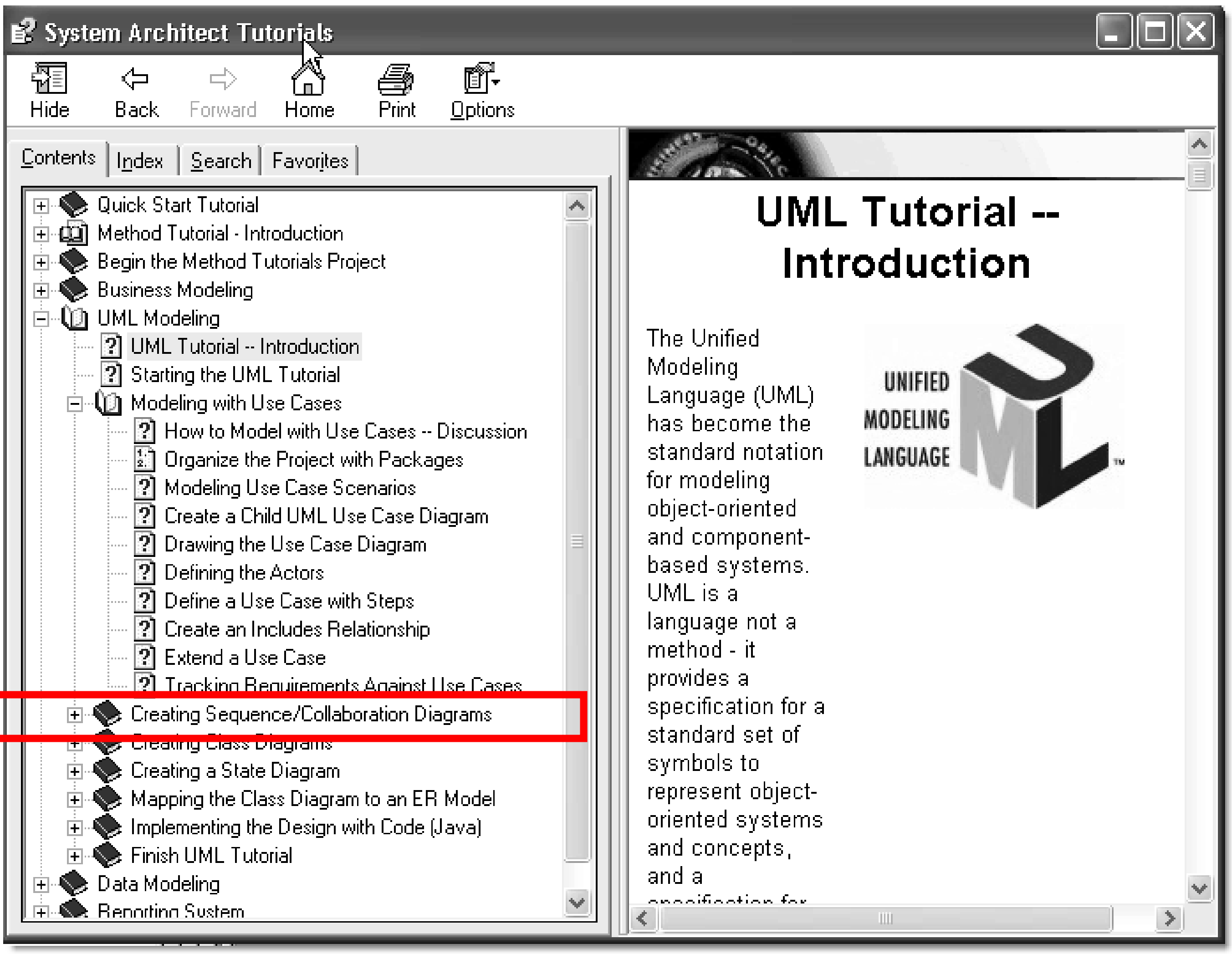
Sequence & Collaboration Diagrams

in UML

This material has been abstracted and slightly modified from the System Architect tutorial File

Robin Beaumont 03/01/2004 14:50


System Architect Tutorials

Hide Back Forward Home Print Options

Contents | Index | Search | Favorites

- Quick Start Tutorial
- Method Tutorial - Introduction
- Begin the Method Tutorials Project
- Business Modeling
- UML Modeling
  - UML Tutorial -- Introduction
  - Starting the UML Tutorial
  - Modeling with Use Cases
    - How to Model with Use Cases -- Discussion
    - Organize the Project with Packages
    - Modeling Use Case Scenarios
    - Create a Child UML Use Case Diagram
    - Drawing the Use Case Diagram
    - Defining the Actors
    - Define a Use Case with Steps
    - Create an Includes Relationship
    - Extend a Use Case
    - Tracking Requirements Against Use Cases
  - Creating Sequence/Collaboration Diagrams
  - Creating Class Diagrams
  - Creating a State Diagram
  - Mapping the Class Diagram to an ER Model
  - Implementing the Design with Code (Java)
  - Finish UML Tutorial
- Data Modeling
- Reporting System

UML Tutorial -- Introduction

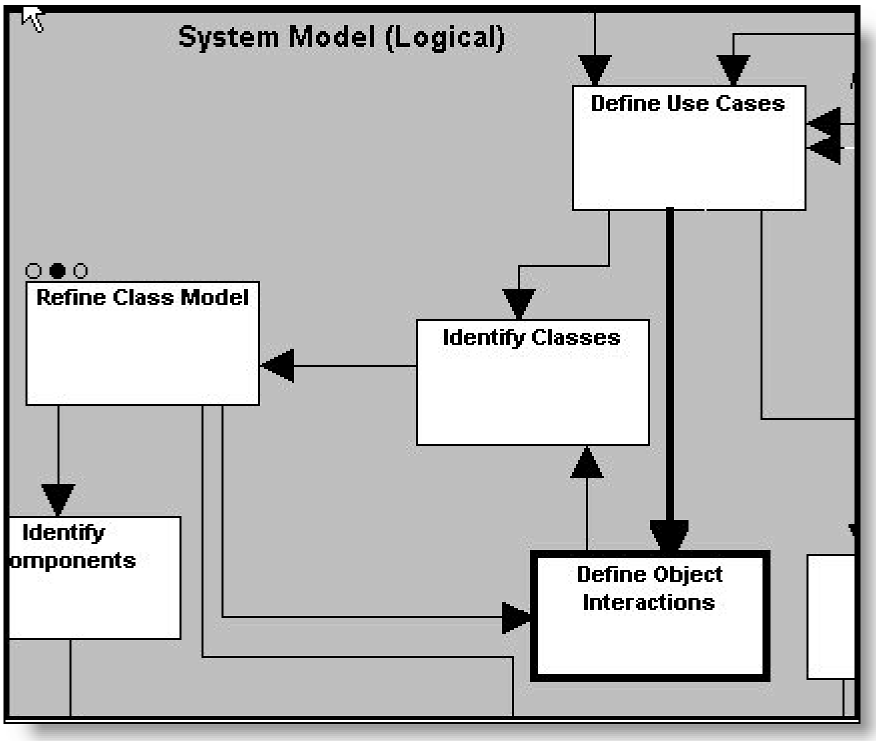
The Unified Modeling Language (UML) has become the standard notation for modeling object-oriented and component-based systems. UML is a language not a method - it provides a specification for a standard set of symbols to represent object-oriented systems and concepts, and a

Contents

Comments I have added are in red - RB

1. Introduction Sequence and Collaboration Diagrams

Whereas you use Use Cases to gain a high-level business view of what goes on in the system, with the Sequence diagram you detail the interactions among objects in the system. The Collaboration diagram offers a different view of the same information as the Sequence diagram. The two are sometimes referred to as Interaction diagrams. Within the Process Chart that we are using to navigate this tutorial, we are at the following place, highlighted in bold in the picture below:



Sequence Diagram

Like Use Case diagrams, you use Sequence diagrams to model scenarios in the system. They offer a different level of detail, however. Whereas Use Cases and the steps or textual descriptions that define them offer a high-level-analysis view of a system, the Sequence diagram offers more specific analysis and design information about the interactions among objects in the system.

In a Sequence diagram, an object appears on the diagram as a vertical dashed line with the name of the object at the top and the name of the object's class optionally drawn below the object name. Events appear as horizontal lines between objects. Event lines are drawn in chronological order, from the top of the diagram to the bottom. They do not necessarily have a one-to-one correspondence with the steps defined for a Use Case scenario.

Collaboration Diagram

The Collaboration diagram shows how the objects in a scenario interrelate. In a Collaboration diagram, the modeler can show detail such as visibility.

Each object in a Collaboration diagram instantiates a particular class in the system. The objects are connected by links, each link representing an instance of an association between the respective classes. The link shows messages sent between the objects, and the type of message passed (synchronous, asynchronous, simple, balking, and timeout).

For the modeler trying to understand all of the effects on a given object, Collaboration diagrams offer a better view of a scenario than a Sequence diagram. Collaboration diagrams are therefore good for procedural design.

Keeping the Diagrams in Synch

To show the implementation details of a scenario, you can choose to draw either a Sequence or a Collaboration diagram, or both. System Architect automatically creates a Collaboration diagram from a Sequence diagram, and vice versa, when you invoke the **Synchronize Diagram** command from the **Draw** menu.

2. Sequence Diagram

The completion time of this section is approximately 60 minutes.

1.1. Create Sequence/Collaboration Diagrams

Sequence/Collaboration Diagrams

In this section of the tutorial, we will create a Sequence diagram. In UML, you model a Sequence and (optionally) a Collaboration diagram for every scenario in the system.

Our first task is to find the objects, classes, and messages of the system. To do this, we'll examine the description of the Use Case. If you have modeled the description of the Use Case as a sequence of steps (as we have done in this tutorial), then you can 'walk through' the steps to discover what objects are necessary for the steps to occur.

