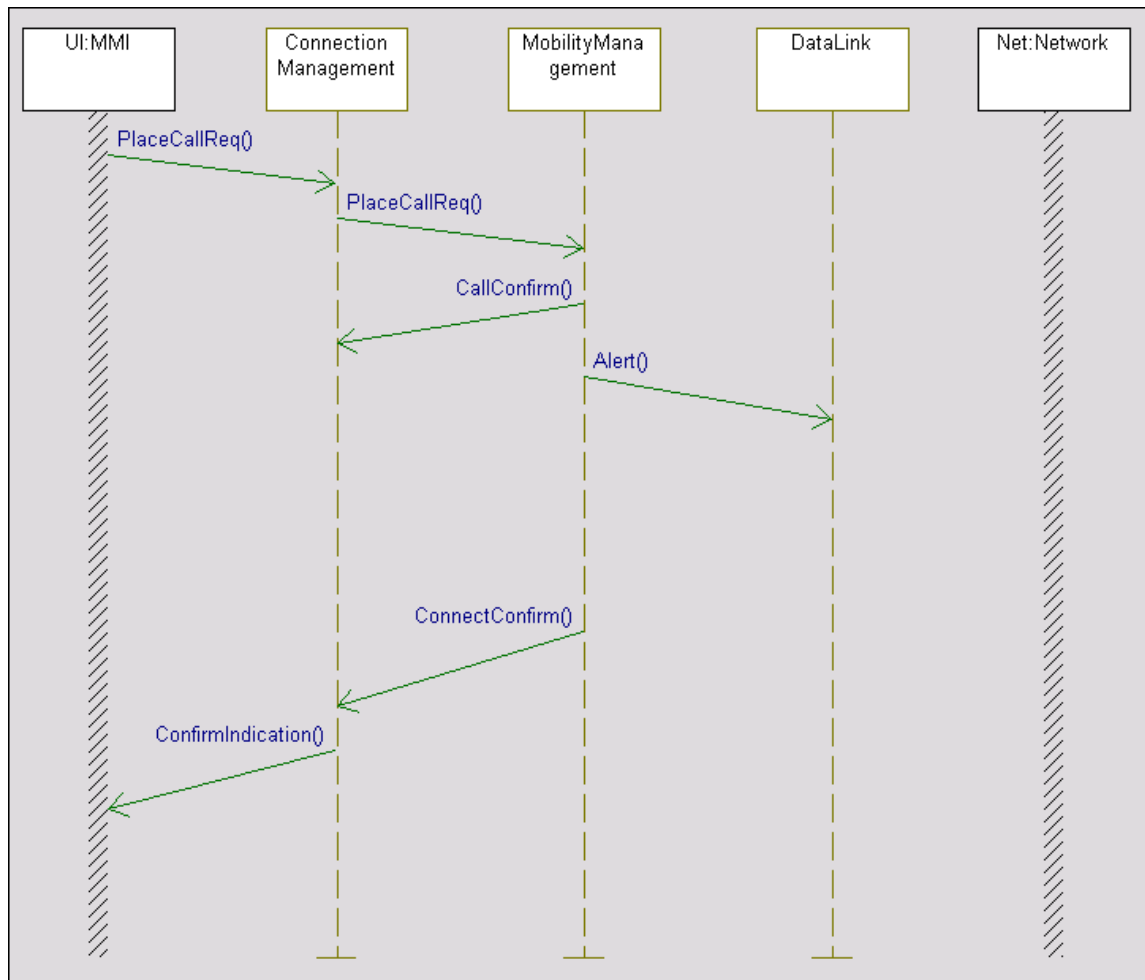
**Note:** To add white space to (or remove it from) a sequence diagram (such as between actors lines and classifier roles), press **Shift** and drag the actor line or classifier role to its new location.

### Drawing Messages


A **Message** represents an interaction between objects, or between an object and the environment. A message can be an event, a triggered operation, or a primitive operation.

In this procedure, you draw events that represent the exchange of information when placing a call, as shown in the following illustration. These events are as follows:

- ♦ The **UI** actor issues a request to connect when placing a call.
- ♦ Call and connect confirmations occur between **MobilityManagement** and **ConnectionManagement**.
- ♦ Alerts occur between **MobilityManagement** and **DataLink**.
- ♦ The user receives confirmation from **ConnectionManagement**.



To draw messages:

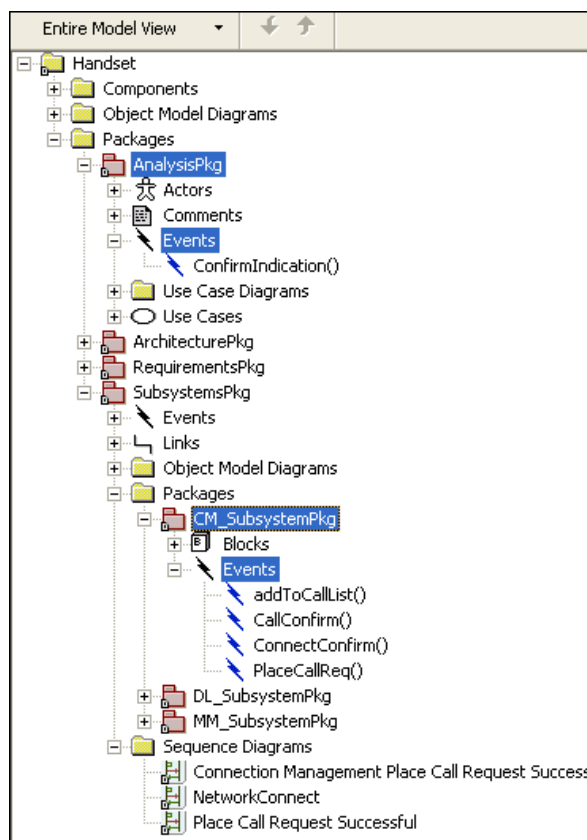
1. Click the **Message** tool  in the Diagram Tools.
2. Click the **UI** actor line to show that the first message comes from the UI actor when the user issues the command to place a call request.
3. Click the **ConnectionManagement** line to create a downward-slanted diagonal line. A message with the default name **event\_n()**, where **n** is an incremental integer starting with 0, is created.
4. Rename the message **PlaceCallReq**. Modeler asks if you want to realize the message. Click **Yes**.

5. Draw the following messages:

- ♦ From **ConnectionManagement** to **MobilityManagement** called **PlaceCallReq**
- ♦ From **MobilityManagement** to **ConnectionManagement** called **CallConfirm**
- ♦ From **MobilityManagement** to **DataLink** called **Alert**
- ♦ From **MobilityManagement** to **ConnectionManagement** called **ConnectConfirm**
- ♦ From **ConnectionManagement** to the **UI** actor called **ConfirmIndication**

**Note:** When prompted, click **Yes** to realize each new message.

6. In the **Browser Window**, expand the **CM\_SubsystemPkg** package and the **Events** category to view the new realized events: **PlaceCallReq**, **CallConfirm**, and **ConnectConfirm**, and expand the **AnalysisPkg** package and the **Events** category to view the new realized event: **ConfirmIndication**, as shown in the following illustration.

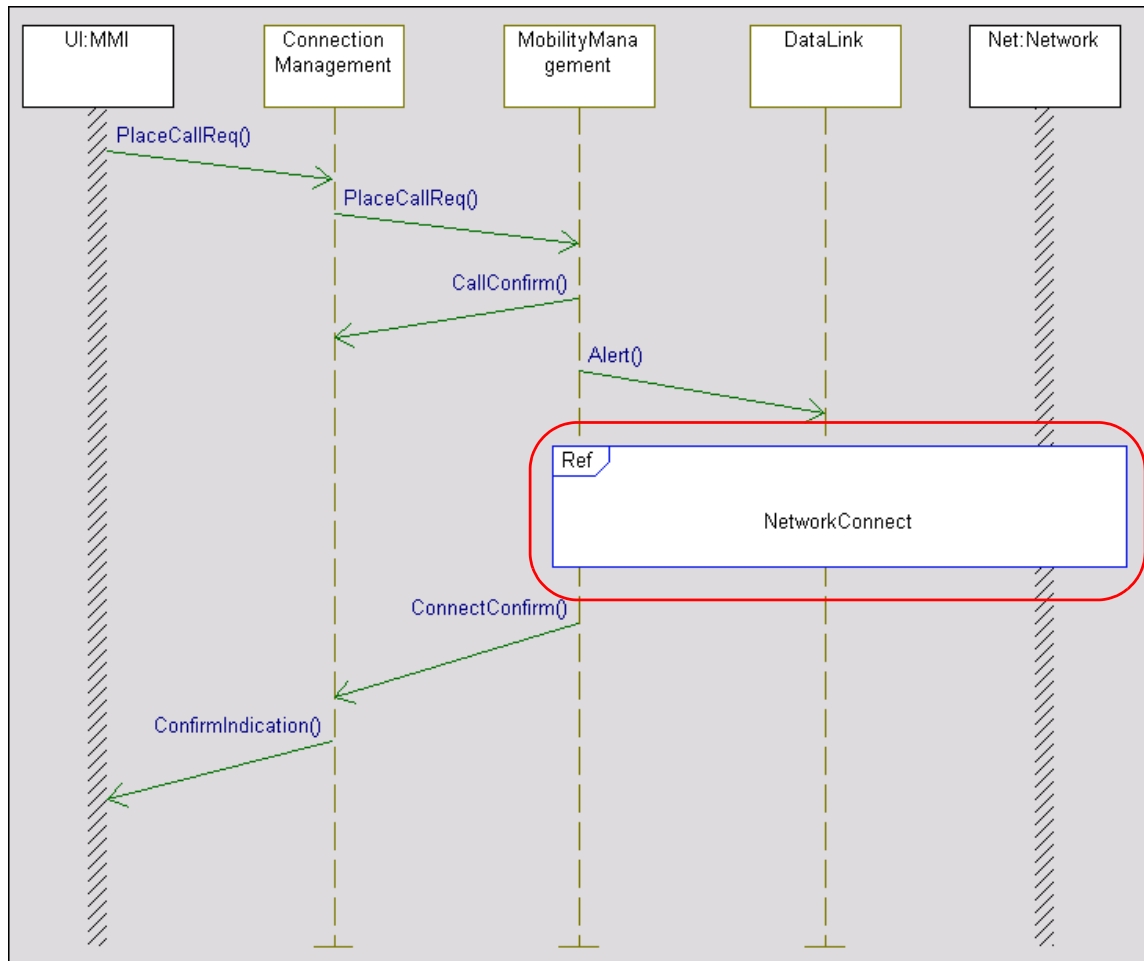


7. Click **ConfirmIndication** and drag it to the **Events** category under the **CM\_SubsystemPkg** package.


## Drawing an Interaction Occurrence

An **Interaction Occurrence** (or reference sequence diagram) enables you to refer to another SD from within an SD. It allows you to break down complex scenarios into smaller scenarios that can be reused.

