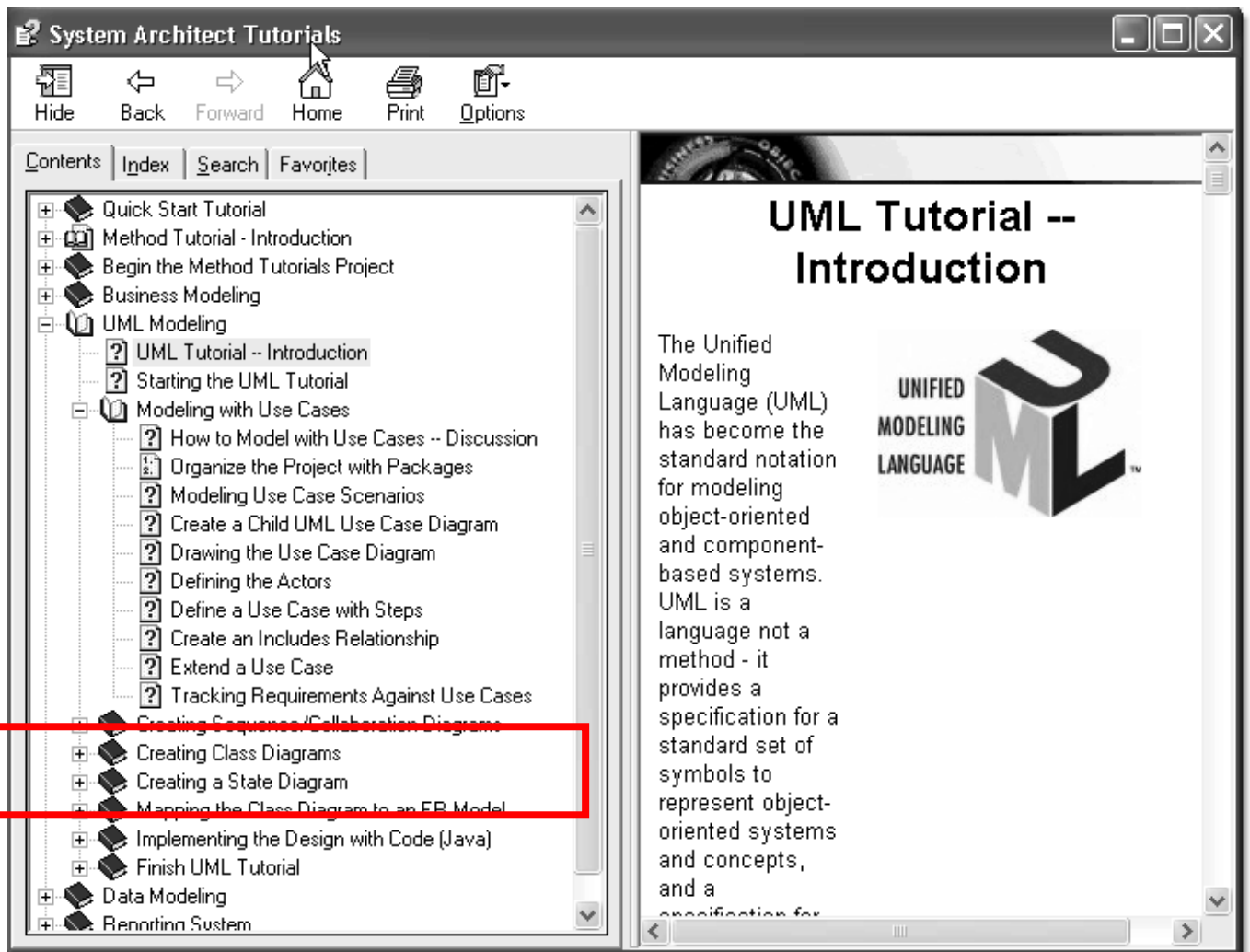


Class and State Diagrams

Robin Beaumont robin@organplayers.co.uk 03/01/2004 15:07

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



Contents

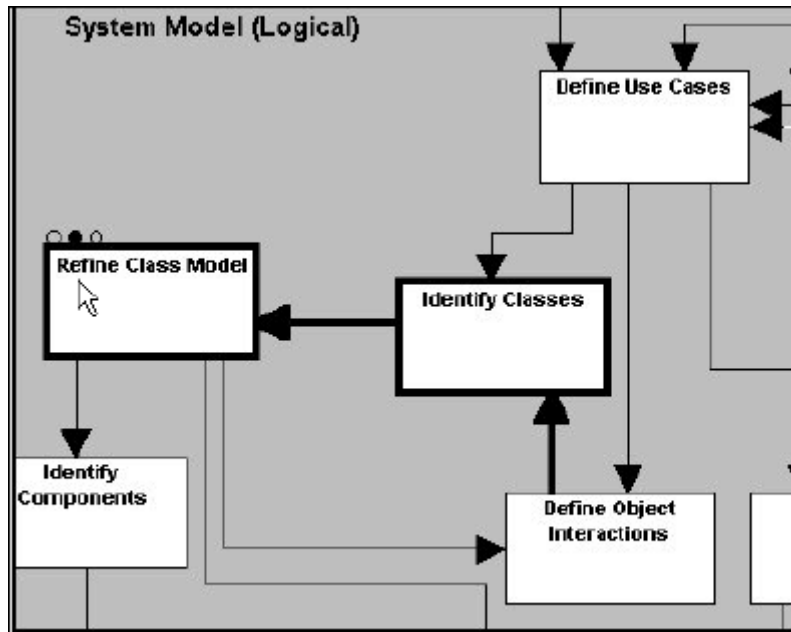
Contents	2
1. Introduction Class and State Diagrams	3
2. Begin a New UML Class Diagram	4
1.1. Building Relationships Between Classes	5
1.2. Defining an Association.....	6
1.3. Adding Associations Between Other Classes.....	8
1.4. Building Inheritance.....	9
1.5. Cross Referencing Class and Sequence Diagrams.....	11
3. Create a UML State Diagram	13
1.6. Determine the States and Transitions for a Class.....	14
1.7. Drawing the State Diagram	15
1.8. Defining a State.....	17
4. Mapping Class Diagram to ER Model.....	18
5. UML Class Design	20

Comments I have added are in red RB

1. Introduction Class and State Diagrams

The completion time of this section is approximately 50 minutes.

The Class diagram is the main analysis/design diagram for a system. In it, you describe the structure of classes in the system - their identity, relationships to other classes, attributes, and operations. Class diagrams model static behavior. 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:



Iterative Design

The class diagram is developed in an iterative fashion, in other words, through a repeated cycle of analysis, design, and implementation.

Often issues that affect the design arise during implementation. Changes made to implementation code need to be reversed back into the analysis model so that iterative design can continue. System Architect facilitates this process by enabling you to implement the design in C++, Java, and Visual Basic, and then reverse that code back into the existing class diagram, automatically updating the information stored on the diagram and in the underlying encyclopedia.

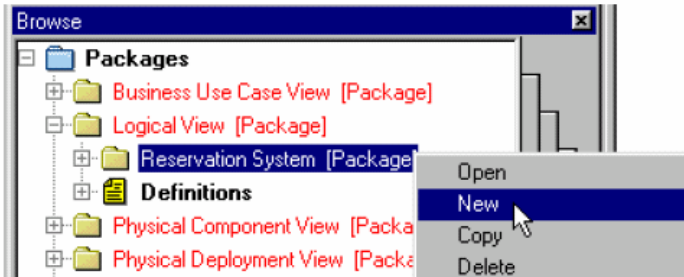
In the following exercises, you will

1. Build a very simple class diagram for the hotel reservations system, using classes found through Use Case and Sequence diagram analysis,
2. Add some new classes and design details found through brainstorming,
3. Perform a design check of the Class diagram against the previously modeled Sequence diagrams,
4. Analyze one of the important classes in the application by building a State diagram for it,
5. Map the persistent classes to an Entity Relation diagram to begin the design of the relational database affiliated with the application, and
6. Implement the partial design of the application by generating Java code.

2. Begin a New UML Class Diagram

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

We will start by creating a new class diagram.



1. Within the **UML** tab of the browser, select the Logical View, Reservation System package, right-mouse click on it and select **New** from the drop-down menu. The **Select new type UML** dialog appears.
2. Under the **Diagrams** list, double-click on **Class** to create a new Class diagram.
3. In the **New Diagram** dialog, name the diagram

Hotel Reservation System. Click **OK**. A dialog will present you with the package that this diagram is contained in. Leave it at **Reservation System** and click **OK**.