1.2. Finding Objects By Examining Use Case Scenarios

The Use Case technique has become a popular vehicle for finding objects during an object-oriented analysis of a system. After modeling the scenarios for the way a system works or should work, you can then examine those scenarios to find the objects necessary for the scenarios to take place.

Examine Nouns and Verbs

One way to find objects is to examine the textual description or steps of every Use Case. Consider nouns in the textual description to be classes or class attributes and verbs to be methods of the classes.

For example, in the Make Reservation Use Case, in order for a Customer to request a reservation, two objects are necessary: Customer and Reception. In order for the step Check Diary for Room Availability to occur, the objects Room and Diary must exist. In order for the step Customer accepts accommodation to occur, the objects Customer and Room must exist, etc. Looking at the steps of Make Reservation, we highlight possible objects in bold, possible class attributes in bold underline, and possible methods in italic.

1. **Customer** *Queries* for **<u>Available</u>** **Rooms**
2. *Store* **Customer** **<u>Details</u>**
3. *Check* **Diary** for **Room** **<u>Availability</u>**
4. **Room** is Available
5. *Advise* **Customer** of **<u>Availability</u>**
6. **Customer** *Requests* **Reservation**
7. *Provisionally Book* **Room**
8. *Figure Out* **<u>Price</u>**, *Advise* **Customer**
9. **Customer** *Accepts* **<u>Terms</u>**
10. *Provisionally Book* **Room**
11. *Check* **Customer** **<u>Credit</u>**
12. **Customer** **<u>Credit</u>** Is OK
13. *Reserve* **Room**.

Important: Before you begin this tutorial make sure you have the Tutorial encyclopaedia open (see the previous tutorial for details).

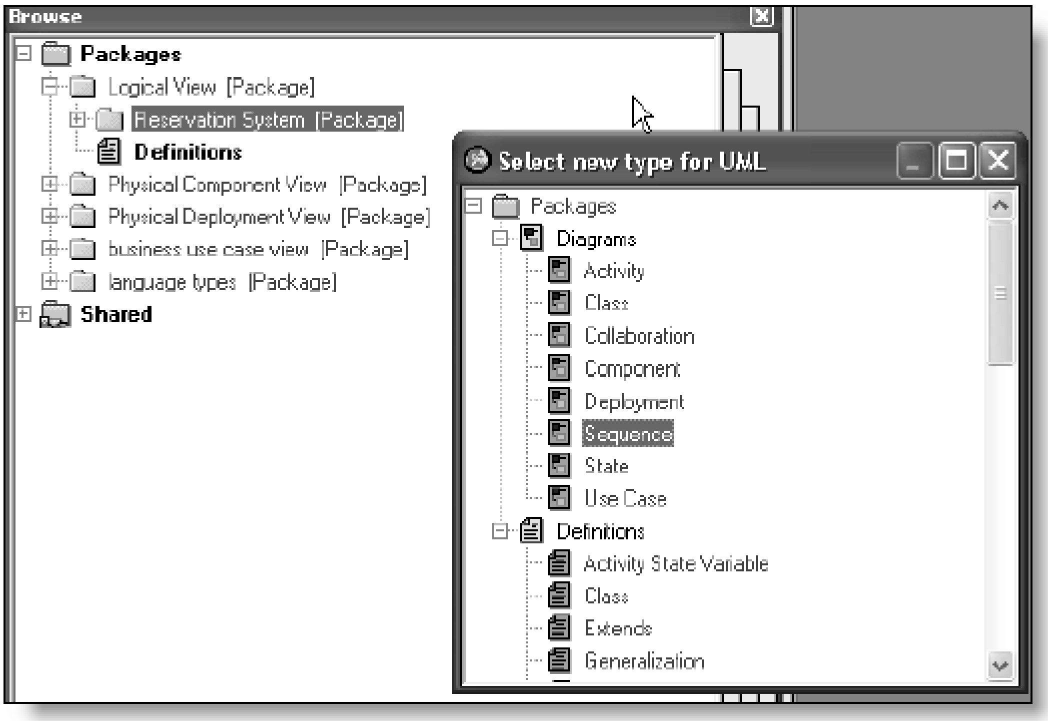
1.3. Create a Sequence Diagram

Let's model this scenario with a Sequence diagram. We will create a Sequence diagram, and make it a child of the Use Case diagram we created in the earlier tutorial. This will create a hypertext link between the two diagrams. For this tutorial, our Use Cases were in the Business Use Case View package. We will build our Sequence diagrams in a **Reservation System** package within the **Logical View** package.

1.3.1. Create the Sequence Diagram

To create a Sequence diagram:

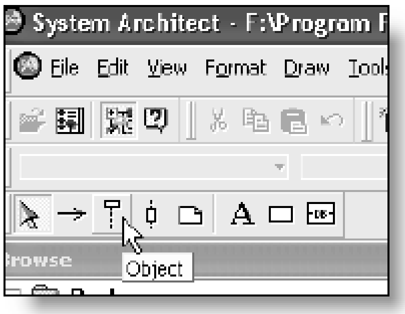
1. In the **UML** tab of the browser, click on the **+** mark next to the **Logical View** package to expand it, revealing the **Reservation System** package underneath it.
2. Right-mouse click on the **Reservation System** package and select **New** to open the **Select New Type for UML** dialog.



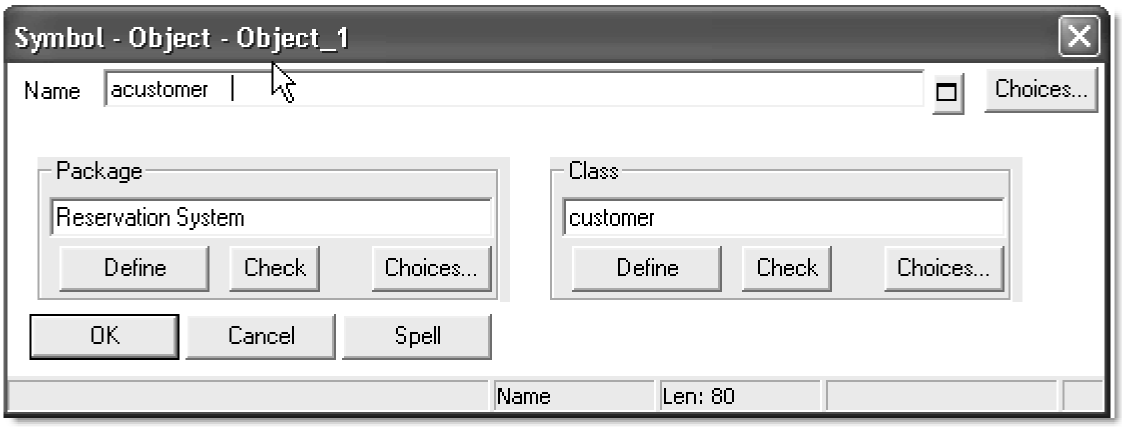
3. Within the **Diagrams** list, double-click on **Sequence** to create a new Sequence diagram.
4. Type in **Make Reservation** in the **New Diagram** dialog.
5. Another dialog box will appear asking which Package you wish the diagram to belong to – accept the default value and hit the **OK** button

1.3.2. Drawing an Object Lifeline

In a Sequence Diagram, an object is shown as a long, vertical, dashed line. This is considered the object lifeline. It represents the life of an object during a scenario. We discovered above that the objects Customer, Reservation, Room, and Diary are needed for the Make Reservation scenario. Let's model them.



1. Select the object symbol from the toolbar and place an object symbol down on the diagram. The **Symbol - Object** (name) dialog appears.
2. In the **Name** field, type in the name **acustomer,** and in the **Class - (Class)** field, type in the name **Customer.** (We can define the class at this point, but we won't. Let's draw the object first and then go back and define the class.)



Symbol - Object - Object_1

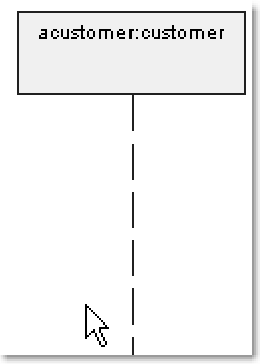
Name: acustomer

Package: Reservation System

Class: customer

Name | Len: 80

You are specifying the object name, and the class that this object instantiates. This class, **Customer**, is a different class than the one we modeled as an actor in the Use Case of the **Reservations** package of the previous section. We are purposely separating business classes from logical classes. We could have used the same class (by clicking on the **Choices** button and dragging it in, or modeling this Sequence diagram in the same package as the Use Cases). This is a design decision we have made.