In this procedure, you draw an interaction occurrence, as shown in the following illustration.



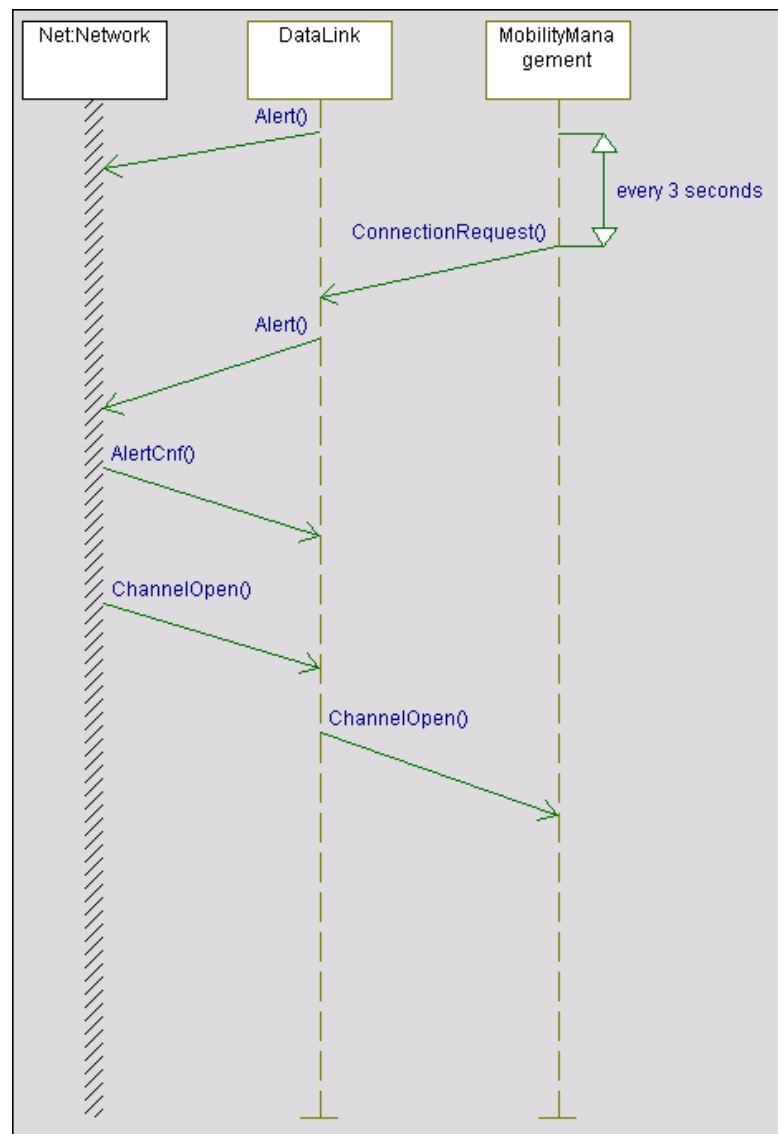
To draw an interaction occurrence:

1. Click the **Interaction Occurrence** tool  in the Diagram Tools.
2. Draw the interaction occurrence below the **Alert** message and across the **MobilityManagement** instance line and the **Net** actor line. The interaction occurrence appears as a box with the **ref** label in the top corner.
3. Type “NetworkConnect” and press **Enter**.

## Exercise 4.2: Drawing the NetworkConnect SD

The NetworkConnect SD shows the scenario of connecting to the network when placing a call. It is a generic interaction that can be reused within voice, data, supplementary services, and short message services.

In this exercise, you develop a NetworkConnect SD, as shown in the following illustration.

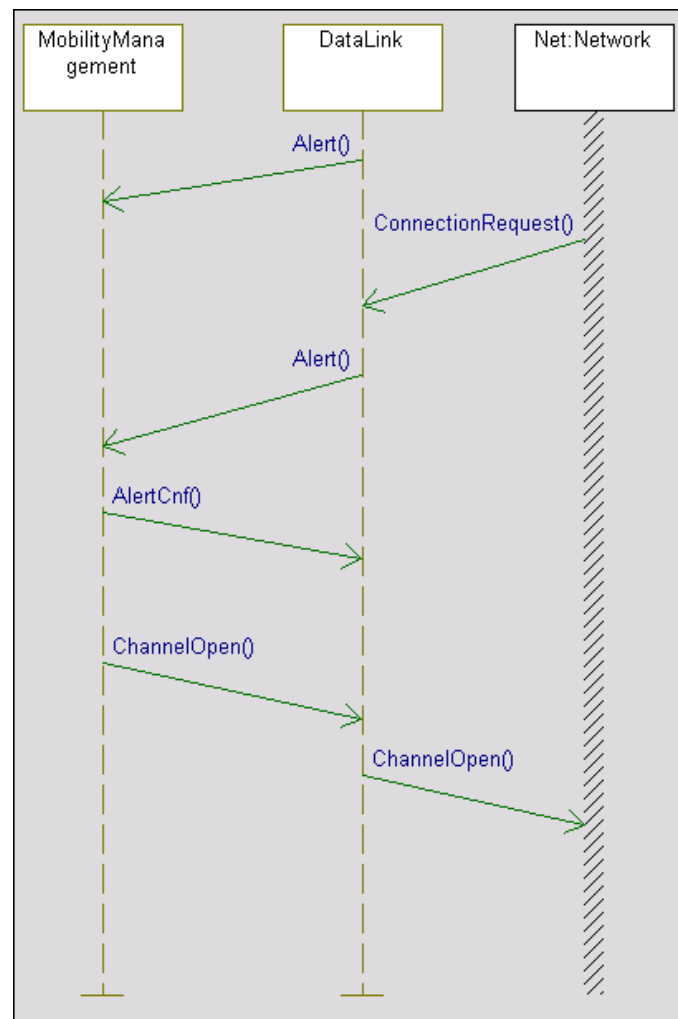


### Creating the NetworkConnect Diagram

To create the NetworkConnect SD, right-click the interaction occurrence in the Place Call Request Successful SD and click **Create Reference Sequence Diagram**. Modeler opens the new diagram in the drawing area containing the three functions the interaction occurrence crosses, and adds the SD to the browser window.


### Drawing Messages

In this procedure, you draw events, as shown in the following illustration.





To draw messages:

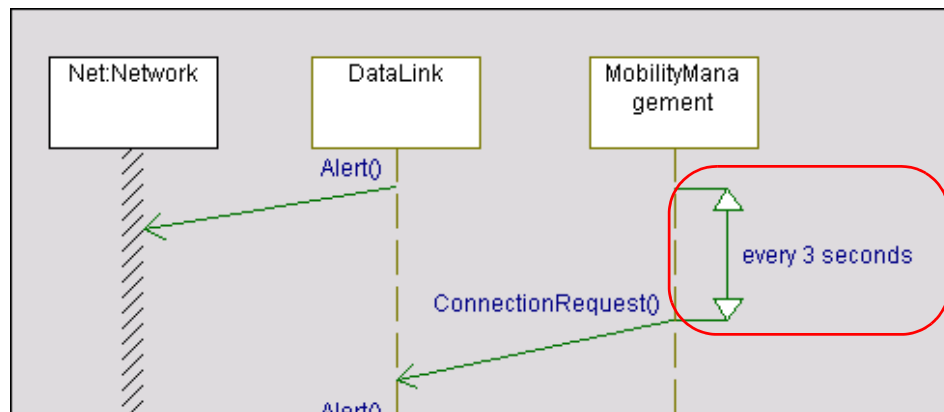
1. Click the **Message** tool  in the Diagram Tools.
2. Draw the following messages:
  - ♦ From **DataLink** to **Net** called **Alert**
  - ♦ From **MobilityManagement** to **DataLink** called **ConnectionRequest**
  - ♦ From **DataLink** to **Net** called **Alert**
  - ♦ From **Net** to **DataLink** called **AlertCnf**
  - ♦ From **Net** to **DataLink** called **ChannelOpen**
  - ♦ From **DataLink** to **MobilityManagement** called **ChannelOpen**

**Note:** When prompted, click **Yes** to realize each new message.


### Drawing Time Intervals

Sequence diagrams can specify the maximum amount of time that can elapse between two points. A **Time Interval** is a vertical annotation that shows how much (real) time should pass between two points in the scenario. The name of the time interval is free text; it is not constrained to be a number or unit.

In this procedure, you set a time interval of 3 seconds in which **MobilityManagement** checks for a connection request, as shown in the following illustration.



To draw a time interval:

1. Click the **Time Interval** tool  in the Diagram Tools.
2. Click near the top of the **MobilityManagement** line, then click the origin of the **ConnectionRequest** event.

Modeler draws two horizontal lines at the start and end points of the time interval, and a two-headed vertical arrow in the middle, indicating the time lapse between the two points.

3. Type “every 3 seconds” and press **Enter**.

## Moving Events

The **ArchitecturePkg** package contains the sequence diagrams that detail the design of the system and the flow of information. When you draw messages on the SDs, several messages are added to the **Events** category in the **ArchitecturePkg** package. To make these events available for model execution, you need to move them from the **ArchitecturePkg** package to the **SubsystemsPkg** package.

To move events:

1. In the **Browser Window**, expand the **ArchitecturePkg** package and the **Events** category.
2. Click **Alert** and drag it to the **SubsystemsPkg** package.