Reuse Existing Classes

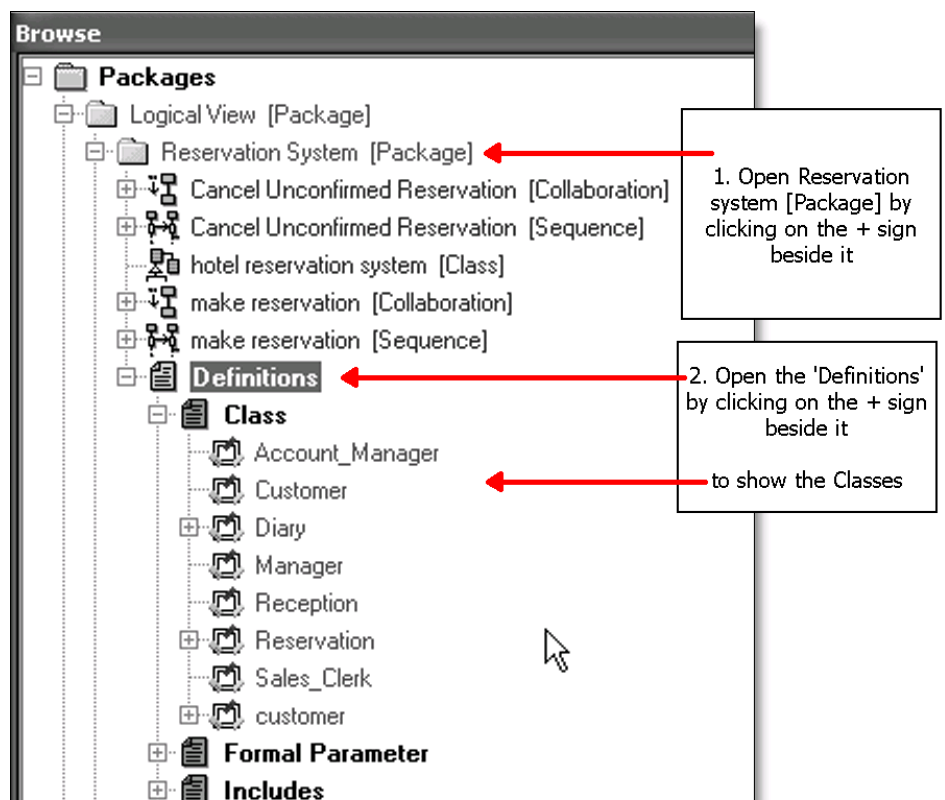
At this point we will reuse the classes that we have already established in the Sequence diagram that we've built within the Reservation System package (and some other classes that already existed in this **Tutorial** encyclopedia). To place existing classes on the new class diagram:

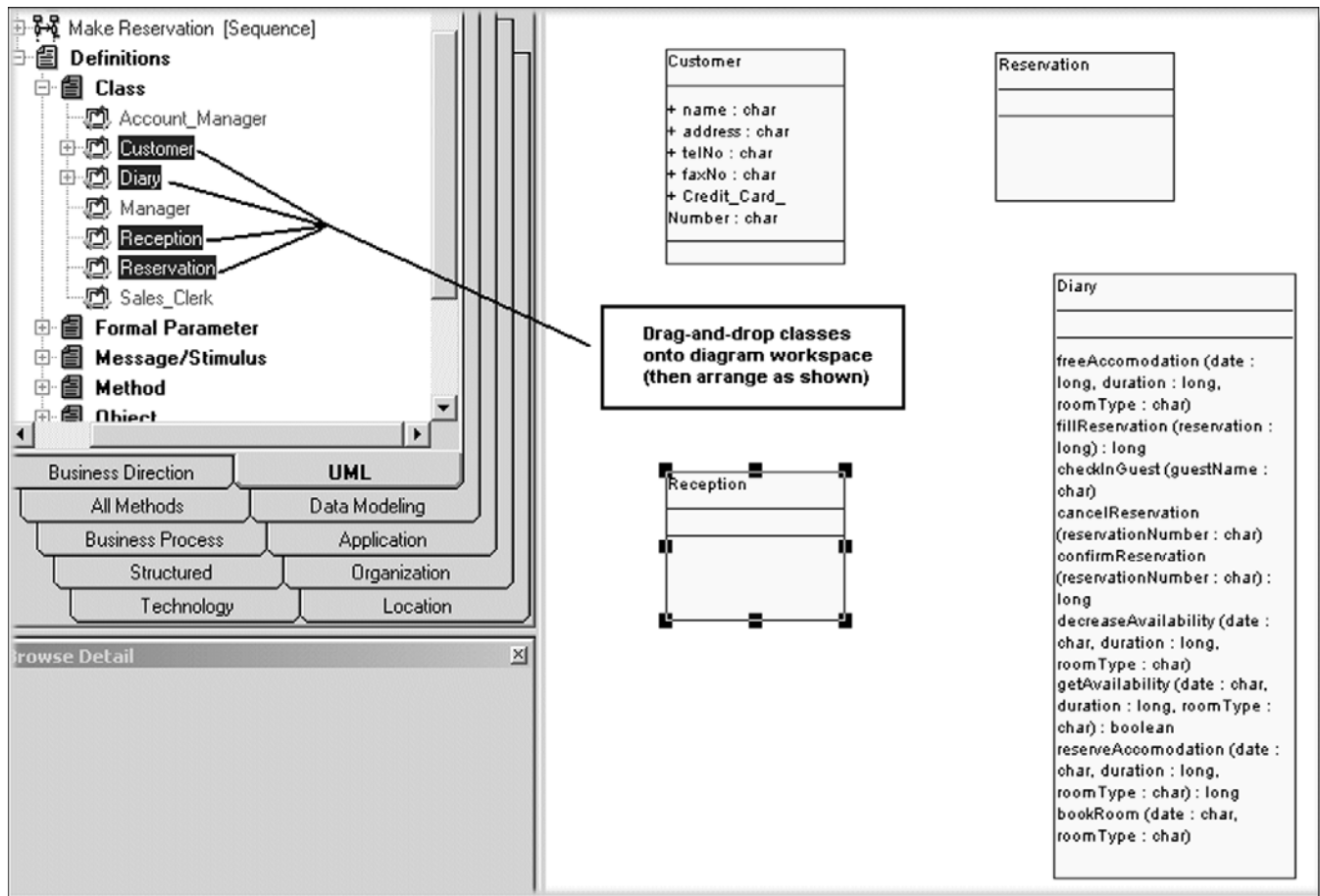
4. With the **UML** tab still selected in the browser, **open the Reservations System package, by clicking on the + mark beside it**. This will show the **Definitions** that is all the UML definition types that have definitions in this package.

5. Click the **+** mark by **Class** to reveal the classes already defined in the tutorial encyclopedia.

6. Hold your **Control** key down and multi-select the classes **Reservation, Reception, Diary,** and **Customer** and drag them onto the desktop of the open diagram. The classes are automatically drawn.

7. Arrange the classes as shown in the diagram below. Add space in between the classes to facilitate the drawing of associations between them in the next section.





1.1. Building Relationships Between Classes

In the UML Sequence diagram, we already established that some of our objects are sending messages to one another. Within the scope of the class diagram, there should be respective associations between the classes that these objects instantiate. At this point we will want to graphically show that these relationships exist. Later, depending on the implementation language, we will want to specify the nature of these relationships (for example, in C++ a class can reference another class, or point to it; in Java, no such notion as 'pointing' exists, but a class can reference another). **Ignore the sentences about implementation languages we will not be travelling that far down the path!**

In the Sequence diagram we modeled that **Diary creates Reservation**. It might also *cancel* it (we would model that in another Sequence diagram) or *update its status*. We can say Diary **Maintains** Reservation. **We will now model this.**

Select Straight – Orthogonal Line Style

SA offers a number of line styles to select when drawing a line. For this exercise, we will draw using the Straight -- Orthogonal choice. We will switch later to **Straight -- Any Orientation**.