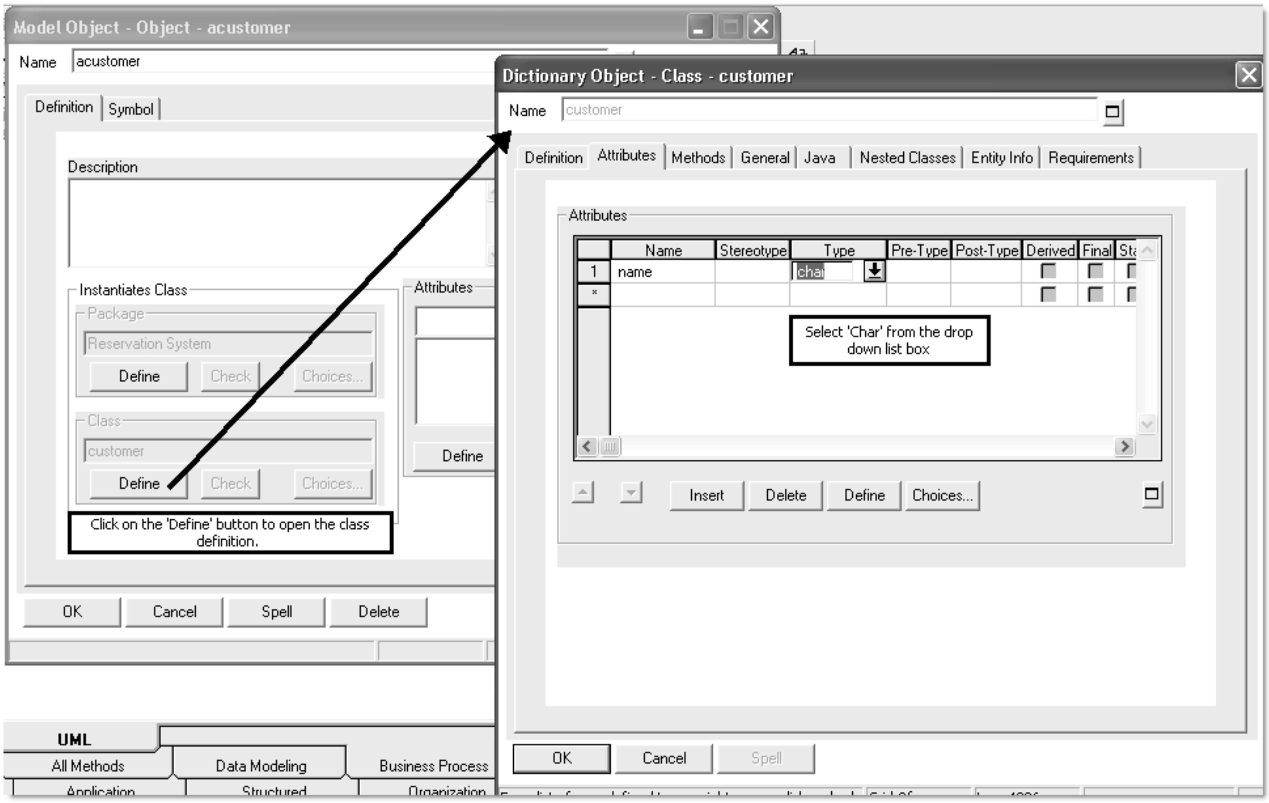


3. Click **OK** to close the dialog. The object lifeline is drawn on the diagram.

1.3.3. Define the Class

Let's go back and define the class of the object we just drew.

1. Double-click on the object lifeline symbol to open its definition dialog.
2. Click the **Define** button in the **Class - (Class), Customer** field. Let's take a look at its definition.


Model Object - Object - acustomer

Name: acustomer

Package: Reservation System

Class: customer

Click on the 'Define' button to open the class definition.

Dictionary Object - Class - customer

Name: customer

	Name	Stereotype	Type	Pre-Type	Post-Type	Derived	Final	Sta
1	name		char					
*								

Select 'Char' from the drop down list box

Adding Class Attributes

A class Customer may have many attributes associated with it, for example, Social Security Number, email, age, sex, height, weight, etc. Although many such attributes may immediately come to mind, you must be careful to add only those attributes that are relevant to your design. For example, to make a hotel reservation, we might also want the customer to provide the Credit Card Number that they will be charging. If it were a reservation over the Internet, we might require a return email address. At this point it would be hard to defend why we would need their Social Security Number or their Height and Weight for them to make a reservation.

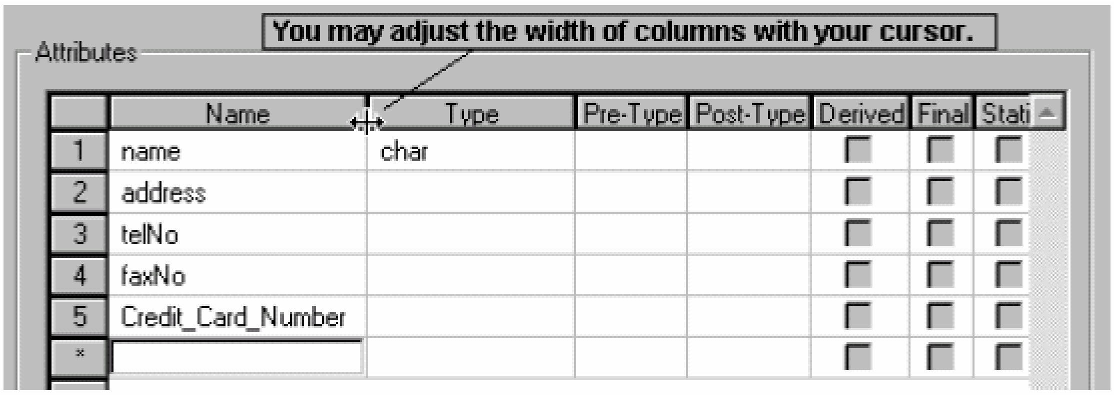
Let's add the attribute Credit Card Number to the Customer class. To add an attribute to the class:

3. Within the class definition, with the **Attributes** tab selected, place your cursor in the next empty cell of the Attributes grid, type **name** and hit your **Enter** key.
4. In the **Type** cell, hit the '**c**' key on your keyboard to quickly select the **char** type for this attribute.*[Note (RB): I found that this shortcut did not work I needed to enter the whole word'char' or select it from the drop down list box]*

Let's add a few more attributes. We'll add the attributes first, then go back and specify a type for each one.

5. Add the following attributes to the grid, hitting your **Enter** key after each one to enter it:

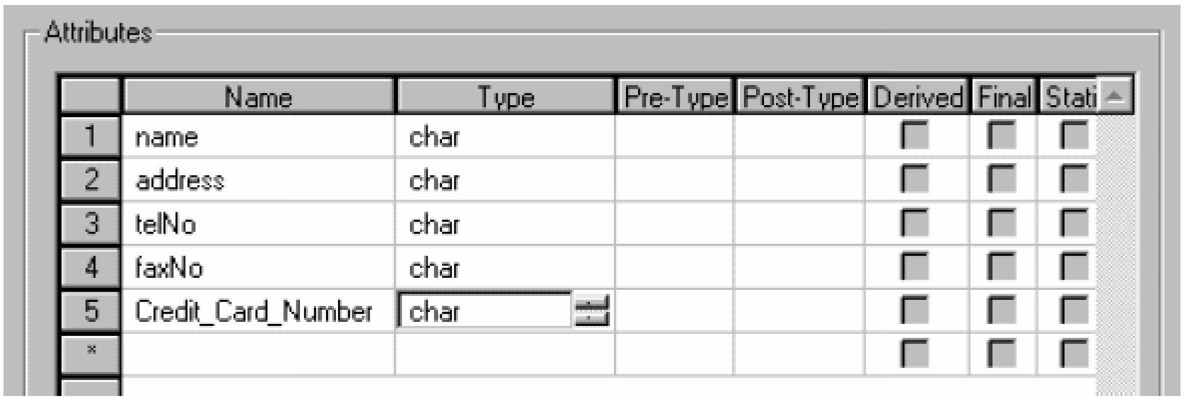
address
telNo
faxNo
Credit_Card_Number



	Name	Type	Pre-Type	Post-Type	Derived	Final	Stati
1	name	char			☐	☐	☐
2	address				☐	☐	☐
3	telNo				☐	☐	☐
4	faxNo				☐	☐	☐
5	Credit_Card_Number				☐	☐	☐
*					☐	☐	☐

Note: When typing in the name of an attribute, you may elect to put spaces in between words (embedded spaces). For example, you could have specified the last attribute added above as Credit Card Number. This is completely okay during analysis. However, most implementation languages (i.e., Java, C++) do not allow embedded spaces in attribute or method names. In this tutorial we will not be changing the name of this attribute later during design, so we did not use embedded spaces in the name.

6. Specify each attribute as type **char**.



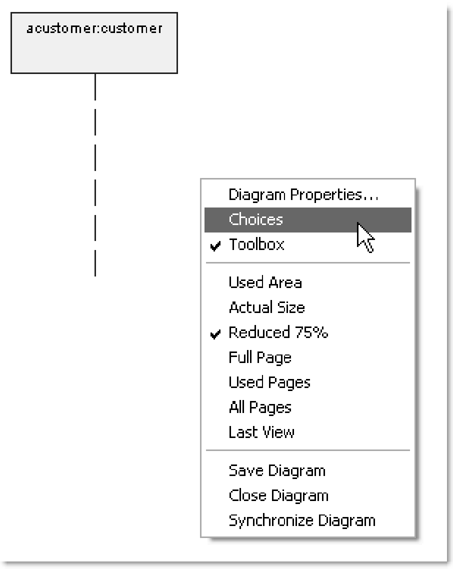
	Name	Type	Pre-Type	Post-Type	Derived	Final	Stati
1	name	char			☐	☐	☐
2	address	char			☐	☐	☐
3	telNo	char			☐	☐	☐
4	faxNo	char			☐	☐	☐
5	Credit_Card_Number	char			☐	☐	☐
*					☐	☐	☐

7. Click **OK** twice to close all dialogs.

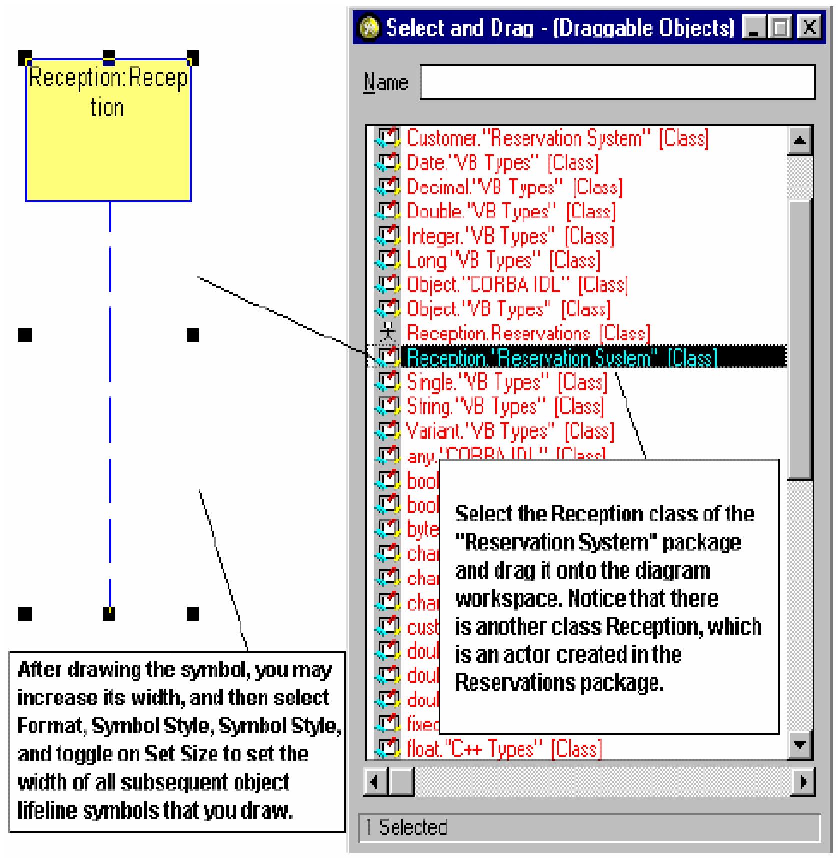
1.4. Adding Additional Objects to the Sequence Diagram

So far we have added the object aCustomer (instantiating the class Customer) to the Sequence diagram. Now let's add the other objects that we found in the steps of the Make Reservation Use Case: Reception, Reservation, Diary, and Room.

