Telelogic

Rhapsody

Systems Engineering Tutorial

IBM

Rhapsody®

Systems Engineering Tutorial

Before using the information in this manual, be sure to read the “Notices” section of the Help or the PDF available from **Help > List of Books**.

This edition applies to Telelogic Rhapsody 7.4 and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Rhapsody Basics for Systems Engineers

Welcome to the *Systems Engineering Tutorial*. Rhapsody allows systems engineers to capture and analyze *requirements* quickly and then design and validate system behaviors.

A Rhapsody systems engineering project includes the UML and SysML diagrams, packages, and simulation configurations that define the model. Systems engineers may use the SysML Profile Features or the Harmony profile and process to guide software development through this iterative development process:

- Perform system analysis to define and validate system requirements
- Design and specify the system architecture
- Systems analysis and design
- Software analysis and design
- Software implementation
- Validate and simulate the model to perform detailed system testing

Rhapsody's system engineering features allow system designers to hand off their work to software developers accurately and easily.

This tutorial provides step-by-step instructions demonstrating the tasks systems engineers can accomplish using the SysML profile in Rhapsody.

Installing and Launching Systems Engineering

Rhapsody's Systems Engineering add-on requires these extra installation steps and start-up steps:

1. Follow the instructions in the *Rhapsody Installation Guide* to install the development environments you need.
2. When the installation program displays the **Setup Type** window, select **Typical**. Click **Next**.
3. The **Add-on Installation** window displays. Check the **System Engineering Add-On** option**,** as shown.

4. Click **Next** and complete the installation as instructed.
5. When you want to start Rhapsody, select the Windows **Start** > **Programs** > **Telelogic** > **Telelogic Rhapsody** (version) > (Rhapsody Edition) > **Rhapsody**.

The *Custom* installation makes the systems engineering features available to support the UML and SysML standards in these specifications:

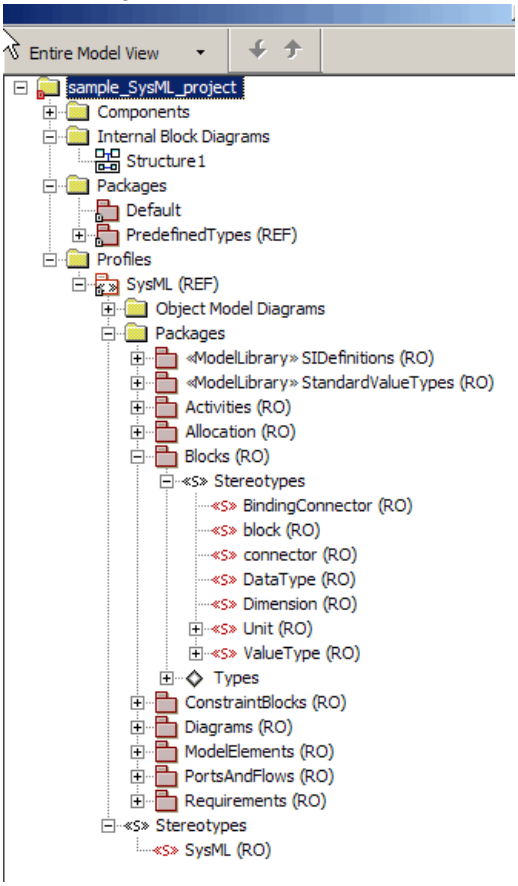
- **UML specification**
- **SysML specification**

SysML Profile Features

In this tutorial, you select the SysML profile for your project. With this profile, Rhapsody provides a starting point with a blank Block Definition Diagram (named `Model1`), packages, and predefined types, as shown in SysML Profile Elements. This profile is Rhapsody's implementation of the **OMG SysML Specification**. The Rhapsody SysML profile provides this additional functionality for your model:

- SysML enhancements to standard UML diagrams including the Use Case, Requirements, Activity, Sequence diagrams and Statecharts
- SysML's Block Definition, Internal Block, Package, and Parametric diagrams
- XMI 2.1 support

SysML Profile Elements



The SysML profile also contains read-only (RO) packages and a read-only ProfileStructure object model diagram for you to use as reference of the available Rhapsody SysML features in the profile.

Note

The items listed under `Profiles` in the browser are not intended to be used as part of a working model. They are for information purposes only.

When you create a new project, Rhapsody 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 Rhapsody browser. Rhapsody provides several default elements in the new project, including a default package, component, and configuration.

Handset Model Problem Statement

This tutorial shows you how to use Rhapsody to analyze, design, and build a model of a wireless telephone. 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.

Note

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

Starting Rhapsody

This section presents the basic concepts to start a Rhapsody project and use the Rhapsody interface.

To start Rhapsody, follow these steps:

1. Select **Start** > **All Programs** > **Telelogic** > **Telelogic Rhapsody** (version) > (Rhapsody Edition) > **Rhapsody**.
2. If the system opens with the **Tip of the Day** dialog box, close it to start working in Rhapsody.

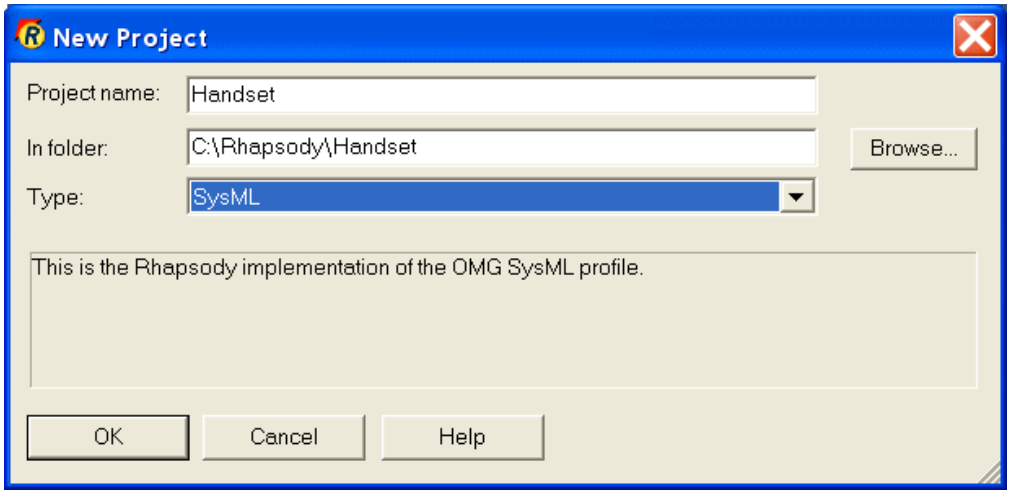
Creating a New SysML Project

A Rhapsody project includes the UML and SysML diagrams, packages, and simulation configurations that define the model. When you create a new project, Rhapsody 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 Rhapsody browser. Rhapsody provides several default elements in the new project, including a default package, component, and configuration.

To create a new SysML project, follow these steps

1. Click the **New** icon  on the main toolbar or select **File** > **New**. The New Project dialog box opens.
2. In the **Project name** field, type `Handset` as the name of the project.
3. In the **In folder** field, enter the directory in which the new project will be located, or click the **Browse** button to select the directory.

4. In the **Type** field, select the `SysML` *profile* so that you will be able to use the SysML modeling language and all of the systems engineering diagrams. Your dialog box should resemble this example.



5. Click **OK**. Rhapsody verifies that the specified location exists. If it does not, Rhapsody asks whether you want to create it.

6. Click **Yes**. Rhapsody creates a new project in the `Handset` subdirectory, opens the project, and displays the browser in the left pane.

Note

If the browser does not display, select **View** > **Browser**.

Saving a Project

Use the **Save** command to save the project in its current location. The Save command saves only the modified units, reducing the time required to save large projects. To save the project to a new location, use the **Save As** command.

Rhapsody performs an autosave every ten minutes to back up changes made between saves. Modified units are saved in the autosave folder, along with any units that have a time stamp older than the project file.

To save the project in the current location, use one of the following methods:

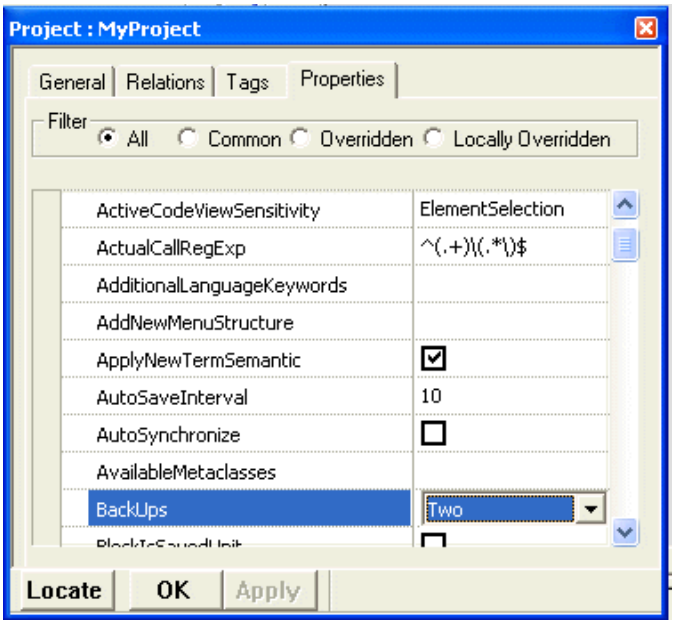
1. Select **File** > **Save**.

2. Click the **Save** icon  in the toolbar.

Creating Backups

To set up automatic backups for your new project, follow these steps:

1. In the browser, right-click the `Handset` item in the browser list.
2. Select **Features** from the pop-up menu.
3. Click the **Properties** tab at the top of the dialog box that then displays.
4. Click the **All** radio button to display all of the properties for this project.
5. Expand the **General** and then the **Model** property lists. (Rhapsody descriptions use a notation method with double colons to identify the location of a specific property, for example, `General::Model::BackUps.`)
6. Locate the **BackUps** property and select **Two** from the pull-down menu, as shown below. With this setting, Rhapsody creates up to two backups of every project in the project directory.



7. Click **OK** to save this property change.

After this change, saving a project more than once creates <projectname>_bak2.rpy contains the most recent backup and the previous version in <projectname>_bak1.rpy. To *restore* an earlier version of a project, you can open either of these backup files.

Project Files and Directories

Rhapsody creates the following files and subdirectories in the project directory:

- A project file, called `<project_name>.rpy`
- A repository directory, called `<project_name>_rpy`, which contains the unit files for the project, including diagrams, packages, and code generation configurations
- An event history file, called `<project_name>.ehl`, which contains a record of events injected during animation, and active and nonactive breakpoints
- Log files, which record when projects were loaded and saved in Rhapsody
- A `.vba` file, called `<project_name_>.vba`, which contains macros or wizards
- Backup project files and directories
- An `_RTC` directory, which holds any tests created using the Rhapsody TestConductor™ add-on

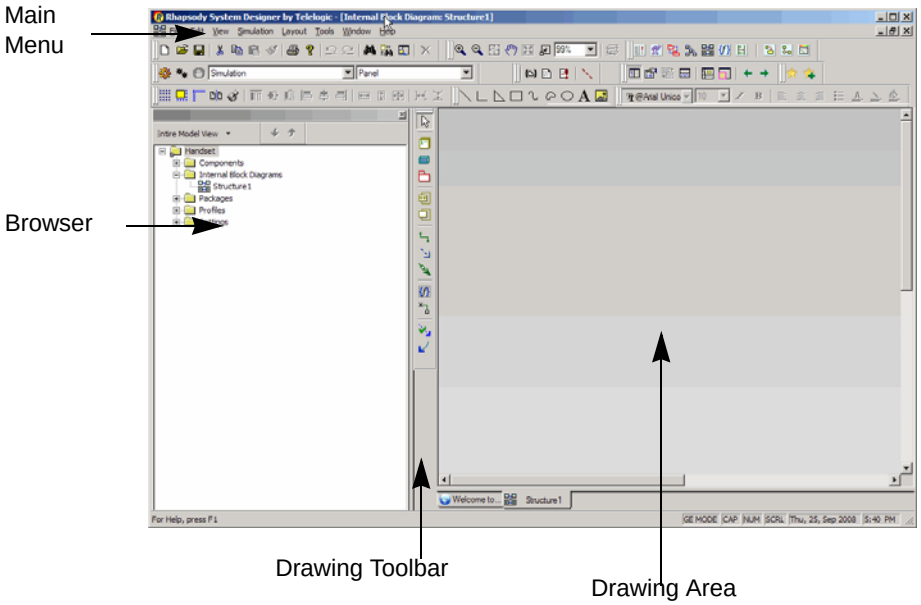
Note

Rhapsody requires the project file (`<project_name>.rpy`) and the repository directory (`<project_name>_rpy`) to simulate the model.

Rhapsody Guided Tour for Systems Engineers

Before proceeding with this tutorial, you need to become familiar with the main features of the Rhapsody user interface. The Rhapsody GUI has three primary work areas (browser, drawing area, and output window), a Drawing toolbar in the center of the interface, and a main menu across the top with project management and diagram icons under it.

The following example shows the Rhapsody interface with features for systems engineers.



Main Menu

The main menu across the top of the window provides file management capabilities and access to special tools. Many of the menu options can also be performed using the icons below the menu.

Drawing Toolbar

Rhapsody displays different icons on the Drawing toolbar for each UML or SysML diagram type. The uses of the individual icons are demonstrated within this tutorial as they are used.

Browser

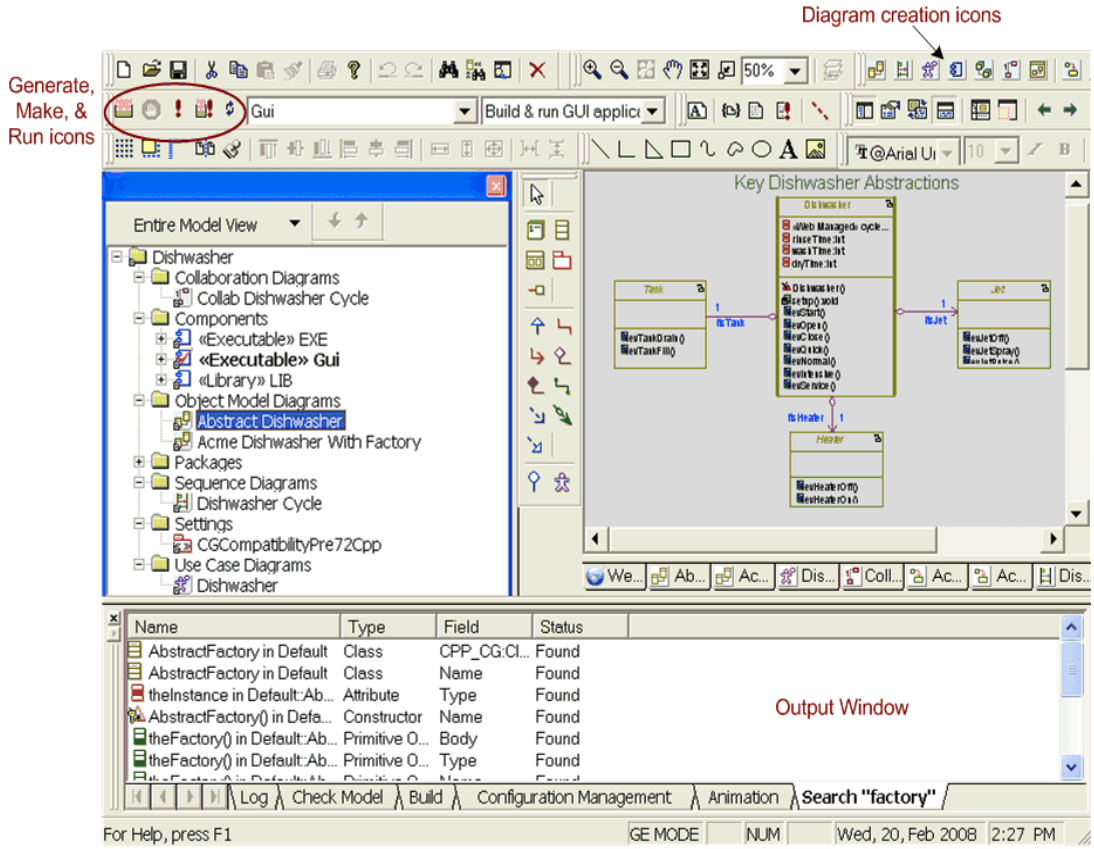
The browser shows the contents of the project in an expandable tree structure. By default, it is the upper, left-hand part of the Rhapsody GUI. The top-level folder, which contains the name of the project, is the *project folder* or *project node*. Although this folder contains no elements, the folders

that reside under it contain elements that have similar characteristics. These folders are referred to as *categories*.

A project consists of at least one package in the `Packages` category. A package contain elements, such as and diagrams. Rhapsody automatically creates a default package called `Default`, which it uses to save model parts unless you specify a different package.

Output Window and Icons

The output window displays Rhapsody messages when animating a model or performing other tasks, such as a search or check model operation. These tabbed output windows display at the bottom of the window, but they can be moved.



Naming Conventions and Guidelines

To assist all members of your team in understanding the purpose of individual items in the model, it is a good idea to define naming conventions. These conventions help team members to read the diagram quickly and remember the model element names easily.

Note

Remember that the names used in the Rhapsody models are going to be automatically written into the generated code. Therefore, the names should be simple and clearly label all of the elements.

Standard Prefixes

Lower and upper case prefixes are useful for model elements. The following is a list of common prefixes with examples of each:

- Event names = “ev” (evStart)
- Trigger operations = “op” (opPress)
- Condition operations = “is” (isPressed)
- Interface classes = “I” (IHardware)

Guidelines for Naming Model Elements

The names of the model elements should follow these guidelines:

- Class names begin with an upper case letter, such as “System.”
- Operations and attributes begin with lower case letters, such as “restartSystem.”
- Upper case letters separate concatenated words, such as “checkStatus.”
- The same name should not be used for different elements in the model because it will cause code generation problems. For example, no two elements, such as a class, an interface, and a package, should not have exactly the same name.

Inserting a Diagram Title

Each diagram has its name in the diagram table and in the title bar of the window that displays the diagram. However, it is also useful to add a title to a diagram to help other members of your team understand the content and purpose of a diagram. It may be any text you wish. To add this optional title to your sequence diagram, follow these steps:

1. With the new diagram displayed in the drawing area, click the **Text** icon  at the top of the window.
2. Click above the items in the diagram and type the title of the diagram.
3. You may reposition the title by dragging it into a new location.
4. You may change the font style using the text icons at the top of the window.

Using Packages to Organize a System Model

You may use packages to structure the model into functional domains or subsystems consisting of blocks, block types, functions, variables, and other logical artifacts. Packages can also provide a hierarchy for high-level partitioning that might include the following:

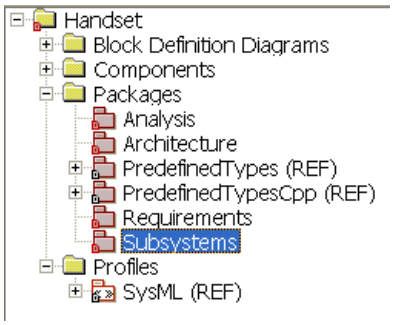
- `Requirements` contains the system's functional requirements.
- `Analysis` contains the use case diagrams to identify the system requirements.
- `Architecture` contains the block definition diagram detailing the system model design and information flow.
- `Subsystems` contains the lower-level of system decomposition.

Note

Any of the model elements within the above packages can be linked to other elements to demonstrate requirements traceability.

To organize your model into the logical packages, follow these steps:

1. In the browser, expand the `Packages` category.
2. Double-click the `Default` package and rename it `Requirements`.
3. Right-click `Packages` in the browser and select **Add New Package**. Rhapsody creates a package with the default name `package_n`, where *n* is greater or equal to 0.
4. Rename more packages named `Architecture` and `Subsystems`, as shown in this browser example.



Requirements Capture and Analysis

This section describes importing requirements using the Rhapsody Gateway and using *use case diagrams* (UCDs) to show the main system functions and the entities that are outside the system (actors). The use case diagrams specify the requirements for the system and demonstrate the interactions between the system and external actors.

Importing Requirements into Rhapsody

You may use the *Rhapsody Gateway* product to define specific requirements to support your analysis.

This add-on product allows Rhapsody to hook up seamlessly with third-party requirements and authoring tools for complete requirements traceability. The Rhapsody Gateway includes the following features:

- Traceability of requirements workflow on all levels, in real-time
- Automatic management of complex requirements scenarios for intuitive and understandable views of upstream and downstream impacts
- Creates impact reports and requirements traceability matrices to meet industry safety standards
- Connects to common requirements management/authoring tools including DOORS, Requisite Pro®, Word®, Excel® Powerpoint PDF®, ASCII, Framemaker, Interleaf, Code and Test files
- A bidirectional interface with the third-party requirements management and authoring tools
- Monitoring of all levels of the workflow, for better project management and efficiency

Modeling the Handset Requirements

To create the handset's Requirements and Functional Overview as a use case diagrams, you must identify the system requirements including the actors, the major function points of the system, and the relationships between them.

Note

Rhapsody supports both black-box (probing the external behavior of a program with inputs) and the next white-box (understanding the program code) analysis approaches.

First, consider the actors that interact with the system:

- `MMI`—Handset user interface, including the keypad and display
- `Network`—System network or infrastructure of the signalling technology

Next, consider the system function points:

- 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 the Requirements Diagram

The Requirements diagram graphically shows the relationship among textual requirement elements.

To create the Requirements diagram, follow these steps:

1. Right-click the `_Requirements` package in the browser and select **Add New** > **Use Case Diagram** from the pop-up menu.
2. Type `Requirements` for the name.
3. Click **OK**.

Adding the Requirements as Textual Annotations

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

The handset model contains the following requirements:

Name	Specification
Req.1.1	The mobile shall be fully registered before a place call sequence can begin.
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.

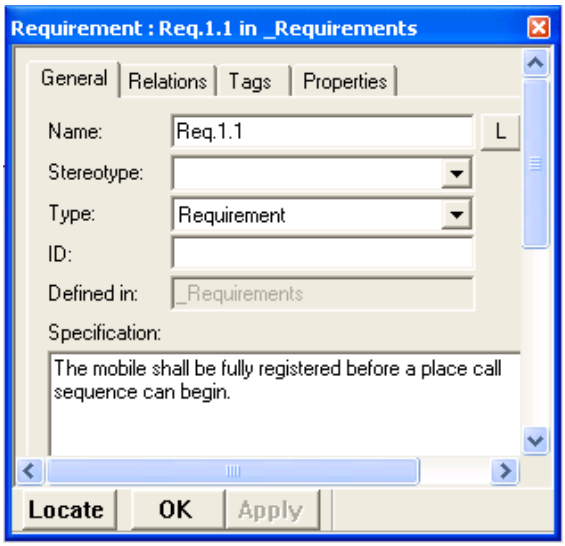
To add these requirements to Rhapsody, follow these steps:

1. In the browser, right-click the `_Requirements` package, and select **Add New** > **Requirement** from the pop-up menu. Rhapsody creates the `Requirements` category and a requirement with a default name of `requirement_n`, where *n* is greater than or equal to 0.
2. Rename the requirement `Req.1.1`.
3. Double-click `Req.1.1`, and the dialog box opens.

4. Type the following in the **Specification** field:

```
The mobile shall be fully registered before a place call sequence can begin.
```

The dialog box should be similar to this example.



5. Click **OK** to apply the changes and close the dialog box.
6. Add the remaining requirements and their specifications in the same manner.

Adding the Requirements to the Diagram

To add these requirements to the Requirements diagram, follow these steps:

1. In the browser, expand the **Packages** and the `_Requirements` category.
2. Select `Req.4.2` and drag it to the top left of the Requirements drawing area.
3. Select `Req.4.1` and drag it below `Req.4.2`.
4. Select `Req.3.2` and drag it to the top center of the drawing area.
5. Select `Req.4.0` and drag it to the lower left side of `Req.3.2`.
6. Select `Req.5.6` and drag it to the lower right side of `Req.3.2`.
7. Select `Req.6.2` and drag it below `Req.5.6`.

Note

For each requirement, right-click on the drawing of the requirement and select **Display Options**. Click the **Name only** radio button and then **OK** to show the requirement name and not the path on the diagram.