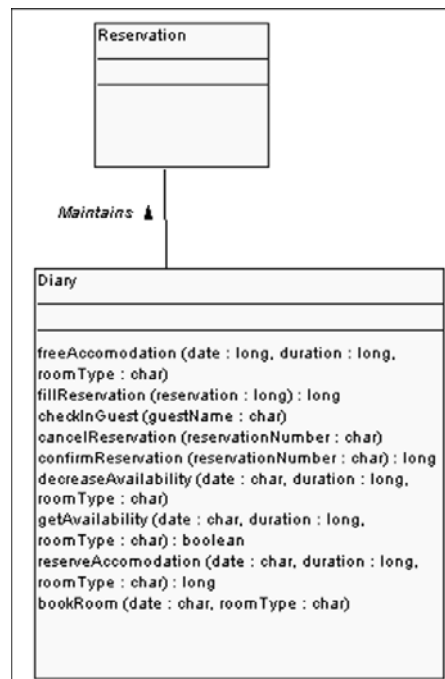
1. Select **Format, Symbol Format, Line**.
2. In the **Line Style** dialog, toggle on **Straight – orthogonal** and click **OK** to close the dialog.

Add an Association Between Classes

To add an association to the diagram:



1. Select the **Association** line tool from the toolbar, and draw a line **from** the class **Diary** **to** the class **Reservation** (it is important to draw it in this order). Remember, you must start with a plus line (+) on the first symbol, release your mouse, and end with a plus line on the second symbol. Name the line **Maintains** and press **Enter**. **Enter Select mode now by pressing Esc or clicking on the Select mode icon on the toolbar (far left)**
2. Move the **Maintains** name to the right so that it is not covering the line.



1.2. Defining an Association

Most of the interesting information about an association is attached to its ends. (The association end is part of the association, not part of the class.) An association between two classes contains two association ends. A joint association between three classes (referred to as a ternary association) contains three association ends, etc.

1. Open the definition dialog of the association **Maintains** between **Reservation** and **Diary** (double-click on the line or right-mouse click on it and choose **Edit**).

The definition of an association contains a grid of the **class roles** that the association is involved in. Each row in the grid represents an **Association End**.

Role Names

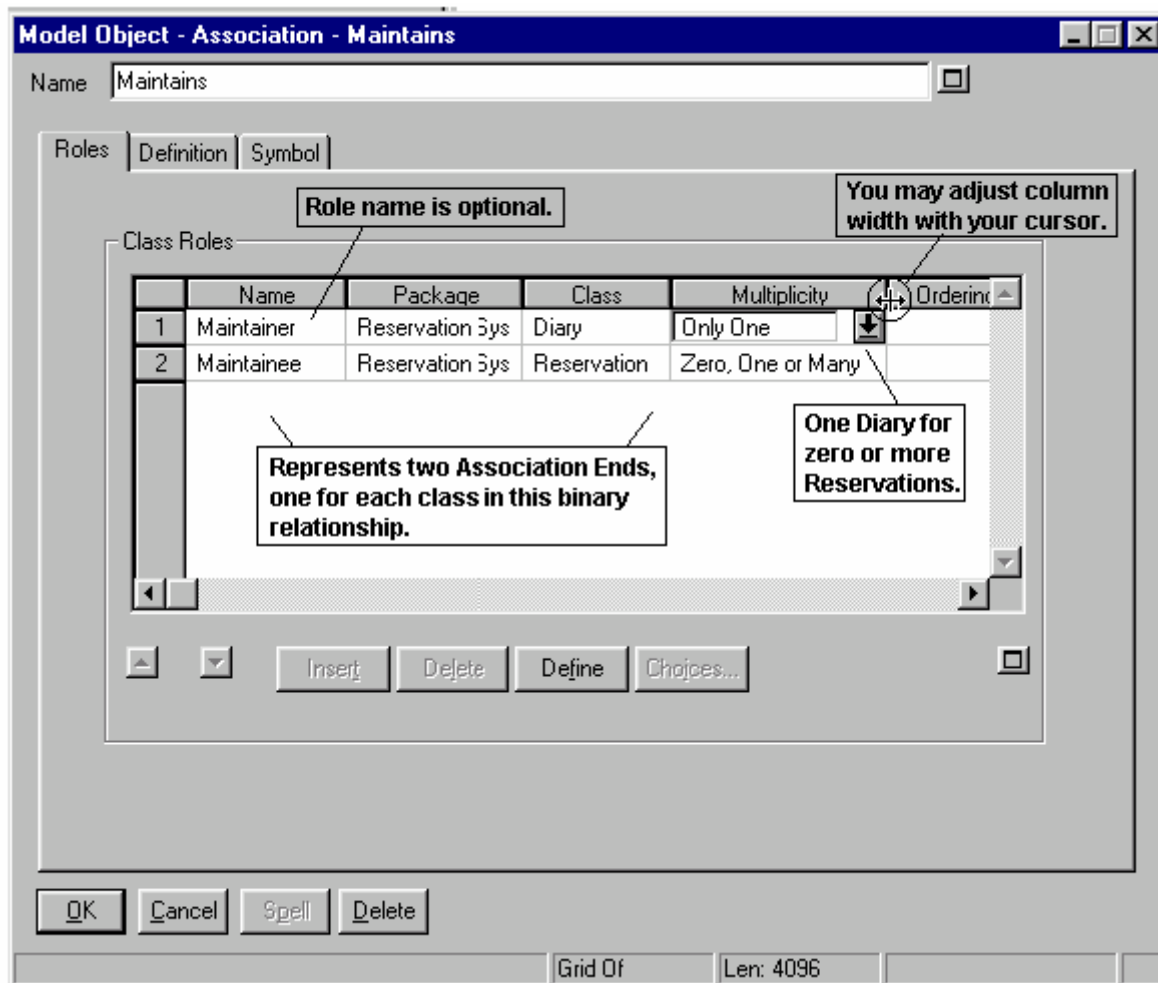
The name of the association end is the name of the **Role** played by that the class for that end. For example, if you had an association **Married** between the class **Man** and the class **Woman**, the role name on the Man side would be **Husband** and the role name on the Woman side would be **Wife**. Naming the association end is optional; the name will appear on the association at the end of the class it specifies.

- Place your cursor in the **Name** cell for the **Diary** class and type in **Maintainer** -- this is the role that the **Diary** class plays in this association.
- Place your cursor in the **Name** cell for the **Reservation** class and type in **Maintainee** -- this is the role that the **Reservation** class plays in this association.

Specifying Multiplicity

A *single Diary* maintains *zero or more Reservations* (zero because it might be possible that at a point in time there are no reservations, although this is unlikely). These numbers are considered the multiplicity of the association.

- Place your cursor in the **Multiplicity** grid for **Diary**, and select **Only One** from the drop-down list (you may adjust the column width with your cursor). Select **Zero, One or Many** as the multiplicity for the **Reservation** side of the association.



Specifying a Qualifier

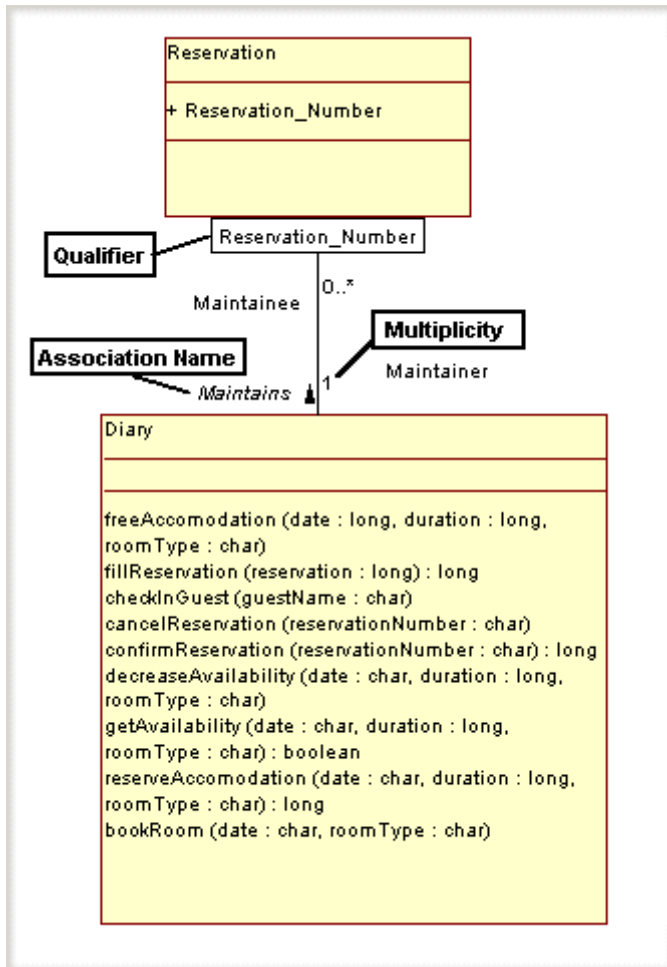
A qualifier narrows down the multiplicity of an association. For example, the relationship between classes **Diary** and **Reservation** has a multiplicity of one-to-zero-or-more, in other words, one **Diary** maintains many **Reservations**. The addition of a qualifier such as **Reservation_Number** reduces the multiplicity to one-to-one, in other words, a **Diary** maintains one **Reservation_Number**.