To add the object Reception, we will use a different technique than that used previously - we will reuse the class Reception of the package Reservation System that has been preloaded into this project encyclopedia (to make taking the tutorial faster).



1. Right-mouse click on the Sequence diagram workspace, and select **Choices**. You can view a list of classes and objects already established in this encyclopedia
2. Select the **Reception."Reservation System"** class and drag and drop it onto the diagram workspace. Note that there are two Reception classes -- the Reception actor of the Reservations package that we created earlier, [during the Use Case tutorial if you saved it! RB] and a Reception class of the Reservations System package. Make sure you drag this latter one onto the diagram workspace. An object **Reception**, instantiating the class **Reception**, is created. Close the **Select and Drag** dialog.


Reception:Reception
Select and Drag - (Draggable Objects)
Name
Customer."Reservation System" [Class]
Date."VB Types" [Class]
Decimal."VB Types" [Class]
Double."VB Types" [Class]
Integer."VB Types" [Class]
Long."VB Types" [Class]
Object."CORBA IDL" [Class]
Object."VB Types" [Class]
Reception.Reservations [Class]
Reception."Reservation System" [Class]
Single."VB Types" [Class]
String."VB Types" [Class]
Variant."VB Types" [Class]
float."C++ Types" [Class]
1 Selected

After drawing the symbol, you may increase its width, and then select Format, Symbol Style, Symbol Style, and toggle on Set Size to set the width of all subsequent object lifeline symbols that you draw.

Select the Reception class of the "Reservation System" package and drag it onto the diagram workspace. Notice that there is another class Reception, which is an actor created in the Reservations package.

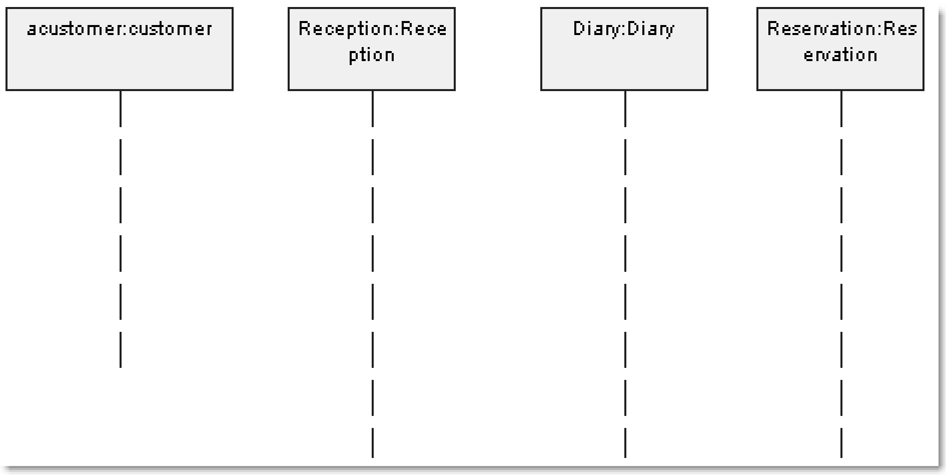
3. Position the object **Reception** so that it is parallel to the first object drawn, **Customer**.

Moving Objects on a Sequence Diagram and Diagram Grid Options: By default, the Sequence diagram employs an invisible, horizontal grid that you draw and move symbols on. You may change the flexibility of this grid, or turn it off altogether, by selecting **Format, Diagram Format, Grid and Reduced View.** If you want to see the grid on the diagram workspace, select **Format, Diagram Format, Display Options**, and in the **Display Options** dialog, toggle on **Show Grid**. We will not show the grid in this tutorial.

We will now add two more instances of classes in the same manner:

- Diary
- Reservation [RB]

4. Right-mouse click on the diagram workspace and select **Choices**. From the **Select and Drag** dialog, drag and drop **Diary."Reservation System"**
5. Repeat the process to drag and drop **Reservation."Reservation System"** onto the diagram. Close the **Select and Drag** dialog. Position the two objects in parallel to the other object lifelines to end up with something like this:

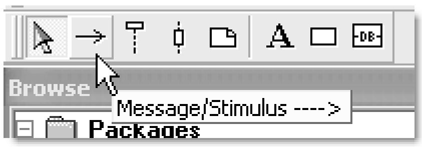


It is rather confusing to have an instance (‘object) of a class given the same name as the Class such as Reception:Reception. I try to use a name that indicates it is a (single) instance for example “acustomer” representing a Customer similarly it probably would be best to have the instance of the Reception class called TheReception or something similar. RB

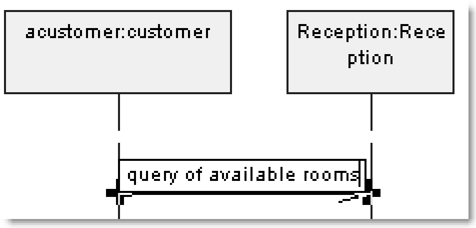
1.5. Draw Message/Stimulus Lines Between Objects

Message/Stimulus lines are drawn between objects to show how and when they communicate. These lines (which we'll refer to simply as message lines) are drawn chronologically from the top of the Sequence diagram to the bottom.

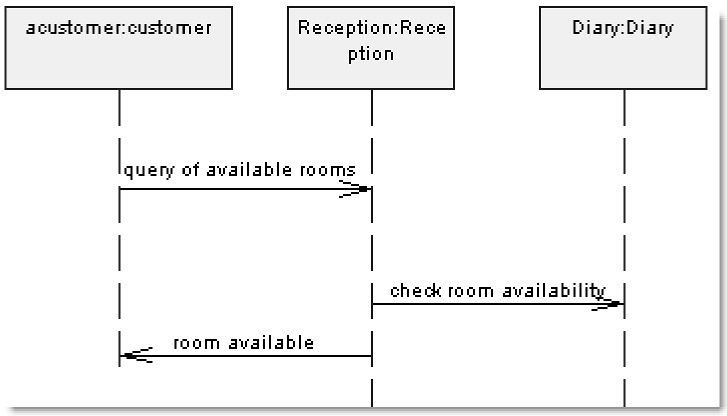
The message line represents a message sent from one object to another, in which the 'from' object is requesting an operation be performed by the 'to' object. The 'to' object performs the operation using a method that its class, or one if its superclasses, contains. Later in this tutorial, we will specify the method that a 'from' object is requesting the 'to' object to invoke. First, let's concentrate on the messages passed between the objects:



1. Select the **Message/Stimulus** tool from the toolbox and draw a message/stimulus line, starting at the object **Customer** and ending at the **Reception** object. When attaching the Event line to an object lifeline, make sure that you see a **+** mark on the intersection. This will tell you that the lines are connected. Name the line **Query for Available Rooms** and press **Enter**.



Reception responds to the event **Query for Available Rooms** by performing an action: it must check the Diary for room availability.



2. Draw a message line from the **Reception** object to the **Diary** object. Name it **Check Room Availability**. We will not model the return message from **Diary** back to **Reception** -- we will leave it implied.

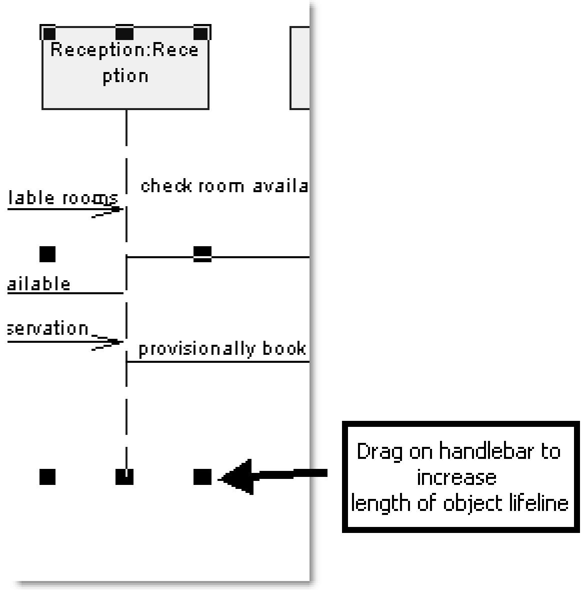
3. Draw a message line from the **Reception** object to the **Customer** object and name it **Room Available**.

The **Customer** may now request to reserve the room (we are assuming the Customer already knows the price per night of a room; perhaps this is information provided on the website, or perhaps we've missed this detail and should add it later to our Sequence Diagram). **Reception** will provisionally book the room in the **Diary**, which will cause the **Reservation** to be created. (In this way, no one else can reserve the room out from under this customer while **Reception** is asking for the **Customer** to provide a credit card -- an embarrassing situation.