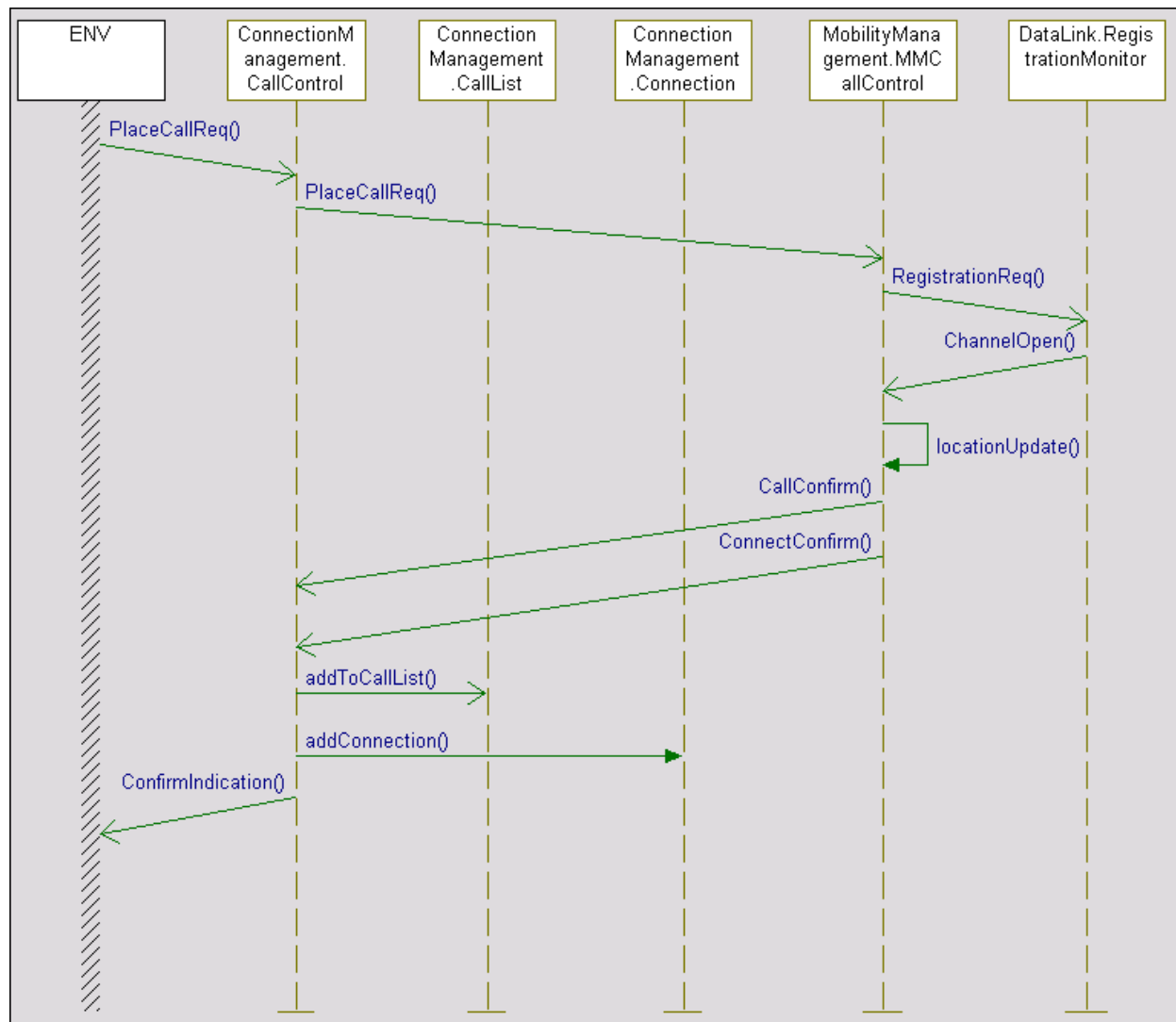
Modeler automatically creates the **Events** category in the **SubsystemsPkg** package and adds the **Alert** event.

3. Repeat Steps 1 and 2 for the remaining events.

## Exercise 4.3: Drawing the Connection Management Place Call Request Success SD

The Connection Management Place Call Request Success SD shows the interaction of the subsystems. It identifies the part decomposition interaction when placing a successful call.

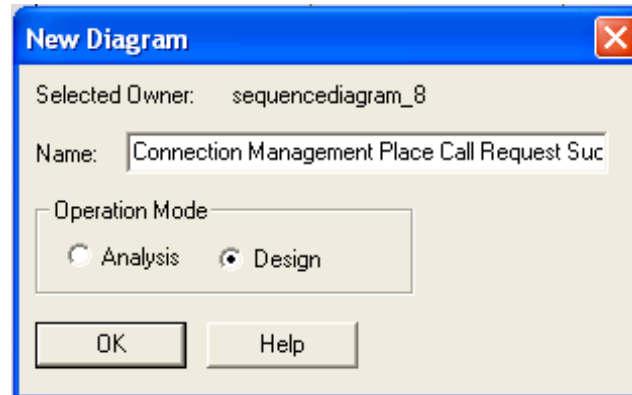
In this exercise, you develop a Connection Management Place Call Request Success SD, as shown in the following illustration.



## Creating the Connection Management Place Call Request Success SD

To create the Connection Management Place Call Request Success SD:

1. In the **Browser Window**, expand the **SubsystemsPkg** package, right-click **Sequence Diagrams** and click **Add New Sequence Diagram**. The **New Diagram** dialog box appears.




2. Type "Connection Management Place Call Request Success."
3. Under **Operation Mode**, click **Design** and then click **OK** to close the dialog box.
4. Click **OK** to close the dialog box.

Modeler automatically creates the **Sequence Diagrams** category and the new SD to the **SubsystemPkg** package in the **Browser Window**. A new drawing area opens.

### Drawing the System Border

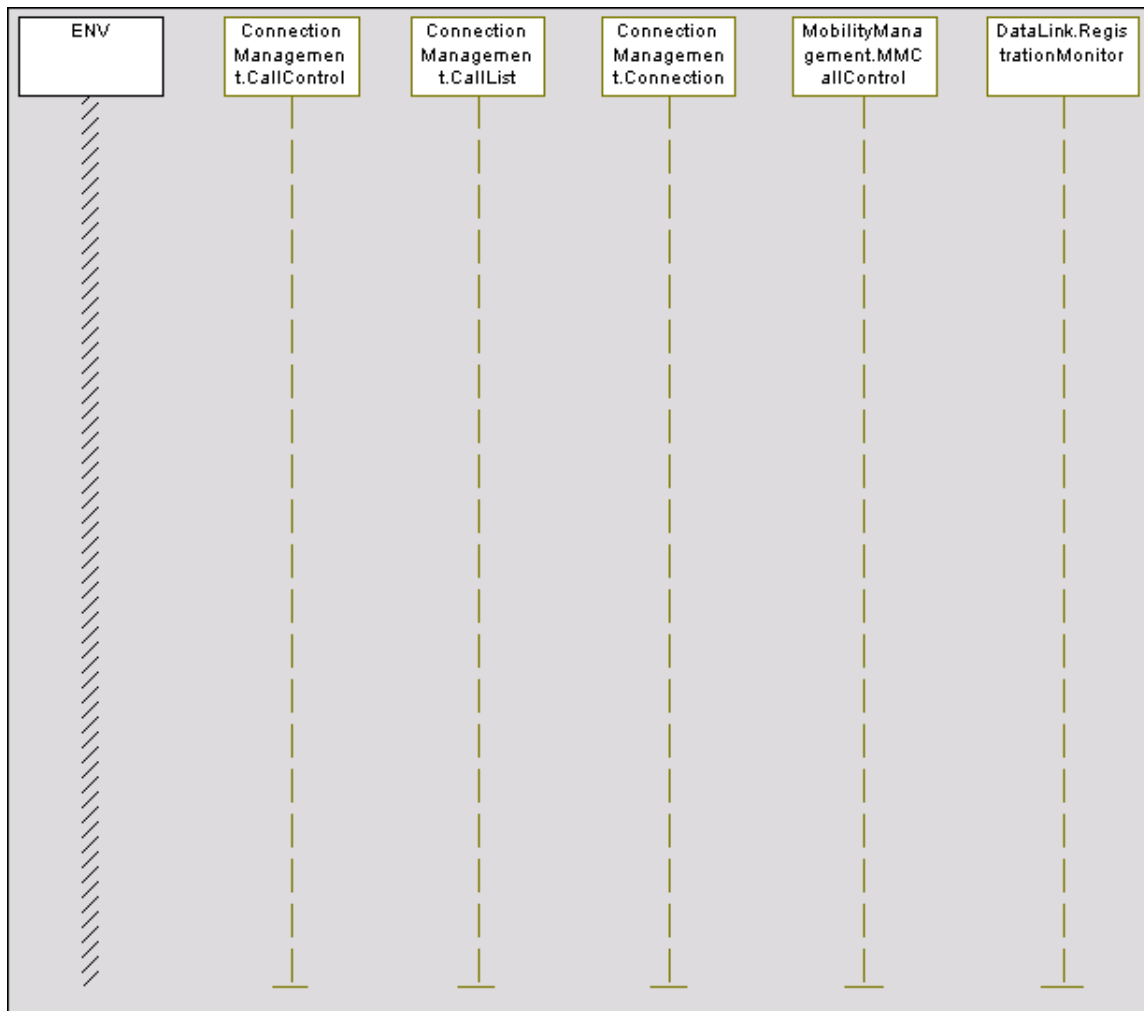
The **System Border** represents the environment and is shown as a column of diagonal lines. Events or operations that do not come from instance lines are drawn from the system border. You can place a system border anywhere an instance line can be placed; the most usual locations are the left or right side of the SD.

To draw the system border:

1. Click the **System border** tool  in the Diagram Tools.
2. Click on the left side of the diagram to place the border.

### Drawing Classifier Roles

In this procedure, you draw the classifier roles that represent the internal functions of the subsystems as shown in the following illustration.



To draw classifier roles:

1. In the **Browser Window**, expand the **ConnectionManagement** block and the **Parts** category.
2. Click **CallControl** and drag it next to the system border. A classifier role with the name of the function in the names pane is created.
3. Click **CallList** and drag it next to **CallControl**.