Drawing and Defining the Dependencies

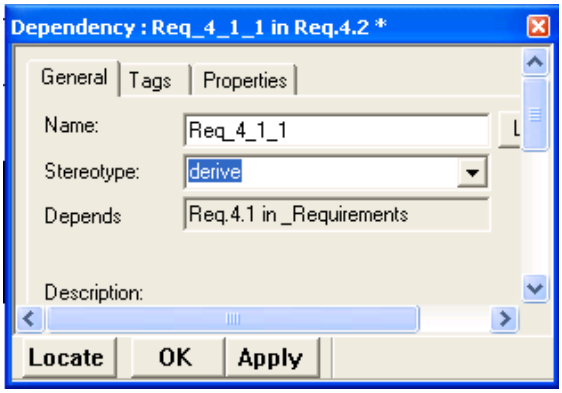
A *dependency* is a direct relationship in which the function of an element requires the presence of and may change another element. You can show the relationship between requirements, and between requirements and model elements using dependencies.

In this example, you set the following types of dependency stereotypes:

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

Now you will define the relationships between requirements with dependencies:

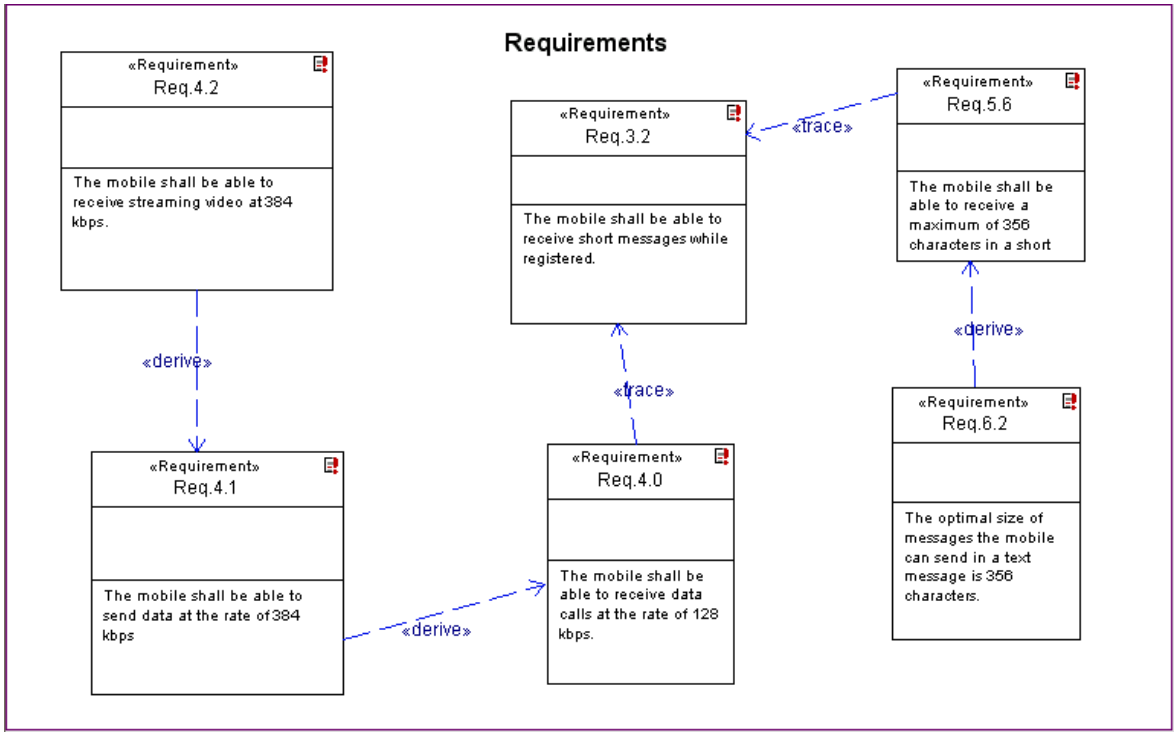
1. Click the **Dependency** icon  .
2. Draw a dependency line from `Req.4.2` to `Req.4.1`. Right-click on the line and select **Features** from the menu. Select `derive` as the **Stereotype**, as shown in this example.



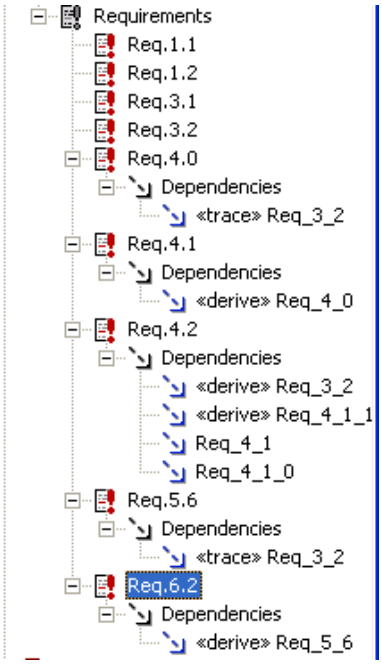
3. Click **OK** to save the change and close the dialog box.
4. Draw a dependency line from `Req.4.1` to `Req.4.0`. Right-click on the line and select **Features** from the menu. Select `derive` as the **Stereotype** and click **OK.**
5. Draw a dependency line from `Req.4.0` to `Req.3.2`. Right-click on the line and select **Features** from the menu. Select `trace` as the **Stereotype** and click **OK.**
6. Draw a dependency line from `Req.5.6` to `Req.3.2`. Right-click on the line and select **Features** from the menu. Select `trace` as the **Stereotype** and click **OK.**

7. Draw a dependency line from `Req.6.2` to `Req.5.6`. Right-click on the line and select **Features** from the menu. Select `derive` as the **Stereotype** and click **OK.**

At this point the Requirements diagram should be similar to this example.



Rhapsody automatically adds the dependency relationships to the browser as shown in this example.



Creating a Use Case Diagram

To create a new use case diagram (UCD) containing the actors and basic use cases, follow these steps:

1. Start Rhapsody if it is not already running and open the handset model if it is not already open.
2. In the browser right-click the `Analysis` package, and select **Add New** > **Use Case Diagram** from the pop-up menu. The New Diagram dialog box opens.
3. In the **Name** field replace the generated name with `Functional_Overview`, then click **OK**.

Rhapsody automatically adds the `Use Case Diagrams` category and the Functional_Overview in the browser. Then it opens the new diagram in the drawing area.

Drawing the Boundary Box

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

To draw the boundary box, follow these steps:

1. Click the **Create Boundary box** icon  on the Drawing toolbar.
2. Click in the upper, left corner of the drawing area and drag to the lower right. Rhapsody creates a boundary box, named `System Boundary Box`.
3. Rename the boundary box `Handset Protocol Stack`.

Drawing the Actors

Create the following two actors that interact with the system: `MMI` and `Network`.

To draw the actors, follow these steps:

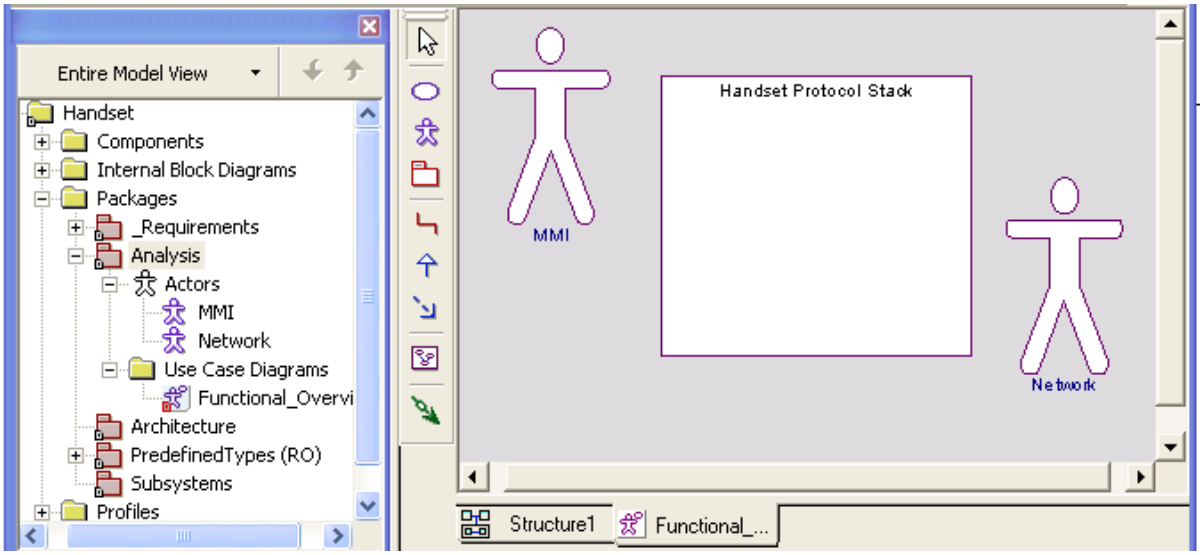
1. Click the **Create Actor** icon  on the Drawing toolbar.
2. Click the left side of the drawing area. Rhapsody creates an actor with a default name of `actor_n`, where *n* is greater than or equal to 0.
3. Rename the actor `MMI`, then press Enter.

Note

Because code can be generated using the specified names, do not include spaces in the names of actors.

4. Draw an actor on the right side of the drawing area named `Network`, then press Enter.

At this point, the browser and diagram should be similar to this example.



Adding Use Cases to the Functional Overview

The Functional Overview needs the following use cases:

- `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.

Rhapsody automatically adds the `Use Case Diagrams` category and the new UCD to the package in the browser and opens the new diagram in the drawing area.

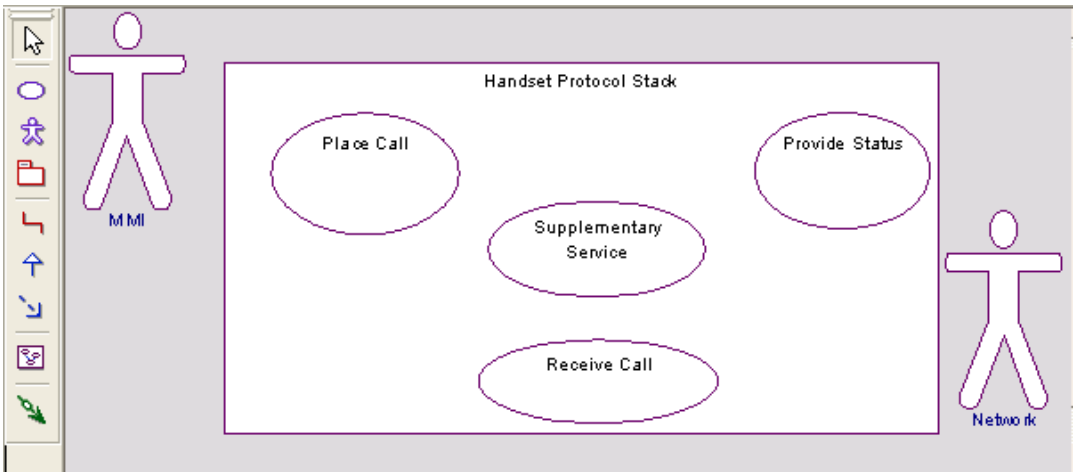
To draw these use cases in the Functional Overview, follow these steps:

1. Click the **Create Use Case** icon on the Drawing toolbar.

2. Click in the upper left of the boundary box. Rhapsody creates a use case with a default name of `usecase_n`, where *n* is equal to or greater than 0.

3. Rename the use case `Place Call`.

4. Create three more use cases inside the boundary box named `Supplementary Service`, `Receive Call`, and `Provide Status`. At this point the Functional Overview diagram should be similar to this example.



Defining Use Case Features

You can define the features of a use case, associate the use case with a different main diagram, and enter a description using the Features dialog box. You can access the Features dialog box from the browser or the diagram.

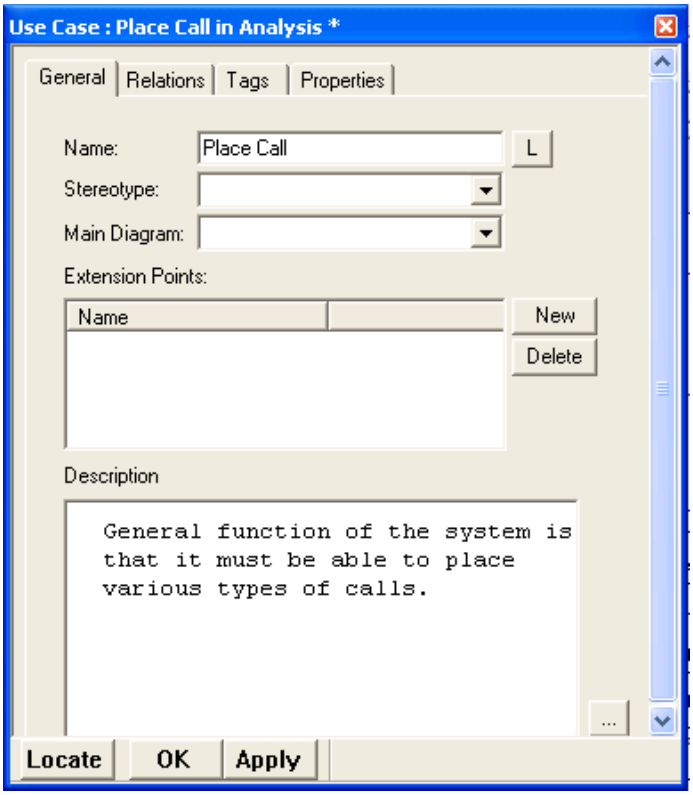
To define use case features, follow these steps:

1. In the browser, expand the `Analysis` package and `Use Cases` category. Double-click the `Place Call` use case, or right-click and select **Features** from the pop-up menu. The Features dialog box opens.

2. In the **Description** text field, type the following text to describe the purpose of the `Place Call` use case:

```
General function of the system is that it must be able to place
various types of calls.
```

To use the internal editor to enter the text, click the ellipse button. The completed dialog box should resemble this example.



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

4. Open the Features dialog box for the `Supplementary Service` use case, and type the following in the **Description** text field to describe its purpose:

```
A supplementary service is a short message, call forwarding, call
holding, call barring, or conference calling.
```

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

6. Open the Features dialog box for the `Receive Call` use case, and type the following in the **Description** text field to describe its purpose:

```
General function of the system is that it must be able to
receive and terminate calls.
```

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

8. Open the Features dialog box for the `Provide Status` use case, and type the following in the **Description** text field to describe its purpose:

```
The stack must be able to communicate with the network in order
to provide the user with visual status such as signal strength
and current registered network. It must also be able to handle
user requests for network status and location registration.
```

9. Click **OK** to apply the changes and close 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. In this example, you create the associations showing the connections between actors and the relevant use cases using association lines.

To draw association lines, follow these steps:

1. Click the **Create Association** icon  on the Drawing toolbar.
2. Click the edge of the `MMI` actor, then click the edge of the `Place Call` use case. Rhapsody creates an association line with the name label highlighted. You do not need to name this association, so press Enter.
3. Create an association between the `MMI` actor and the `Receive Call` use case, then press Enter.
4. Create an association between the `Network` actor and the `Receive Call` use case, then press Enter.
5. Create an association between the `Network` actor and the `Provide Status` use case, then press Enter.
6. In the browser, expand the `Actors` category to view these relations for the actors and use cases.

   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 also 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

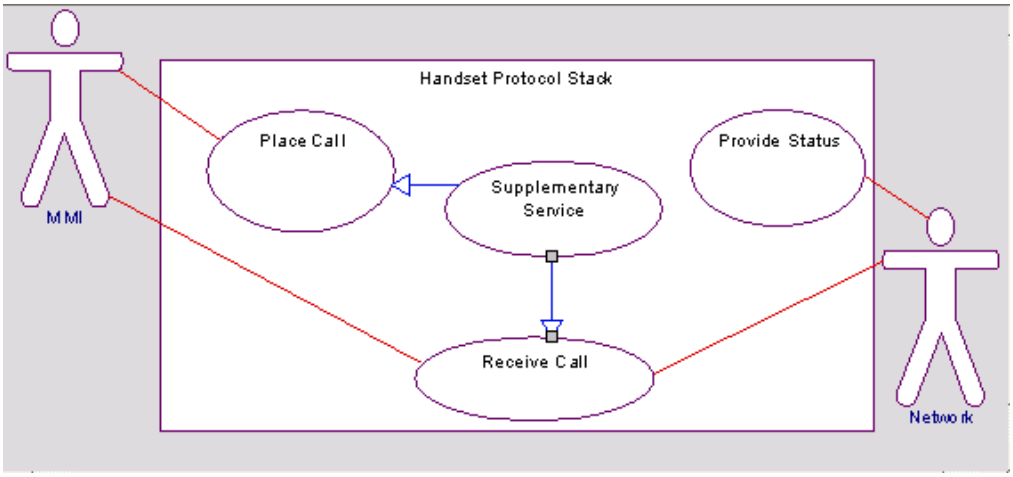
A *generalization* is a relationship between a general element and a more specific element. The more specific element inherits the properties of the general element and is substitutable for the general element. A generalization lets you derive one use case from another.

The `Supplementary Service` use case is a more specific case of placing a call, and it is a more specific case of receiving a call. In this example, you will draw generalizations indicating that `Supplementary Service` is derived from the `Place Call` use case and the `Receive Call` use case.

To draw a generalization, follow these steps:

1. Click the **Create Generalization** icon  on the Drawing toolbar.
2. Click the `Supplementary Service` use case and draw a line to the `Place Call` use case.
3. Click the `Supplementary Service` use case and draw a line to the `Receive Call` use case.
4. In the browser, expand the `Supplementary Service` use case and note that `Place Call` and `Receive Call` are SuperUseCases for `Supplementary Service`.

You have completed drawing the Functional Overview UCD. Your diagram should be similar to this example.



Drawing 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.

To create the Place Call Overview UCD, follow these steps:

1. In the browser, right-click the `Use Case Diagrams` category in the `Analysis` package, and select **Add New Use Case Diagram** from the pop-up menu. The New Diagram dialog box opens.
2. Type `Place Call Overview`, then click **OK**.

Rhapsody automatically adds the name of the new UCD to the browser, and opens the new diagram in the drawing area.

Drawing the Use Cases

The Place Call Overview UCD contains the following uses cases:

- `Place Call`—The user can place various types of calls. 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, follow these steps:

1. In the browser, expand the `Analysis` package and `Use Cases` category.
2. Select the `Place Call` use case and drag it to the top center of the UCD.
3. Click the **Create Use Case** icon  on the Drawing toolbar.
4. Create a use case in the lower left of the drawing area, named `Data Call`.
5. Create a use case in the lower right of the drawing area, named `Voice Call`.

Defining Use Case Features

Now you add descriptions to the `Data Call` and `Voice Call` use cases as follows:

1. In the Place Call Overview UCD or browser, double-click the `Data Call` use case, or right-click and select **Features** from the pop-up menu. The Features dialog box opens.
2. In the **Description** text field, type the following text to describe its purpose:

   ```
   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.
   ```

3. Click **OK** to apply the changes and close the Features dialog box.
4. Double-click the `Voice Call` use case, or right-click and select **Features** from the pop-up menu. The Features dialog box opens.
5. In the **Description** text field, type the following text to describe its purpose:

   ```
   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.
   ```

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

Drawing Generalizations

In this example, draw generalizations to show that the `Data Call` use case and the `Voice Call` use case derive from the `Place Call` use case as follows:

1. Click the **Create Generalization** icon  on the Drawing toolbar.
2. Click the edge of the `Data Call` use case and draw the line to the edge of the `Place Call` use case.
3. Click the edge of the `Voice Call` use case and draw the line to the edge of the `Place Call` use case.

In the next section, you add the requirements elements to the model and then draw the requirements on the Place Call Overview UCD.

Drawing Requirements

You can add requirement elements to UCDs to show how the requirements trace to the use cases.

To add the requirements to the use case diagram, follow these steps:

1. Select `Req.1.1` and drag it to the right of the `Place Call` use case.
2. Select `Req.4.1` and drag it to the lower left of the `Data Call` use case.
3. Select `Req.4.2` and drag it to the lower right of the `Data Call` use case.

Setting the Display Options for Model Elements

You can set the type of information and the graphical format to display for model elements using the Display Options dialog box.

In this example, you will set the display options to **Name** to show only the name of the requirement on the diagram as follows:

1. Right-click `Req.1.1` in the diagram and select **Display Options** from the pop-up menu. The Requirement Display Options dialog box opens.
2. The **Show** group box specifies the information to display for the requirement. Select the **Name** radio button to display the name of the requirement.
3. Click **OK**.
4. Set the display options for `Req.4.1` and `Req.4.2` to **Name**.

Drawing Dependencies

In this example, draw dependencies between the requirements and the use cases as follows:

1. Click the **Dependency** icon  on the Drawing toolbar.
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, expand the `Requirements` category to check that the dependency relationship is listed there.

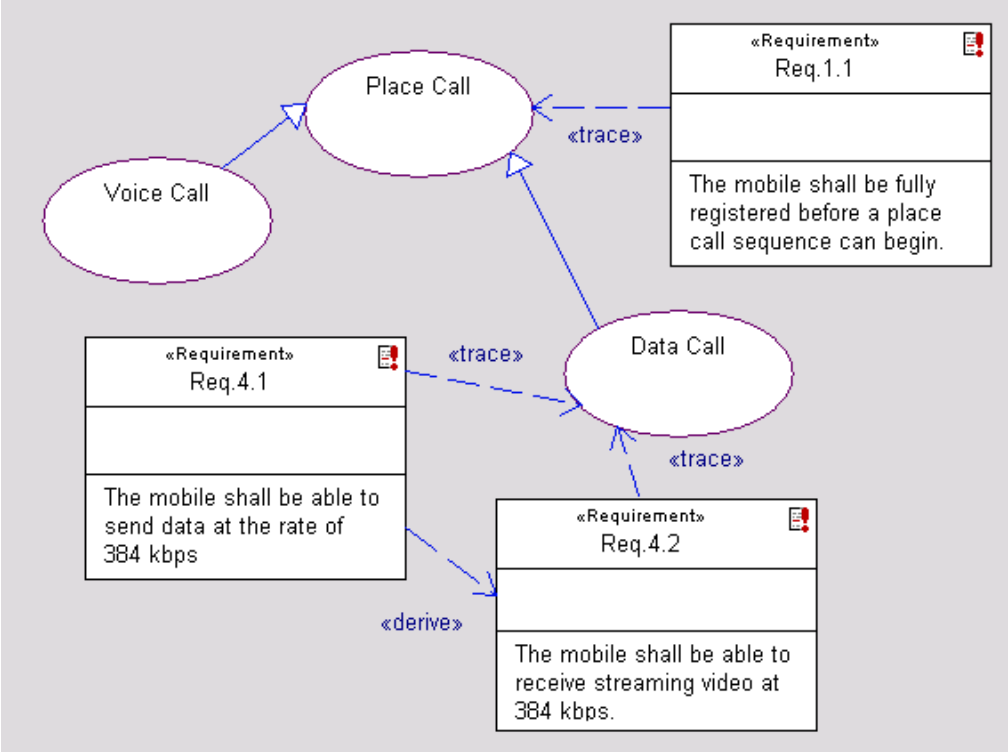
Defining the Stereotype of a Dependency

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. Rhapsody includes predefined stereotypes, and you can also define your own stereotypes. Stereotypes are enclosed in guillemets on diagrams, for example, «derive».

To define the stereotype of a dependency, follow these steps:

1. Double-click the dependency between `Req.1.1` and `Place Call`, or right-click and select **Features** from the pop-up menu.
2. Select `trace` from the **Stereotype** pull-down list.
3. Click **OK** to apply the changes and close the Features dialog box.
4. Set the stereotype of the dependency between `Req.4.1` and `Data Call` to `trace`.
5. Set the stereotype of the dependency between `Req.4.2` and `Data Call` to `trace`.
6. Set the stereotype of the dependency between `Req.4.1` and `Req.4.2` to `derive`.

Your completed drawing the Place Call Overview UCD should be similar to this example.



Capturing the Design Structure

Internal and Block Definition 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.

This section demonstrates the creation of the following block diagrams:

- Protocol Stack Architecture (Block Definition Diagram) identifies the system-level components and flow of information
- ConnectionManagement (Internal Block Diagram)
- DataLink Connections (Internal Block Diagram)
- MobilityManagement (Internal Block Diagram)

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 system's functions into subsystem components.