Put your cursor in the **Maintainee** Name cell of the **Reservation** association end, and click on the **Define** button at the bottom of the grid to open the Association End's full definition dialog.

- Click on the **Qualifier** tab.
- Click on the **Choices** button in the **Qualifier - (Class Attribute)** list box.

7. Drag and drop the attribute **Reservation_Number** into the list box.
8. Click **OK** to close the **Association End** definition dialog. **This also automatically closes the drag and drop choice dialog.**
9. Click **OK** to close the **Association** definition dialog.

Notice that on the diagram, the qualifier appears in a box attached to the Reservation class. You may adjust the positioning of the association name, the role names, and the multiplicities. Notice that System Architect prevents you from moving the role name and multiplicity to the side of the other class. It can only reside on the side of the class it represents.



1.3. Adding Associations Between Other Classes

As we also modeled in the Sequence diagram, **Reception** uses **Diary** to gain information for the customer who wants to make a reservation. The association is a one-to-one relationship: there is **one** *Reception* class that deals with **one** *Diary* class.

In addition, **each** *Customer* makes **one-or-more** *Reservations*.

Let's model these associations:

1. Draw an association line from **Reception** to **Diary**, and name it **uses**.
2. Draw an association line from **Customer** to **Reservation**, and name it **makes**.
3. Get out of drawing mode by hitting your **Esc** key or choosing the **Select Mode** pointer in the toolbox.

4. Open the definition of the **uses** association between **Reception** and **Diary** and set the multiplicity to **Only One** for the **Reception** class, and **Only One** for the **Diary** class. **Click the OK button to save your changes.**
5. Open the definition of the **makes** association between **Customer** and **Reservation** and set the multiplicity to **Only One** for **Customer**, and **One or Many** for **Reservation**. **Click the OK button to save your changes.**

1.4. Building Inheritance

In our system, we will have various types of customers. In the beginning of this tutorial, we analyzed the problems of the business. We stated that we would be concentrating on business travelers and vacationers. These two classes of customer will have different enough behaviors that we may want to introduce them into our design as separate classes that inherit from the basic Customer class. For example, although a Business Traveler has a name, address, telephone number, fax number, and credit card number that a standard customer has, the business traveler also has a corporate discount, and may be guaranteed a room if their company is involved in hosting a conference in the hotel or at an affiliated conference hall. A vacationer may partake in vacation package discounts.

Note: SA offers a number of line styles to select when drawing a line. We have been using the **Straight -- Orthogonal** choice. We will now switch to **Straight -- Any Orientation**.

How to Select Straight – Any Orientation Style

1. Select **Format, Symbol Format, Line**.
2. In the **Line Style** dialog, toggle on **Straight – any orientation** and click **OK** to close the dialog.

Model the Subclasses

To model the subclasses Business Traveler and Vacationer on the class diagram:

1. Select the **Class** tool from the toolbar and add a new class to the diagram, placing it under the **Customer** class. Name it **Business_Traveler**. **This can be a problem if you have accidentally duplicated a class by mistyping the name or missing out a capital or putting one in when defining a class always look through the choices dialog before typing in a name for a new class to avoid problems latter on with adding inheritance lines.**
2. Select the **Inherits From** line from the toolbar and draw an inheritance relationship from **Business_Traveler** to **Customer**. **If you are unable to do this see the above step?**
3. Draw a second class on the diagram, placing it under **Customer**, and name it **Vacationer**.
4. Select the **Inherits From** line from the toolbar and draw an inheritance relationship from **Vacationer** to **Customer**.

The existing attributes and methods for Customer do not need to be repeated in the class Business_Traveler or Vacationer. The inheritance line semantically specifies that these 'subclasses' inherit all attributes and methods of the 'parent' class, plus have their own attributes and methods.

5. Open the definition dialog for **Business_Traveler** (double-click on the symbol or right-mouse click on it and select **Edit**), select the **Attributes** tab and type in **Corporate_Discount** as a new attribute. In the **Type** column, select **char**. Click **Ok** to close the dialog.
6. Open the definition dialog for **Vacationer**, select the **Attributes** tab and type in **Vacation_Package** as a new attribute. In the **Type** column, select **char**. Click **Ok** to close the dialog.

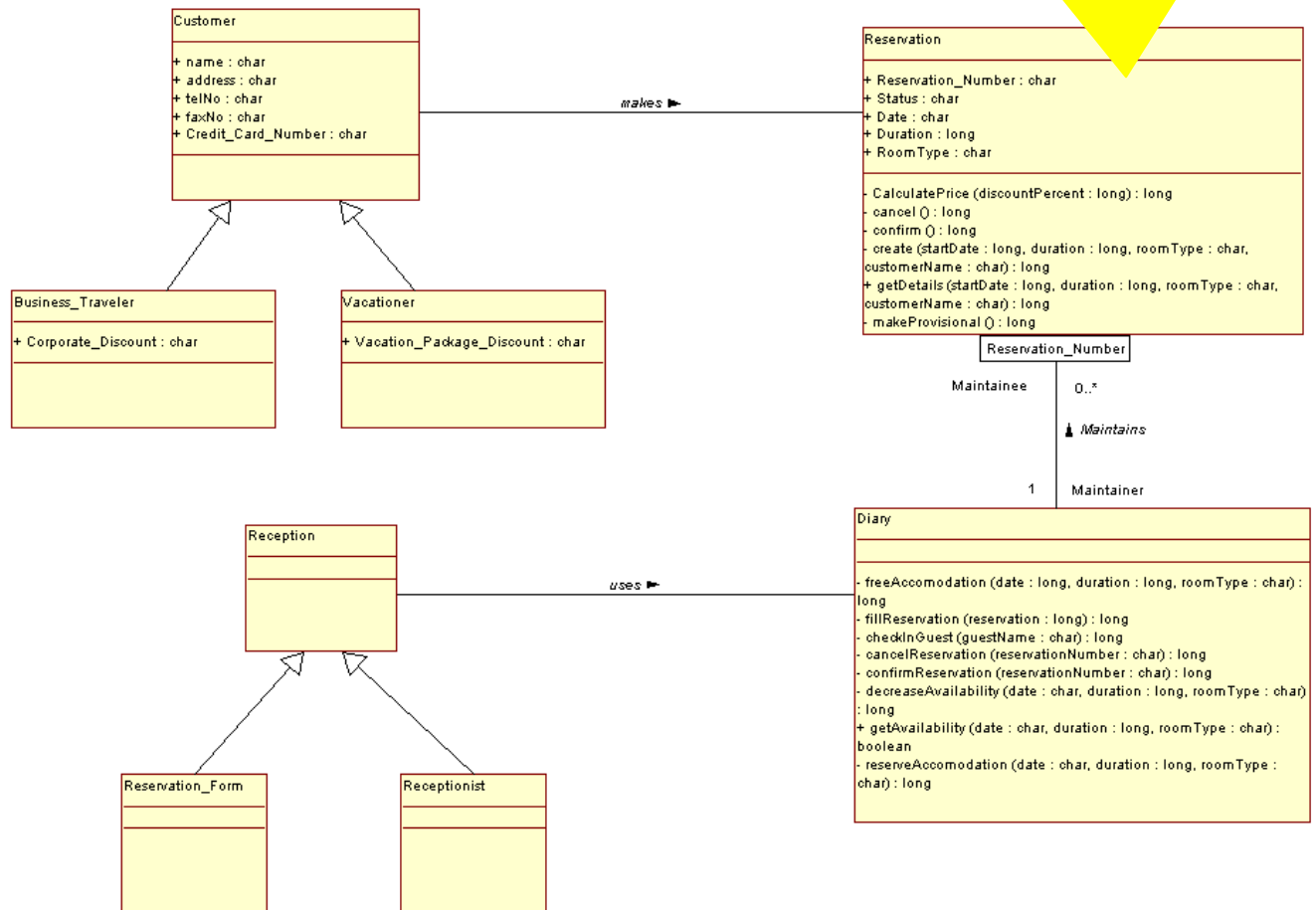
Building a Second Inheritance Structure

Now let's take a look at the Reception class - this is a class that will interface with the end user, or Customer. In our system, Reception may well be a Receptionist that answers the phone at the front desk of a hotel, or at the Hotel's call center at its 1-800 number. Or, it could be a Reservation Order form on the hotel's website. These various forms of reception should be modeled with different classes, since their

behavior and capabilities will be different, although they perform the same basic functions, to interface and get information from the customer.