4. Click **Connection** and drag it next to **CallList**.
5. In the **Browser Window**, expand the **MobilityManagement** block and the **Parts** category.
6. Click **MMCallControl** and drag it next to **Connection**.
7. In the **Browser Window**, expand the **DataLink** block and the **Parts** category.
8. Click **RegistrationMonitor** and drag it next to **MMCallControl**.

### Drawing Messages

When the system receives a request to place a call, it validates and registers the user; once registered, it monitors the user's location. The call and connection are confirmed, the connection is set up, and confirmation is provided.

In this procedure, you draw events using slanted lines, and primitive operations using horizontal lines and messages-to-self.

To draw messages:

1. Click the **Message** tool  in the Diagram Tools.
2. Draw the following events using slanted lines:
  - ♦ From the system border to the **CallControl** line called **PlaceCallReq**
  - ♦ From **CallControl** to **MMCallControl** called **PlaceCallReq**
  - ♦ From **MMCallControl** to **RegistrationMonitor** called **RegistrationReq**
  - ♦ From **RegistrationMonitor** to **MMCallControl** called **ChannelOpen**
3. Draw a message-to-self on the **MMCallControl** instance line called **locationUpdate**.

**Note:** Message names are case-sensitive.

4. Draw the following events:
  - ♦ From **MMCallControl** to **CallControl** called **CallConfirm**
  - ♦ From **MMCallControl** to **CallControl** called **ConnectConfirm**
5. Draw the following primitive operations using horizontal lines:
  - ♦ From **CallControl** to **CallList** called **addToCallList**
  - ♦ From **CallControl** to **Connection** called **addConnection**

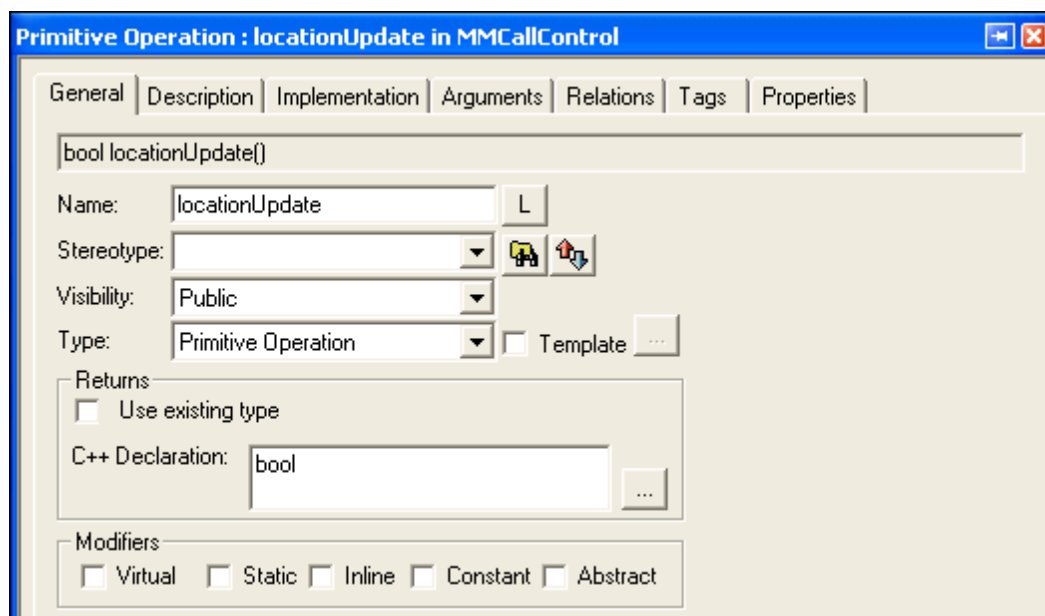
6. Draw an event from **CallControl** to the system border called **ConfirmIndication**.

Modeler adds the new realized events and primitive operations to the part to which the message is passed. For example, Modeler adds **locationUpdate** to the **Operations** category in the **MMCallControl** part.

### Setting the Features of locationUpdate

In this procedure, you set the return type and implementation for **locationUpdate**.

1. In the **Browser Window**, double-click **locationUpdate**. The **Primitive Operations Features** dialog box appears.
2. Click the **General** tab.
3. Under the **Returns**, clear the **Use existing type** check box, and type “bool” in the **C++ Declaration** box, as shown in the following illustration.



4. Click the **Implementation** tab, type “return TRUE;”
5. Click **OK** to close the **Primitive Operations Features** dialog box.



## Summary

In this module, you created SDs, which identify the message exchange between subsystems and subsystem modules when placing a call. You became familiar with the parts of an SD and created the following:

- ◆ System border
- ◆ Classifier roles and actor lines
- ◆ Interaction occurrences
- ◆ Events and primitive operations
- ◆ Time intervals

You are now ready to proceed to the next module, where you identify the functional flow of users placing a call and registering users on the network using activity diagrams.

### Note

---

With Rational Rhapsody Developer, you can generate code and animate the parts of the model you have created, showing the actual behavior on a generated sequence diagram.



# Module 5:

## Creating Activity Diagrams

---

**Activity Diagrams** show the dynamic aspects of a system and the flow of control from activity to activity. They describe the essential interactions between the system and the environment, and the interconnections of behaviors for which the subsystems or components are responsible. They can also be used to model an operation or the details of a computation. In Rational Rhapsody Developer, you can animate activity diagrams to verify the functional flow.

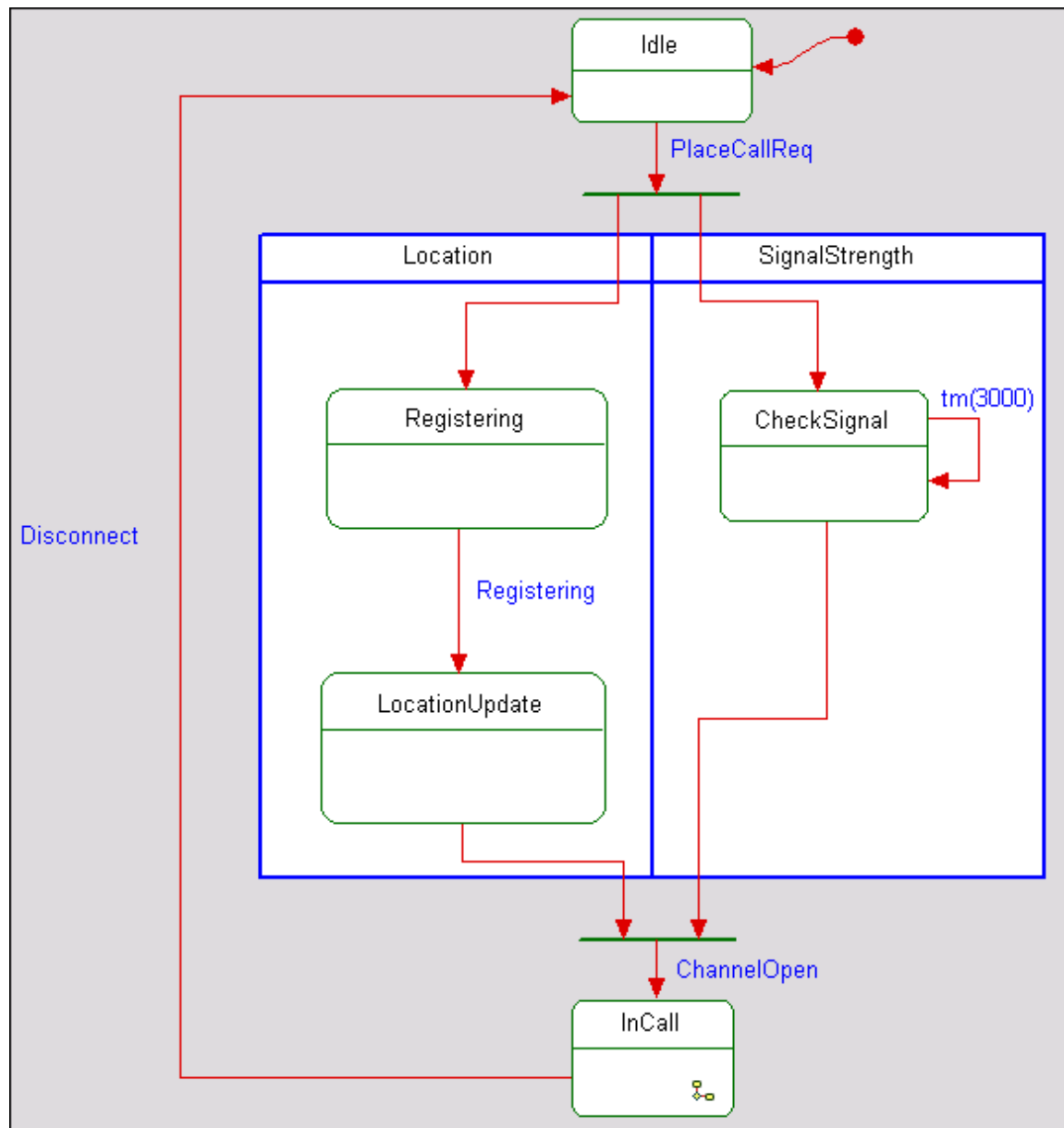
### Goals for this Module

In this module, you create the following activity diagrams:

- ♦ **MMCallControl** - identifies the functional flow of users placing a call, which includes registering users on the network, providing their current location, and obtaining an acceptable signal strength
- ♦ **InCall** - identifies the flow of information once the system connects the call
- ♦ **RegistrationMonitor** - identifies the functional flow of registering users on the network, which includes monitoring registration requests and sending received requests to the network

## Exercise 5.1: Creating the MMCallControl Activity Diagram

In this exercise, you develop a MMCallControl Activity Diagram, as shown in the following illustration.



To create the MMCallControl Activity Diagram:

1. In the **Browser Window**, expand the **SubsystemsPkg** package, the **Packages** category, the **MM\_SubsystemPkg** package, the **Blocks** category, the **MobilityManagement** block, and the **Parts** category.
2. Right-click the **MMCallControl** part, click **Add New**, and then click **Activity Diagram**.