Block diagrams define the components of a system and the flow of information between components. *Structure diagrams* can have the following parts:

- *Block* contains parts and may also include links inside a block.
- *Actors* are the external interfaces to the system.
- *Service Port* is a distinct interaction point between a class, object, or block and its environment.
- *Dependency* shows dependency relationships, such as when changes to the definition of one element affect another element
- *Flow* specifies the exchange of information between system elements at a high level of abstraction.

Creating an Block Definition Diagram

An Block Definition Diagram identifies the system components (blocks) and describes the flow of data between the components from a black-box perspective. The following sections describe the decomposition of these system components (blocks):

- Select
- Block
- Part
- Create Port
- Link
- Flow

To create an block definition diagram, follow these steps:

1. Start Rhapsody if it is not already running and open the handset model if it is not already open.
2. In the browser, right-click the `Architecture` package, then select **Add New** > **Block Definition Diagram** from the pop-up menu. The New Diagram dialog box opens.
3. Type `High Level Architecture` for the diagram name.
4. Click **OK** to close the dialog box.

Rhapsody automatically creates the `Block Definition Diagrams` category in the browser and adds the name of the new block definition diagram. In addition, Rhapsody opens the new diagram in the drawing area.

Drawing Blocks

Blocks specify the components of the system. The `Handset` model contains the following three system components or functions:

- `ConnectionManagementBlock` handles the reception, setup, and transmission of incoming and outgoing call requests.
- `MobilityManagementBlock` handles the registration and location of users.
- `DataLinkBlock` monitors registration.

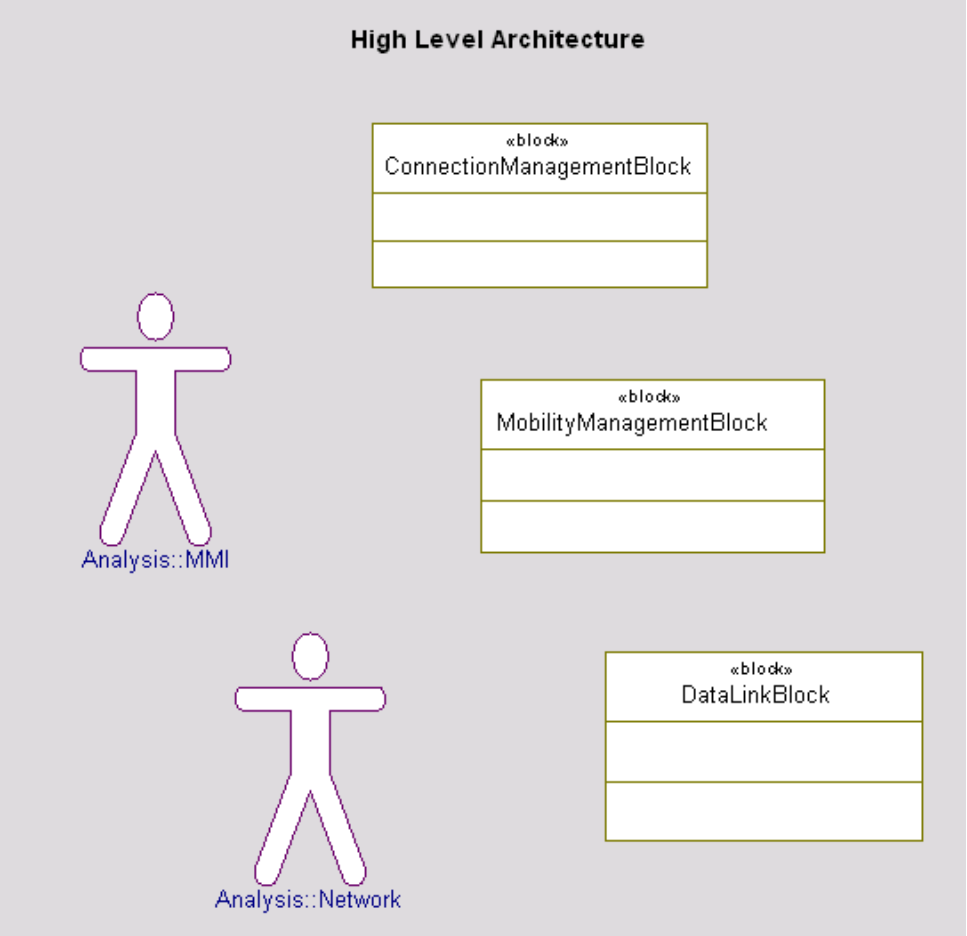
To draw the blocks, follow these steps:

1. Click the **Block** icon  in the Drawing toolbar.
2. In the top center of the drawing area, click-and-drag or just click. Rhapsody creates a block with a default name of `block_n`, where *n* is equal to or greater than 0.
3. Rename the block `ConnectionManagementBlock`.
4. In the upper right of the drawing area, create a block named `MobilityManagementBlock`.
5. In the bottom right of the drawing area, create a block named `DataLinkBlock`.

Adding Actors to the Diagram

To create the data flow from and to the actors and the blocks, follow these steps:

1. Open the `Analysis` and `Actors` sections of the browser.
2. Drag the `MMI` and `Network` actors from the browser into the block diagram. At this point your diagram should be similar to this example.



Drawing Service Ports, Flows, and Links

A service port is a distinct interaction point between a Block or Part and its environment. *Service 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 Block or Part. To draw service ports, follow these steps:

1. Click the **Service Port** icon  in the Drawing toolbar.
2. Click on the left edge of `ConnectionManagementBlock` and create a service port named `call_req`. This service port sends and relays messages to and from the user interface.
3. Click on the right edge of `ConnectionManagementBlock` and create a service port named `network`. This service port sends and relays messages from `MobilityManagementBlock`.
4. Click on the right edge of `MobilityManagementBlock` and create a service port named `mm_network`. This service port sends and relays messages from ConnectionManagementBlock.
5. Click on the bottom edge of `MobilityManagementBlock` and create a service port named `mm_dl`. This service port relays registration information to DataLinkBlock.
6. Click on the top edge of `DatalinkBlock` and create a service port named `dl_in`. This service port relays information between DataLinkBlock and MobilityManagementBlock.
7. Click on the left edge of `DatalinkBlock` and create a service port named `data_net`. This service port relays information between DataLinkBlock and the network.

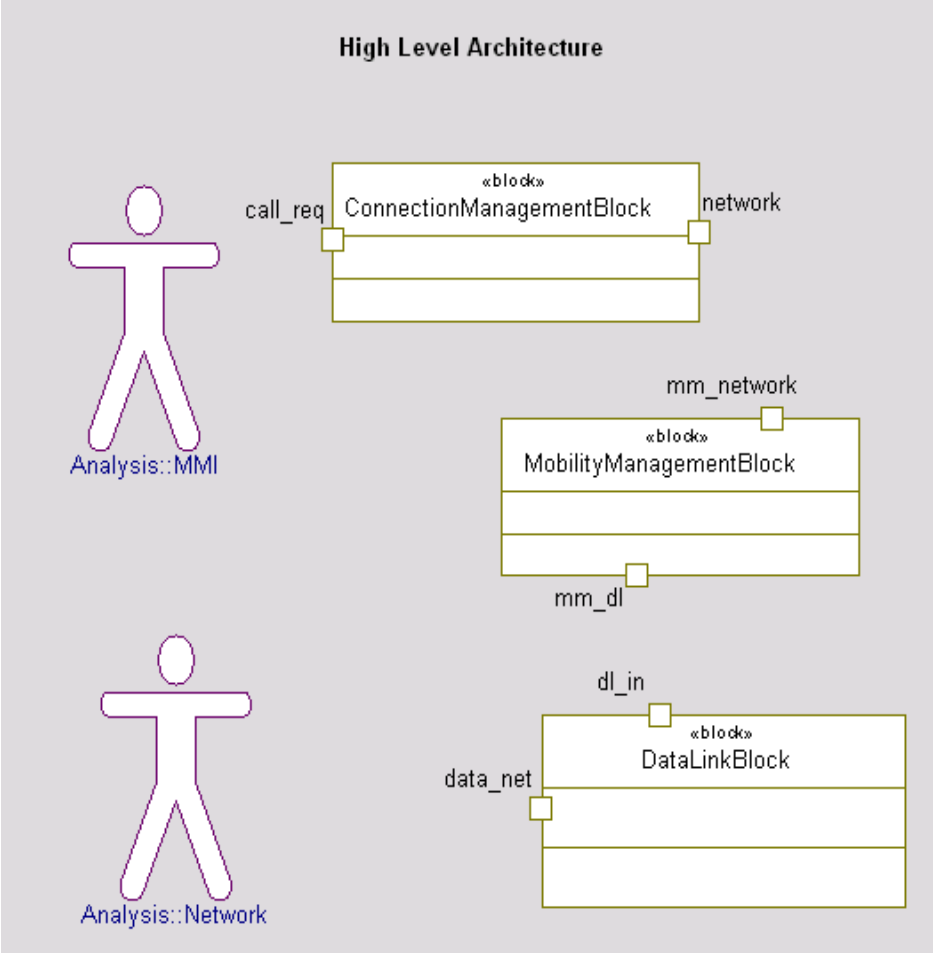
Rhapsody automatically adds the service ports you created to the Blocks category in the browser.

Specifying Service Port Attributes

You can specify ports as behavioral ports to 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 example, set the `call_req` port to be behavioral port using these steps:

1. Double-click the `call_req` port or right-click and select **Features** from the pop-up menu.
2. In the **General** tab, set `Behavior` for the **Attributes** value.
3. Click **OK** to apply the changes and close the dialog box. At this point your diagram should resemble this example.



Connecting the Architecture through Parts and Links

After specifying the blocks and actors, it is necessary to define the connections between them before specifying what types of items that flow on those connections. To accomplish this, specify parts from the blocks and create links between the service ports to define the path the information flow.

Follow these steps first to change the actors and blocks into parts:

1. Right-click on the `ConnectionManagementBlock` block and select **Make an Object**.
2. Right-click on the `MobilityManagementBlock` block and select **Make an Object**.
3. Right-click on the `DataLinkBlock` block and select **Make an Object**.
4. Right-click on the `MMI` actor and select **Make an Object**.
5. Right-click on the `Network` actor and select **Make an Object**.

Note

The number 1 that appears in the upper left hand corner of the object boxes indicates the number of *parts* in a particular block or actor that exists in this architecture. The user can set the number manually, but for this example, the number of parts is limited to 1 for each element.

Creating Ports on Actors

Now create Service Ports on the Actor objects to specify with which Block Parts they communicate.

1. Select the **Service Port** icon  on the Drawing bar and click on the top of the `itsMMI:MMI` object and name the service port `ui_req`.
2. Place another service port on `itsNetwork:Network` by clicking on the right side of the object and name the service port `net_in`.

Connecting the Architecture

To create the link connections between the parts of the architecture so that flow information can be specified, follow these steps:

1. Select the **Link** icon  on the Drawing bar.
2. Click once on the `ui_req` service port and once on the `call_req` service port to create a link. Press **Enter**.
3. Connect the `network` and `mm_network` service ports in the same manner.
4. Connect the `mm_dl` and `dl_in` service ports.
5. Connect the `data_net` and `net_in` service ports.

Drawing Flows

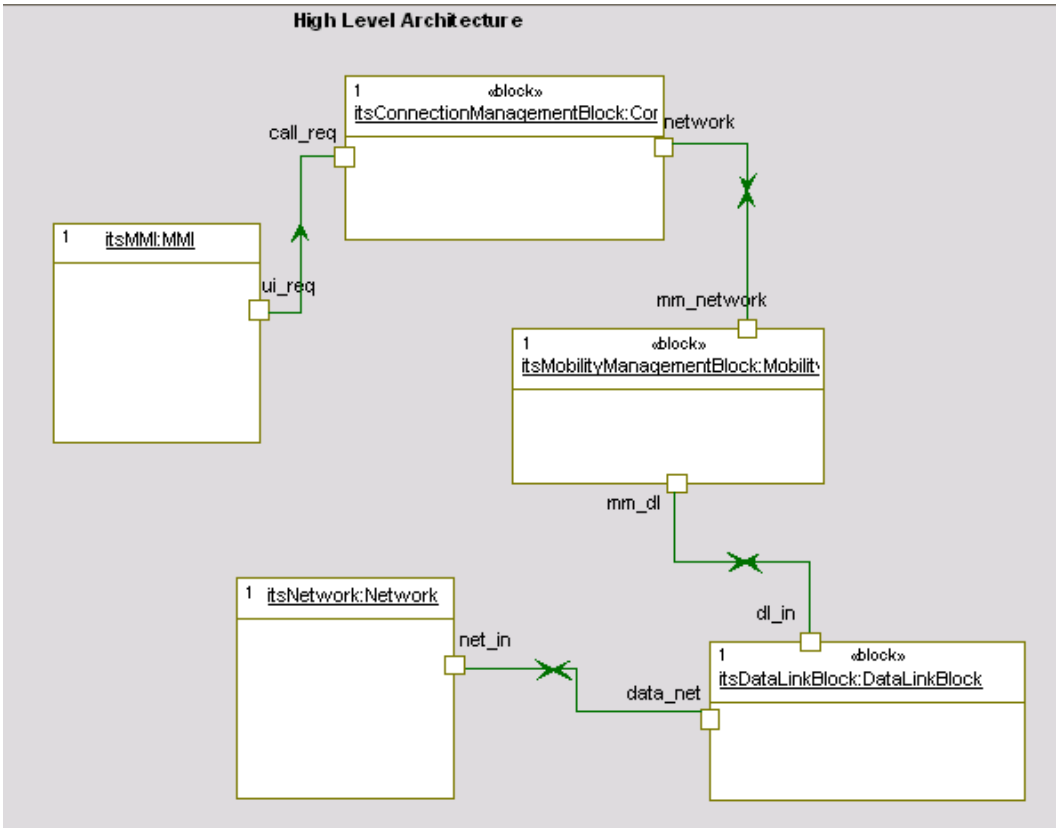
Flows specify the exchange of information between system elements. They allow you to describe the flow of data and commands within a system at an early stage, before committing to a specific design. To draw flows between ports, objects, and blocks, follow these steps:

1. Click the **Flow** icon  on the Drawing toolbar.
2. To create a flow, click in the middle of link line between `ui_req` and `call_req` service ports. Press **Enter**.
3. Create a flow on the link between `network` and `mm_network` service ports. Press **Enter**.
4. Create a flow on the link between `mm_dl` and `dl_in` service ports. Press **Enter**.
5. Create a flow on the link between `data_net` and `net_in` service ports. Press **Enter**.

Changing the Direction of the Flow

Information can flow from one element to another or between elements in either direction. Changing the flows between the blocks to bidirectional indicates that information can flow in either direction between system elements. To change the direction of the flow or make the flow bidirectional, follow these steps:

1. Double-click the *flow arrowhead* between `ConnectionManagementBlock` and `MobilityManagementBlock`, or right-click and select **Features** from the pop-up menu.
2. In the **General** tab, select `Bidirectional` from the **Direction** pull-down menu.Click **OK** to apply the changes and close the dialog box.
3. Using the same method, set the flow between `MobilityManagementBlock` and `DataLinkBlock` to also be bidirectional.
4. Set the flow between the `DataLinkBlock` and the `Network` objects. At this point your diagram should resemble this example.



Specifying the Flow Items

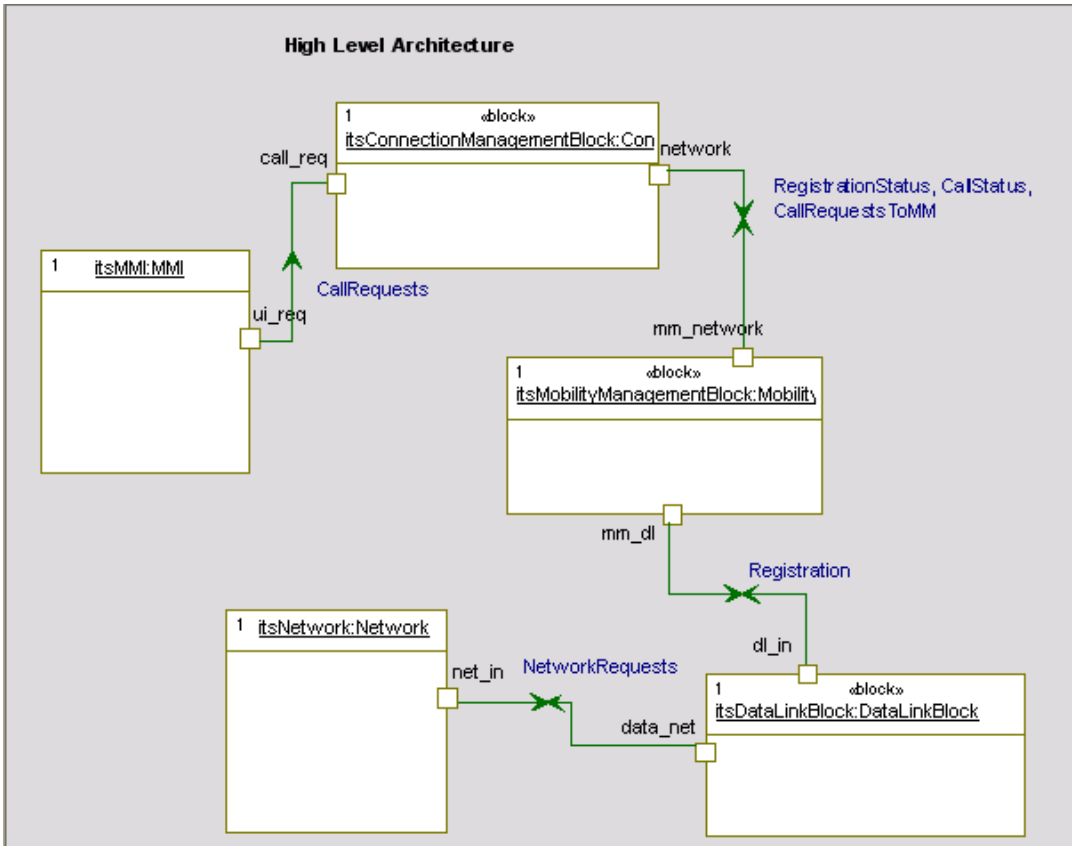
Once you have determined how communication occurs through flows, you can specify the information that passes over a flow using a *flow item*. A flow item can represent either pure data, *data instantiation*, or *commands* (*events*).

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.

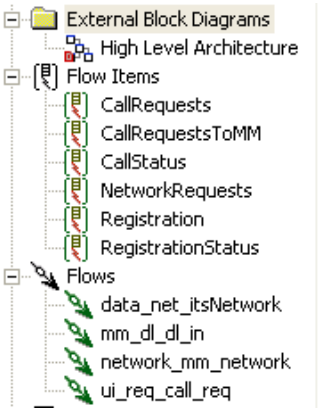
To specify the flow items, follow these steps:

1. Double-click the *point of the arrowhead* in the link between the `MMI` and `call_req` port. Select the **Details** tab of the Features dialog box that opens.
2. Click the **<Add>** row in the list of information elements and select **FlowItem** from the pop-up list. The flow items Features dialog box opens.
3. Enter `CallRequests` as the **Name**. This flow item represents all user interface requests into the system.
4. Click **OK** to apply your changes and close the dialog box for the new element.
5. Click **OK** to apply your changes and close the dialog box for the flow.
6. Using the same method, create three flow items to describe the flow between the `network` port and the `mm_network` port. Name the three flows `RegistrationStatus`, `CallStatus`, and `CallRequestsToMM`. These flow items represent the relay of information between the main call control logic (`ConnectionManagement`) and user location (`MobilityManagement`).
7. Create a flow item for the flow between the `mm_dl` port and the `dl_in` port named `Registration`. This flow item represents network registration status information.

8. Create a flow item for the flow between the `data_net` port and the `Network` actor named `NetworkRequests`. This flow item represents all network information into and out of the system. At this point the diagram should resemble this example.



9. In the browser, expand the Flows category and the Flow Items category to view the newly created flows and flow items, as shown in this example.



Changing the Line Shape

Rhapsody has three line shapes that can be used when drawing line and arrow elements: straight, spline, and rectilinear.

To change the line shape, right-click the line in the drawing area, select **LineShape** from the pop-up menu, and then one of the following options:

- **Straight** changes the line to a straight line.
- **Spline** changes the line to a curved line.
- **Rectilinear** changes the line to a group of line segments connected at right angles. This is the default line shape.

The last option, **Reroute**, is used to remove excess control points to make the line more fluid.

Specifying the Port Contract

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

- **Contract-based ports** allow you to 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 actors or parts). 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.

- **Noncontract-based 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.

In this example, you specify the provided and required interfaces for the mm_dl and dl_in ports.

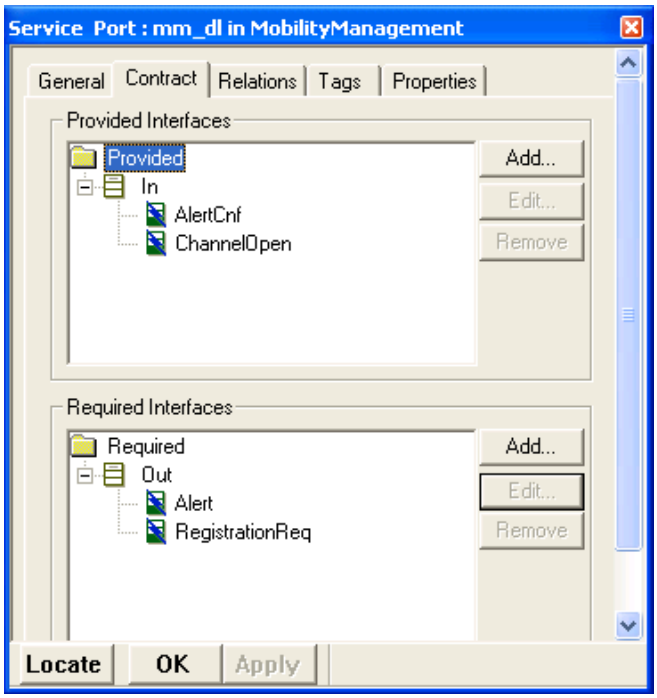
Note

Depending on your workflow, you might identify the communication scenarios using sequence diagrams before defining the port contracts. Refer to the **Creating a Sequence Diagram section** for more information.

To specify the port contract, follow these steps:

1. Double-click the `mm_dl` port, or right-click and select **Features** from the pop-up menu.
2. Select the **Contract** tab.
3. Select the **Provided** folder icon and click the **Add** button. The Add new interface dialog box opens.
4. Select `In` from the pull-down list, then click **Apply** to save the changes and leave the dialog box open.

5. Select the **Required** folder icon and click the **Add** button. Select `Out` from the pull-down list. Click **Apply** adds the provided and required interfaces and leave the dialog box open.The dialog box lists the interfaces you just specified.



6. In the **Operations** tab, click **<New>** and select **Reception** from the pop-up menu.

7. Type `AlertCnf` for the Event name and click **OK**. A message displays that an event with the selected name could not be found. Click **Yes** to create the new event. Rhapsody adds the reception to the Operations tab.

8. In the **Operations** tab, click **<New>** and select `Reception` from the pop-up menu. The New Reception dialog box opens.

9. Add the receptions `ChannelOpen`, `Alert`, and `RegistrationReq`.

10. Click **OK** to close the dialog box and return to the original dialog box.

11. Click **OK** to close the dialog box. Rhapsody adds the provided and required interfaces to the mm_dl port in the Block Definition Diagram. Rhapsody also adds the receptions to the Events category in Architecture package in the browser.

12. To specify the port interfaces for dl_in, double-click the dl_in port, or right click and select Features from the pop-up menu. The Features dialog box opens.

13. Select the **General** tab and select **In** from the **Contract** pull-down list.

14. Select the **Contract** tab. Rhapsody automatically adds the provided interfaces defined as In.

15. Select the **Required** folder icon, then click the **Add** button. The Add new interface dialog box opens.

16. Select **Out** from the pull-down list, then click **OK**. Rhapsody automatically adds the required interfaces defined as Out.