7. Select the class tool from the toolbar and add a new class to the diagram, placing it under **Reception**. Name it **Reservation_Form**.
8. Select the **Inherits From** line from the toolbar and draw an inheritance relationship from **Reservation_Form** to **Reception**.
9. Draw a second class on the diagram, also placing it under **Reception**. Name it **Receptionist**.
10. Select the **Inherits From** line from the toolbar and draw an inheritance relationship from **Receptionist** to **Reception**.

Notice that the reservation class has a set attributes and methods now. This will only be the case if you did this and saved the previous tutorials including the ERD one!



We have completed our first-cut class diagram.

11. Select **File, Save Diagram** to save the class diagram.

1.5. Cross Referencing Class and Sequence Diagrams

The Student Version of System Architect does not allow you to generate SA Word Reports. You may only do this with System Architect's commercial version.

At this point we want to make sure that every message modeled between objects on a Sequence diagram has a corresponding association between the respective, instantiated classes, and vice versa. If we were designing a real system, we might have many Sequence diagrams, perhaps hundreds, detailing the scenarios of our system. To make sure these correlations have been made, we can use an automated SA/Word report provided by System Architect. Let's run this report.

Note: you must have Microsoft Word installed on your local machine to run this report.

1. Select **Reports, Word Reports, Object Model Reports, Sequence Class Xref Report**. Microsoft Word should automatically run, and load a special VBA macro that has been installed into your Word template directory upon installation of System Architect. The **UML Sequence Diagram/Class Diagram Interrelationships Report** should show up in your Word document.
2. Cursor down through the report - at the end of the report, you should see two informative tables -- **Messages with no Association on Any Class Diagram**, and **Associations with no Events on Any Sequence Diagram**.

Messages with no Association on Any Class Diagram

Message	From Object/From Class	To Object/To Class	Sequence Diagram
Display Cancellation Details	Diary/Diary	Diary/Diary	Cancel Unconfirmed Reservation
Cancel Reservation	Manager/Manager	Diary/Diary	Cancel Unconfirmed Reservation
Display Reservation	Diary/Diary	Diary/Diary	Cancel Unconfirmed Reservation
Show Unconfirmed Reservations	Manager/Manager	Diary/Diary	Cancel Unconfirmed Reservation
Provide Credit Card	Customer/Customer	Reception/Reception	Make Reservation
Request Credit	Reception/Reception	Customer/Customer	Make Reservation
Accept Terms	Customer/Customer	Reception/Reception	Make Reservation
Advise Customer	Reception/Reception	Customer/Customer	Make Reservation
Indicate Total Price	Reservation/Reservation	Reception/Reception	Make Reservation
Request Reservation	Customer/Customer	Reception/Reception	Make Reservation
Room Available	Reception/Reception	Customer/Customer	Make Reservation
Query for Available Rooms	Customer/Customer	Reception/Reception	Make Reservation

Associations with no Events on Any Sequence Diagram

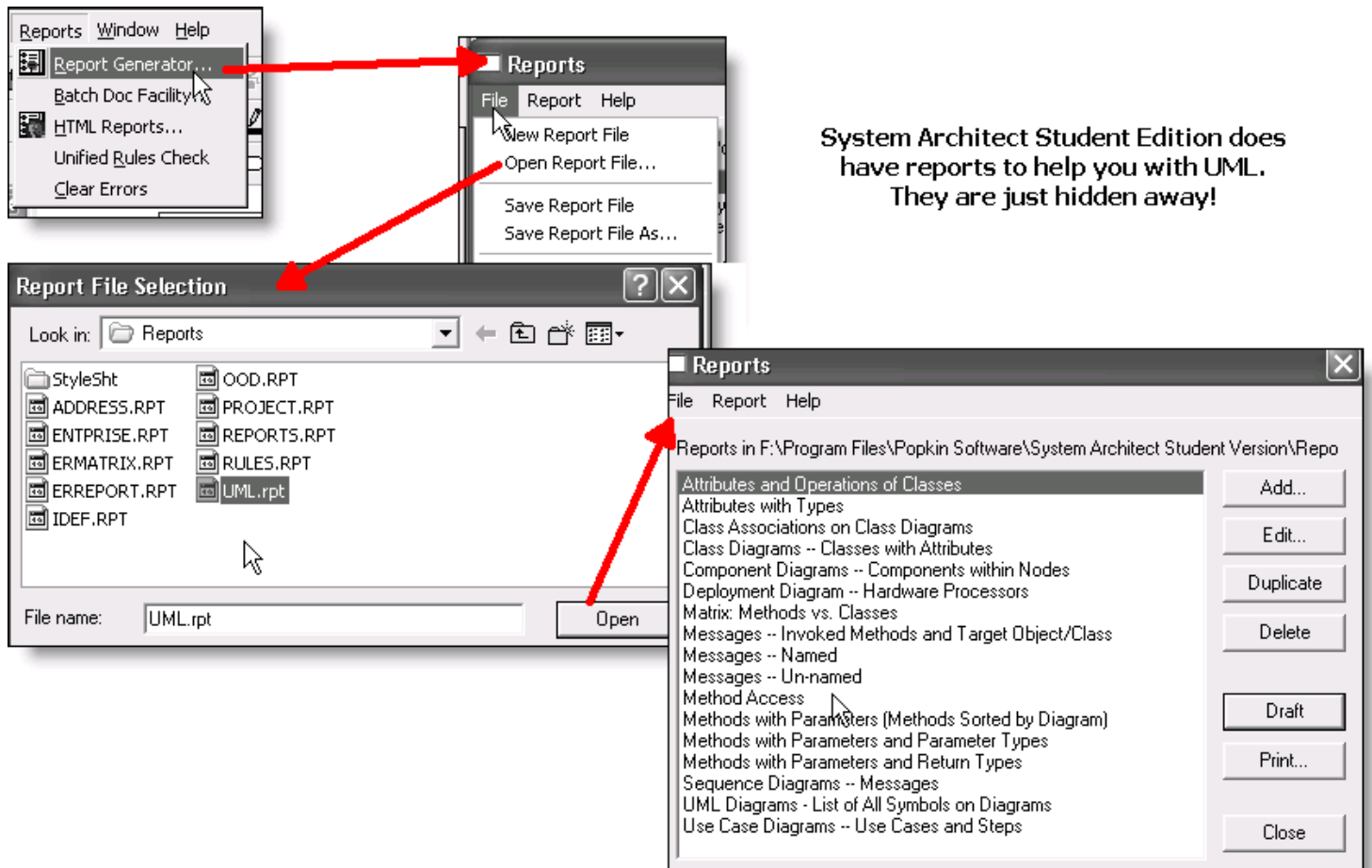
Association	From Class	To Class	Class Diagram
makes	Customer	Reservation	Hotel Reservation System

The first table tells us is that we have not modeled an association between Reception and Customer on the class diagram, even though a Sequence diagram tells us that there is a relationship between corresponding objects that instantiate them. The second table tells us we modeled an association on the class diagram between Customer and Reservation, but no such relationship exists between any two instantiating objects on any Sequence diagram.

In a real project, we would now go back and make appropriate changes to our models to achieve consistency.

3. Close the word report and select **No** when asked to save the document.

While you do not have the facility to create word reports you can still create certain ones. See the diagram below.



3. Create a UML State Diagram

The State diagram shows the events that cause a transition from one state of a class to another state. It is a network of *states* and *events*. It captures the class's received stimuli, responses, and actions. Each state receives one or more events, at which time the class *transitions* to the next state. The next state depends on the current state as well as the events.

Modeling a state diagram is useful for understanding the dynamic behavior of *important* classes in the system. By important, we mean a class that performs important tasks and may change state often based on system or business events. What is also implied is that you do not need to build a state diagram for every class in the system, just ones you'd like to understand better.