Modeler automatically creates an **Activity Diagram** in the **Browser Window** under the **MMCallControl** part. A new drawing area opens.

### Drawing the MMCallControl Activity Diagram

The MMCallControl activity diagram shows the functional flow that supports the mobility of users when placing a call, which includes registering users on the network, providing their current location, and obtaining an acceptable signal strength. When the user places a call, the system leaves the `Idle` action element, and checks for an acceptable signal strength and to see if the wireless telephone is registered. It then waits for the call to connect and enters a connection action element.



Draw an activity diagram using the following general steps:

1. Draw swimlanes.
2. Draw action elements.
3. Draw states.
4. Draw a subactivity.
5. Draw a default connector.
6. Draw transitions.

### Drawing Swimlanes

**Swimlanes** organize activity diagrams into sections of responsibility for actions and subactions. Vertical, solid lines separate each swimlane from adjacent swimlanes. To draw swimlanes, you first need to create a swimlane frame and then a swimlane divider.

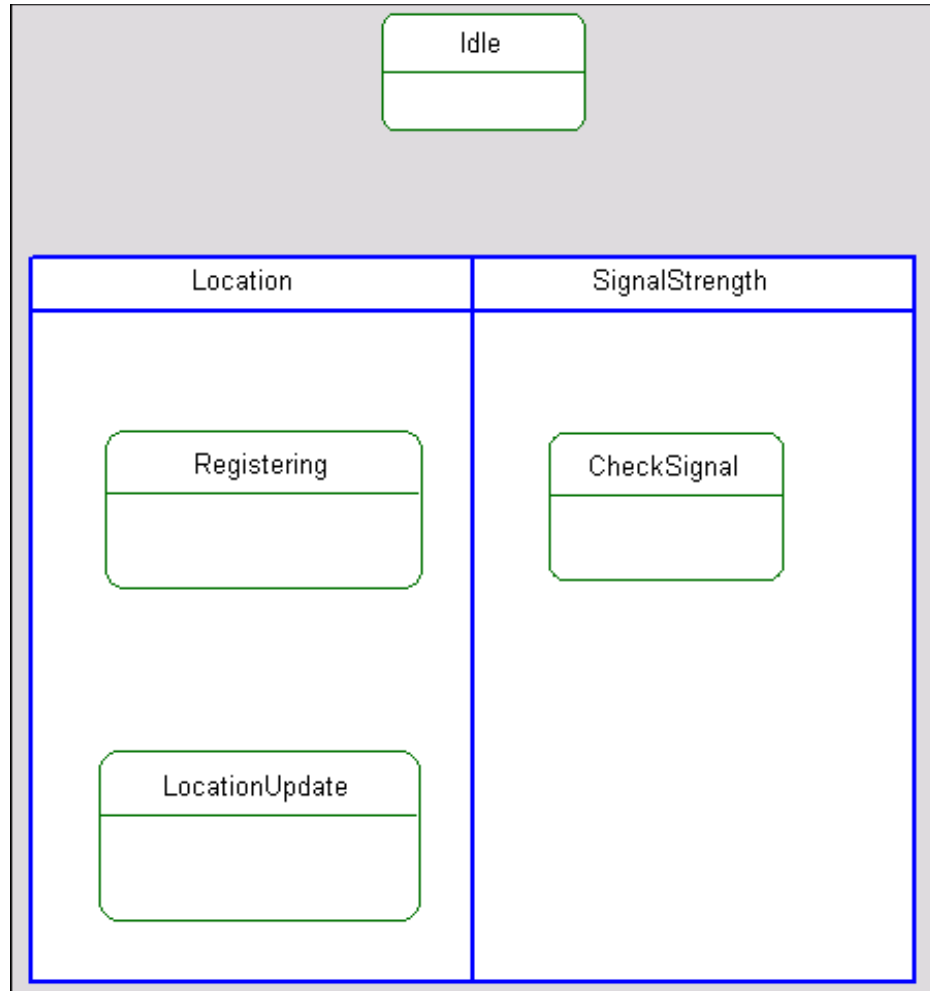
To draw swimlanes:

1. Click the **Swimlanes Frame** tool  in the Diagram Tools.
2. Click to place one corner, then drag diagonally to draw the swimlane frame.
3. Click the **Swimlanes Divider** tool  in the Diagram Tools.
4. Click the middle of the swimlane frame. Two swimlanes, called **swimlane\_*n*** and **swimlane\_*n*+1**, where *n* is an incremental integer starting at 0, are created.
5. Rename the swimlane on the left **Location**. This swimlane tracks the location of users.
6. Rename the swimlane on the right **SignalStrength**. This swimlane tracks the signal strength of users.

## Drawing Action Elements


**Action Elements** represent function invocations with a single exit transition when the function completes. In this procedure, you draw the action elements that represent the functional processes, and then add names to the action elements.

In this procedure, you draw action elements, as shown in the following illustration.



To draw action elements:

1. Click the **Action** tool  in the Diagram Tools.

**Note:** You can use the **Stamp** tool  for repetitive drawing actions. For example, when creating more than one action element, click the **Stamp** icon first before clicking the **Action** icon. Once you have created an action element, you can keep creating new action elements without clicking the **Action** icon each time. Click the **Stamp** icon again when you are done drawing action elements.

2. In the top section of the drawing area, click-and-drag to create an action element and then press **Ctrl+Enter**.
3. Double-click the action element. The **Features** dialog box appears.
4. In the **Name** box, type “Idle” and then click **OK**. This indicates that no call is in progress.

**Note:** For each action element, set the display options (right-click the action element and click **Display Options**) to **Name** to show the action element name on the diagram.

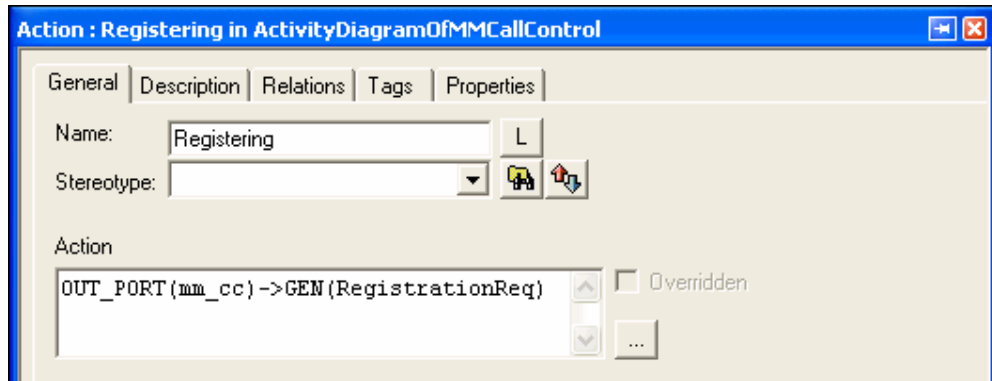
5. In the lower section of the **Location** swimlane, draw an action element and then press **Ctrl+Enter**.
6. Double-click the action element. The **Features** dialog box appears.
7. In the **Name** box, type “LocationUpdate” and then click **OK**.
8. In the **SignalStrength** swimlane draw an action element and then press **Ctrl+Enter**.
9. Double-click the action element. The **Features** dialog box appears.
10. In the **Name** box, type “CheckSignal” and then click **OK**.
11. In the **Location** swimlane, above **LocationUpdate**, draw an action element and then press **Ctrl+Enter**.
12. Double-click the action element. The **Features** dialog box appears.
13. In the **Name** box, type “Registering” and then click **OK**.



## Defining an Action

To define an action:

1. Double-click the **Registering** action element. The **Features** dialog box appears.



2. In the **Action** box, type the following code:

```
OUT_PORT(mm_cc) -> GEN(RegistrationReq)
```


This command sends an asynchronous message out the **mm\_cc** port for registration requests.

3. Click **OK** to apply the changes and to close the **Features** dialog box.

## Drawing a Default Connector

One of the states of the object must be the *default* action element. This is the initial action element of the object. **Idle** is in the default action element as it waits for call requests.

To draw a default connector:


1. Click the **Default flow** icon  in the Diagram Tools.
2. Click to the right of the **Idle** action element, click its edge, and then press **Ctrl+Enter**.

### Drawing a Subactivity

A **Subactivity** represents the execution of a non-atomic sequence of steps nested within another activity. It looks like an action element with a subactivity icon in its lower, right corner, depicting a nested activity diagram.

In this procedure, you draw the **InCall** subactivity, which indicates that the call has been established.

To draw a subactivity:

1. Click the **Subactivity** icon  in the Diagram Tools.
2. In the bottom section of the drawing area, click-and-drag.
3. Type “InCall” and then press **Ctrl+Enter**.

### Drawing Transitions

**Transitions** represent the response to a message in a given action element. They show the next action element. In this procedure, you draw the following transitions:

- ♦ Transitions between states
- ♦ Fork and join transitions
- ♦ Timeout transition

#### Note


---

To change the line shape of a transition, right-click the line, click **Line Shape**, and then click one of the following: **Straight**, **Spline**, **Rectilinear**, or **Reroute**.

### Drawing Transitions Between States

In this procedure, you draw two transitions: **Disconnect** and **Registering**.

To draw transitions between actions:


1. Click the **Activity Flow** icon  in the Diagram Tools.
2. Click the **InCall** subactivity action and then click the **Idle** action.
3. Type “Disconnect” and then press **Ctrl+Enter**.
4. Repeat Steps 1 through 3 to draw a transition from **Registering** to **LocationUpdate** and call it “Registering.”

### Labeling Elements

Modeler enables you to assign a descriptive label to an element. A labeled element does not have any meaning in terms of generated code, but enables you to easily reference and locate elements in diagrams and dialog boxes. A label can have any value and does not need to be unique.

In this procedure, you label the transition between **Registering** and **LocationUpdate**.

To label an element:

1. Double-click the transition between **Registering** and **LocationUpdate**. The **Features** dialog box appears.
2. Click the **L** button icon  next to the **Name** box. The **Name and Label** dialog box appears.
3. In the **Label** field, type “Registering” and click **OK** to close the **Name and Label** dialog box.
4. Click **OK** to close the **Features** dialog box.