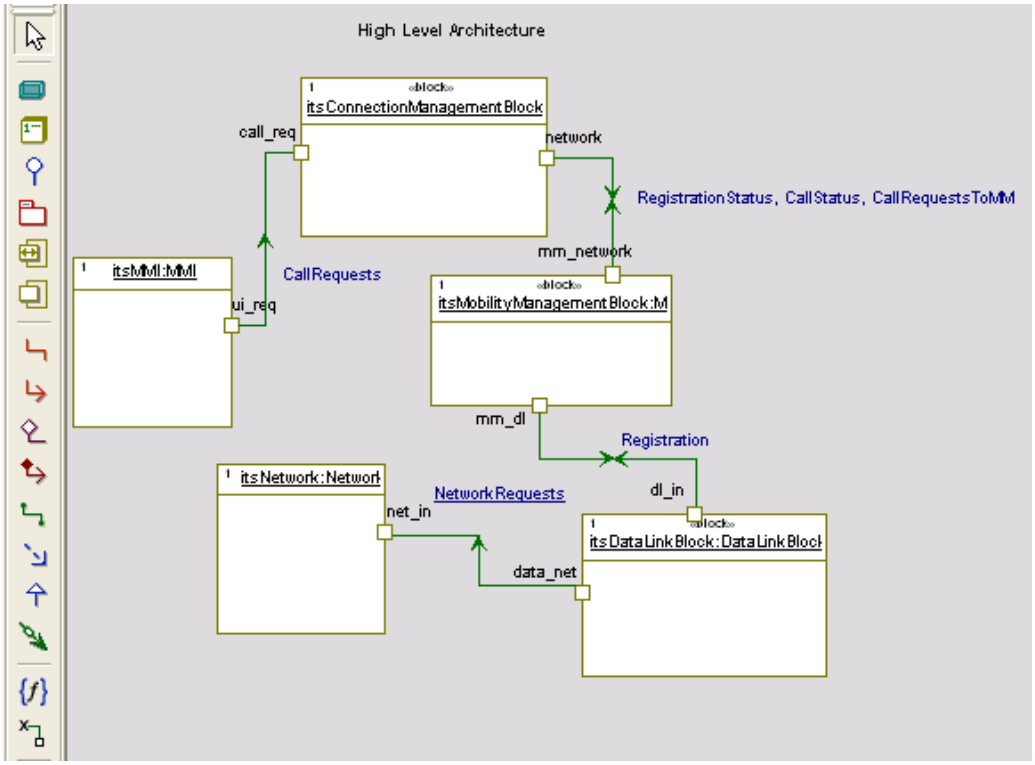
17. Click **OK** to apply the changes and close the dialog box.

Reversing a Port

To change the provided interfaces into the required interfaces and the required interfaces into the provided interfaces, reverse the ports, as follows:

1. Open the **Features** dialog box for the `dl_in` service port.
2. In the **General** tab, set `Reversed` for the **Attributes**. The bottom of the Contract tab displays a message in red that the contract is reversed.
3. Click **OK** to apply the changes and close the dialog box. You have completed the High Level Architecture diagram.

Be certain that the Display Options for each object are set to show All Operations, and your High Level Architecture diagram should resemble this example.



Allocating the Functions Among Blocks

Now that you have captured the architectural design in the Block Definition 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 chapter includes both the system external and internal block 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 system's functions into subsystem components.

Organizing the Blocks Package

Packages let you divide the system into functional domains, or subsystems, which consist of blocks, parts, functions, variables, and other logical artifacts. They can be organized into hierarchies to provide a high level of partitioning. In this example, you will create the following subpackages, which represent the functional subsystems: ConnectionManagement, DataLink, and MobilityManagement.

To create packages within the Subsystems package, follow these steps:

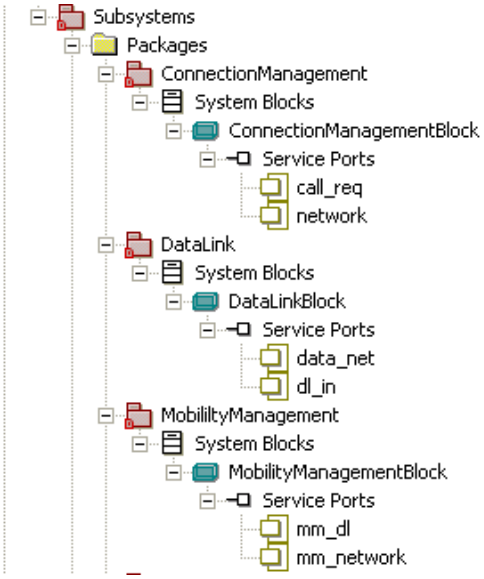
1. In the browser, right-click **Subsystems** and select **Add New** > **Package**. Rhapsody create a new Packages category within Subsystems and a package with the default name package_n, where n is greater or equal to 0.
2. Rename the package `ConnectionManagement`.
3. Right-click **Packages**, select Add **New Packages** from the pop-up menu, and create two additional packages named `DataLink` and `MobilityManagement`.

Organizing Elements

To allocate the blocks from the Block Definition Diagram in the Architecture package, organize them into the elements into their packages using these steps:

1. In the browser, expand the Architecture package and the Blocks category.
2. Click the `ConnectionManagementBlock` and drag it down into the new `ConnectionManagement`.
3. Click the `DataLinkBlock` and drag it into the package.

4. Click the `MobilityManagementBlock` and drag it into the package. The blocks are removed from the Architecture package and added to the Subsystem packages. Your browser should resemble this example.



Note

It is good practice to test the model as it is developed using the simulation feature. This allows you to determine whether the model meets the requirements and to find defects early in the design process. For more information, refer to the **Simulating the Model section**.

Creating the Connection Management Diagram

You can decompose the system-level blocks in the internal block diagram into sub-blocks and corresponding internal block diagrams to show their decomposition. To accomplish this, you create the following subsystem internal block diagrams:

- ConnectionManagement from the ConnectionManagementBlock
- DataLink from the DataLinkBlock
- MobilityManagement from the MobilityManagementBlock

Drawing the Internal Block Diagram

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

To draw the ConnectionManagement internal block diagram, follow these steps:

1. In the browser, expand the ConnectionManagement subsystem package and the Blocks category. Right-click ConnectionManagement and select **Add New** > **Internal Block Diagram**. The New Diagram dialog box opens.
2. Type `ConnectionManagement`, then click OK. Rhapsody automatically creates the Internal Block Diagrams category in the `ConnectionManagementBlock`, and adds the name of the new internal block diagram. In addition, Rhapsody opens the new diagram in the drawing area, which contains the ConnectionManagementBlock and its service ports, as defined in the Internal Block Diagram.

If the service ports are not visible, follow these steps:

1. Right-click the `ConnectionManagementBlock`.
2. From the context menu, select **Ports** > **Show All Ports**.

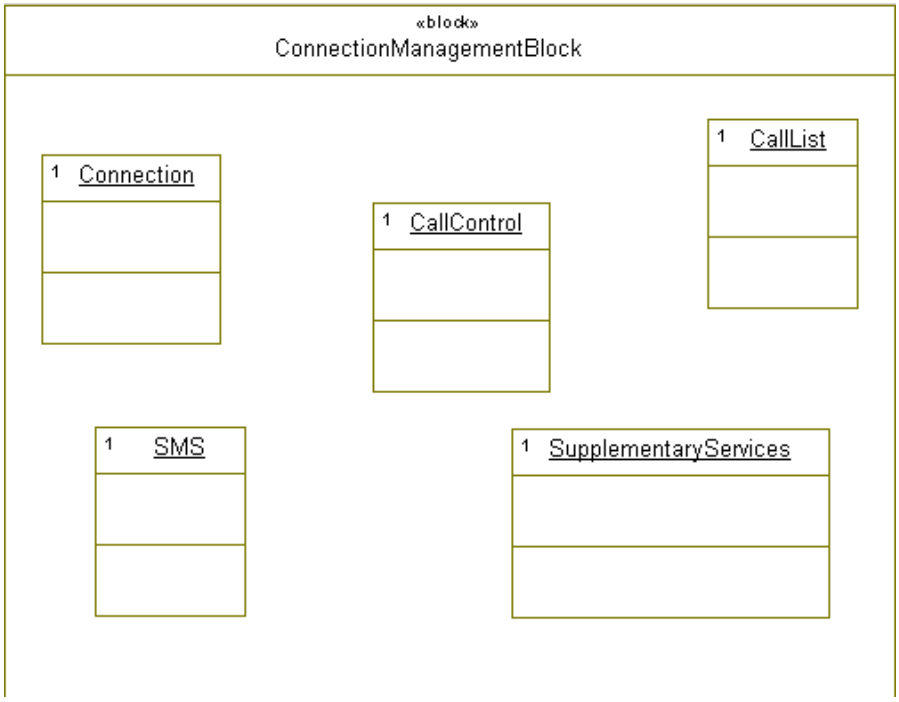
Drawing Parts

In this example, draw the following parts to represent these activities performed by Connection Management:

- Connection—Tracks the number of valid connections
- CallList—Maintains the list of currently active calls
- CallControl—Manages incoming and outgoing calls
- SMS—Manages the short message services.
- SupplementaryServices—Manages the supplementary services, including call waiting, holding, and barring

To draw parts, follow these steps:

1. Click the **Part** icon  in the Drawing toolbar.
2. In the upper, left corner of ConnectionManagement, click or click-and-drag. Rhapsody creates an object with a default name of part_n, where n is equal to or greater than 0.
3. Rename part to be `Connection`, then press **Enter.**
4. Draw additional parts named `CallList`, `CallControl`, `SMS`, and `SupplementaryServices`. At this point your diagram should be similar to this example.



Drawing Service Ports

To draw service ports, follow these steps:

1. Click the **Service Port** icon  in the Drawing toolbar.
2. Click the left edge of the `CallControl` part and create a service port named `cc_mm`. This service port relays messages to and from MobilityManagement.
3. Click the right edge of the `CallControl` part and create a service port named `cc_in`. This service port relays messages from the user interface.

Changing the Placement of Ports

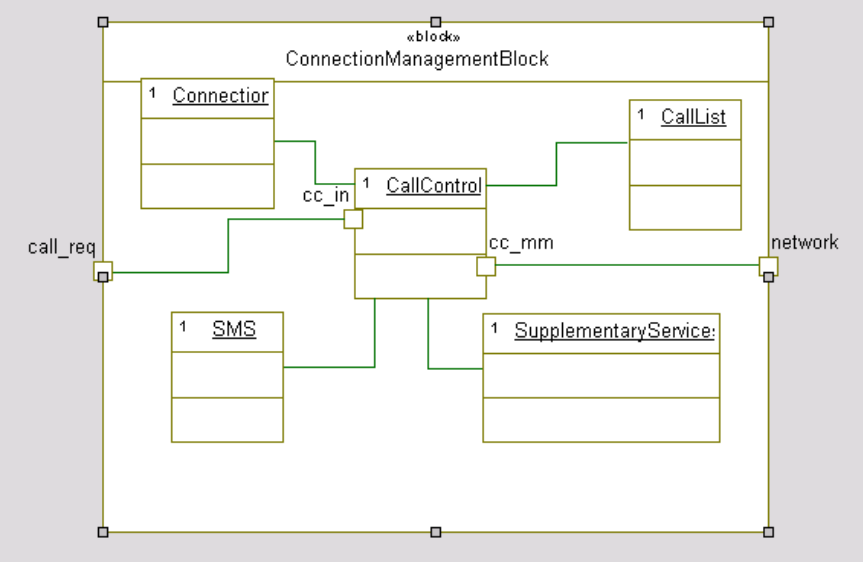
When Rhapsody adds the ConnectionManagementBlock to the diagram, it places the ports defined in the internal block diagram on the boundary. You can change the port placement by selecting the port and dragging it to another location on the part or block.

Drawing Links

A link is an instantiation of an flow. 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 example, draw links between objects and ports, as follows:

1. Right-click on the `ConnectionManagementBlock` and select the **Ports** > **Show All Ports** option. The previously created `call_req` and `network` ports appear on the outside edges of the block.
2. Click the **Link** icon  in the Drawing toolbar.
3. Click the `cc_mm` port, then click the `network` port.
4. Click the `cc_in` port, then click the `call_req` port.
5. Click the `CallControl` part, then click the `Connection` part.
6. Click the `CallControl` part, then click the `CallList` part.
7. Click the `CallControl` part, then click the `SMS` part. Click the `CallControl` part, then click the `SupplementaryServices` part.
8. In the browser, expand the ConnectionManagement category to view the newly created objects under the Parts category and the links under the Links category.

9. You have completed drawing the Connection Management diagram. Your block diagram should be similar to this example.



Creating the DataLink Internal Block Diagram

An internal block diagram of the data link decomposes the DataLinkBlock into its subsystems. It identifies how the system monitors registration.

To create the DataLink diagram as an internal block diagram, follow these steps:

1. In the browser, expand the `DataLink` subsystem package and the `Blocks` category.
2. Right-click `DataLinkBlock` and select **Add New** > **Internal Block Diagram**. The New Diagram dialog box opens.
3. Type `Datalink` for the name, then click **OK**.

Rhapsody automatically creates the `Internal Block Diagrams` category in the `DataLinkBlock`, and adds the name of the new diagram. In addition, Rhapsody opens the new diagram in the drawing area, which contains the DataLinkBlock and its ports with the required and provided interfaces as defined in the Internal Block Diagram.

If the ports are not visible, follow these steps:

1. Right-click the internal block diagram in the drawing area.
2. From the context menu, select **Ports** > **Show All Ports**.

Drawing the RegistrationMonitor Part

In this example, draw the `RegistrationMonitor` part to represent the activity performed by the DataLinkBlock as follows.

1. Click the **Part** icon  in the Drawing toolbar.
2. Click or click-and-drag in the center of `DataLinkBlock`.
3. Type `RegistrationMonitor` and press **Enter**.

Drawing the Registration Request Port

To draw ports, follow these steps:

1. Click the **Service Port** icon  in the Drawing toolbar.
2. Click the right edge of `RegistrationMonitor` and create a port named `reg_request`. This port relays registration requests and results.

Linking the Request and DataLink Ports

To draw links, follow these steps:

1. Click the **Link** icon  in the Drawing toolbar.
2. Click the `reg_request` port, then click the `dl_in` port. Press **Enter**.

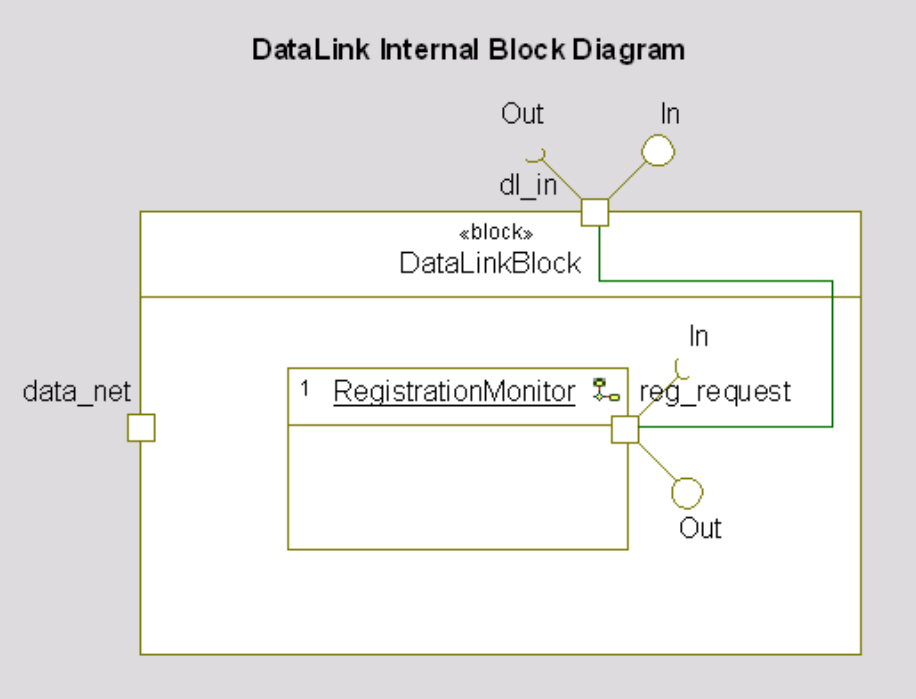
Specifying the Port Contract and Attributes

Now you specify the port contract and features for reg_request as follows:

1. Double-click the `reg_request` port, or right-click and select Features from the pop-up menu. The Features dialog box opens.
2. In the **General** tab, click the `Behavior` and `Reversed` radio buttons to set them as the **Attributes**. Click **Apply** to save the changes and keep the dialog box open.
3. Select the **Contract** tab.
4. Select the **Provided** folder icon and click the **Add** button. The Add new interface dialog box opens.
5. Select `In` from the pull-down list, then click **Apply** to save the changes and leave the dialog box open.
6. Select the **Required** folder icon and click the **Add** button. Select `Out` from the pull-down list.
7. Click **OK** to apply the changes and close the dialog box. Rhapsody automatically adds the provided and required interfaces.

You have completed drawing the DataLink internal block diagram. Rhapsody automatically adds the newly created parts, links, and ports to the DataLinkBlock in the browser.

Your diagram should be similar to this example.



Creating the MobilityManagement Internal Block Diagram

The MobilityManagement internal block diagram decomposes the MobilityManagementBlock into its subsystems. MobilityManagement supports the mobility of users including registering users on the network and providing their current location.

To create the MobilityManagement internal block diagram, follow these steps:

1. In the browser, expand the `MobilityManagement` package.
2. Right-click `MobilityManagement` and select **Add New** > **Internal Block Diagram**.
3. Type `MobilityManagementBlock` for the name and click **OK**.

Rhapsody automatically creates the new diagram and opens it in the drawing area.

If the ports are not visible, follow these steps:

1. Right-click the block.
2. From the context menu, select **Ports** > **Show All Ports**.

Drawing the Registration, Location, and MMCallControl Parts

In this example, draw the following parts to represent the activities performed by MobilityManagement:

- Registration—Maintains the registration status
- Location—Tracks the location of users
- MMCallControl—Maintains the logic for MobilityManagement

To draw parts, follow these steps

1. Click the **Part** icon  in the Drawing toolbar.
2. In the upper, left corner of MobilityManagement, click or click-and-drag.
3. Type `Registration`, and then press **Enter**.
4. Draw two more parts named, `Location` and `MMCallControl`.

Drawing Service Ports and Links

To draw service ports, follow these steps:

1. Click the **Service Port** icon in the Drawing toolbar.
2. Click the left edge of the `MMCallControl` part and name the service port `mm_cc`. This service port relays information to ConnectionManagementBlock
3. Click the right edge of the `MMCallControl` part and name the port `cc_in`. This service port sends and receives information from the DataLinkBlock

To draw a link between two parts, follow these steps:

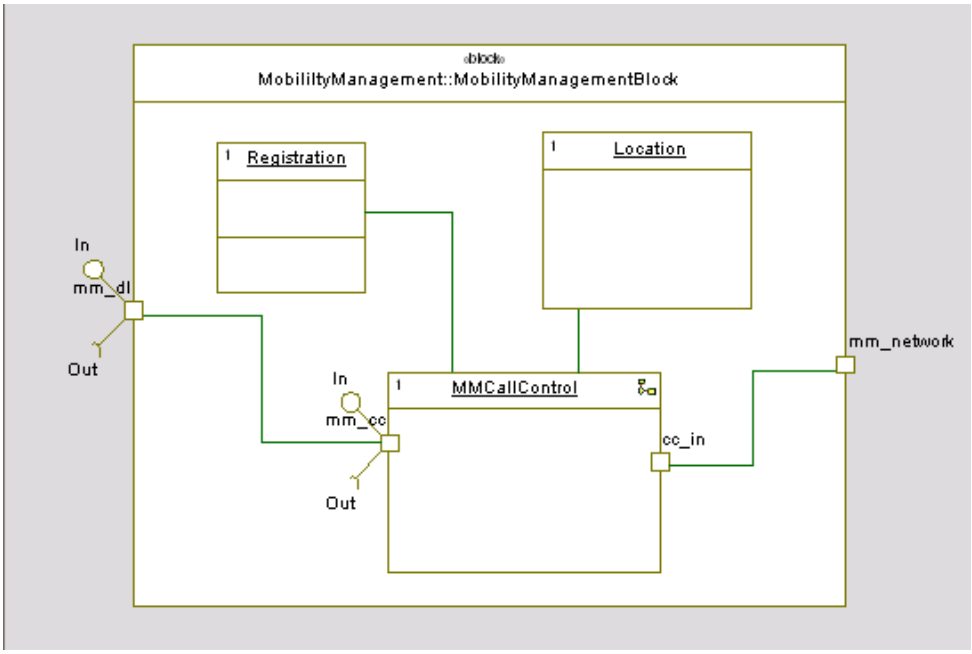
1. Click the **Link** icon on the Drawing toolbar.
2. Click the `cc_in` service port, then click the `mm_network` service port.
3. Click the `MMCallControl` part, and click the `Registration` part.
4. Click the `MMCallControl` part, and click the `Location` part.

Specifying the Port Contract and Attributes

Now you specify the port contract and attributes for the mm_cc service port as follows:

1. Double-click the mm_cc port to display the Features dialog box.
2. In the **Contract** tab, select the **Provided** folder icon and click the **Add** button. The Add new interface dialog box opens.
3. Select `In` from the pull-down list, then click **Apply** to save the changes and leave the dialog box open.
4. Select the **Required** folder icon and click the **Add** button. Select `Out` from the pull-down list.
5. Click **OK** to apply the changes and close the dialog box. Rhapsody automatically adds the provided and required interfaces.

The completed MobilityManagementBlock internal diagram should resemble this example.



Capturing Equations in Parametric Diagrams

Parametric diagrams allow you to capture *equations* graphically. These diagrams are non-executable diagrams that force Rhapsody to determine *user locations* and store the information in the system.

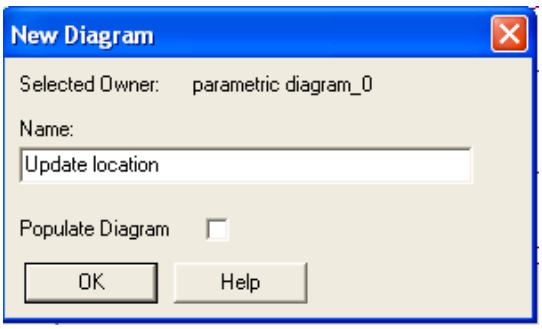
This is accomplished by binding the input data to parametric constraints, performing the calculation in the parametric constraint, and then flowing the output into other parametric constraints. Each parametric constraint shows the piece of the equation that it calculates.

Creating a Parametric Diagram

In this example, you graphically describe the function of locating a user within the MobilityManagementBlock. In order to accomplish this task, you need to describe the algorithm used to locate the user in a parametric diagram.

To create the parametric diagram, follow these steps:

1. Click the **Parametric diagram** icon {f} above the window.
2. The system displays a dialog box allowing you to select the `Packages` category and `Subsystems` and `MobilityManagement` for the diagram. Click **New**.
3. Name the new parametric diagram `Update location`, as shown here.