4. Draw a message line between **Customer** and **Reception** named **Request Reservation**.

5. Draw an event line between **Reception** and **Diary** named **Provisionally Book Room**.

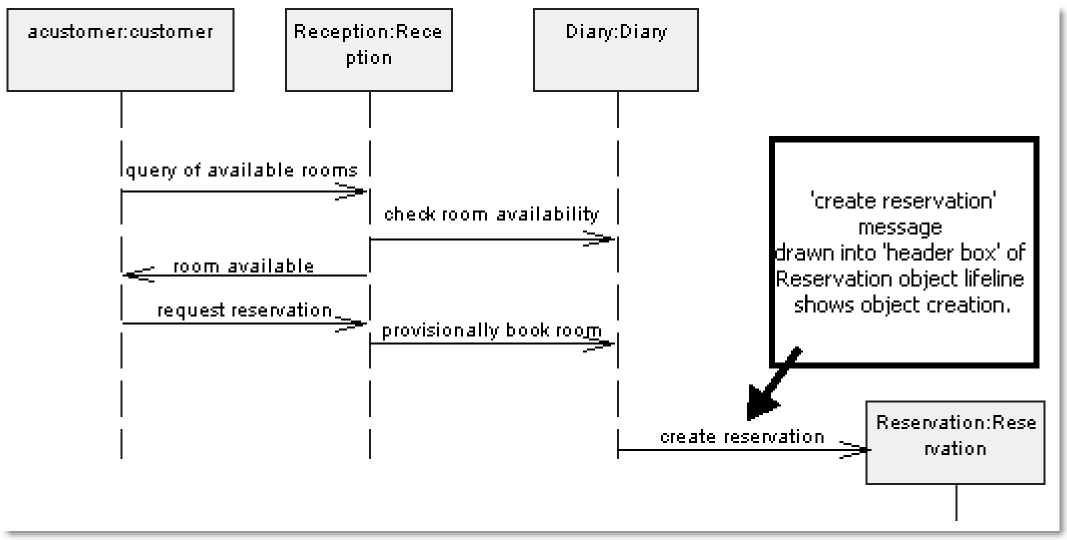
6. Increase the length of the first three object lifeline symbols (all but Reservation), by selecting the symbol and dragging its bottom handlebar downward.



1.6. Show Object Creation

The UML notation for showing that an object on a Sequence diagram is created is to draw a message line into the head of the name box at the top of the object lifeline. To show that Reservation is being created in this scenario, we will move the Reservation object lifeline downward so that we can draw a line into its header box.

1. Select the **Reservation** object and move it down on the diagram so that the next line drawn can be drawn into its 'header box'.
2. Draw a message line from **Diary** to the header box of **Reservation** and name it **Create Reservation**.

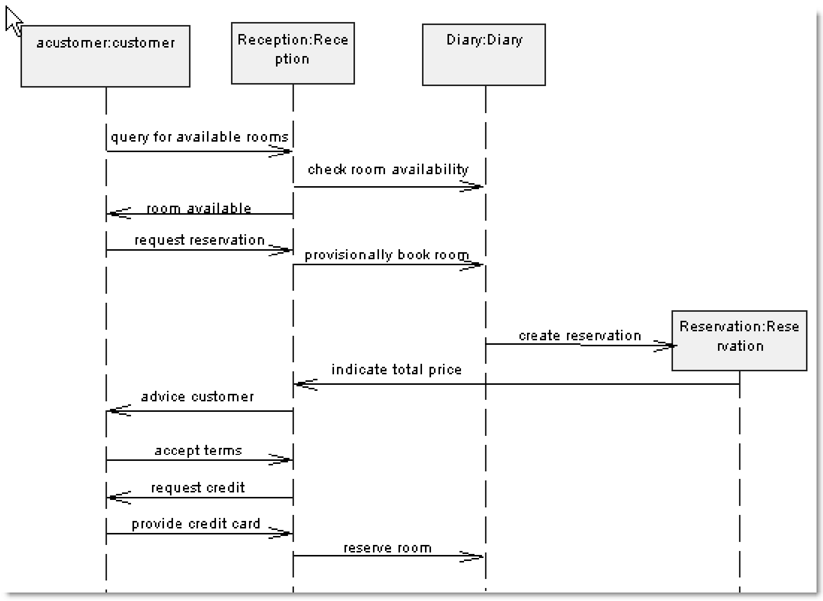


1.7. Complete the Sequence Diagram

The **Reservation** object will compute its price and notify **Reception**, which will advise the **Customer** (**Diary** is not involved in this pricing process). If the **Customer** accepts the terms, he/she notifies **Reception**, which reserves the room in the diary. **Diary** updates the **Reservation** status to **Reserved**.

1. Draw a message line between **Reservation** and **Reception** and name it **Indicate Total Price**.
2. Draw a message line between **Reception** and **Customer** and name it **Advise Customer**.
3. Draw a message line between **Customer** and **Reception** and name it **Accept Terms**.
4. Draw a message line between **Reception** and **Customer** and name it **Request Credit**.
5. Draw a message line between **Customer** and **Reception** and name it **Provide Credit Card**.
6. Draw a message line between **Reception** and **Diary** and name it **Reserve Room**.

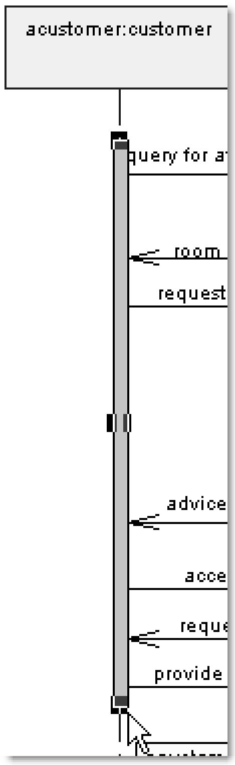
The Sequence diagram should now look like the diagram pictured below.

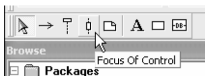


Note: We might also want to send a message from Reception to another object in the system or outside the system to check the customer's credit before reserving a room. We will not do this in our example tutorial to keep it simple; let's assume the client's credit is checked when he/she confirms the room.

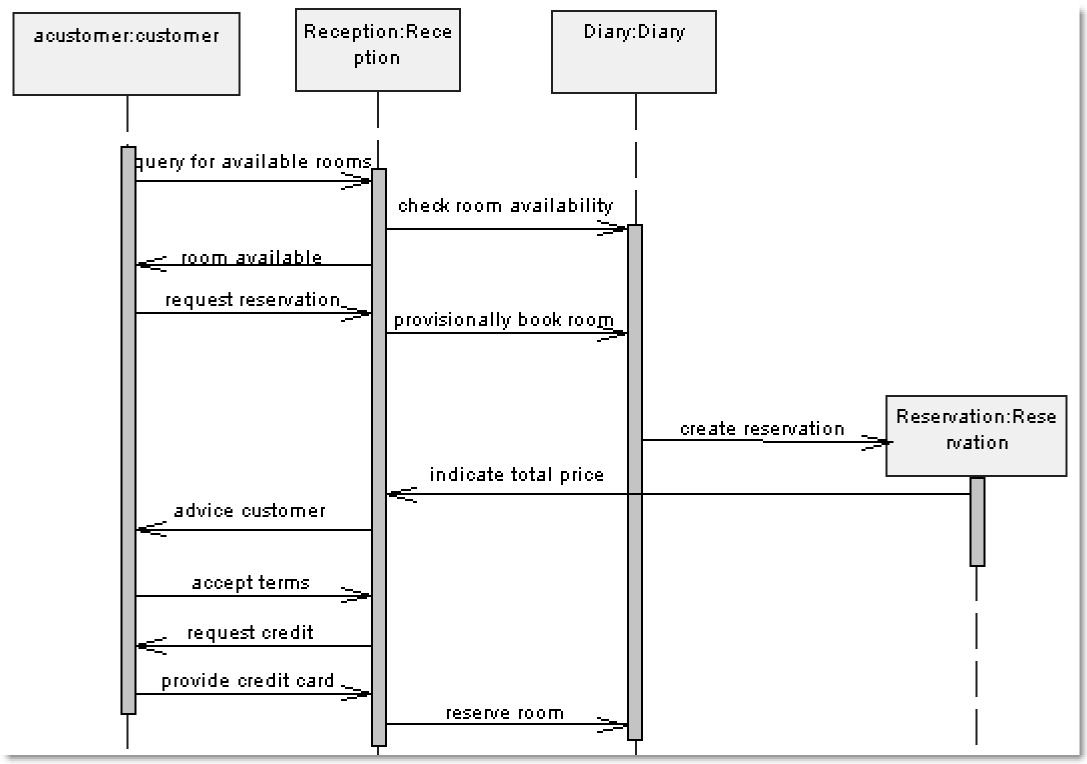
1.7.1. Add 'Focus of Control', or Activation Bars

A focus of control rectangle, or activation bar, is drawn on an object lifeline to show the period during which the object is performing an action. The activation bar represents both the duration of the action in time and the control relationship between the activation and its callers. The terms 'focus of control' and 'activation bar' are interchangeable.





1. Select the **Focus of Control** tool on the toolbox, and drop one onto the **Customer** object lifeline. Notice that you can only drop a Focus of Control bar onto an object lifeline symbol. System Architect does not allow you to drop it onto empty diagram workspace.
2. Choose the **Select Mode** pointer from the toolbox (or hit your **Esc** key) to get back your pointer. Select the **Focus of Control** bar, and drag on its bottom and top handlebars so that it covers the object lifeline at all points where it sends or receives a message.
3. Repeat steps 1 and 2 for the other object lifelines on the diagram. Your diagram should look like the one pictured below.