### Note


---

To display the label, right-click the transition and click **Display Options** and then click **Show Label**.

### Drawing a Fork Synchronization

A **Fork Synchronization** represents the splitting of a single flow into two or more outgoing flows. It is shown as a bar with one incoming transition and two or more outgoing transitions.



To draw a fork synchronization bar:

1. Click the **Draw Fork Synch Bar** icon  in the Diagram Tools.
2. Click-and-drag between the **Idle** action and the swimlanes. Modeler adds the fork synchronization bar.
3. Click the **Activity Flow** icon and draw a single incoming transition from **Idle** to the synchronization bar.
4. Type “PlaceCallReq” and then press **Ctrl+Enter**. This transition indicates that the interface has initiated a call request.
5. Draw the following outgoing transitions from the synchronization bar:
  - a. To the **Registering** action and then press **Ctrl+Enter**.
  - b. To the **CheckSignal** action and then press **Ctrl+Enter**.

### Drawing a Join Synchronization

A **Join Synchronization** represents the merging of two or more concurrent flows into a single outgoing flow. It is shown as a bar with two or more incoming transitions and one outgoing transition.

To draw a join synchronization bar:


1. Click the **Draw Join Synch Bar** icon  in the Diagram Tools.
2. Click-and-drag between the swimlanes and **InCall**. Modeler adds the join synchronization bar.
3. Click **Activity Flow** icon  and draw the following incoming transitions to the synchronization bar:
  - a. From **LocationUpdate** and then press **Ctrl+Enter**.
  - b. From **CheckSignal** and then press **Ctrl+Enter**.
4. Draw one outgoing transition from the synchronization bar to **InCall**.
5. Type “ChannelOpen” and then press **Ctrl+Enter**. This transition indicates that the channel is open and the call can be established.

### Drawing a Timeout Transition

A **Timeout Transition** causes an object to transition after a specified amount of time has passed. It is an event with the form  $tm(n)$ , where  $n$  is the number of milliseconds the object should wait before making the transition.

In this procedure, you draw a timeout transition that monitors the signal strength of transmissions every three seconds.

To draw a timeout transition:

1. Click the **Activity Flow** icon  in the Diagram Tools.
2. Draw a transition originating and ending with **CheckSignal**.
3. Type “tm(3000)” and then press **Ctrl+Enter**.

**Note:** Rational Rhapsody Developer creates real time timeouts in the generated code for the tm transition.

### Specifying an Action on a Transition

In this procedure, you specify actions for **Disconnect** and **ChannelOpen**.

To specify an action on a transition:

1. Double-click the **Disconnect** transition. The **Features** dialog box appears.
2. In the **Action** box, type the following code:

```
OUT_PORT(mm_cc) ->GEN(Disconnect);
```

This command sends an asynchronous message out the **mm\_cc** port when disconnecting.

3. Click **OK** to apply the changes and to close the **Features** dialog box. Modeler displays the transition name with the action command.
4. Double-click the **ChannelOpen** transition. The **Features** dialog box appears.
5. In the **Action** box, type the following code:

```
locationUpdate();  
OUT_PORT(cc_in) ->GEN(CallConfirm);  
OUT_PORT(cc_in) ->GEN(ConnectConfirm);
```

The OUT\_PORT commands send asynchronous messages out the **cc\_in** port.

6. Click **OK** to apply the changes and to close the **Features** dialog box. Modeler displays the transition name with the action command.

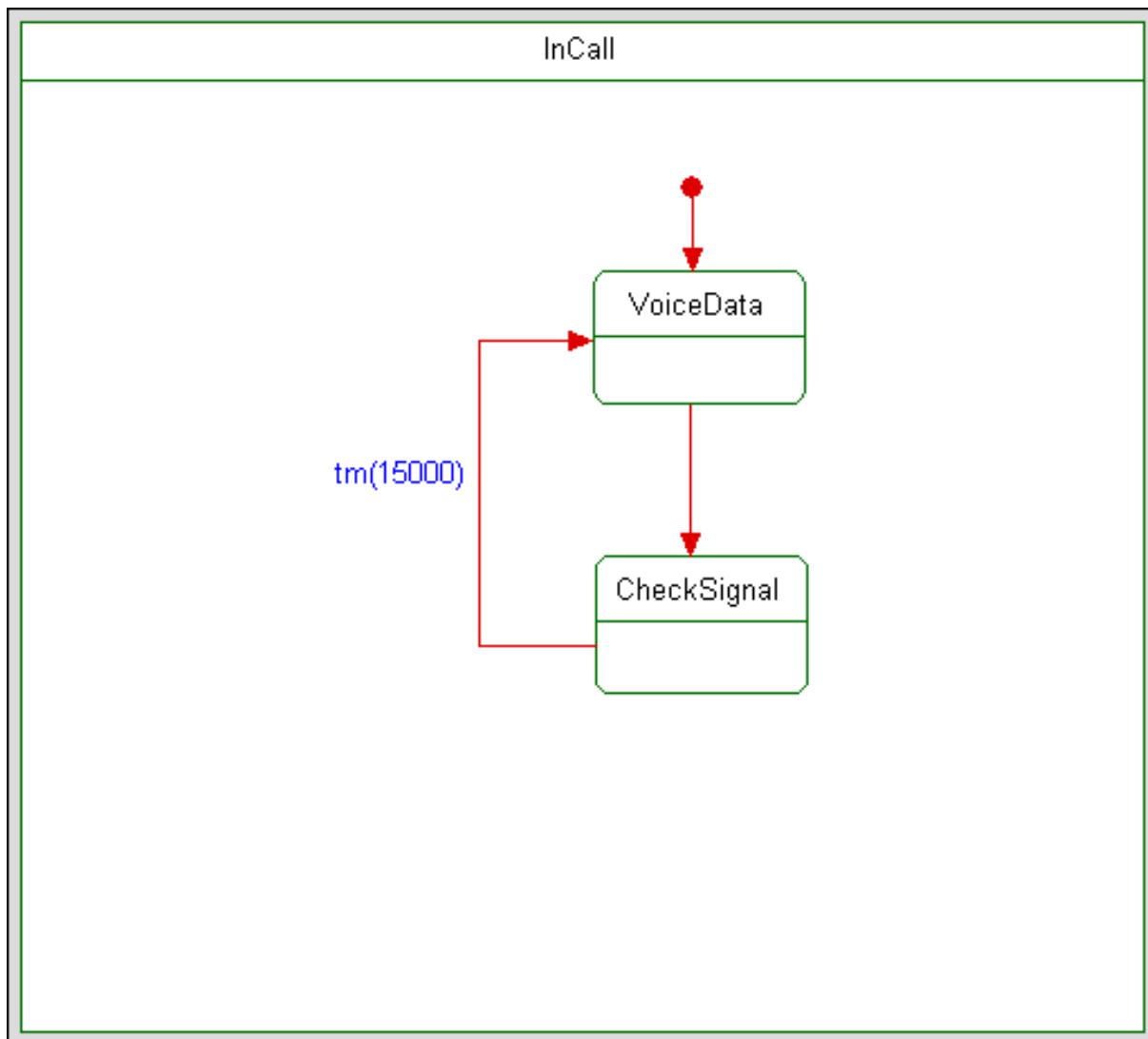
**Note:** To display the transition name without the action, type the transition name as the label using the **Features** dialog box. Then right-click the transition and click **Display Options** and then click **Show Label**.

Modeler automatically adds the action elements and transitions to the **MMCallControl** part in the **Browser Window**.

## Exercise 5.2: Drawing the InCall Subactivity Diagram

Subactivity states represent nested activity diagrams. The InCall subactivity diagram shows the flow of information once the system connects the call. The system monitors the signal strength for voice data every 15 seconds.

In this exercise, you develop an InCall Subactivity Diagram, as shown in the following illustration.



## Opening the InCallSubactivity Diagram


To open the **InCall** subactivity diagram, right-click **InCall** in the **MMCallControl** activity diagram, and click **Open Sub Activity Diagram**. Modeler displays the subactivity diagram with the **InCall** activity in the drawing area.

## Drawing Action Elements

In this procedure, you draw the following two actions states, and then add names to the action elements.

To draw the action elements:

1. Click the **Action** icon  in the Diagram Tools.

**Note:** You can use the **Stamp** tool  for repetitive drawing actions. For example, when creating more than one action element, click the **Stamp** icon first before clicking the **Action** icon. Once you have created an action element, you can keep creating new action elements without clicking the **Action** icon each time. Click the **Stamp** icon again when you are done drawing action elements.

2. In the top section of the **InCall** action element, click-and-drag and then press **Ctrl+Enter**.
3. Double-click the new action element. The **Features** dialog box appears.
4. In the **Name** box, type “VoiceData” and then click **OK**. The **VoiceData** action element processes voice data.
5. In the bottom section of the **InCall** action element, click-and-drag and then press **Ctrl+Enter**.
6. Double-click the new action element. The **Features** dialog box appears.
7. In the **Name** box, type “CheckSignal” and then click **OK**. The **CheckSignal** action element checks the signal strength on the network.

### Note

---

For each action element, set the display options to **Name** to show the name on the diagram.



## Drawing a Default Connector


The subactivity diagram must have an initial action element. Execution begins with the initial action element when an input transition to the subactivity action element is triggered.

To draw the default connector:

1. Click the **Default Flow icon**  in the Diagram Tools.
2. Click above **VoiceData**, then click **VoiceData**, and then press **Ctrl+Enter**.


## Drawing Transitions

To draw a transition:

1. Click the **Activity Flow icon**  in the Diagram Tools.
2. Draw a transition from **VoiceData** to **CheckSignal** and press **Ctrl+Enter**.