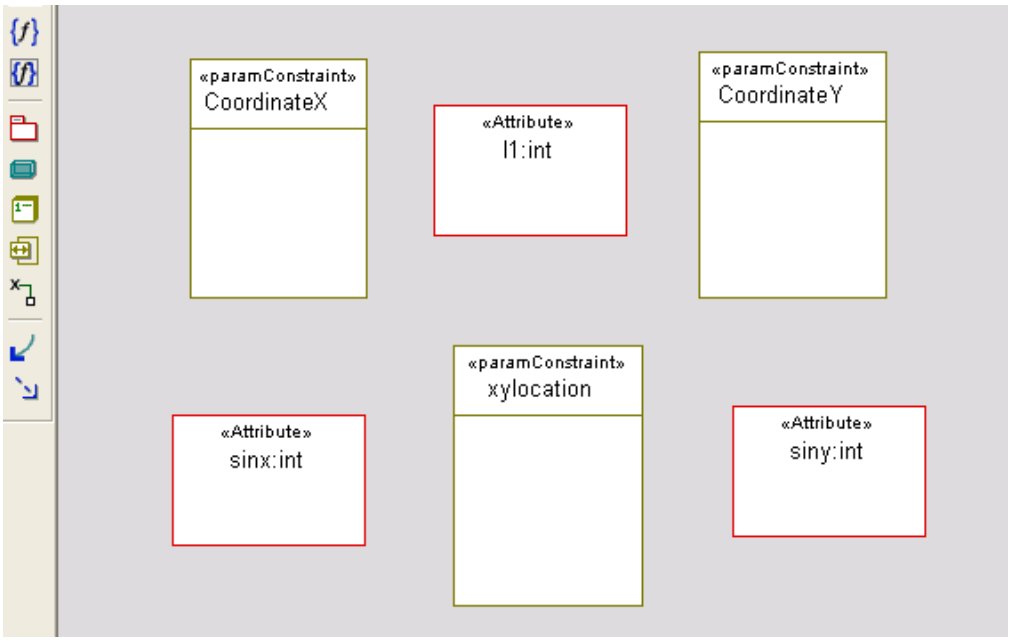


4. Click **OK**.
5. To draw the three parametric constraints, click the **Parametric constraints** icon {f} on the Drawing toolbar. (It is the same symbol used to create the diagram, but it is on the Drawing Toolbar in the center of the window.) Draw the required three constraints to calculate the location on a grid:
   - `CoordinateX`
   - `CoordinateY`
   - `xylocation`

Linking the Diagram to the Model

Now the parametric diagram needs to be linked to the model by following these steps:

1. In the browser, click `Parts`.
2. Right-click `Location` and select **Add New** > **Attribute**.
3. Type these three new attributes under the `Location` part in the browser:
   - `l1`
   - `sinx`
   - `siny`
4. Drag these three data attributes from the browser onto the parametric diagram. At this point your diagram should resemble this example.



Creating Flow Ports and Flows

To show how the data is bound to each constraint, you need to add flow ports and flows to the parametric diagram.

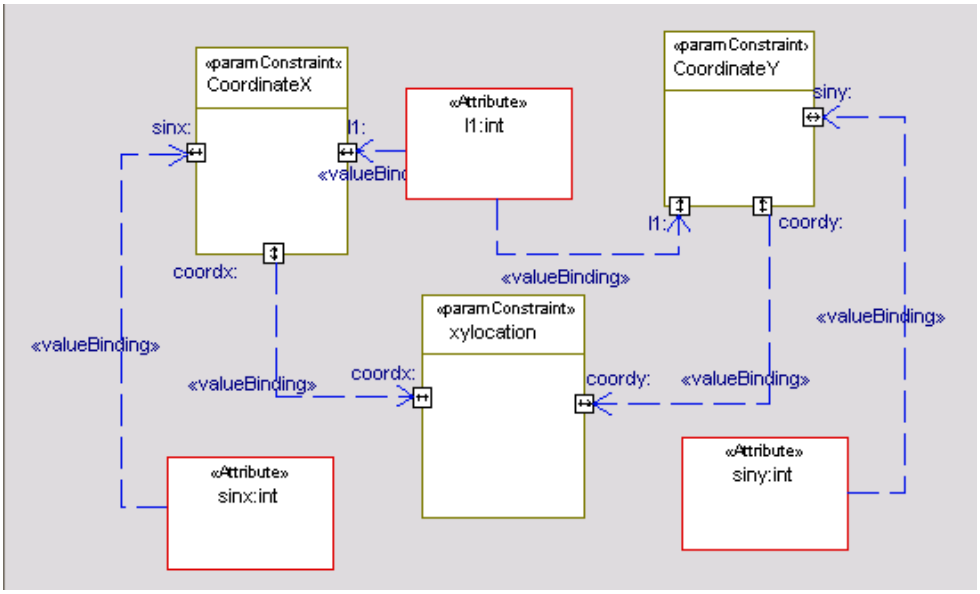
Note

Flow ports are not supported in Rhapsody in J.

Follow these steps:

1. Click the **Flow Port** icon  on the Drawing toolbar. Click on the outside edges of the constraints to create these ports:
   - **CoordinateX** - add `sinx` and `l1` ports
   - **CoordinateY** - add `siny` and `l1` ports
   - **xylocation** - add `coordx` and `coordy` ports
2. Click the **Value Binding** icon  to create the data flow as follows:
   - From `sinx` attribute to the `sinx` port on `CoordinateX` and name this connection `valueBinding`.
   - From `l1` attribute to `l1` port on `CoordinateX` and name this connection `valueBinding`.
   - From `l1` attribute to `l1` port on `CoordinateY` and name this connection `valueBinding`.
   - From `siny` attribute to `siny` port on `CoordinateY` and name this connection `valueBinding`.
3. The output from the parametric constraints is used as the input for the calculations. Select the **Flow Port** icon again to create the output flow ports from `CoordinateX` and `CoordinateY`. Name them `coordx` and `coordy` respectively.
4. Click the **Value Binding** icon and draw `valueBinding` flows from the new flow ports on the CoordinateX and CoordinateY constraints to the corresponding ports on xy`location`.

At this point your parametric diagram should be similar to this example.

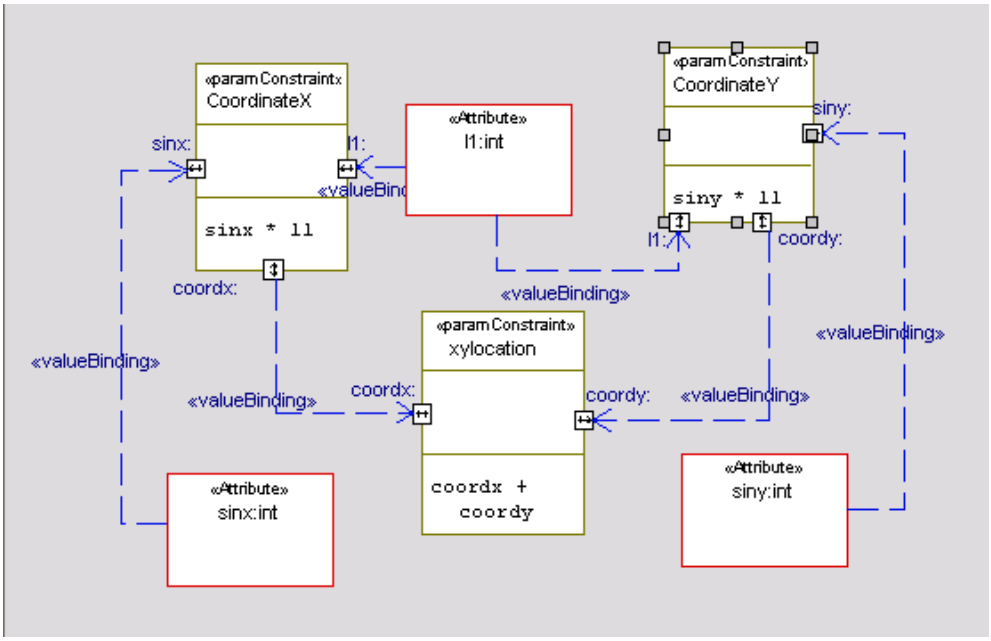


Adding Equations

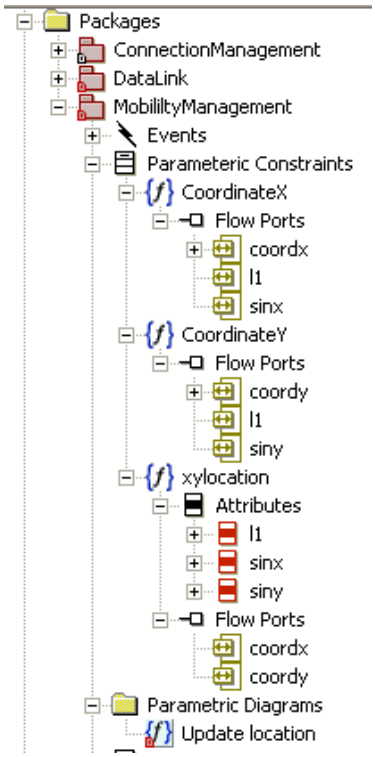
To add the required equations to all of the constraints, follow these steps:

1. Right-click on the `CoordinateX` constraint to display the **Features** dialog.
2. Type `sinx * l1` in the **Description** area and click **Apply** to keep the dialog box open.
3. Right-click on the `CoordinateY` constraint and type `siny * l1` in the **Description** area. Click **Apply** to keep the dialog box open.
4. Right-click the xy`location` constraint and type `coordx + coordy` and click **OK**.
5. To display the equation in the xy`location` constraint, right-click xy`location` and select **Display Options** from the menu.Click the **Compartments** button.
6. Select **Description** form the Available list and click the **Display** button to move it to the Displayed column. Click **OK** to save this change. You may wish to perform this action on each of the boxes containing an equation.

7. Click **OK** to save the Display Options dialog. You have completed the parametric diagram. It should resemble this example.



8. Expand the Parametric items in the browser. Your list in the browser should resemble this example.



System Behaviors

Sequence diagrams describe how structural elements communicate with one another over time and identify the required relationships and messages. Sequence diagrams also show the interactions between actors, use cases, and blocks.

Sequence diagrams have an executable aspect and are a key simulation tool. When you simulate a model, Rhapsody dynamically builds sequence diagrams that record interaction between actors, use cases, and blocks.

Sequence Diagrams Describing Scenarios

Sequence diagrams shows how subsystems interact during a scenario. For example, a sequence diagram showing a successful request to place a call identifies the order and exchange of messages between the parts and blocks as represented in the Internal Block Diagrams. By describing the flows through *scenarios*, you create the logical interfaces of the blocks. For example, if a message is shown going into the `DataLinkBlock`, you can see that the message belongs to the block as an event or operation.

In this lesson, 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 chapter 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 block definition diagram, before defining the communication scenarios of the functions.

Creating a Sequence Diagram

To create a new sequence diagram, follow these steps:

1. Start Rhapsody if it is not already running and open the handset model if it is not already open.
2. In the browser, right-click the `Subsystems` package and select **Add New** >**Sequence Diagram** from the pop-up menu.
3. Type `Place Call Request Successful` into the **Name** field of the dialog box.
4. Click the **Analysis** radio button for the **Operation Mode**.

   Rhapsody enables you to create sequence diagrams in two modes:

   a. In *analysis mode*, you draw message sequences without adding elements to the model. This enables you to brainstorm your analysis and design without affecting the simulated source code. Once the design is finalized, you can realize the instance lines and messages so that they display in the browser, and can have code simulated for them.

   b. In *design mode*, every instance line and message you create or rename can be realized as an element (class, part, operation, or event) that appears in the browser, and for which code can be simulated. When you draw a message, Rhapsody will ask if you want to realize it. Click **Yes** to realize the message.
5. Click **OK** to save your changes and close the dialog box.

Note

You can also create a sequence diagram using the Tools menu or the Sequence Diagram icon at the top of the Rhapsody window.

Adding the 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.

To draw the diagram and actor lines, follow these steps:

1. In the browser, expand the `Analysis` and then the `Actors` group.
2. Click the `MMI` actor and drag-and-drop it to the far left side of the sequence diagram. Rhapsody creates the actor line as an environment boundary.

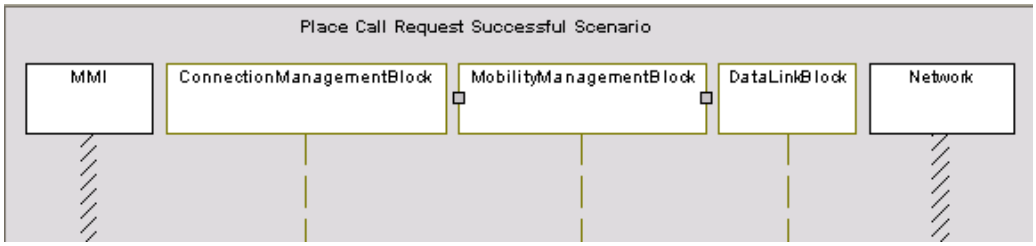
3. Click the `Network` actor and drag-and-drop it to the far right side of the sequence diagram.

Drawing Classifier Roles

Classifier roles or instance lines are vertical timelines labeled with the name of an instance to 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 example, you draw the classifier roles that represent the system components, `ConnectionManagement`, `MobilityManagement`, and `DataLink`, by dragging them from the browser to the diagram as follows:

1. In the browser, expand the `Subsystems` package and then the `ConnectionManagementBlock` subsystem and `Block` group.
2. Click `ConnectionManagementBlock` and drag-and-drop it next to `MMI`. Rhapsody creates the classifier role with the name of the function in the name pane.
3. In the browser, expand the `MobilityManagement` subsystem and the `Block` group. Click `MobilityManagementBlock` and drag-and-drop it next to `ConnectionManagementBlock`.
4. In the browser, expand the `DataLink` subsystem and the `Block` group. Click the `DataLinkBlock` and drag-and-drop it next to the `MobilityManagementBlock`.
5. Right-click the actors and blocks at the top of each line and select **Display Options** from the menu. Change all of them to show the **Label** and not the Name. The default is to show the Name. At this point your sequence diagram should resemble this example.



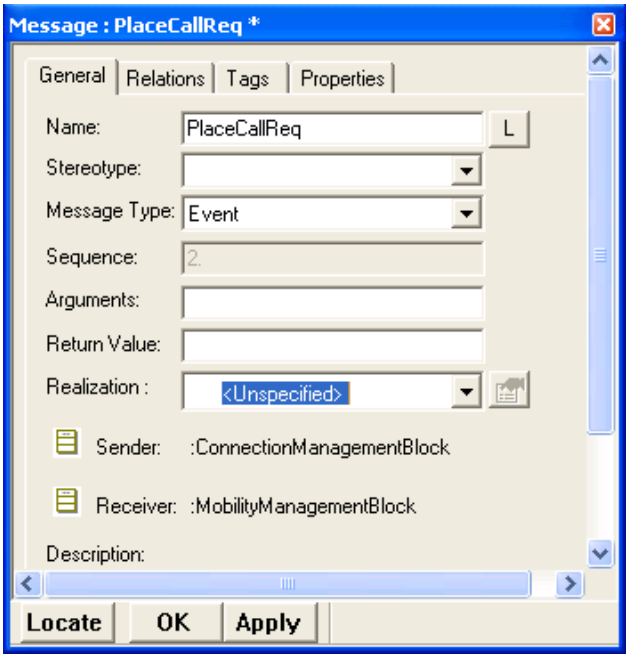
6. To add white space to (or remove it from) a sequence diagram (such as between actor lines and classifier roles), press the Shift key and drag the actor line or classifier role to its new location.

Drawing Messages

A *message* represents an interaction between parts, or between a part and the environment. A message can be an *event*, a *triggered operation*, or a *primitive operation*. In this example, you draw events that represent the exchange of information when placing a call. The actor issues a request to connect when placing a call. Call and connect confirmations occur between `MobilityManagementBlock` and `ConnectionManagementBlock`. Alerts occur between `MobilityManagementBlock` and `DataLinkBlock`. The user receives confirmation from `ConnectionManagement`.

To draw messages, follow these steps:

1. Click the **Message** icon  on the Drawing toolbar.
2. Click the `MMI` actor line to show that the first message comes from the `MMI` actor when the user issues the command to place a call request.
3. Click the `ConnectionManagementBlock` line to create a straight line. Rhapsody creates a message with the default name `message_n()`, where *n* is an incremental integer starting with 0.
4. Double-click the message line and in the dialog box, enter the **Name** as `PlaceCallReq`. The **Message Type** should be `Event` with `<Unspecified>` as the **Realization**. Your changes should resemble this example.



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

6. Draw the following messages with **Message Type** of `Event` and `<Unspecified>` as the **Realization**:

   a. From `ConnectionManagementBlock` to `MobilityManagementBlock`, named `PlaceCallReq`

   b. From `MobilityManagementBlock` to `ConnectionManagementBlock`, named `CallConfirm`

   c. From `MobilityManagementBlock` to `DataLinkBlock`, named `Alert`

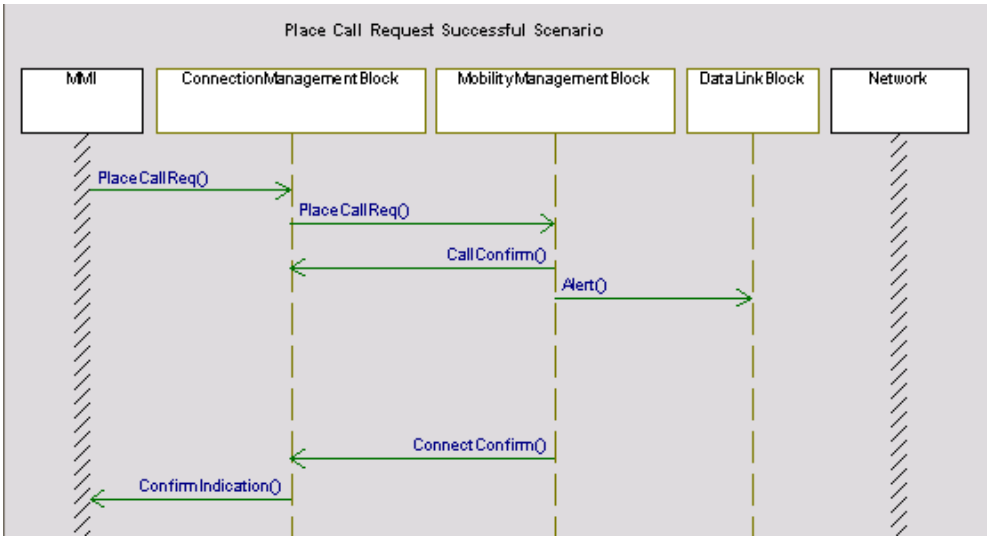
7. Leave a space on the lines for the interaction occurrence (reference sequence diagram) that you create in the next section.

8. Then draw the remaining messages also with **Message Type** of `Event` and `<Unspecified>` as the **Realization**:

   a. From `MobilityManagementBlock` to `ConnectionManagementBlock`, named `ConnectConfirm`

   b. From `ConnectionManagementBlock` to the `MMI` actor, named `ConfirmIndication`

At this point your diagram should resemble this example.



Drawing an Interaction Occurrence

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

To draw an interaction occurrence, follow these steps:

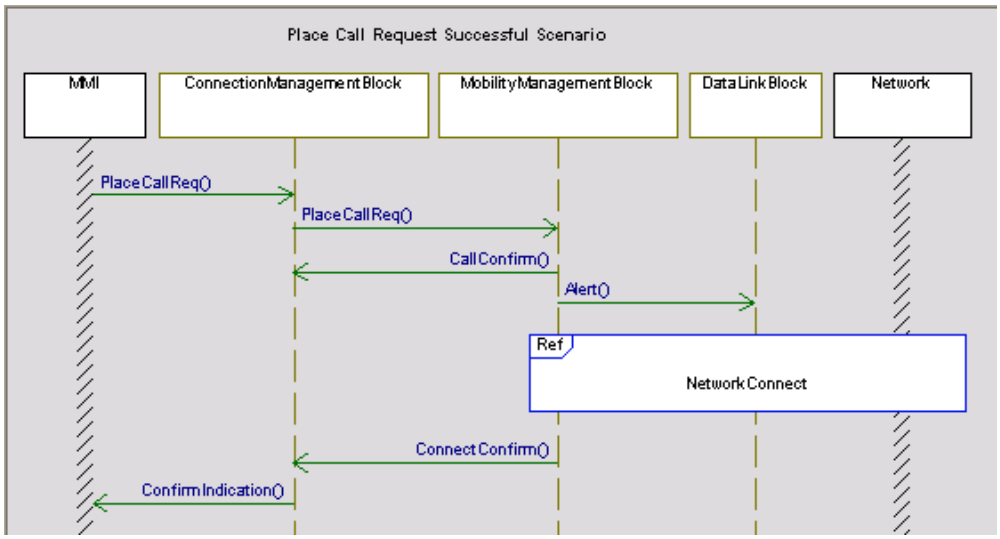
1. Click the **Interaction Occurrence** icon on the Drawing toolbar.
2. Draw the interaction occurrence below the `Alert` message and across the `MobilityManagementBlock` instance line to the `Network` actor line. The interaction occurrence appears as a box with the `ref` label in the top corner.
3. Type `NetworkConnect` as the name of the Interaction Occurrence.

You have completed drawing the Place Call Request Successful sequence diagram.

Note

Be sure to check the arrow heads on the messages. They must all be open to be events.

At this point you diagram should resemble this example.



Diagramming the Network Connection Scenario

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

Creating the NetworkConnect Sequence Diagram

To create the NetworkConnect sequence diagram, follow these steps:

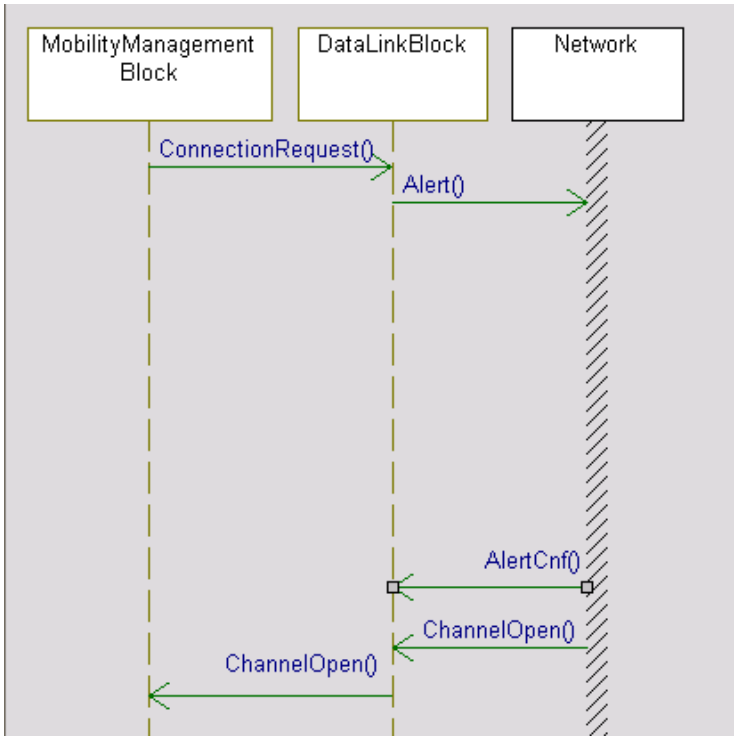
1. In the Place Call Request Successful sequence diagram, right-click the interaction occurrence (NetworkConnect) and select **Create Reference Sequence Diagram** from the pop-up menu. Rhapsody opens the new diagram in the drawing area containing the three functions that the interaction occurrence crosses and adds the sequence to the browser.
2. Select the items in the sequence diagram and right-click to select the **Display Options** from the menu.
3. Change the display to show the **Label** and not the Name of each item.

Drawing Messages

In this example, draw these events:

1. Click the **Message** icon  in the Drawing toolbar.
2. Draw the following messages (with the arrows horizontal) with **Message Type** of `Event` and `<Unspecified>` as the **Realization**:
   a. From `MobilityManagementBlock` to `DataLinkBlock`, named `ConnectionRequest`
   b. From `DataLinkBlock` to `Network`, named `Alert`
   c. From `Network` to `DataLinkBlock`, named `AlertCnf`
   d. From `Network` to `DataLinkBlock`, named `ChannelOpen`
   e. From `DataLinkBlock` to `MobilityManagementBlock`, named `ChannelOpen`

At this point your diagram should resemble this example.



Drawing Interaction Operators

To draw the `alt` interactive operation in the sequence diagram, follow these steps:

1. Click the **Interaction Operator** icon and draw a large box across all three lines and over the last three messages.
2. Right-click on the Interactive Operator box and select **Features** from the menu.
3. In the **General** tab, change the **Type** to be `alt`. Click **OK**. This change appears in the upper left corner of the box.
4. Click the **Interactive Operand Separator** icon and click on the interaction operator box to divide it into two sections.
5. Edit the two labels to read `[alert=false]` and `[else]`.
6. Click the **Interaction Operator** icon again and draw another box within the first box, in the `[alert=false]` section, and across all three lines.

7. Right-click this box to open the **Features** dialog and change the **Type** to `loop`. Click **OK**. This change appears in the upper left corner of the box.