For our system, we may not be interested too much about the dynamic behavior of the Reception class -- Reception is an interface class that is basically getting and providing information from/to a Customer and inputting it into the system. However, we may want to take a deeper look at the Reservation class, since it will be in various states -- like provisionally booked, booked, created, destroyed, on hold, etc. -- based on various events. It is important to understand how long it is in these states, and what events make it change state.

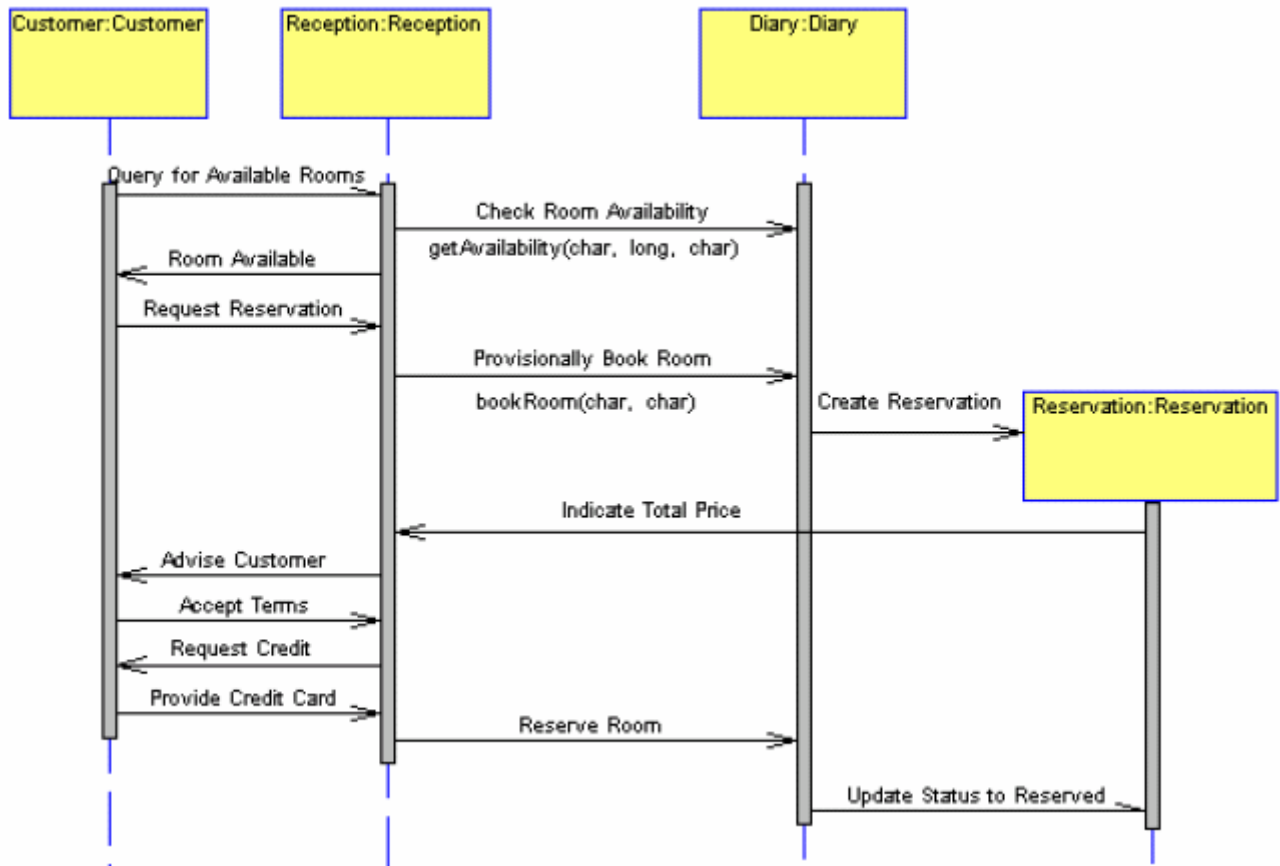
Let's examine the behavior of the class **Reservation**. Let's create a UML State diagram as a child diagram to the **Reservation** class:

1. Open the UML class diagram you have built, **Hotel Reservation System** (if not already open).
2. Right-mouse click on the class **Reservation** and select **Create Child**. The **Child Create** dialog appears.
3. Select **State** as the type and leave its name at the default **Reservation**. Press **OK**.
4. Select **Yes** to the message asking if you want to save changes to the current diagram. The **Diagram - UML State** - dialog appears, with the package (**Reservation System**) and class (**Reservation**) properties already filled in.
5. Press **OK** to create the state diagram.

1.6. Determine the States and Transitions for a Class

As a general rule, you can look at event lines **entering** objects of a class on a Sequence diagram to find **Transitions** for your State diagram. Likewise, you can study the **outgoing** event lines to find actions that the class carries out while in a particular State.

Let's take another look at the Sequence diagram we built named **Make Reservation**.



Note the messages on the Sequence diagram that enter the object **Reservation**. These messages, such as **Create Reservation** and **Update Status to Reserved**, cause transitions that cause the **Reservation** to move from one state to another (from the state of being **Provisionally Created** to the state of being **Reserved**). If we continued to add details to this Sequence diagram, we might add event lines corresponding to the Customer confirming the reservation, which would put **Reservation** into the state of being **Confirmed**. On another Sequence diagram, we might model the scenario for the customer canceling the reservation, transitioning **Reservation** into the state of being **Cancelled**. On yet another Sequence diagram, we might model the scenario for the customer's credit **not** being ok, in which case the **Reservation** is **Denied**. You should look at all of the Sequence diagrams on which objects of a class are modeled when trying to capture the behavior of a class in a state diagram.

Sometimes it is not clear from the Sequence diagram that an action is performed while a class is in a particular State. With the State diagram, you can discover and capture actions not included in the Sequence diagram.

1.7. Drawing the State Diagram

Here are some basic rules for drawing a state diagram:

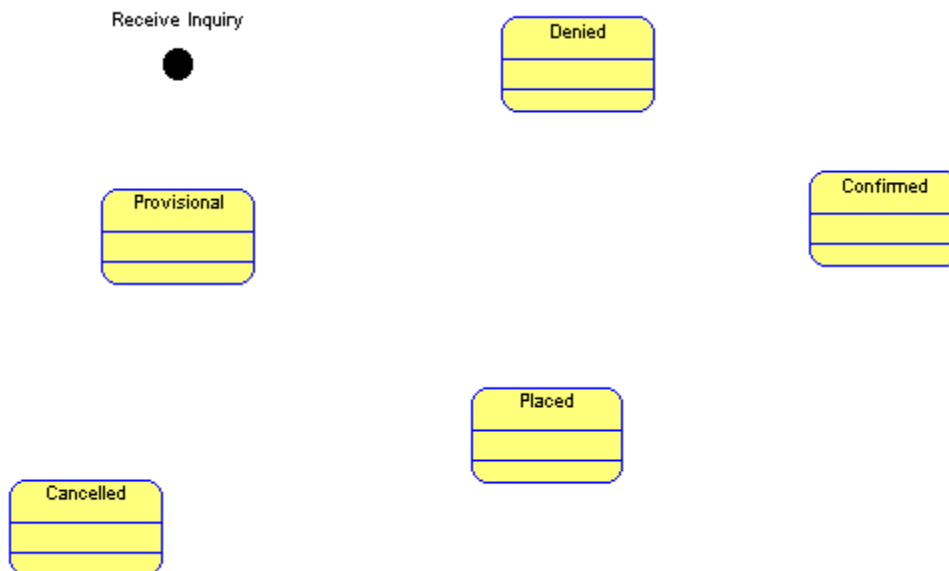
States represent the conditions of objects at certain points in time. Events represent incidents that cause objects to move from one state to another. Transition lines depict the movement from one state to another. Each transition line is labeled with the event that causes the transition. Actions occur when an object arrives in a state.