## Drawing a Timeout Transition

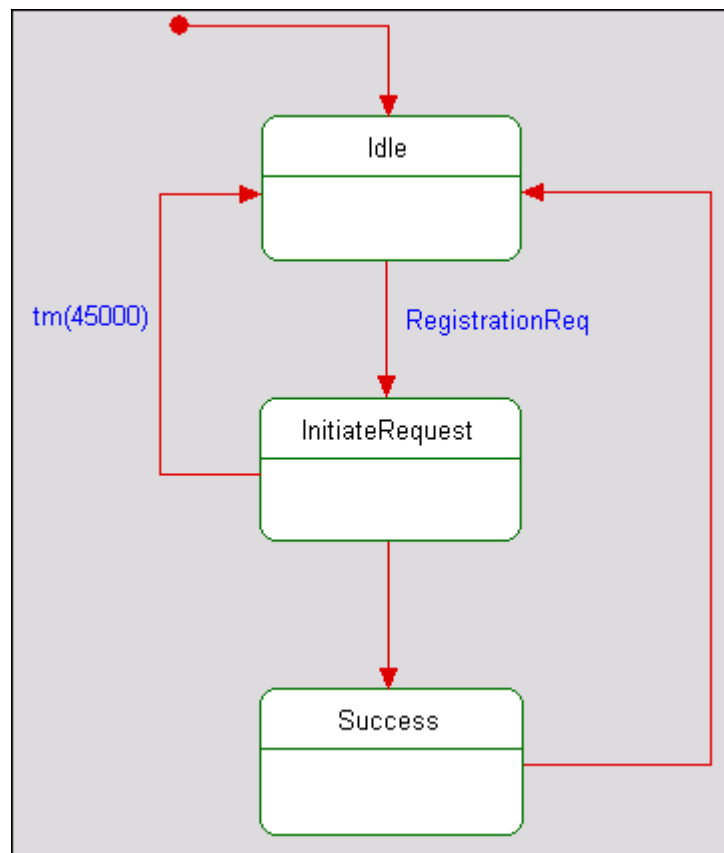
To draw a timeout transition to check for voice data every 15 seconds:

1. Click the **Activity Flow icon** .
2. Draw a transition from **CheckSignal** to **VoiceData**.
3. Type “tm(15000)” and then press **Ctrl+Enter**. Modeler automatically adds the newly created action elements and transitions to the **Browser Window**.

## Exercise 5.3: Drawing the RegistrationMonitor Activity Diagram

The RegistrationMonitor activity diagram shows the functional flow of network registration requests. The system checks for registration requests and then sends received requests to the network.

In this exercise, you develop a RegistrationMonitor Activity Diagram, as shown in the following illustration.



## Creating the RegistrationMonitor Activity Diagram

To create the RegistrationMonitor Activity Diagram:

1. In the **Browser Window**, expand the **SubsystemsPkg** package, the **Packages** category, the **DL\_SubsystemPkg** package, the **Blocks** category, the **DataLink** block, and the **Parts** category.
2. Right-click the **RegistrationMonitor** part, click **Add New**, and then click **Activity Diagram**.


Modeler automatically creates an **Activity Diagram** in the **Browser Window** under the **RegistrationMonitor** part. A new drawing area opens.

### Drawing Action Elements

In this procedure, you draw three actions states and then add names to the action elements.

To draw the action elements:

1. Click the **Action** tool  in the Diagram Tools.

**Note:** You can use the **Stamp** tool  for repetitive drawing actions. For example, when creating more than one action element, click the **Stamp** icon first before clicking the **Action** icon. Once you have created an action element, you can keep creating new action elements without clicking the **Action** icon each time. Click the **Stamp** icon again when you are done drawing action elements.

2. In the upper section of the drawing window, click and drag and then press **Ctrl+Enter**.
3. Double-click the new action element. The **Features** dialog box appears.
4. In the **Name** box, type “Idle” and then click **OK**.
5. Below the **Idle** action element, click and drag and then press **Ctrl+Enter**.
6. Double-click the new action element. The **Features** dialog box appears.
7. In the **Name** box, type “InitiateRequest” and then click **OK**.
8. Below the **InitiateRequest** action element, click and drag and then press **Ctrl+Enter**.
9. Double-click the new action element. The **Features** dialog box appears.
10. In the **Name** box, type “Success” and then click **OK**.

**Note:** For each action element, set the display options to **Name** to show the name on the diagram.

### Defining an Action

In this procedure, you specify an action for the **InitiateRequest** action element.

To define an action:

1. Double-click **InitiateRequest**. The **Features** dialog box appears.
2. In the **Action** box, type the following code:


```
OUT_PORT (reg_request) ->GEN (ChannelOpen) ;
```

This command sends an asynchronous message out the **reg\_request** port when the channel is open.

3. Click **OK** to apply the changes and to close the **Features** dialog box.


### Drawing a Default Connector

To draw a default connector:

1. Click the **Default Flow** icon  in the Diagram Tools.
2. Click above the **Idle** action element, then click the **Idle** action element, and then press **Ctrl+Enter**.


### Drawing Transitions

To draw transitions between actions states:

1. Click the **Activity Flow** icon  in the Diagram Tools.
2. Draw a transition from the **Idle** action element to the **InitiateRequest** action element.
3. Type “RegistrationReq” and then press **Ctrl+Enter**.
4. Draw a transition from **InitiateRequest** to **Success** and then press **Ctrl+Enter**.
5. Draw a transition from **Success** to **Idle** and then press **Ctrl+Enter**.

## Drawing a Timeout Transition

To draw a timeout transition to return to the **Idle** action element after 45 seconds if no response is received from the network:

1. Click the **Activity Flow** icon  in the Diagram Tools.
2. Draw a transition from the **InitiateRequest** action element to the **Idle** action element.
3. Type the transition label “tm(45000)” and then press **Ctrl+Enter**.

Modeler automatically adds the newly created action elements and transitions to the **RegistrationMonitor** part in the **Browser Window**.

## Summary

In this module, you created activity diagrams and a subactivity diagram, which show the functional flow of placing a call and registering users. You became familiar with the parts of an activity diagram and created the following:

- ♦ Swimlanes
- ♦ Action elements
- ♦ Subactivity states
- ♦ Default connectors
- ♦ Transitions and timeout transitions
- ♦ Fork synchronization bar and join synchronization bar

You are now ready to proceed to the next module, where you identify the action element-based behavior when the system receives call requests and connects calls using a statechart.

### Note

---

In Rational Rhapsody Developer, you can generate code and animate the parts of the model you have created. This lets you determine whether the model meets the requirements, and identify defects early on in the design process.



# Module 6:

## Creating a Statechart

---

**Statecharts** (SCs) define the behavior of classifiers (actors, use cases, or classes), objects, and blocks, including the states that they can enter over their lifetime and the messages, events, or operations functions that cause them to transition from state to state.

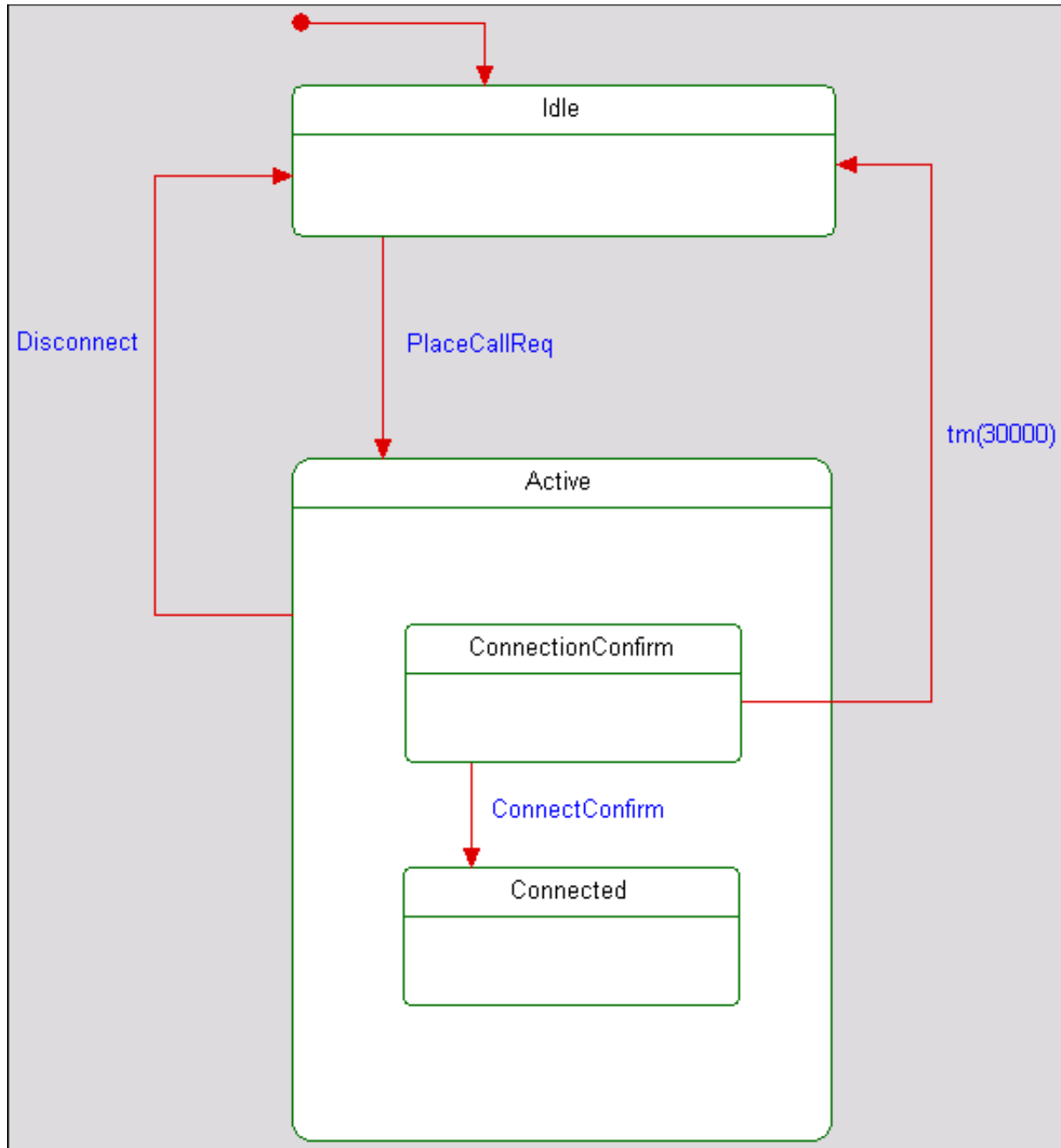
Statecharts are a key animation tool used to verify the functional flow and moding. Statecharts can be animated to view the design level of abstraction and graphically show dynamic behavior.