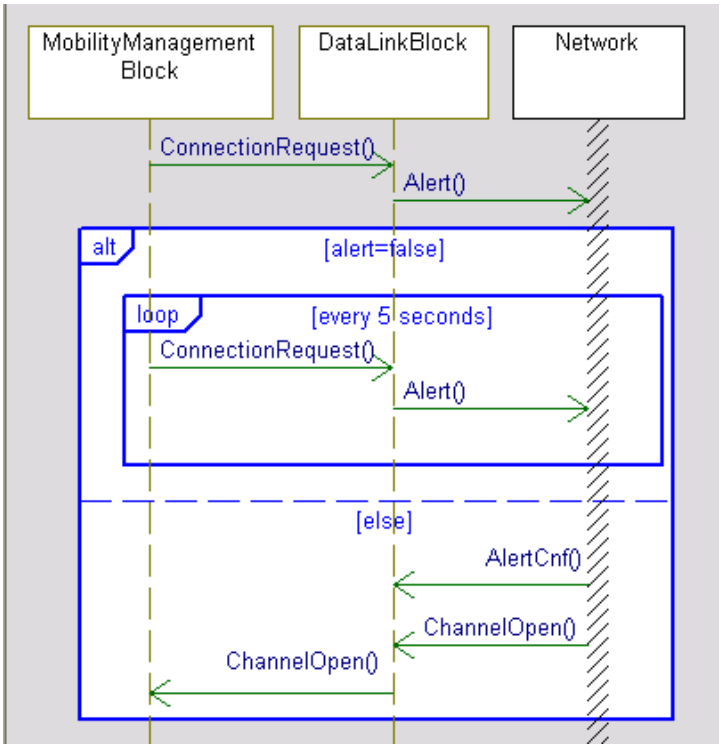
8. Edit the label in the new box name to be `[every 5 seconds]`.

9. Inside the loop box, draw these two messages (horizontal arrows) with **Message Type** of `Event` and `<Unspecified>` as the **Realization**:

   a. From `MobilityManagementBlock` to `DataLinkBlock`, named `ConnectionRequest`

   b. From `DataLinkBlock` to `Network`, named `Alert.`

At this point your diagram should resemble this example.



Creating the Connection Management Sequence Diagram

The ConnectionManagement Place Call Request Success sequence diagram shows the interaction of the functions. It identifies the part decomposition interaction when placing a successful call.

To create a new sequence diagram, follow these steps:

1. In the browser, right-click the `Subsystems` package and select **Add New** >**Sequence Diagram** from the pop-up menu.
2. Type `ConnectionManagement Place Call Request Success` into the **Name** field of the dialog box.
3. Select `Analysis` for the **Operation Mode**.
4. Click **OK** to close the dialog box.

Rhapsody creates the `Sequence Diagrams` category, adds the name of the new sequence diagram, and opens the new diagram in the drawing area. You may wish to type a title on the diagram at this point.

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 sequence diagram.

To draw the system border, follow these steps:

1. Click the **System border** icon  in the Drawing toolbar.
2. Click on the left side of the diagram to place the environment border.

Drawing Classifier Roles

In this example, you draw the classifier roles that represent the system components, `ConnectionManagementBlock`, `MobilityManagementBlock`, and `DataLinkBlock` by dragging them from the browser to the diagram as follows:

1. In the browser, expand the `ConnectionManagement` subsystem and then the `ConnectionManagementBlock` and `Parts` group.
2. Click `CallControl` and drag-and-drop it next to the system border. Rhapsody creates the classifier role with the name of the function in the names pane.
3. Click `CallList` and drag-and-drop it next to `CallControl`.
4. Click `Connection` and drag-and-drop it next to `CallList`.
5. In the browser, expand the `MobilityManagementBlock` and the `Parts` category.
6. Click `MMCallControl` and drag-and-drop it next to `Connection`.
7. In the browser, expand the `DataLinkBlock` and the `Parts` category.
8. Click `RegistrationMonitor` and drag-and-drop it next to `MMCallControl`.

Note

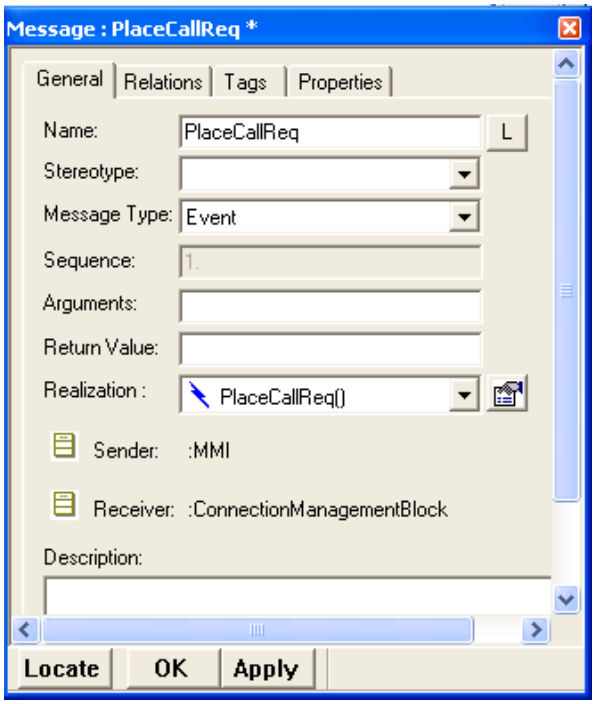
Names that are too long to fit in the pane continue past the divider and behind the lower pane. To view these names, enlarge the size of the pane or change the font or font size.

Drawing Messages

In this scenario when the system receives a request to place a call, it validates and registers the user. Once it is registered, it monitors the user's location. The call and connection are confirmed, the connection is set up, and confirmation is provided.

To draw the event messages, follow these steps:

1. Click the **Message** icon  in the Drawing toolbar.
2. Draw the following events using horizontal lines:
   a. From the system border to the `CallControl` line, named `PlaceCallReq`
   b. From `CallControl` to `MMCallControl`, named `PlaceCallReq`
   c. From `MMCallControl` to `RegistrationMonitor`, named `RegistrationReq`
   d. From `RegistrationMonitor` to `MMCallControl`, named `ChannelOpen`
3. For each of the four messages, double-click the message line and set the **Message Type** as `Event` and select **<New>** from the **Realization** pull-down menu. Then the system generates a realization name and inserts it in the Realization field, as shown in this example.



4. Draw a message-to-self on the `MMCallControl` instance line, named `locationUpdate`. **(Message names are case-sensitive.**)

5. Double-click the `locationUpdate` message-to-self line and set the **Message Type** as `PrimitiveOperation` and select **<New>** from the **Realization** pull-down menu. The system adds `locationUpdate()` as the Realization name.

6. Draw the following events using horizontal lines and double-click the message lines and set the **Message Type** as `Event` and select **<New>** from the **Realization** pull-down menu.

   a. From `MMCallControl` to `CallControl`, named `CallConfirm`

   b. From `MMCallControl` to `CallControl`, named `ConnectConfirm`

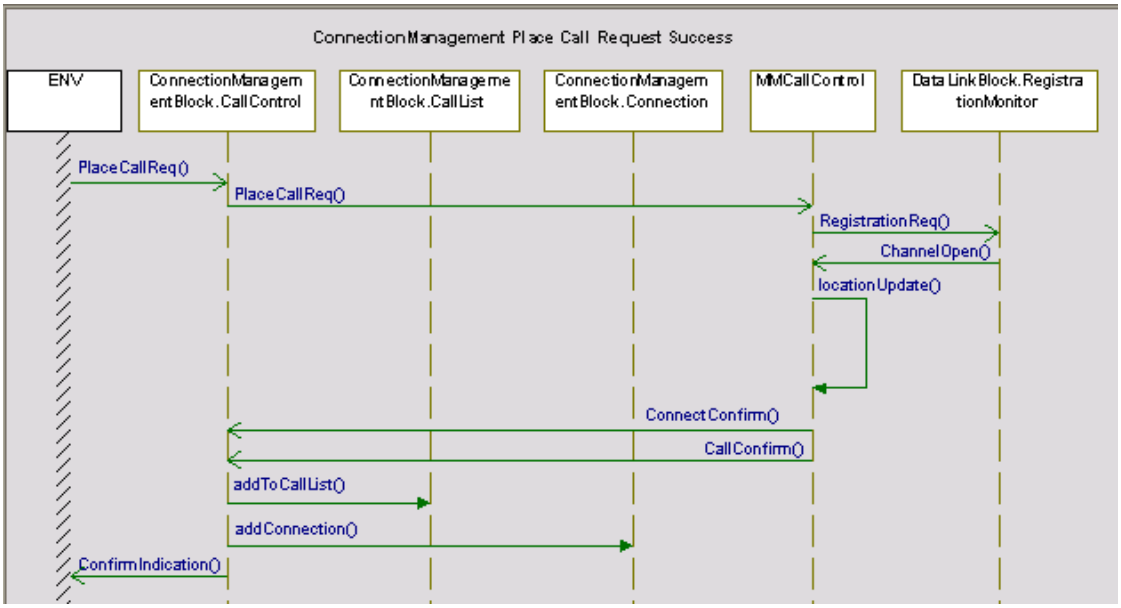
7. Draw the following horizontal message lines and double-click each message line to set the **Message Type** as `PrimitiveOperation` and select **<New>** from the **Realization** pull-down menu.

   a. From `CallControl` to `CallList`, named `addToCallList`

   b. From `CallControl` to `Connection`, named `addConnection`

8. Draw an event from `CallControl` to the system border, named `ConfirmIndication` and double-click the message line and check to be certain the **Message Type** is `Event`.

At this point, your diagram should resemble this example.



Implementation Using an Action Language

In order to show actions in a model, the designer needs an implementation language. Rhapsody includes an Action Language, a subset of C++ that uses a C++ compiler to allow you to simulate the model. This language provides the following:

- Message passing
- Data checking
- Actions on transitions
- General model execution

Examples of this language appear in several of the following diagrams as the Action implementations.

Basic Syntax Rules

This streamlined version of C++ has these basic syntax rules:

- It is case-sensitive, so “evGo” is different from “evgo.”
- Names must follow these rules:
  - No Spaces (“Start motor” is not correct.)
  - No Special Characters (“StartMotor@3” is not correct.)
  - Must start with a Letter (“2ToBegin” is not correct.)
- All statements must end in a semicolon
- Do not to use reserved words such as *id*, *for, next.*

Frequently Used Statements

To add some simple operations to your model, you may use the following:

- These increment/decrement operators provide standard functions:
  - X++; (Increment X)
  - X--; (Decrement X)
  - X=X+5; (Add 5 to X)
- To print out on the screen, use one of these:
  - cout << “hello” << endl;
  - cout << attribute_name << endl;
  - cout << “hello : “ << attribute_name << endl;

Reserved Words

The Action Language reserved words are listed below. All reserved words for built-in functions are lower case, for example, *if*.

asm	continue	float	int	params	sizeof	typedef
auto	default	for	IS_IN	private	static	union
break	delete	friend	IS_PORT	protected	struct	unsigned
case	do	GEN	long	public	switch	virtual
catch	double	goto	new	register	template	void
char	else	id	operator	return	this	volatile
class	enum	if	OPORT	short	throw	while.
const	extern	inline	OUT_PORT	signed	try	

Defining Flow of Control in 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 addition, you can animate activity diagrams to verify the functional flow.

In this lesson, 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.

Creating an Activity Diagram

To create an activity diagram, follow these steps:

1. Start Rhapsody if it is not already running and open the handset model if it is not already open.
2. In the browser, expand the `Subsystems` package, the `MobilityManagement` package, the `MobilityManagementBlock`, and the `Parts` category. Right-click `MMCallControl` and select **Add New** > **Activity Diagram** from the pop-up menu or click the **Activity Diagram** icon at the top of the window.
   The blank diagram opens in the drawing area. You may want to add a title to the diagram.
3. Right-click the new diagram and select **Diagram Properties** from the context menu.
4. On the General tab of the features dialog, clear the Analysis Only checkbox.
5. Click OK to close the features dialog.

Defining the MMCallControl Functional Flow

The MMCallControl activity diagram shows the functional flow that supports the mobility of users when placing a call to include the following:

- Registering users on the network
- Providing their current location

- Obtaining an acceptable signal strength

When the user places a call, the system leaves the `Idle` state 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 state.

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, follow these steps:

1. Click the **Swimlanes Frame** icon  on the Drawing toolbar.
2. Click to place one corner, then drag diagonally to draw the swimlane frame.
3. Click the **Swimlanes Divider** icon  on the Drawing toolbar.
4. Click the middle of the swimlane frame. Rhapsody creates two swimlanes, named `swimlane_n` and `swimlane_n+1`, where *n* is an incremental integer starting at 0.
5. Name the swimlane on the left `Location`. This swimlane tracks the location of users.

   Note: If you drag the swimlane left or right, it also resizes the swimlane frame. Once you have repositioned the swimlane divider, resize the columns to make them wide enough for the drawings.
6. Name the swimlane on the right `SignalStrength`. This swimlane tracks the signal strength of users.

Setting Activity Diagram Properties

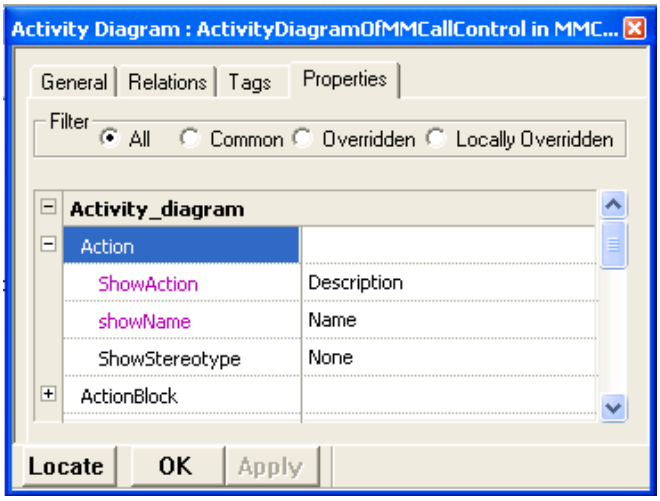
Action states represent function invocations with a single exit transition when the function completes. In this example, you will draw the action states that represent the functional processes, and then add names to the action states.

The default settings are used when you add an Action and type a name in the action state on the diagram. That name becomes the action text, not the name of the action. Before adding actions, set the *properties* for the diagram, following these steps:

1. Right-click outside the Swimlanes frame and select **Diagram Properties**.
2. Select the **Properties** tab and click the **All** radio button for the Filter.
3. Open the **Action** category and change the **showName** and **ShowAction** properties to use these values:

```
Activity_diagram::Action::showName = Name
Activity_diagram :: Action :: ShowAction = Description
```

This second property allows informal text to be displayed on the diagram, while the actual action is described formally using an *executable language*. Your dialog box should be similar to this example.



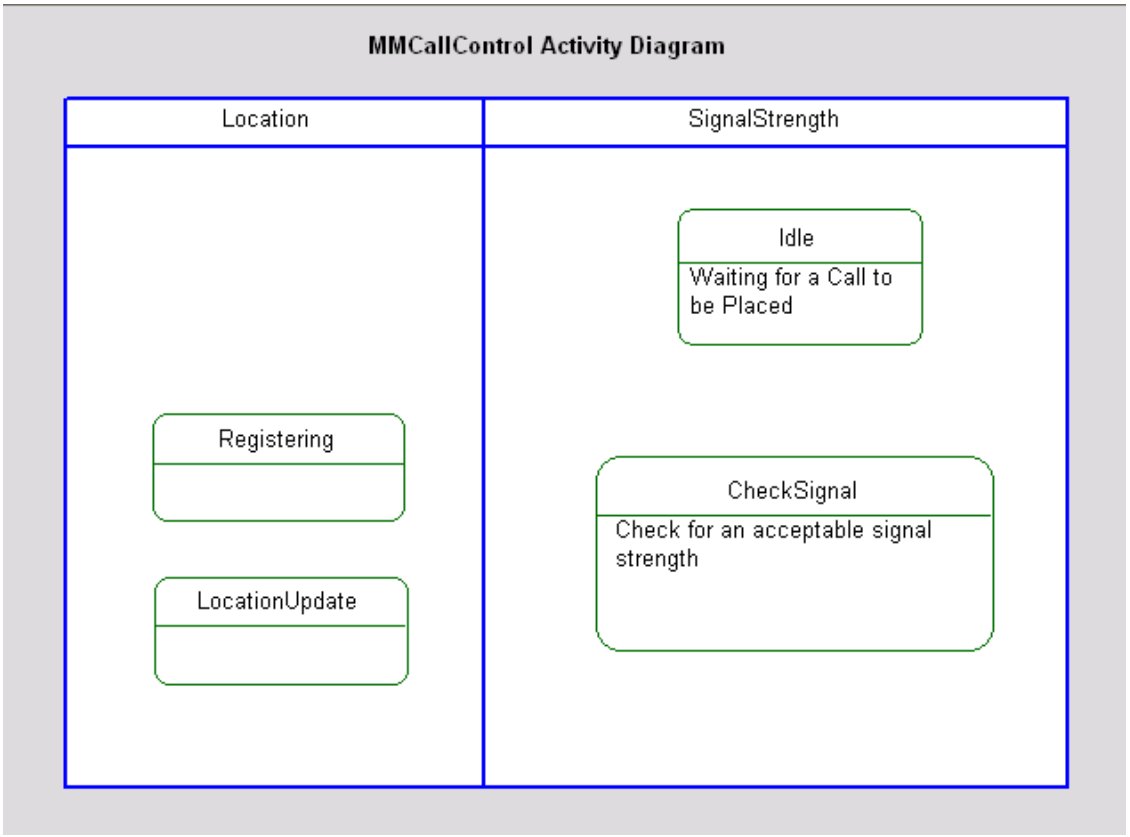
4. Click **OK** to save the changes and close the dialog box.

Drawing Action States

To draw action states in the diagram, follow these steps:

1. Click the **Action** icon on the Drawing toolbar.
2. In the top section of the drawing area inside the upper `SignalStrength` swimlane frame, click or click-and-drag to create an action state. Name the new action `Idle`.
3. Click the action state, or right-click and select **Features** from the pop-up menu.
4. In the description field, type `Waiting for a Call to be Placed`.
5. Click **OK** to apply the changes and close the **Features** dialog box.
6. In the lower section of the `Location` swimlane, draw an action state and name it `LocationUpdate`.
7. In the `SignalStrength` swimlane draw an action state and name it `CheckSignal`.
8. Double-click the `CheckSignal` action state to display the **Features** dialog box.
9. Type `Check for an acceptable signal strength` in the **Description** field. Then click **OK**.

10. Click the **Action** icon.
11. Click or click-and-drag above the `LocationUpdate` action in the `Location` swimlane.
12. Name the new action `Registering`. At this point, your diagram should resemble this example.



Defining an Action using an Action Language

To define action states, Rhapsody provides an action language that is a subset of C++. For more information, refer to the **Implementation Using an Action Language section**.

To define an action, follow these steps:

1. Double-click the `Registering` action state, or right-click and select **Features** from the pop-up menu.
2. Type the following action language in the **Action** field:

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

This command sends an asynchronous message out the `mm_cc` port for the registration requests.

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

Drawing a Default Connector

One of the Action States must be the *default* state. This is the initial state of the Activity. `Idle` is the default state as it waits for call requests.

To identify the default state, follow these steps:

1. Click the **Default Flow** icon  on the Drawing toolbar.
2. Click to the right of the `Idle` action state and then click its edge. Press **Ctrl**+**Enter** stop drawing the connector and not label it.

Drawing a Subactivity State

A *subactivity state* represents the execution of a non-atomic sequence of steps nested within another activity. In this example, draw the `InCall` subactivity state to indicate that the call has been established using these steps:

1. Click the **Subactivity** icon  on the Drawing toolbar.
2. In the bottom section of the `SignalStrength` swimlane, click or click-and-drag to draw the subactivity state.
3. Name the subactivity state `InCall`.
4. To display the subactivity icon in the lower right corner of the state drawing, right-click the `InCall` box and select **Display Options** from the menu.
5. Click the **Icon** radio button for the **Show Stereotype** selections and click **OK**.

Drawing Transitions

Transitions represent the response to a message in a given state. They show what the next state will be. In this example, you will 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, select **Line Shape** from the pop-up menu, and then **Straight**, **Spline**, **Rectilinear**, or **Reroute**.

Drawing Transitions Between States

To draw transitions between states, follow these steps:

1. Click the **Activity Flow** icon  on the Drawing toolbar.
2. Click the `InCall` subactivity state, then click the `Idle` state.
3. Name the transition `Disconnect`.
4. Draw a transition from `Registering` to `LocationUpdate`, then press **Ctrl**+**Enter**.

   Note: Rhapsody enables you to assign a descriptive label to an element. A labeled element does not have any meaning in terms of an executable action, but the label helps you to reference and locate elements in diagrams and dialog boxes. A label can have any value and does not need to be unique.

   In this example, label the transition between `Registering` and `LocationUpdate` as follows:

5. Double-click the transition between `Registering` and `LocationUpdate` or right-click and select **Features** from the pop-up menu. The Features dialog box opens.
6. Click the **L** button next to the **Name** field. The Name and Label dialog box opens.
7. Type `Registering` in the **Label** field.
8. Click **OK** to close the Name and Label dialog box.
9. Click **OK** to close the Features dialog box.
10. To display the label, right-click the transition and select **Display Options** > **Display Name** > **Label** from the pop-up menu.

Note

When drawing activity flows, it is a good practice to not cross the flow lines. This makes the diagram easier to read.

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, follow these steps:

1. Click the **Draw Fork Synch Bar** icon  on the Drawing toolbar.

2. Click or click-and-drag between the `Idle` action state and the `CheckSignal` action state. Rhapsody adds the fork synchronization bar.

3. Click the **Activity Flow** icon, and draw a single incoming transition from `Idle` to the synchronization bar. Type `PlaceCallReq`, then press **Ctrl**+**Enter**. This transition indicates that a call request has been initiated.

4. Draw the following outgoing transitions from the fork bar:

   a. To the `Registering` action, then press **Ctrl**+**Enter**

   b. To the `CheckSignal` action state, 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, follow these steps:

1. Click the **Draw Join Synch Bar** icon  on the Drawing toolbar.
2. Click or click-and-drag between `CheckSignal` action state and the `InCall` subactivity. This line remains within the `SignalStrength` swimlane. Rhapsody adds the join synchronization bar.
3. Click the **Activity Flow** icon and draw the following incoming transitions to the synchronization bar:
   a. From `LocationUpdate`, then press **Ctrl**+**Enter**
   b. From `CheckSignal`, then press **Ctrl**+**Enter**
4. Draw one outgoing transition from the synchronization bar to `InCall`. 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 for CheckSignal

A *timeout transition* causes a transition to be taken after a specified amount of time has passed. It is an event with the form `tm(n)`, where *n* is the number of milliseconds that should pass before the transition is made.

In this example, you will draw a timeout transition that monitors the signal strength of transmissions every three seconds as follows:

1. Click the **Activity Flow** icon  on the toolbar.
2. Draw a transition originating and ending with `CheckSignal`.
3. Type `tm(3000)`and then press **Ctrl**+**Enter**.