From our previous discussion, we note that the states that **Reservation** can be in are:

- **Provisional**
- **Reserved** (we will call this **Placed**)
- **Confirmed**
- **Denied**
- **Cancelled**

In addition, every state diagram normally has a start state, which in UML notation is a filled in circle, or large bullet. Let's model this:

1. Select the **Initial State** symbol from the toolbox or **Draw** menu and place an initial state on the diagram. Name it **Receive Inquiry**.
2. Select the **State** symbol from the toolbox and place a state symbol on the diagram. Name the state **Provisional**.
3. With the State symbol drawing tool still selected (your mouse should resemble a pen), drop down additional states and name them **Placed**, **Confirmed**, **Denied**, and **Cancelled**. Position the symbols on the diagram workspace as shown in the picture below:

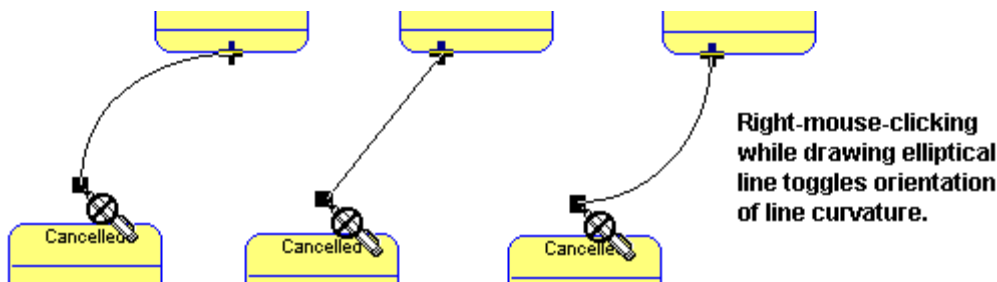


A transition is represented by a vectored line labeled with the name of the event that causes the transition. So far, throughout most of this tutorial, we have been drawing straight or straight orthogonal lines. For the state diagram, however, we might want to use elliptical arcs for aesthetic reasons. Let's switch the line style.

Change the Line Style to Elliptical Arcs

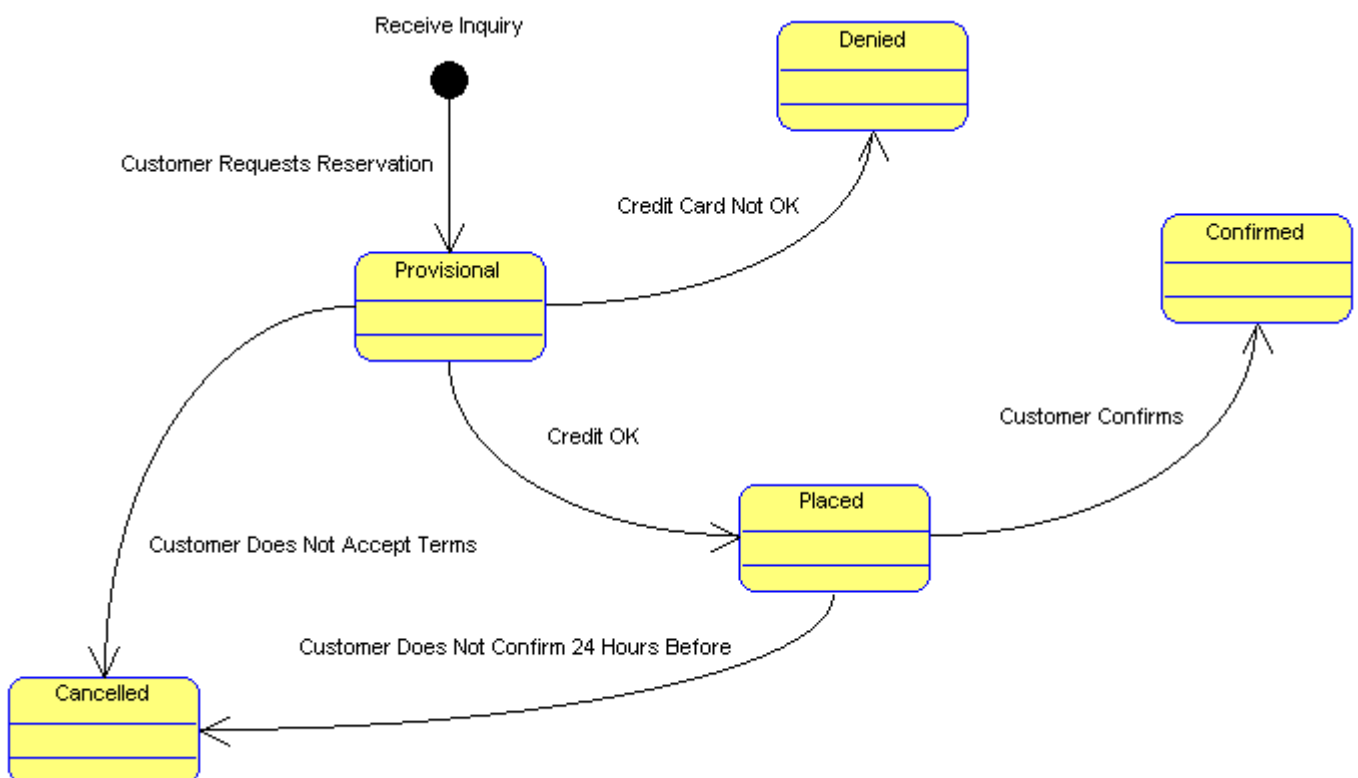
1. Select **Format, Symbol Format, Line**. In the **Line Style** dialog, toggle on **Elliptical Arcs** and click **OK**.

When drawing elliptical arc lines, you can change the direction of the loop -- after left mouse-clicking to attach the start point of the line to a state symbol, release your mouse, and then, before you left-mouse click again to attach the end point to the target state, right-mouse click -- you will notice the arc orientation of the loop in the line change. Continue right-mouse clicking (before left-mouse clicking to attach the line) to toggle between arc orientations.



2. From the toolbox, select the **Transition** line tool, and draw a transition line between the initial state **Receive Inquiry** to the state **Provisional**. Name it **Customer Requests Reservation**.
3. Using the diagram below as a guide, draw the following transition lines:
 - Draw a transition line from the state symbol **Provisional** to the symbol **Cancelled** and name it **Customer Does Not Accept Terms**
 - Draw a transition line from the state symbol **Provisional** to the symbol **Denied** and name it **Credit Card Not OK**
 - Draw a transition line from the state symbol **Provisional** to the symbol **Placed** and name it **Credit OK**
 - Draw a transition line from the state symbol **Placed** to the symbol **Cancelled** and name it **Customer did not confirm in 24 Hours**
 - Draw a transition line from the state symbol **Placed** to the symbol **Confirmed** and name it **Customer Confirms**

The diagram should now resemble the one shown below:



1.8. Defining a State

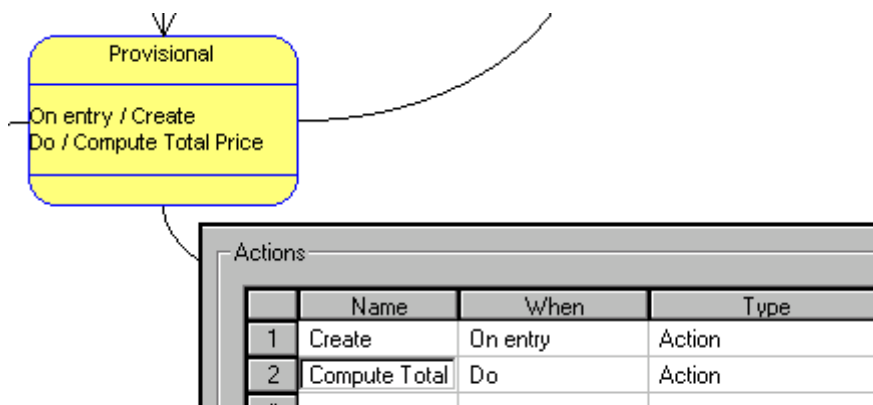
Each **state** is defined by the **actions** that are performed during a **transition** into the state (**entry actions**), the activities that the class performs while in the state (**'do' actions**), internal actions that are performed while in the state, and actions that are performed while the class transitions out of the state (**exit actions**).

Let's take a look at the **Provisional** state, and reference that Sequence diagram we built. When the **Reservation** class enters the **Provisional** state, the reservation is **created**. This is an **entry action** (it is performed on entry into this state). While in the state, the Reservation class **computes its total price**. This is a **'do' action**. While in this state, another class, **Reception**, will find out if the customer's credit is good. We will not model this as an action of the class **Reservation**. However, if the credit is good, the **Diary** class will tell the **Reservation** to update its status to **reserved**. This event will cause a transition out of the **Provisional** state to the **Placed** state.

You will only be able to do the following step if you did and saved the previous tutorials including the ERD and Sequence diagram ones!