1.8. Adding a Method to an Message Line

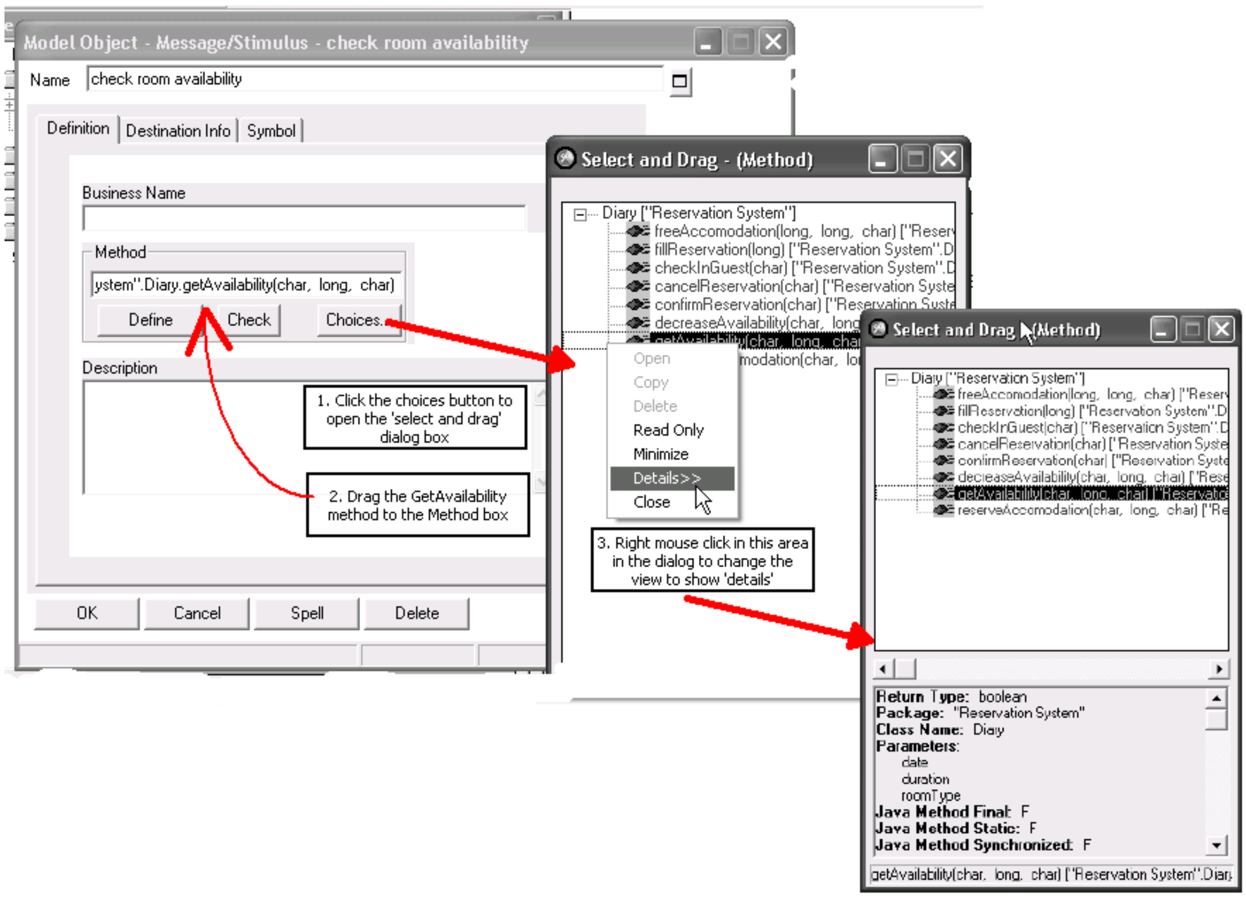
When one object sends a message to another object, it is in essence asking it to do something. The object receiving the message must be able to perform this task and return an answer to the sending object. The receiving class performs an operation - it uses a specific method to perform this operation. This method can be represented in a programming language such as Java or C++ or Visual Basic, etc.

On the Sequence diagram, you can specify what method is 'invoked' by the sending of a message from one object to another. The method invoked belongs to the class that the receiving object instantiates. In System Architect you may specify what method is associated with each message within the definition of the line. This method should become part of the 'to' class definition, or the definition of a superclass of the 'to' class.

At an early stage of analysis, you would normally not spend too much time figuring out the methods invoked; this would be performed at a later stage of design, when we would return to this diagram. For time's sake, in this tutorial, we will add methods at this point.

Let's model the following: when **Reception** sends the message **Check Room Availability** to the object **Diary**, the method **getAvailability** is invoked. To reserve the accommodation, we need a start date, the duration of the stay, and the room type. These will be parameters of the method.

1. Open the definition of the message line **Check Room Availability** that is sent from the object **Reception** to the object **Diary**. (To open the definition, double-click on the line or right-mouse click on it and select **Edit**.)
2. In the definition dialog, select the **Choices** button for the **Methods** property. The **Select and Drag** dialog will open, providing you with a list of all methods of the target class, **Diary**. Notice these are only methods for the class **Diary** (the System Architect notation is to put the method name (with parameters in parenthesis), a period, and the class name.
3. Select the method **getAvailability(Date, Duration, roomType).Diary** and, holding your left mouse button down, drag it into the **Methods** field. Close the **Select and Drag** dialog.


Model Object - Message/Stimulus - check room availability
Name
check room availability
Definition
Destination Info
Symbol
Business Name
Method
ystem".Diary.getAvailability(char, long, char)
Define
Check
Choices..
Description
OK
Cancel
Spell
Delete
Select and Drag - (Method)
Diary ["Reservation System"]
freeAccomodation(long, long, char) ["Reser
fillReservation(long) ["Reservation System".D
checkInGuest(char) ["Reservation System".D
cancelReservation(char) ["Reservation Syste
confirmReservation(char) ["Reservation Syste
decreaseAvailability(char, long
getAvailability(char, long, char
modation(char, lo
Open
Copy
Delete
Read Only
Minimize
Details>>
Close
Select and Drag - (Method)
Diary ["Reservation System"]
freeAccomodation(long, long, char) ["Reser
fillReservation(long) ["Reservation System".D
checkInGuest(char) ["Reservation System".D
cancelReservation(char) ["Reservation Syste
confirmReservation(char) ["Reservation Syste
decreaseAvailability(char, long, char) ["Rese
getAvailability(char, long, char) ["Reservatio
reserveAccomodation(char, long, char) ["Re
getAvailability(char, long, char) ["Reservation System".Diar

1. Click the choices button to open the 'select and drag' dialog box

2. Drag the GetAvailability method to the Method box

3. Right mouse click in this area in the dialog to change the view to show 'details'

Return Type: boolean
Package: "Reservation System"
Class Name: Diary
Parameters:
date
duration
roomType
Java Method Final: F
Java Method Static: F
Java Method Synchronized: F

4. Once you drag the method into the Operations field, you may optionally click on the **Define** button to view the full method definition.
5. Click **OK** to close the **Message** line definition dialog. You will notice that the name of the method has been added to the diagram, under the Message line.
6. Let's turn off the display of the method name -- right-mouse click on the line and select **Display Mode** to open the **Display Mode** dialog. Toggle off **Method** and click **OK**.

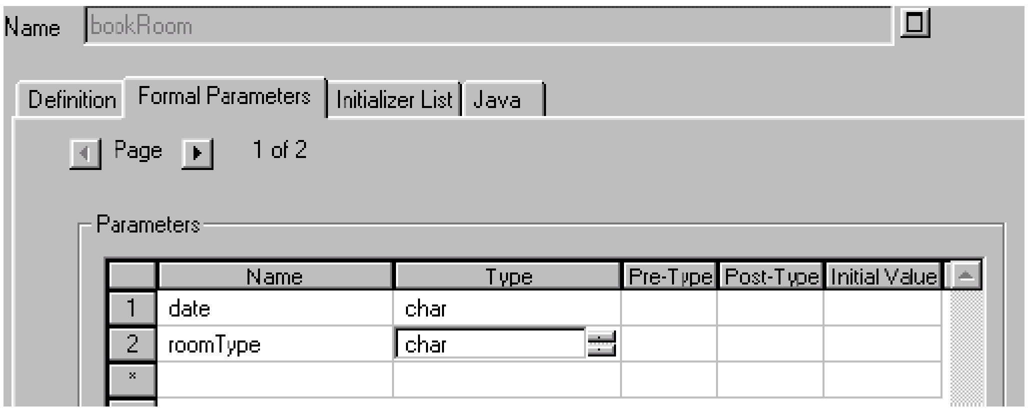
1.9. Adding a New Method to a Class

Next, let's add the method invoked by the message **Provisionally Book Room** that is sent from **Reception** to **Diary**. The method will request details (possess parameters) about the date and the room type. Let's first add this method to the **Diary** class:

1. Open the definition of the **Diary** object lifeline (right-mouse click and select **Edit** or double-click on the symbol).
2. In the **Definition** tab of the definition dialog, click on **Define** next to **Class - (Class) Diary** to open the class's definition.
3. Within the class **Diary**'s definition dialog, click on the **Methods** tab. You will notice there are already a number of operations defined for **Diary**, but no method to book a room. We will add a new method.
4. Cursor to the bottom of the **Methods** grid and type in **bookRoom** in the **Name** cell. Click the **Define** button. This opens the methods definition dialog box entitled 'Dictionary object – method – bookroom'.
5. Within this dialog box, that is method's definition dialog box, select the **Formal Parameters** tab. In the **Parameters** grid, type in **date** and press **Enter**. In the **Type** cell select **char**.
6. Type in a second parameter, **roomType**, and specify the type as **char**.

7. Click **OK** to close the method definition dialog, and **OK** again to close the class definition dialog. And again to close the object definition dialog, by which time you will have closed all the dialog boxes.

.