Specifying an Action on the Disconnect Transition

To specify actions for `Disconnect` and `ChannelOpen`, follow these steps:

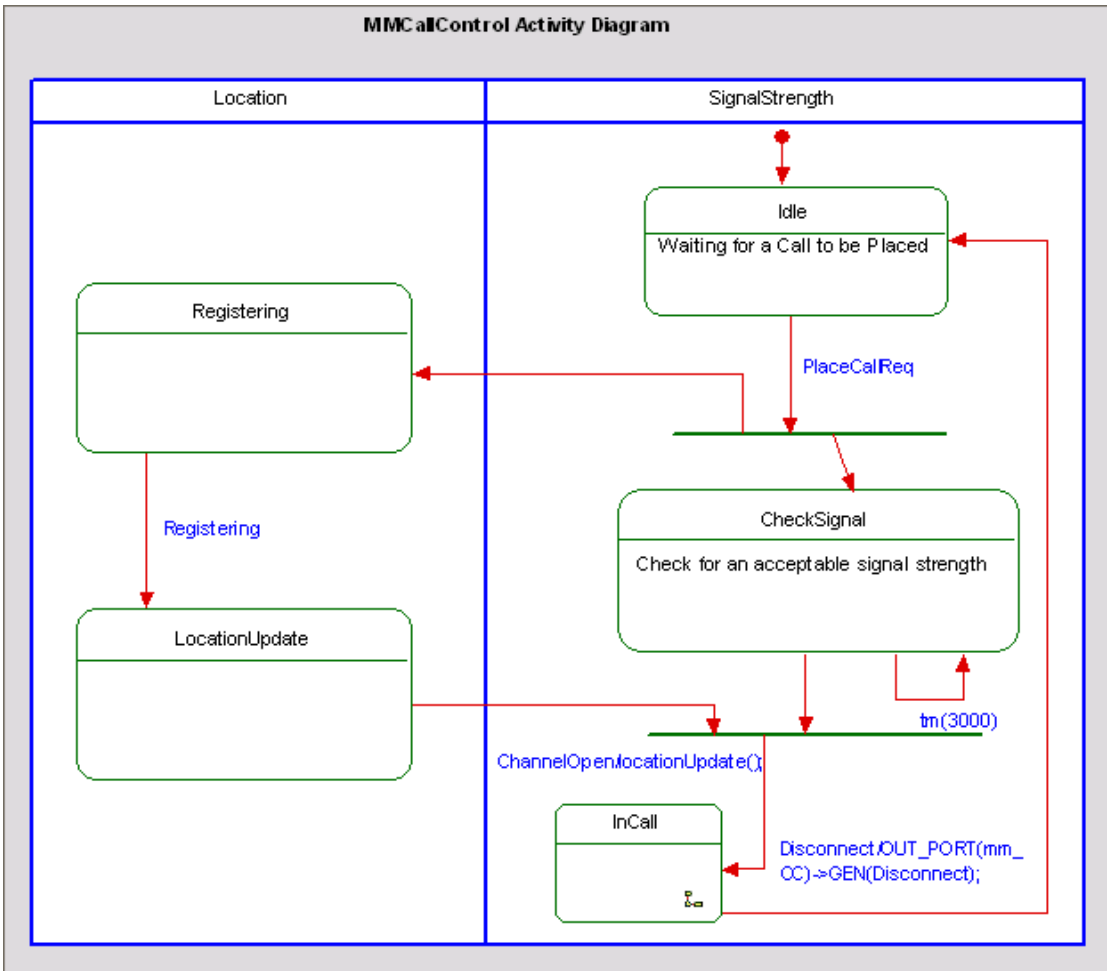
1. Double-click the `Disconnect` transition, or right-click and select **Features** from the pop-up menu. The Features dialog box opens.
2. In the **Action** field, type the following action language code:

   ```
   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 close the **Features** dialog box. Rhapsody displays the transition name with the action command.
4. Double-click the `ChannelOpen` transition, or right-click and select **Features** from the pop-up menu. The Features dialog box opens.
5. In the **Action** field, type the following code:

   ```
   locationUpdate();
   ```

6. Click **OK** to apply the changes and close the **Features** dialog box. Rhapsody displays the transition name with the action command. At this point your diagram should resemble this example.



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 select **Display Options** > **Show Label** from the pop-up menu.

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.

To open the `InCall` subactivity diagram, right-click `InCall` in the MMCallControl activity diagram, and select **Open Sub Activity Diagram** from the pop-up menu. Rhapsody displays the subactivity diagram with the `InCall` activity in the drawing area. This diagram has the same properties as the original diagram.

Drawing Action States

In this example, you will draw the following two actions states, and then add names to the action states:

- `VoiceData`—Processes voice data
- `CheckSignal`—Checks the signal strength on the network

To draw the action states, follow these steps:

1. Click the **Action** icon  on the Drawing toolbar.
2. In the top section of the `InCall` state, click-and-drag or click and name it `VoiceData`. Press **Ctrl**+**Enter**.
3. In the bottom section of the `InCall` state, click-and-drag or click and name it `CheckSignal`. Then press **Ctrl**+**Enter**.

Drawing a Default Connector to VoiceData

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

To draw the default connector, follow these steps:

1. Click the **Default Flow** icon  on the Drawing toolbar.
2. Click above `VoiceData`, then click `VoiceData`. Press **Ctrl**+**Enter**.

Drawing Flow Lines

Draw a flow line between `Voice Data` and `Check Signal` with these steps:

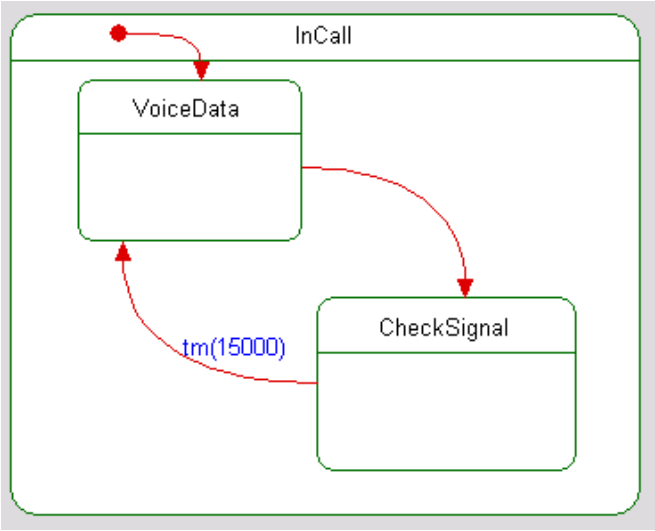
1. Click the **Activity Flow** icon  on the toolbar.
2. Draw a flow line from `VoiceData` to `CheckSignal`. Press **Ctrl**+**Enter**.

Drawing a Timeout Activity Flow

Draw a timeout transition to check for voice data every 15 seconds as follows:

1. Click the **Activity Flow** icon on the toolbar.
2. Draw a flow line from `CheckSignal` to `VoiceData`.
3. Type `tm(15000)`, then press **Ctrl**+**Enter**.

You have completed drawing the InCall subactivity diagram. Rhapsody automatically adds the newly created action states and flows to the browser. Your subactivity diagram should resemble this example.



Creating the RegistrationReq Activity Diagram

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

To create the RegistrationReq activity diagram, follow these steps:

1. In the browser, expand the `DataLink` package, the `DataLinkBlock` and the `Parts` category.
2. Right-click `RegistrationMonitor` and select **Add New** > **Activity Diagram** from the pop-up menu. (This diagram uses the default properties.)
   Rhapsody adds the Activity Diagram category and the new activity diagram to the `RegistrationReq` part in the browser, and opens the new activity diagram in the drawing area.

Note

You may wish to type a title on the activity diagram, such as **RegistrationReq Activity Diagram**.

3. Right-click the new diagram and select **Diagram Properties** from the context menu.
4. On the General tab of the features dialog, clear the Analysis Only checkbox.
5. Click OK to close the features dialog.

Drawing Action States

In this example, you will draw three actions states and then add names to the action states as follows:

1. Click the **Action** icon  on the Drawing toolbar.
2. In the upper section of the drawing window, create an action state, then press **Ctrl** + **Enter**.
3. Open the Features dialog box for this action state, and type `Idle` in the **Name** field. Click **OK**.
4. Create another action state below `Idle`, then press **Ctrl** + **Enter**.
5. Open the **Features** dialog box and type `InitiateRequest` in the **Name** field. Click **OK**.
6. Create another action state below `InitiateRequest`, then press **Ctrl** + **Enter**.
7. Open the Features dialog box and type `Success` in the **Name** field. Click **OK**.

8. For each of the three action states, right-click and select the Display Options. Select these three radio buttons in the three areas **Name, Label,** and **Action** (in that order).

Defining the InitiateRequest Action State

In this example, specify an action for the `InitiateRequest` action state as follows:

1. Double-click `InitiateRequest` in the browser or right-click and select **Features** from the pop-up menu.
2. In the Features dialog box type the following action language in the **Action** field:

   ```
   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 close the Features dialog box.

Drawing a Default Connector

In the activity diagram, draw a default connector using these steps:

1. Click the **Default Flow** icon  on the Drawing toolbar.
2. Click above `Idle` in the diagram to anchor the flow line
3. Click `Idle` to finish the line. Press **Ctrl**+**Enter**.

Drawing Flows

Draw flow lines between actions states as follows:

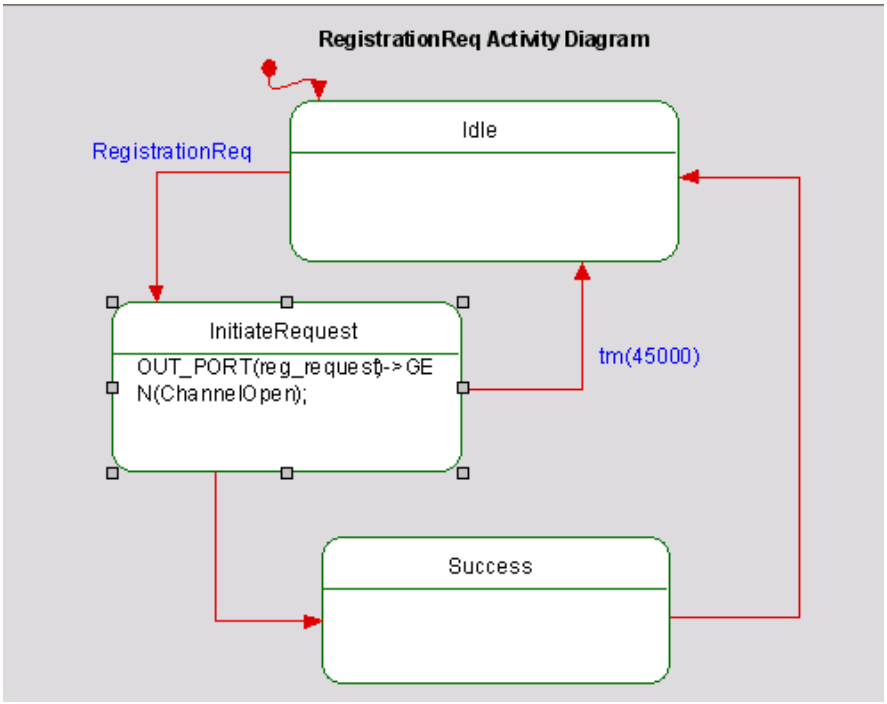
1. Click the **Activity Flow** icon  in the Drawing toolbar.
2. Draw a flow line from `Idle` to `InitiateRequest` and type `RegistrationReq` to label this flow. Press **Ctrl**+**Enter**.
3. Draw a flow from `InitiateRequest` to `Success`. Press **Ctrl**+**Enter**.
4. Draw a flow from `Success` to `Idle`. Press **Ctrl**+**Enter**.

Drawing a Timeout Flow

Draw a timeout flow to return to the `Idle` state after 45 seconds if no response is received from the network as follows:

1. Click the **Activity Flow** icon in the Drawing toolbar.
2. Draw a flow from `InitiateRequest` to `Idle`.
3. Type the flow label `tm(45000)`, then press **Ctrl**+**Enter**.

Your completed RegistrationReq activity diagram should resemble this example.



Modeling Behavior in Statecharts

Statecharts describe a system's behavior over time, specifically the behavior of classifiers (actors, use cases, or classes), parts, and blocks. This includes the states and modes of the system and the triggers that cause them to transition from state to state.

Statecharts constitute an extensive generalization of state-transition diagrams. They allow for multi-level states, decomposed in an and/or fashion, and thus support economical specification of concurrency and encapsulation. They incorporate the timeout operator for specifying synchronization and timing information, and a means for specifying transitions that depend on the history of the system's behavior. Statecharts are a key animation tool used to show dynamic behavior graphically.

Creating a Statechart

To create a statechart, follow these steps:

1. Start Rhapsody if it is not already running and open the handset model if it is not already open.
2. In the browser, expand the `Subsystems` package, the `ConnectionManagement` package, the `ConnectionManagementBlock`, and the `Parts` category.
3. Right-click `CallControl` and select **Add New** > **Statechart** from the pop-up menu.

Rhapsody adds the `Statechart` category and the new statechart to the CallControl part in the browser, and opens the new statechart in the drawing area.

Drawing States

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

Drawing Idle and Active States

In this example, draw two states, Idle and Active, using these steps:

1. Click the **State** icon  in the Drawing toolbar.
2. In the top section of the drawing area, click or click-and-drag. Rhapsody create a state with a default name of state_n, where *n* is equal to or greater than 0.
3. Type `Idle` for the name 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 named `Active`. This state indicates that the call is being set up or is in progress.

Drawing Nested States

In this example, 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 these nested states, follow these steps:

1. Click the **State** icon  in the Drawing toolbar.
2. In the top section of the Active state, draw a state named `ConnectionConfirm`.
3. In the bottom section of the Active state, draw a state named `Connected`.

Drawing Default Connectors

One of a part's state must be the default state, that is, its initial state when it is first activated. `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 these two default connectors, follow these steps:

1. Click the **Default connector** icon  in the Drawing toolbar.
2. Click to the right of the `Idle` state, then click `Idle`. Press Ctrl+Enter.
3. Draw another default connector to `ConnectionConfirm`. Press Ctrl+Enter.

Drawing Transitions

Transitions represent the response to a message in a given state. They show what the next state will be. A transition can have an optional trigger, guard, or action. This example uses transitions with triggers.

Creating a Trigger

To draw transitions with triggers, follow these steps:

1. Click the **Transition** icon  in the Drawing toolbar.
2. Click the `Idle` state and then click the `Active` state.
3. In the label box, type `PlaceCallReq`, then press **Ctrl**+**Enter**.
4. Create another transition from `ConnectionConfirm` to `Connected` and name this transition `ConnectConfirm`, then press Ctrl+Enter.
5. Create another transition from the `Active` state to the `Idle` state and name it `Disconnect`, the 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, select **Line Shape** from the pop-up menu, and then **Straight**, **Spline**, **Rectilinear**, or **Reroute**.

Specifying an Action on a Transition

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

In this example, specify an action for PlaceCallReq and `Disconnect` as follows:

1. Double-click the `PlaceCallReq` transition, or right-click and select **Features** from the pop-up menu. The Features dialog box opens.
2. In the **Action** field, type the following action language:

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

3. This command sends the PlaceCallReq event to the MMCallControl file element.
4. Click **OK** to apply the changes and close the dialog box. The transition now includes an action.
5. Double-click the `Disconnect` transition, or right-click and select **Features** from the pop-up menu. The Features dialog box opens.
6. In the Action field, type the following action language:

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

This command sends the Disconnect event to the MMCallControl file element.

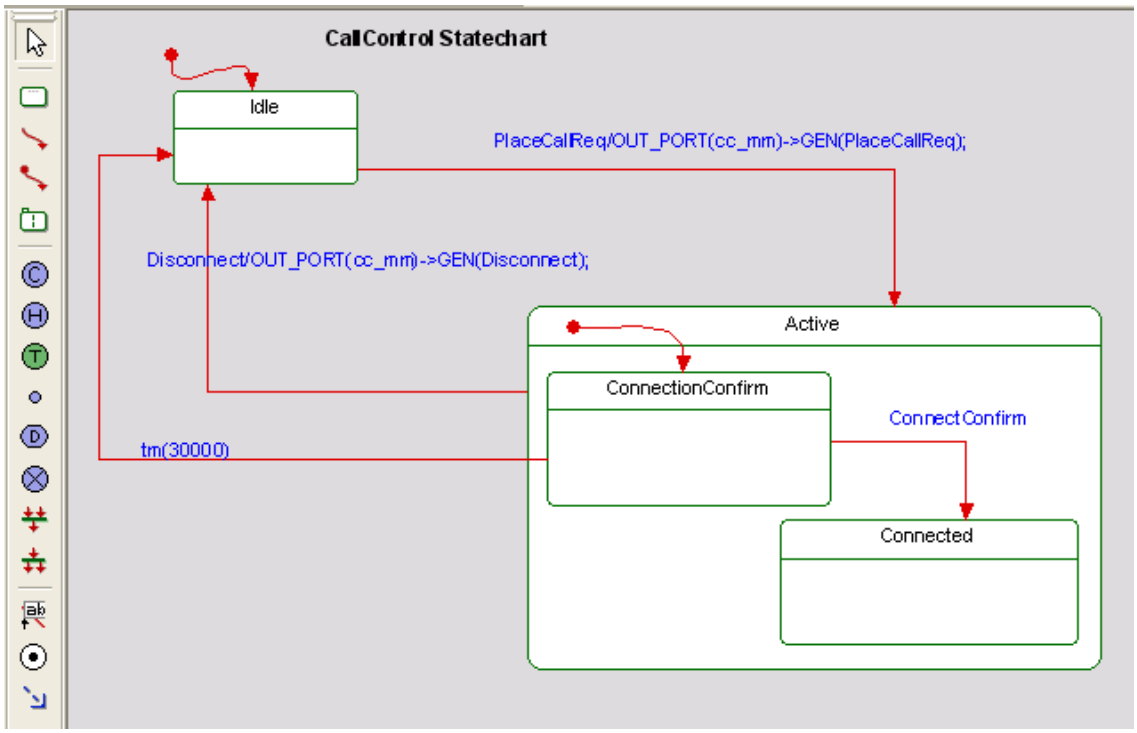
Drawing a Timeout Transition

A *timeout transition* causes a part 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 that the part should wait before making the transition.

In this example, draw a timeout transition in which `ConnectionConfirm` waits thirty seconds before returning to the `Idle` state if a connect confirmation is not made as follows:

1. Click the **Transition** icon on the Drawing toolbar.
2. Draw a transition from `ConnectionConfirm` to `Idle`.
3. Type `tm(30000)`and press **Ctrl**+**Enter**.

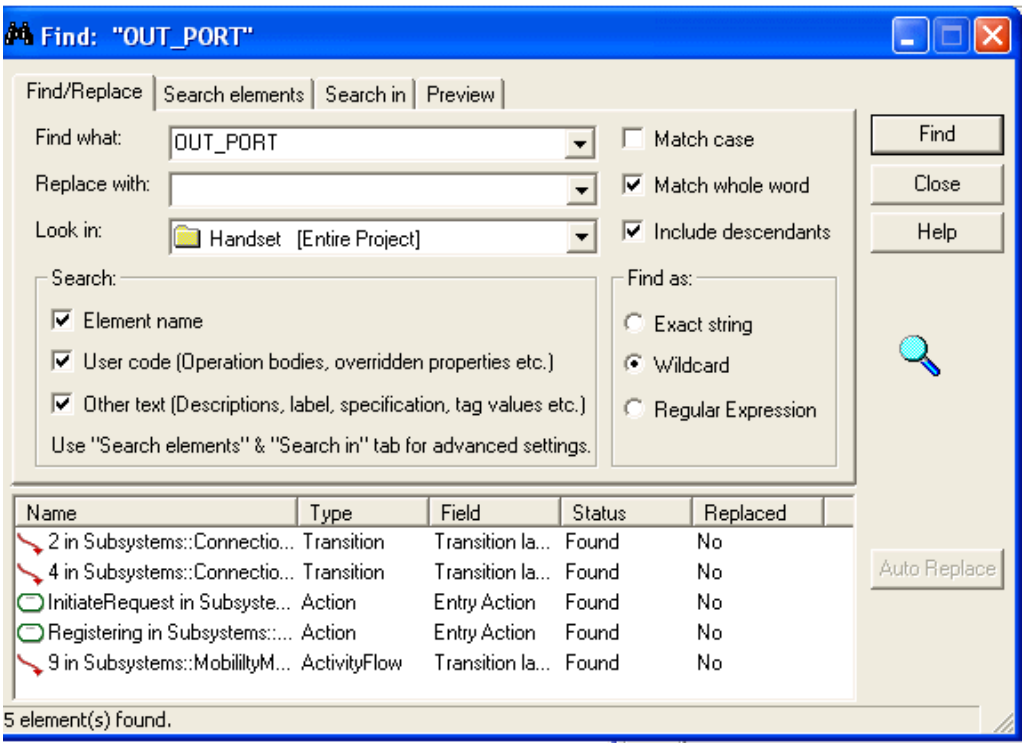
The completed statechart should resemble this example.



Checking Action Language Entries

After entering action language into several diagrams, it is useful to check those entries using the Rhapsody search facility. Follow these steps to check the `OUT_PORT` action language entries:

1. Select the **Edit** > **Search in Model** menu options.
2. Type a portion of the action that you want to use for the search in the **Find What** field. In this case, that is `OUT_PORT`.
3. Click **Find**. Rhapsody lists all of the locations where it found those actions.
4. Click on the entries in the list of elements found, and the system displays the diagram containing that entry and the dialog box with the full action description. Make any corrections that are needed.



System Validation

Rhapsody enables you to visualize the model through simulation. *Simulation* is the execution of behaviors and associated definitions in the model. Rhapsody simulates the behavior of your model by executing its behaviors captured in statecharts, activity diagrams and textual behavior specifications. Structural definitions like blocks, ports, parts and links are used to create a simulation hierarchy of subsystems.

Once you simulate the model, you can open simulated diagrams, which allow you to observe the model as it is running, perform design-level debugging and perform the following tasks:

- Step through the model
- Set and clear breakpoints
- Inject events
- Simulate an output trace

It is good practice to test the model incrementally using model execution. You can simulate pieces of the model as it is developed. This allows you to determine whether the model meets the requirements and find defects early in the design process. Then you can test the entire model. In this way, you iteratively build the model, and then with each iteration perform an entire model validation.

Note

If you are using the System Architect version of Rhapsody, the simulation feature is not available.

Preparing for Simulation

To run a simulation, follow these general steps:

1. Create a component.
2. Create a configuration for your component.
3. Generate component code.
4. Build the component application.
5. Simulate the component application.

The following sections describe these steps in detail.

Creating a Component

A *component* is a level of organization that names and defines a simulatable component. Each component contains configuration and file specification categories, which are used to build and simulate model.

Each project contains a default component, named DefaultComponent. You can use the default component or create a new component. In this example, you can rename the default component Simulation, and then use the Simulate component to simulate the model.

To use the default component, follow these steps:

1. In the browser, expand the `Components` category.
2. Select `DefaultComponent` and rename it `Simulation`.

Setting the Component Features

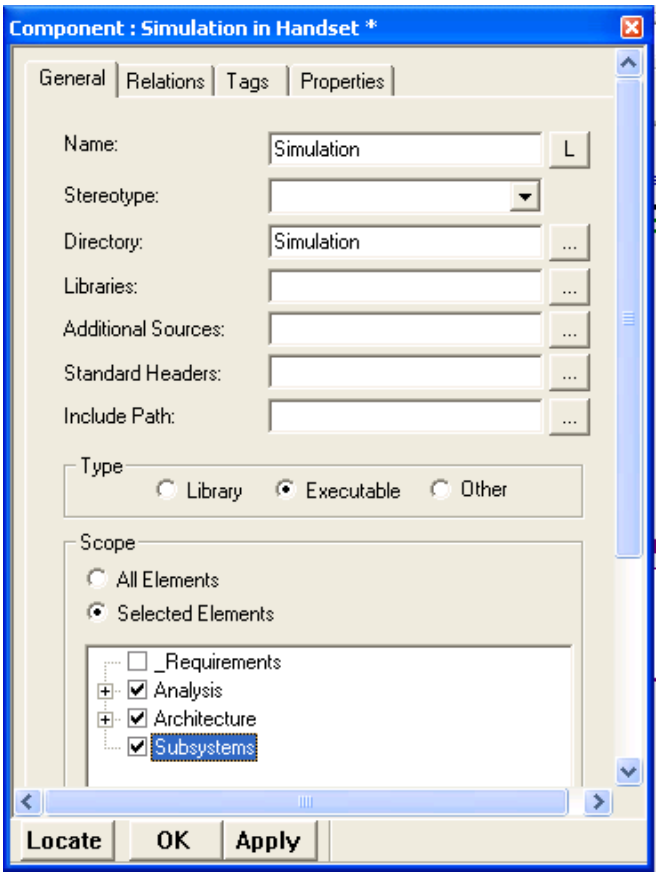
Once you have created the component, you must set its features.

To set the component features, follow these steps:

1. In the browser, double-click `Simulation` or right-click and select **Features** from the pop-up menu. The Component dialog box opens.
2. The **Executable** radio button to set the Type.
3. Select **Selected Elements** as the Scope.
4. Select `Analysis`, `Architecture`, and `Subsystems` as the Selected Elements.

These are the packages for which you create a simulatable component. Do not select the `_Requirements` package because you do not simulate it.

The Component dialog box should resemble this example.



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

Creating a Configuration

A component can contain many configurations. A *configuration* includes the description of the classes to include in code generation, and settings for building and Simulating the model.

Each component contains a default configuration, named `DefaultConfig`. In this example, rename the default configuration to `Debug`, and then use the Debug configuration to simulate the model.