### Goals for this Module

In this module, you create the CallControl statechart, which identifies the state-based behavior when the system receives call requests and connects calls.

## Exercise 6.1: Creating the CallControl Statechart

In this exercise, you develop a CallControl SC, as shown in the following illustration.





To create the CallControl Statechart:

1. In the **Browser Window**, expand the **SubsystemsPkg** package, the **Packages** category, the **CM\_SubsystemPkg** package, the **Blocks** category, the **ConnectionManagement** block, and the **Parts** category.
2. Right-click the **CallControl** part, click **Add New**, and then click **Statechart**.

Modeler automatically adds the **Statechart** category and the new statechart in the **Browser Window** under the **CallControl** part. A new drawing area opens.

### Drawing the CallControl Statechart

The CallControl statechart identifies the state-based behavior of objects when the system receives call requests from users and connects calls. CallControl waits for an incoming call in the `Idle` state. When an incoming call is received, it forwards the message. If it does not receive a confirmation from the network in thirty seconds, it returns to the `Idle` state. If it receives a confirmation, the call connects, and remains connected until it receives a message to disconnect.

Draw statecharts using the following general steps:

1. Draw states and nested states.
2. Draw default connectors.
3. Draw transitions and specify actions on transitions.
4. Draw timeout transitions.


The following sections describe these steps in detail.

### Drawing States

A **State** is a graphical representation of the status of an object. It typically reflects a certain set of its internal data (attributes) and relations.

In this procedure, you draw two states, **Idle** and **Active**.

To draw a state:

1. Click the **State** icon  in the Diagram Tools.
2. In the top section of the drawing area, click-and-drag. A state with a default name of `state_n`, where `n` is equal to or greater than 0, is created.
3. Type “Idle” and then press **Enter**. This state indicates that no call is in progress.


4. In the center of the drawing area, draw a larger state called **Active**. This state indicates that the call is being set up or is in progress.

### Drawing Nested States

In this procedure, you draw the following states nested inside the `Active` state:

- ◆ `ConnectionConfirm` - Waits for a connection and then confirms the connection
- ◆ `Connected` - Connects as a voice or data call


To draw nested states:

1. Click the **State** icon  in the Diagram Tools.
2. In the top section of the **Active** state, draw a state called **ConnectionConfirm**.
3. In the bottom section of the **Active** state, draw a state called **Connected**.

### Drawing Default Connectors

One of the states of the object must be the default state, that is, the state in which the object finds itself when it is first instantiated. **Idle** is in the default state as it waits for call requests and **Active** is in the default state before it confirms the connection.


To draw default connectors:

1. Click the **Default connector** tool  in the Diagram Tools.
2. Click to the right of the **Idle** state, then click the **Idle** state, and then press **Ctrl+Enter**.
3. Draw a default connector to **ConnectionConfirm** and press **Ctrl+Enter**.

### Drawing Transitions

Transitions represent the response to a message in a given state. They show what the next state be. A transition can have an optional trigger, guard, or action. In this procedure, you draw transitions with triggers.

To draw transitions:

1. Click the **Transition** icon  in the Diagram Tools.
2. Click the **Idle** state and then click the **Active** state.
3. In the **Label** box, type “PlaceCallReq” and then press **Ctrl+Enter**.
4. Create a transition from **ConnectionConfirm** to **Connected** called **ConnectConfirm** and then press **Ctrl+Enter**.

5. Create a transition from the **Active** state to the **Idle** state called **Disconnect** and then press **Ctrl+Enter**. This transition indicates that the user has disconnected or the network has terminated the call.

### Note

---

To change the line shape, right-click the line, click **Line Shape**, and then click one of the following: **Straight**, **Spline**, **Rectilinear**, or **Reroute**.

## Specifying an Action on a Transition

You can specify that an object execute a specific action when it transitions from one state to another.

In this procedure, you specify an action for **PlaceCallReq** and **Disconnect**.

To specify an action on a transition:

1. Double-click the **PlaceCallReq** transition. The **Features** dialog box appears.
2. In the **Action** box, type the following code:

```
OUT_PORT(cc_mm) ->GEN(PlaceCallReq);
```

This command sends an asynchronous message out the **cc\_mm** port when placing a call.

3. Click **OK** to apply the changes and to close the **Features** dialog box. The transition now includes an action.
4. Double-click the **Disconnect** transition. The **Features** dialog box appears.
5. In the **Action** box, type the following code:

```
OUT_PORT(cc_mm) ->GEN(Disconnect);
```

This command sends an asynchronous message out the **cc\_mm** port when disconnecting.


6. Click **OK** to apply the changes and to close the **Features** dialog box.

## Drawing a Timeout Transition

A **Timeout Transition** causes an object to transition to the next state after a specified amount of time has passed. It is an event with the form **tm(*n*)**, where ***n*** is the number of milliseconds the object should wait before making the transition.

In this procedure, you draw a timeout transition in which **ConnectionConfirm** waits thirty seconds before returning to the **Idle** state if a connect confirmation is not made.

To draw a timeout transition:

1. Click the **Transition** icon  in the Diagram Tools.
2. Draw a transition from **ConnectionConfirm** to **Idle**.
3. Type “tm(30000)” and then press **Ctrl+Enter**.

Modeler automatically adds the newly created states and transitions to the **CallControl** part in the **Browser Window**.

## Summary

In this module, you created a statechart, which identifies the state-based behavior when the system receives call requests and connects calls. You became familiar with the parts of a statechart and created the following:

- ♦ States and nested states
- ♦ Default connectors
- ♦ Transitions and timeout transitions

### Note

---

Rational Rhapsody Developer generates the full behavioral code for statecharts and also allows model level debugging by animating the statecharts.

**You have completed the handset model.**

# Module 7:

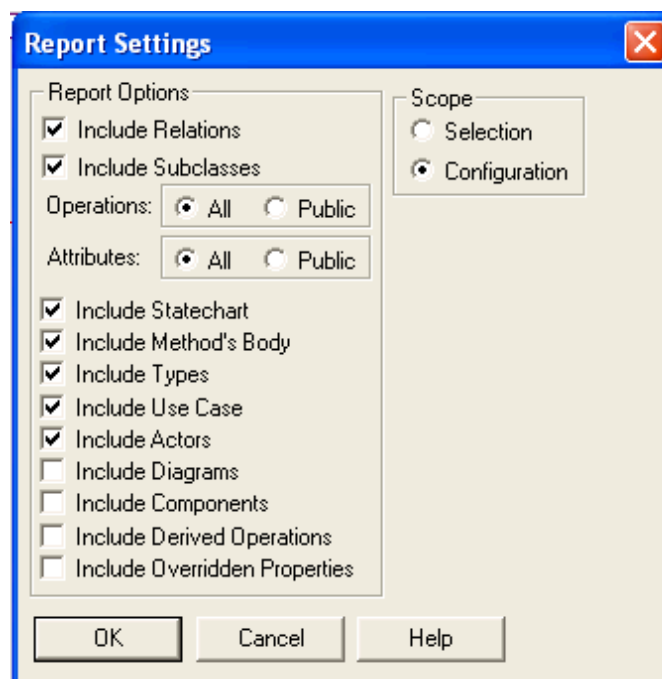
## Generating a Report

---

The internal reporting facility is particularly useful for quick print-outs that the developer needs to use for debugging the model. The reports are not formatted for formal presentations.

### Producing an Internal Report

To create a report using the internal reporter, in the **Tools** menu, click **Report on model**. The **Report Settings** dialog box appears.



The **Report Settings** dialog box contains the following fields:

- ♦ **Report Options** - Specifies which elements to include in the report. The possible values are as follows:
  - **Include Relations** - Include all relationships (associations, aggregations, and compositions). By default, this option is checked.
  - **Include Subclasses** - List the subclasses for each class in the report. By default, this option is checked.
- ♦ **Scope** - Specifies the scope of the report. The possible values are as follows:
  - **Selection** - Include information only for the selected elements.
  - **Configuration** - Include information for all elements in the active component scope. This is the default value.
- ♦ **Operations** - Specifies which operations to include in the report. The possible values are as follows:
  - **All** - Include all operations. This is the default value.
  - **Public** - Include only the public operations.
- ♦ **Attributes** - Specifies which attributes to include in the report. The possible values are as follows:
  - **All** - Include all attributes. This is the default value.
  - **Public** - Include only the public attributes.
- ♦ **Include Statechart** - If the project has a statechart, this option specifies whether to list the states and transitions in the report. By default, this option is checked.
- ♦ **Include Method's Body** - Specifies whether to include the code for all method bodies in the report. By default, this option is checked.
- ♦ **Include Types** - Specifies whether to list the types in the report. By default, this option is checked.
- ♦ **Include Use Case** - Specifies whether to list the use cases in the report. By default, this option is checked.
- ♦ **Include Actors** - Specifies whether to list the actors in the report. By default, this option is checked.
- ♦ **Include Diagrams** - Specifies whether to include diagram pictures in the report. By default, this option is not checked.
- ♦ **Include Components** - Specifies whether to include component information (configurations, folders, files, and their settings) in the report. By default, this option is not checked.
- ♦ **Include Derived Operations** - Specifies whether to include generated operations (when the property `CG::CGGeneral::GeneratedCodeInBrowser` is set to `True`). By default, this option is not checked.

- 
- ♦ **Include Overridden Properties** - Specifies whether to include properties whose default values have been overridden. By default, this option is not checked.

To generate the report, select the appropriate values, then click **OK** to generate the report. The report is displayed in the drawing area with the current file name in the title bar.

## Using the Internal Report Output

When you generate a report in Modeler using, the initial result uses the internal RTF viewer. To facilitate the developer's research, this output may be used:

- ♦ To locate specific items in the report online, in the **Edit** menu, click **Find** and type in the search criteria.
- ♦ To print the initially generated report, in the **File** menu, click **Print** menu option.

The initially generated report is only a view of the RTF file that the facility created. This file is located in the project directory (parallel to the `.rpy` file) and is named `RhapsodyRep<num>.rtf`. If you wish, open the RTF file using a word processor that handles RTF format, such as Microsoft Word.