Name: bookRoom

Definition | Formal Parameters | Initializer List | Java

Page 1 of 2

Parameters

	Name	Type	Pre-Type	Post-Type	Initial Value
1	date	char			
2	roomType	char			
*					

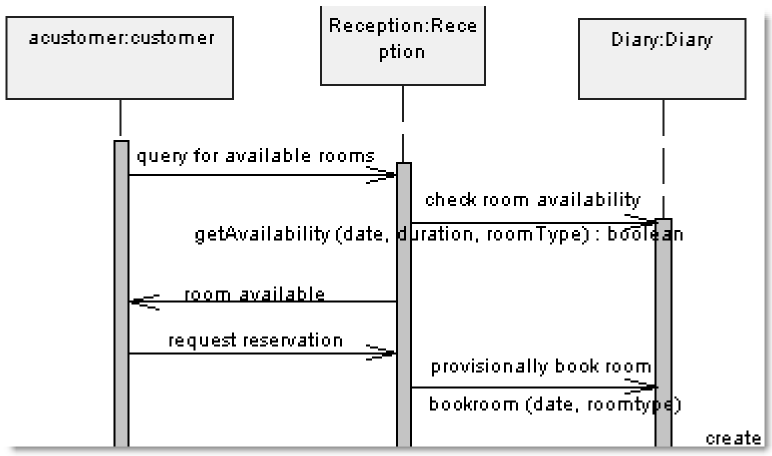
8. Open the definition of the message line **Provisionally Book Room** that is sent from the object **Reception** to the object **Diary**.

9. Click on the **Choices** button for the **Methods** property. The **Select and Drag** dialog will open, providing you with a list of all methods of the target class, **Diary**.

10. Select the method **bookRoom(date, roomType)."Reservation System".Diary** and, holding your left mouse button down, drag it into the **Methods** box. Close the **Select and Drag** dialog and click **OK** to close the other dialog box.

You may want to see what all the above work has actually achieved to get some idea you can turn on again the display of methods on message lines.

You do this by right-mouse clicking on any message line, select Display Mode to open the Display Mode dialog and Toggle on Method and click OK. The result is shown below.



3. Collaboration Diagram

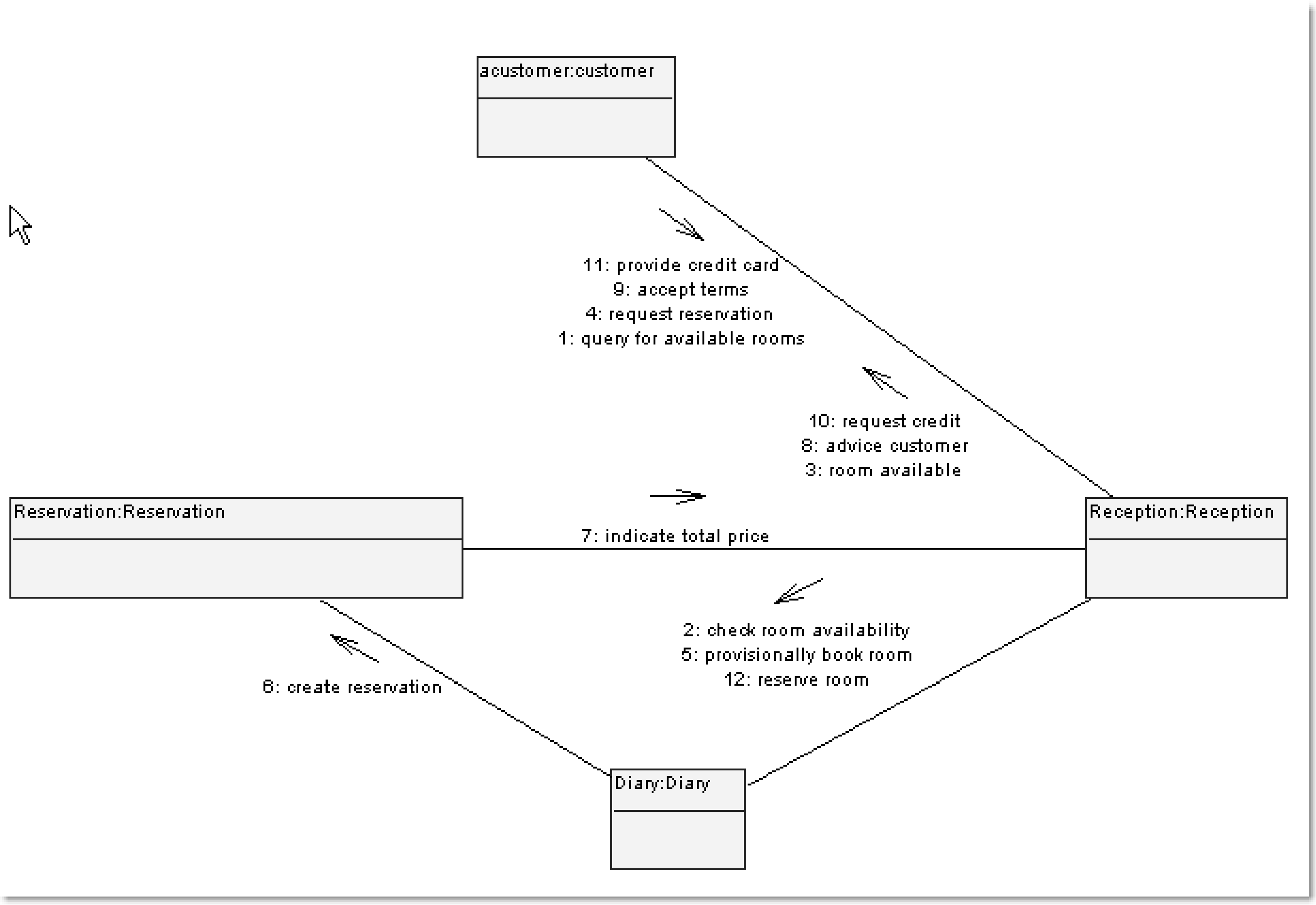
The completion time of this section is approximately 15 minutes.

1.10. Create Collaboration Diagram

The Collaboration diagram is an alternative to the Sequence diagram for modeling the scenarios in your system. Although Sequence diagrams make it easier to see the chronological flow of events, collaboration diagrams enable you to show links between objects, attribute values, and visibility.

Let's have System Architect automatically build a Collaboration diagram for us.

1. With the Sequence diagram open and in focus, select **Draw, Synchronize Diagram**. System Architect will build a Collaboration diagram and, once completed, presents you with a message, **The Sequence and Collaboration diagrams have been synchronized**. Click **OK**. The diagram should look similar to the one below. I have moves the various objects around to make the picture clearer.

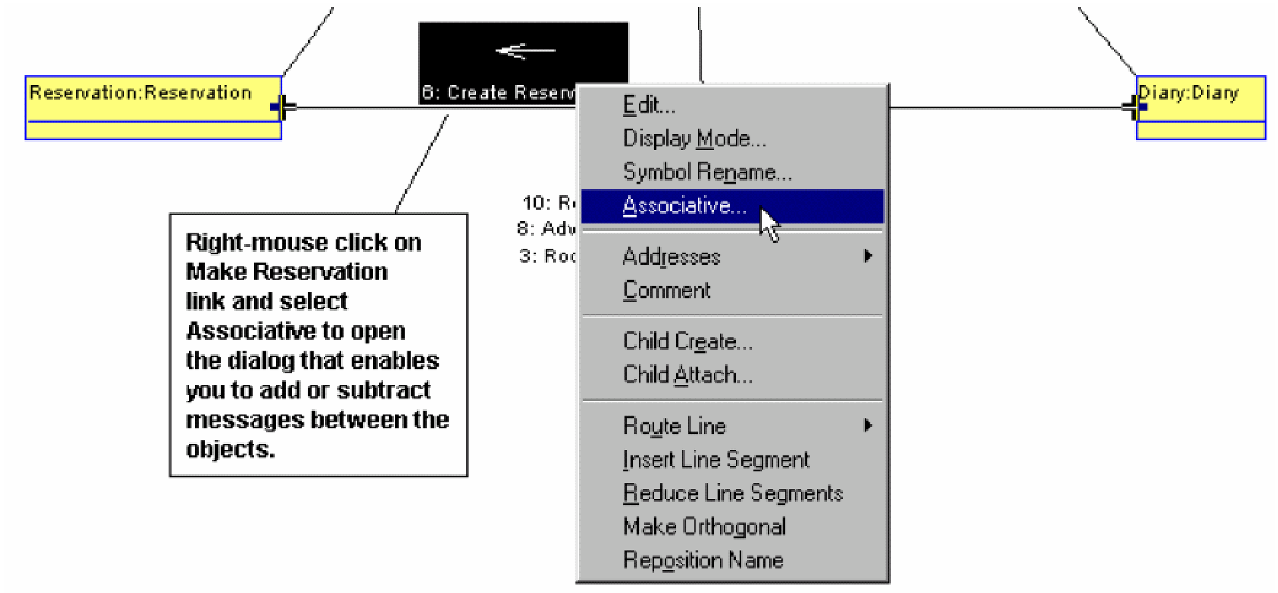


I find these diagrams useful in visualising what I call informally ‘foci’ for example from the above you can clearly see the importance that the Reception plays and possibly how you might want to either accept this situation or think of ways of reducing it’s importance?

1.11. Add a Message to a Link

Let's add a new step, modelled as a message to this Collaboration diagram and its sister Sequence diagram.