To use the default configuration, follow these steps:

1. In the browser, expand the `Simulate` component and the `Configurations` category.
2. Select `DefaultConfig` and rename it `Debug`.

Setting the Configuration Features

Once you have created the Debug configuration, you must set the values for Simulating the model as follows:

1. In the browser, double-click `Debug` or right-click and select **Features** from the pop-up menu. The Configuration Features dialog box opens.
2. Select the Initialization tab and set the following values:
   a. For the **Initial instances** field, select **Explicit** to include the classes which have relations to the selected elements.
   b. Select **Generate Code for Actors**.
3. Select the Settings tab, and set the following values:
   a. Select **Animation** from the Instrumentation Mode pull-down list. This adds instrumentation code allowing you to simulate the model.
   b. Select **Real** (for real time) as the Time Model.
   c. Select **Flat** as the Statechart Implementation. Rhapsody implements states as simple, enumerated-type variables.

   Rhapsody fills in the **Environment Settings**, based on the compiler settings you configured during installation. A compiler is used to build the simulation model.
4. Click **OK** to apply the changes and close the dialog box.

Before you a build a simulation component, you must first set the active configuration. The active configuration is the configuration for which you simulate a simulation component. The active configuration appears in the pull-down list in the Code toolbar.

To simulate the `Debug` configuration, follow these steps:

1. In the browser, right-click the `Debug` configuration, then select **Set as Active Configuration** from the pop-up menu.

2. Choose **Simulation** > **Full Build** from the menu or the **Full Build** icon  . Rhapsody displays a message that the Debug directory does not yet exist and asks you to confirm its creation.

3. Click **Yes**.

Rhapsody displays output messages in the **Build** tab of the Output window (shown below). The messages inform you of the simulatable component creation status including the following:

- Success or failure of internal checks for the correctness and completeness of your model. These checks are performed before simulatable component creation begins.
- Errors or warnings in simulatable component build process.
- Completion of simulatable component build process.

Simulating the Model

To start simulation without including current changes, use the Smart Build feature, as described below:

- Select **Simulation** > **Smart Build** from the menu or the **Smart Build** icon  .

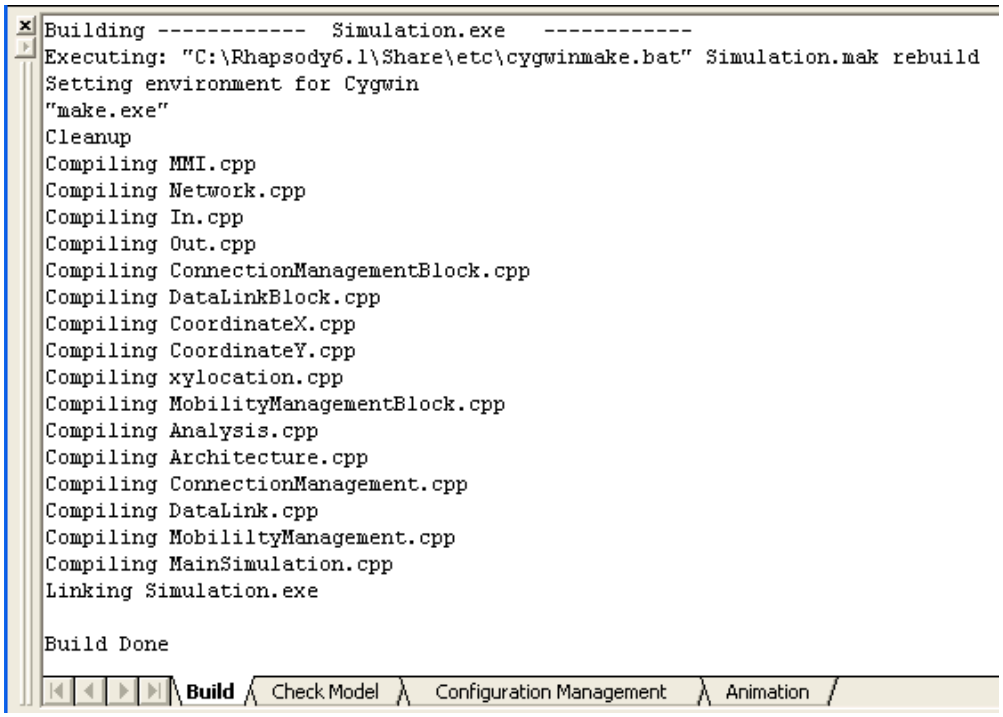
Rhapsody starts simulation and performs the following tasks:

- Displays the simulation toolbar, which enables you to control the simulation process.
- Displays a console window, which provides input to and output from the model. You can position and resize the console and Rhapsody windows so both are visible.
- Displays four output window panes:
  - Build
  - Check Model
  - Configuration
  - Animation

Note

If the output panes are not displayed, select **View** > **Output Window**. The output panes are dockable, so you can move them out of the Rhapsody interface to increase the viewable area for simulations.

If your model builds without errors, your output window should resemble this example.



Creating Initial Instances

It is a good idea to click the **Go Idle** command icon  immediately after starting an executable model so all initial instances are created.

To create instances, click **Go Idle** after starting the model. The initial instances are created (as well as any instances created by those instances) and are listed under the Instances category for the class in the browser.

Break Command

To interrupt a model that is executing, click the **Break** icon  to issue the Break command.

The **Break** command enables you to regain control immediately (or as soon as possible). Issuing a **Break** command also suspends the clock, which resumes with the next **Go** command.

Note

For simple applications, there might be a backlog of notifications. Although the model stops executing immediately, the animator can accept further input only after it has cleared this backlog and displayed any pending notifications.

The **Break** command cannot stop an infinite loop that resides within a single operation. For example, issuing a **Break** cannot stop a `while()` loop:

Preparing to Web-enable the Model

The first step in Web-enabling a working Rhapsody model is to set its configuration and elements as Web-manageable and then to simulate, build, and run the model.

Creating a Web-Enabled Configuration

In this example, create a new configuration and then set its features as follows:

1. Right-click the `Configurations` category and select **Add New Configuration** from the pop-up menu.
2. Type `Panel`.
3. Double-click `Panel` or right-click and select **Features** from the pop-up menu. The Features dialog box opens.
4. Select the **Initialization** tab and set the following values:
   a. For the **Initial instances** field, select **Explicit** to include the classes which have relations to the selected elements.
   b. Select **Generate Code for Actors**.
5. Click **Apply** to save these selections and keep the dialog box open.
6. Select the **Settings** tab, and set the following values:
   a. Select **Web Enabling** for **Webify**.
   b. If desired, click the **Advanced** button to change the default values for the Webify parameters. Rhapsody opens the Advanced Webify Toolkit Settings dialog box.

      This dialog box contains the following fields, which you can modify:

      – Home Page URL—The URL of the home page

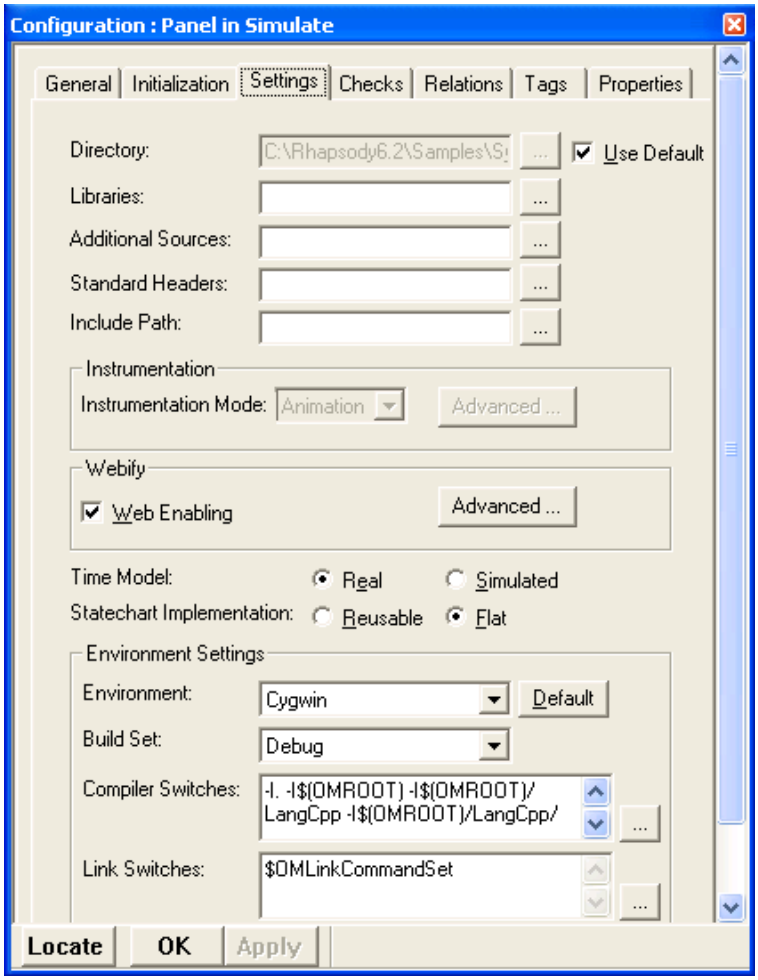
      – Signature Page URL—The URL of the signature page

      – Web Page Refresh Period—The refresh rate in milliseconds

      – Web Server Port—The port number of the Web server

   c. Select **Real** (for real time) as the **Time model**.
   d. Select **Flat** as the **Statechart Implementation**. Rhapsody implements states as simple, enumerated-type variables.

Rhapsody fills in the **Environment Settings** section, based on the compiler settings you configured during installation. This example uses the Cygwin compiler. At this point the dialog box should resemble this example.



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

Selecting Elements to Web-enable

To Web-enable the model, set the elements that you want to control or manage remotely over the Internet using either the Rhapsody `Web Managed` stereotype or the `WebManaged` property.

In this example, you examine how calls are established and disconnected by setting the stereotypes of the following events to Web Managed:

- `CallConfirm`
- `ConnectConfirm`
- `Disconnect`
- `PlaceCallReq`

To select elements to Web-enable, follow these steps:

1. To locate the items you wish to change, select the **Edit** > **Search in Model** option. Type `CallConfirm` into the **Find What** field and click **Find**. The search shows all instances of that text and the browser path for each.
2. Double-click `CallConfirm` located under the `Subsystems` browser category.
3. In the Features dialog, select **Web Managed** from the **Stereotype** pull-down list.
4. Click **OK** to apply the changes and close the dialog box.
5. Make the same change to the remaining three events to make them Web Managed.

Note

If the element already has an assigned stereotype, set the element as Web-managed using a property. In the **Properties** tab, select `WebComponents` as the subject, then set the value of the `WebManaged` property within the appropriate metaclasses to `True`.

Building the Panel

In order to build the Panel, follow these steps:

1. Select **Panel** from the pull-down menu as show here.

2. Click the Full Build icon  .
3. After completing the build, you may use any of the command icons to the left.

Connecting to the Web-enabled Model

Rhapsody includes a collection of default pages that serve as a client-side user interface for the remote model. When you run a Web-enabled model, the Rhapsody Web server automatically simulates a Web site including the file structure and interactive capability. This site contains a default collection of simulated on-the-fly pages that refreshes each element when it changes.

Note

You can also customize the Web interface by creating your own pages or by referencing the collection of pages that come with Rhapsody.

Navigating to the Model through a Web Browser

You can access a Web-enabled model running on your local machine or on a remote machine. In this example, you will connect to the model on your local machine.

To connect to the Web-enabled model on your local machine, follow these steps:

1. Open Internet Explorer.
2. In the address field, type the following URL:

   ```
   http://localhost
   ```

   Other users on the same network can connect to your local model using the IP address or machine name in place of `localhost`.

   If you changed the Web server port using the Advanced Webify Toolkit Settings dialog box, type the following:

   ```
   http://<localhost>:<port number>
   ```

   In this URL, `<localhost>` is `localhost` (or the machine name or IP address of the local machine running the handset model), `<port number>` is the port specified in the Advanced Webify Toolkit Settings dialog box.

By default, the Parts Navigation page of the Rhapsody Web user interface opens.

Note

If you cannot view the right-hand frame in Internet Explorer, go to **Tools** > **Internet Options** > **Advanced** and uncheck the option *Use Java xx for <applet>*.

Viewing and Controlling a Model

The Parts Navigation page provides easy navigation to the Web Managed elements in the model by displaying a hierarchical view of model elements, starting from the top level aggregate. By navigating to and selecting an aggregate in the left frame of this page, you can monitor and control your model in the *aggregate table* displayed in the right frame.

Aggregate tables contain name-value pairs of Rhapsody Web-enabled elements that are visible and controllable through Internet access to the machine hosting the Rhapsody model. They can contain text boxes, combo-boxes, and **Activate** buttons. You can monitor the model by reading the values in the dynamically populated text boxes and combo-boxes. You can control the model by pressing the **Activate** button, which initializes an event, or by editing writable text fields.

Sending Events to Your Model

You can simulate events in the Rhapsody Web user interface and monitor the resulting behavior in the simulated diagrams.

In this example, you simulate the `PlaceCallReq`, `ConnectConfirm`, and `Disconnect` events and view the results in the simulated diagrams as follows:

1. If the simulated Connection Management Place Call Request Success SD is not already open, simulate it and click the **Go** button in the simulation toolbar.
2. If the simulated CallControl statechart is not already open, simulate it.
3. If the simulated MobilityManagement activity diagram is not already open, simulate it.
4. Resize the Rhapsody Web user interface browser window so that you can view the simulated diagrams while sending events to the model.
5. In the navigation frame on the left side of the browser, expand **ConnectionManagement_C[0]**, and click **ConnectionManagement_C::CallControl_C[0]**
6. In the Rhapsody Web user interface, click **Activate** next to `PlaceCallReq`.
7. Open the simulated sequence diagram. Rhapsody displays how the instances pass messages, as shown in the following figure.
8. In the simulated statechart, `Idle` and `PlaceCallReq` transition to the inactive state (olive), and `Active` and `ConnectionConfirm` transition to the active state (magenta). Then `ConnectionConfirm` and `ConnectConfirm` transition to the inactive state (olive), `Active` remains in the active state (magenta), and `Connected` transitions to the active state (magenta).

9. In the simulated activity diagram, `Idle` transitions from the active state to the inactive state. `Registering`, `CheckSignal`, and `LocationUpdate` transition from inactive to active to inactive. Then `InCall` transitions from the inactive state to the active state in the Rhapsody Web GUI, click **Activate** next to `Disconnect`.

   In the simulated statechart, the `Active` and `Connected` states and `Disconnect` change to the inactive state (olive), and `Idle` transitions to the active state (magenta).

   In the simulated activity diagram, `InCall` transitions to the inactive state and `Idle` becomes active.

You can continue generating events and viewing the resulting behavior in the simulated diagrams.

Generating Reports

Rhapsody offers two ways to simulate reports from the models, charts, and other design items:

- A simple and quick internal RTF report generator
- A more powerful reporting tool, Rhapsody *ReporterPlus*

ReporterPLUS

ReporterPlus produces reports that are suitable for formal presentations and can be output in any of these formats:

- HTML page
- Microsoft Word
- Microsoft PowerPoint
- rtf
- text

You can save the file and view it in any program that can read the report's format. In addition, you can create custom report specifications that define the structure, content, and format of reports. The following stylistic definitions all control a report's appearance:

- ReporterPLUS template and selected options
- Output type (Word, PowerPoint, HTML, RTF, text)
- Word or PowerPoint template
- HTML style sheet
- HTML tags in your model, ReporterPLUS template, or in an inserted file

ReporterPlus has extensive Help information available from the interface. Use the **Help Topics** to answer your more detailed questions about the features of this specialized interface.

Examining and Customizing ReporterPlus Templates

Rhapsody includes numerous pre-fabricated report templates that you may want to use as they are or customize to meet your needs.

Note

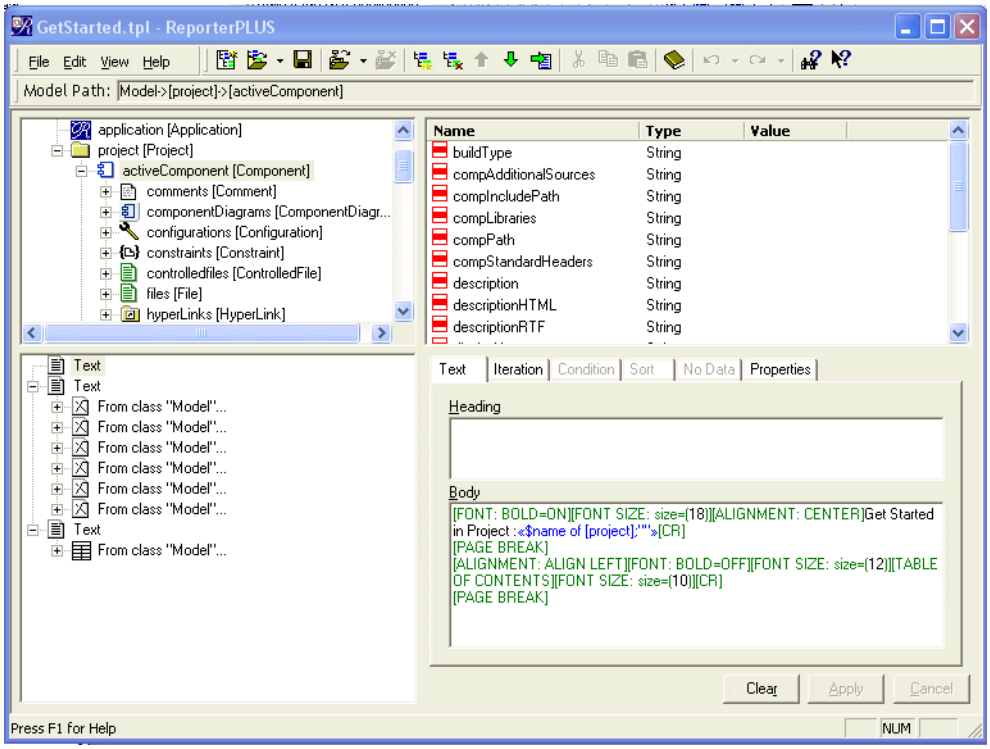
These files are stored in the Rhapsody\reporterplus\Templates directory.

Rhapsody models can be loaded into the ReporterPlus interface and used to create generic or model-specific templates. This interface allows you to create and modify templates graphically using a drag-and-drop method.

To access ReporterPlus and the templates, follow these steps:

1. Select **Tools** > **ReporterPlus** from the menu.
2. From the next menu, select **Create/Edit template with ReporterPlus**.
3. Your model displays in the upper left corner of the ReporterPlus interface. Click the items in the tree to expand and examine the model. When you click part of the model a description of that item appears to the right of the model tree.
4. Select **File** > **Open Template**.
5. Select any of the templates in the dialog box.The template structure appears in the lower left area.

6. Click on individual items in the template and the Q Language definition of the item appears to the right, as shown in the example below.



7. You may also want to add standard headings and text to an existing template. To add this “boilerplate” material, highlight a section of the template in the lower left window and click the **Text** tab in the lower right window. Type text in the Heading and Body sections as desired.

Note

For more complex changes, study the Q Language that ReporterPlus uses to define report expressions. This language is defined in a PDF file accessed from ReporterPlus Help Topics.

Using the System Model Template

ReporterPlus includes a template designed specifically for systems engineers called `SysMLreport.tpl`.

To use this template for your report, follow these steps:

1. With your model displayed in the Rhapsody interface, select **Tools** > **ReporterPlus** from the menu.
2. From the next menu, select **Create/Edit template with ReporterPlus**.
3. Your model displays in the upper left corner of the ReporterPlus interface.
4. Select **File** > **Open Template**.
5. Select the `SysMLreport.tpl` and use it to produce a report for your model.

Note

You may wish to change the template to meet your specific needs.

Report Layout

The main elements of each section are shown along with their page locations and are hyperlinked. The generated report contains the following sections covering the complete SysML profile:

- Requirements diagrams
- Use case diagrams
- Sequence diagrams
- Structure diagrams
- Object model diagrams
- Block Definition diagrams
- Internal Block diagrams
- Parametric diagrams
- Data dictionary
- Model configuration

The report template uses the standard SysML features built into your model when you select the `SysML` project **Type** when you first created your project. Refer the **Creating a New SysML Project section** for more information.

Requirements Diagrams

For any requirements diagrams, the `SysMLreport.tpl` supplies hyperlinks to the location of the definition of any use cases, actors, packages, classes or blocks shown in the diagram.

Note

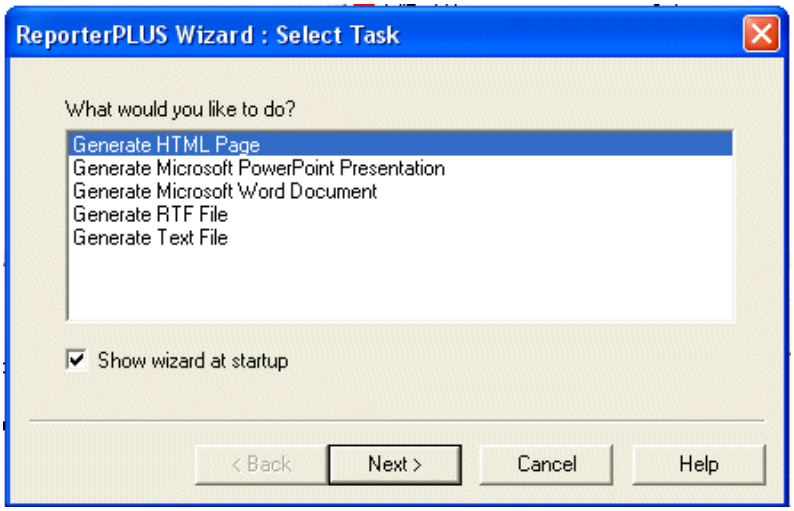
Each requirement must have a **Stereotype** setting so that the reporting feature can extract the requirements data.

Generating Reports

If you are going to generate a report in *Microsoft Word*, be certain that *Word* is closed before you start this procedure to avoid a conflict between *ReporterPlus* and *Word*.

When you have a template that you want to use for a report, follow these steps:

1. With your project open in Rhapsody, select **Tools** > **ReporterPlus** from the menu.
2. From the next menu, select one of these two options:
   - Report on all model elements
   - Report on selected elements
3. Rhapsody displays the ReporterPlus Wizard (shown below) that allows you to select the desired output format and on subsequent dialog boxes, the template, and directory location and name for the finished report.

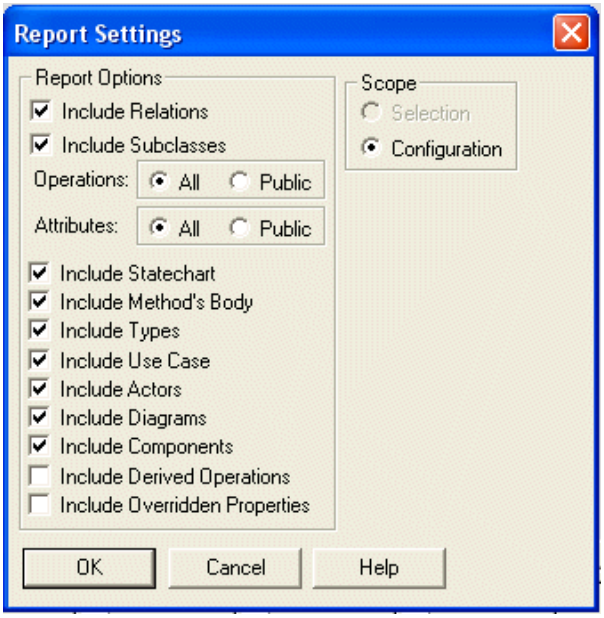


Using the Internal Reporting Facility

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 simple, internal reporter, select **Tools** > **Report on model.** The Report Settings dialog box opens, as shown in the following figure.



The 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.

Select the portions of the model you want to be included in the report, then click **OK** to simulate it. The report is displayed in the drawing area with the current file name in the title bar.

Using the Internal Report Output

When you simulate a report in Rhapsody using **Tools > Report on model**, the initial result uses the internal RTF viewer. To facilitate the developer's research, this output may be used in

- To locate specific items in the report online, select **Edit** > **Find** from the menu and type in the search criteria.
- To print the initially simulated report, select the **File** > **Print** menu option.
- The initially simulated 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.

Index

T

U

V

W

X