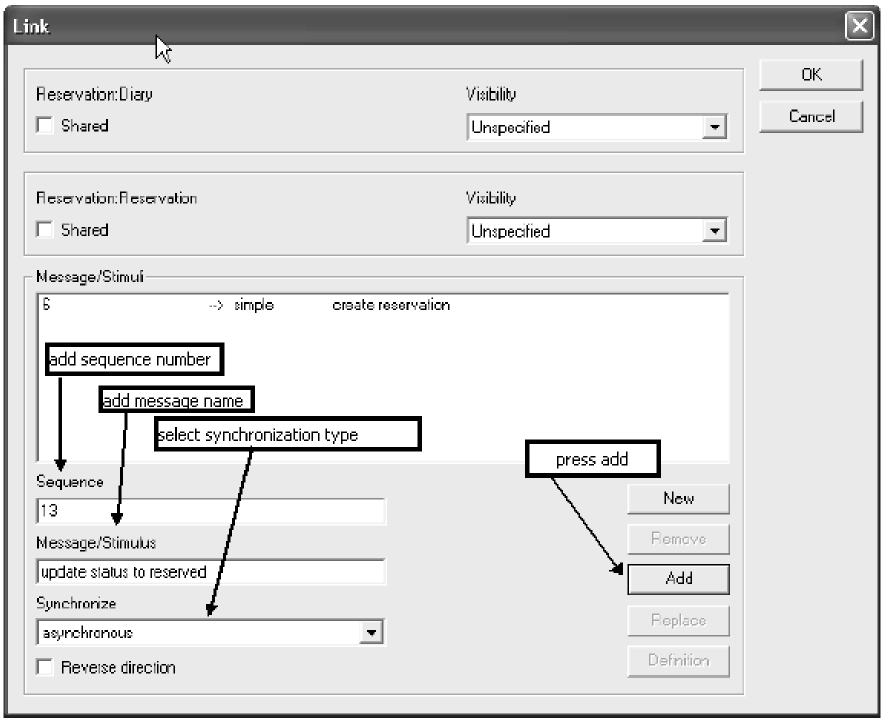
1. Right-mouse click on the **link** line between **Diary** and **Reservation** and select **Associative...** from the drop-down menu.



Right-mouse click on Make Reservation link and select Associative to open the dialog that enables you to add or subtract messages between the objects.

2. The **Link** dialog opens. Press the **New** button and in the **Sequence** box, enter **13** (this is the next step on the diagram).
3. Type in **Update Status to Reserved** in the **Message/Stimuli** box.
4. From the **Synchronize** drop-down list box, select **Asynchronous** as the synchronization type. This is an asynchronous message -- it could happen at any time; opposite of this would be a synchronous message, which happens a preset time after the previous step.
5. Click on the **Add** button.
6. Click **OK** to close the dialog.

This is important remember it! RB



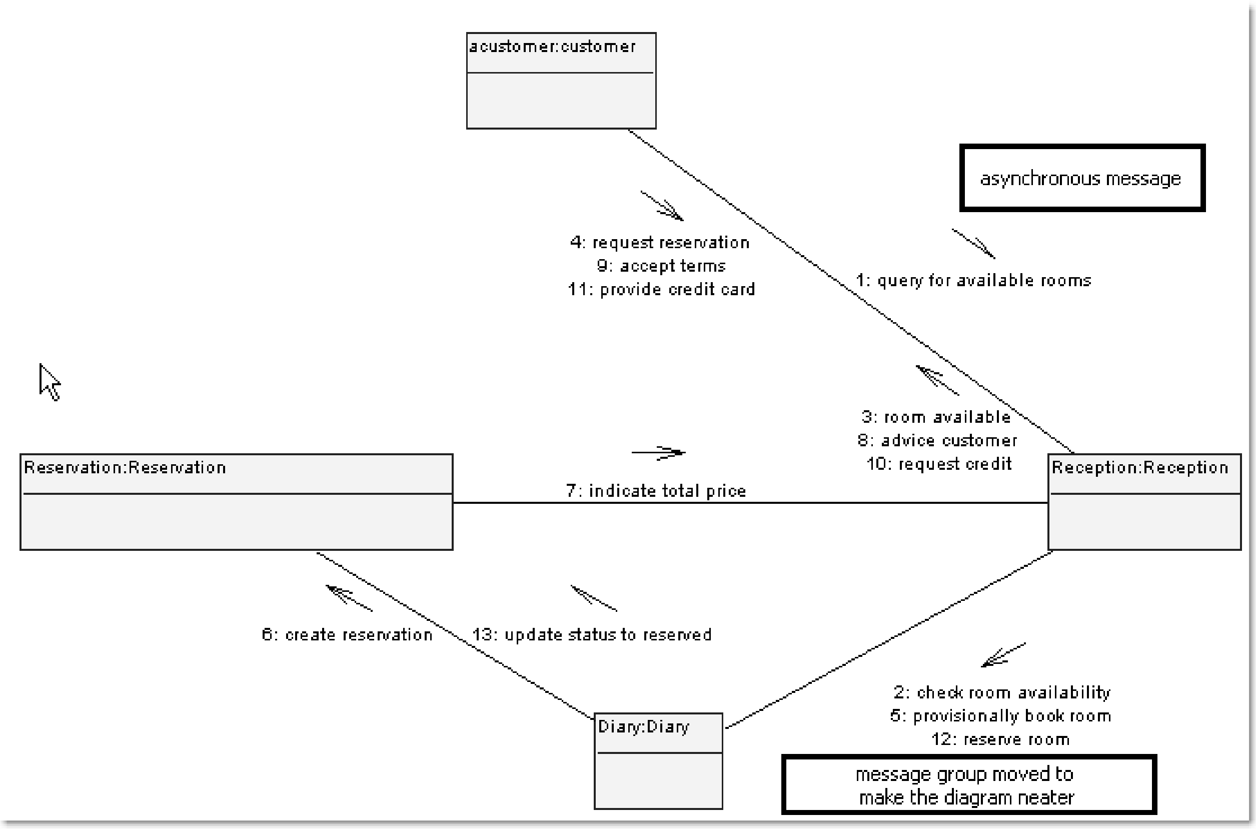
View the diagram and note the changes that you have made in the Collaboration diagram -- a new message appears on the link between **Diary** and **Reservation**. The asynchronous message **Update Status to Reserved** has a symbol next to it that appears as an arrow with a half arrowhead. This is the UML notation for an asynchronous message.

1.12. Modify a Message

To modify a message attached to a link:

1. Right-mouse click on the **link** line between **Reception** and **Customer** and select **Associative...** from the drop-down menu to open the **Link** dialog.
2. Click on Message/Stimuli **1, Query for Available Rooms**, in the **Message/Stimuli** list box.
3. From the **Synchronize** drop down list box, select **Asynchronous**.
4. Click **Replace**.
5. Click **OK** to close the dialog.

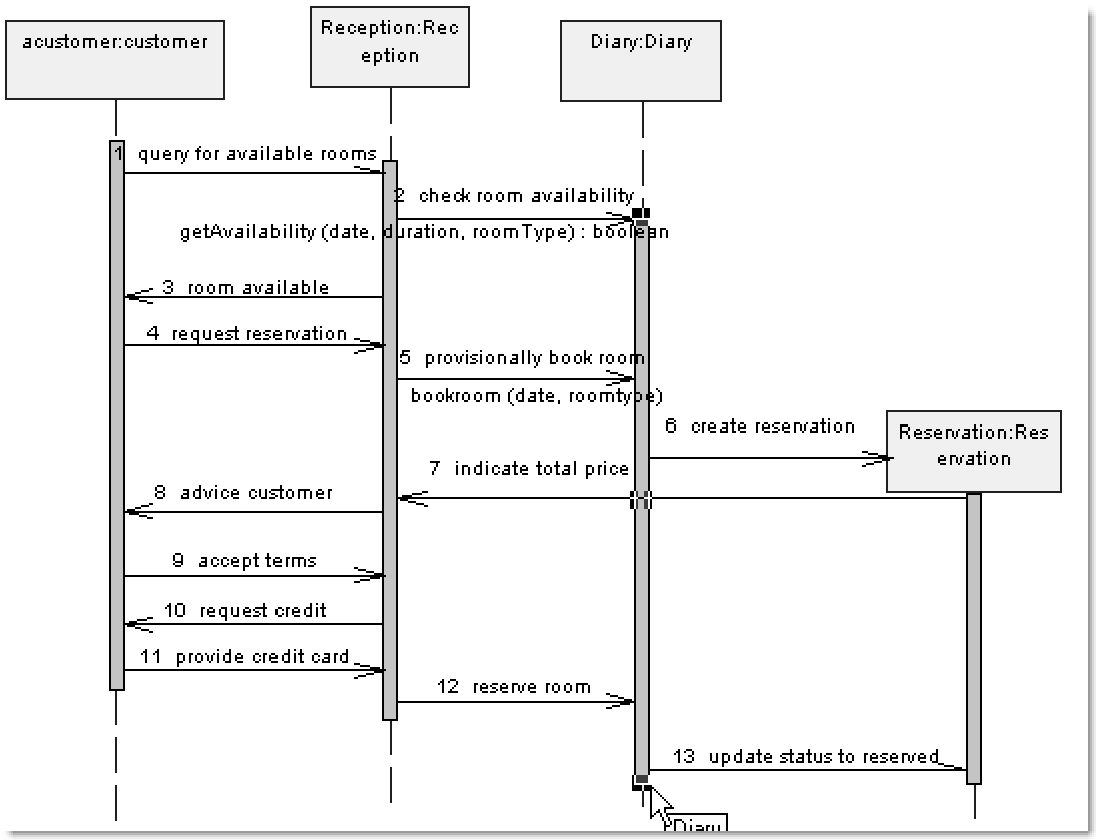
Note: that on the diagram, **Query for Available Rooms** now has an asynchronous arrow next to it. Because the diagram is crowded, you may need to move message groups around to make the diagram neater.



1.13. Synchronize Sequence Diagram

To synchronize the Sequence diagram to include the changes you have made:

1. Select **Draw, Synchronize Diagram**. A dialog box will appear warning you of possible effects – ignore it.
2. Looking at the revised sequence diagram note that the event line **Update Status to Reserved has** been added between the objects **Diary** and **Reservation**, with appropriate synchronization notation. Also the event line for **Query for Available Rooms** has changed to an asynchronous notation. The only thing you need to tidy up is to adjust the length of the activation lines for possibly the Diary and Reservation objects.



Remember to save your work – If you can't remember how to do this have at look at the end of the last tutorial.

END OF TUTURIAL