Add Actions to the State Definition

1. Open the definition of the state **Provisional** (right-mouse click and select **Edit** or double click on it).
2. Select the **Actions** tab, type in the action **Create**, and hit your **Enter** key. By default, its **When** property is set to **On entry**. Leave the default.
3. Put your cursor in the name cell of the **Create** action, and click on the **Define** button at the bottom of the **Actions** grid to open the full definition dialog for the action.
4. Select the **Methods** tab. Click on **Choices**, and select and drag into the list box the method **create(long, long, char, char)."Reservation System".Reservation**. Click **OK** to close the dialog.
5. In the **Actions** grid, type in an action named **Compute Total Price**, and hit your **Enter** key. By default, its **When** property is set to **On entry**. Place your cursor in this field and change the default by selecting **Do**.
6. Put your cursor in the name cell of the **Compute Total Price** action, and click on the **Define** button at the bottom of the **Actions** grid to open the full definition dialog for the action.
7. Select the **Methods** tab. Click on **Choices**, and select and drag into the list box the method **CalculatePrice(long)."Reservation System".Reservation**. Click **OK** to close all dialogs.



Specify the Trigger Event of a Transition

8. Open the definition of the transition **Credit OK**. In the **Trigger Event** field, type the name of the event that triggers this transition -- **Update Status to Reserved**.
9. Select **File, Save Diagram**.

In a real project, we would continue specifying the actions and events for this state diagram to gain a better understanding of the Reservation's behavior. For this tutorial, we will move on to another facet of modeling. Please review the state diagram section of the help for detailed information on advanced state modeling, including super- and sub-states, history, concurrency, etc.

The completion time of this section is approximately 5 minutes.

4. Mapping Class Diagram to ER Model

So far we have created a class diagram to represent business classes we discovered while doing Use Case and Interaction modeling. We have modeled each of these business classes as a persistent class. A persistent class is one that has a persistent store - it exists after an application has run, and generally stores data. It is opposite of a transitory class - one that is created, used, and destroyed during the course of an application running.

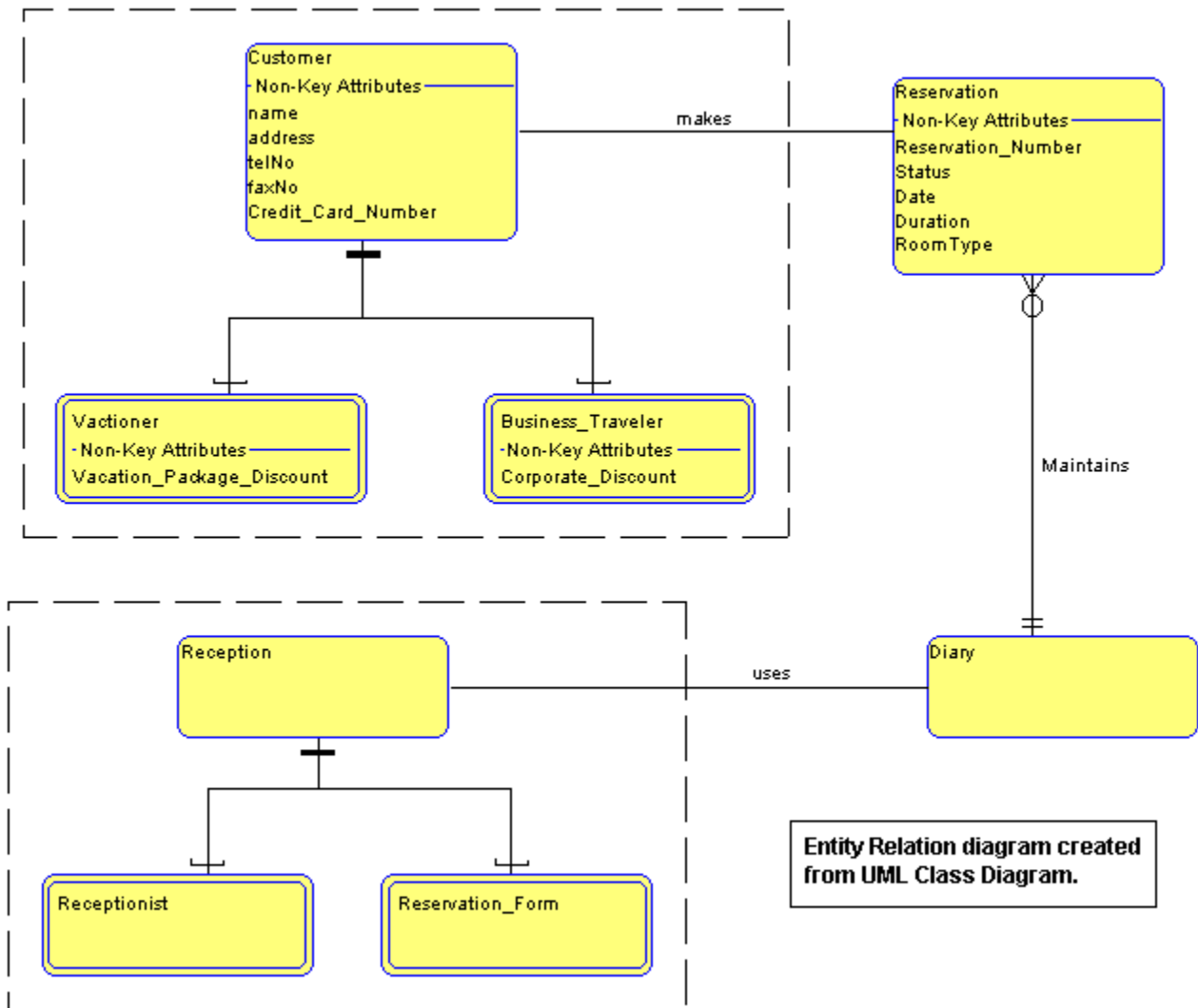
We can use System Architect's automatic mapping capability to map the class diagram we created to a logical Entity Relation (ER) diagram. The ER diagram enables us more capability to model relational concepts, including specification of primary and foreign keys, migration of keys based on relationship types, access paths, etc. Later we can map a logical ER diagram to a physical model representing the specific properties of the DBMS we target for implementation, and then generate working schema (either DDL or to the database directly via ODBC).

During the OO to ER translation, persistent classes and their attributes are mapped to entities. Many-to-many associations between classes are mapped to separate entities in the ER diagram - the name of the entity will be the value defined in the implemented with member variable property of the association, or, if there is no member variable defined, the name of the association itself. Class inheritance structures are mapped directly to entity super/sub relationships in the ER diagram. These super/sub relationships can later be 'flattened' when transforming the logical ER diagram to a physical data model.

To map the UML Class diagram to an ER diagram:

1. Open the class diagram, **Hotel Reservation System**, in the **Reservation System** package.
2. Select **Dictionary, Convert to ER Diagram**. System Architect begins the mapping, and presents you with the **Convert OO - ER** dialog.
3. In the **Convert OO - ER** dialog, select **Hotel Reservation System** from the drop-down list.
4. In the **Destination Diagram** field, type in **Reservation System** as the name of the diagram.
5. Leave the default **All Symbols** selected in the **Transfer** group. This will map all classes on the currently open diagram.

The new ER model is created.



6. Select **File, Save Diagram**.

When I did the above exercise I needed to move the objects around a bit to get the generated diagram looking a neat as the one above.

The following details are just to let you know that it is possible to create programs ('generate code') from the various diagrams we have created. We will not be doing this nor do I expect you to understand this process or the code produced.

The completion time of this section is approximately 15 minutes.

The Student Version of System Architect does not provide Code Generation.

5. UML Class Design

During design, you add further details to associations and class definitions which are necessary for implementing the system. System Architect supports automatic generation of code from a class diagram for a number of languages, including Java, C++, Visual Basic, and CORBA IDL. System Architect also supports reversing code into a class diagram for Java, C++, and Visual Basic.

Code Generation Capabilities

The Student version of System Architect does not provide Code Generation.

System Architect's commercial release provides script-based code generation for a number of languages. Code generation is performed by selecting a code generation script that is located in the **<c>:\Program Files\System Architect\Basgen** directory. There are scripts for Java (java.bas), JavaScript (javasc.bas), and CORBA IDL (corba.bas). The script-based code generator generates code to a file or files. These files can then be used in any Interactive Development Environment (IDE).

For C++, System Architect provides a 'hard-wired' generator, which produces header (.hpp) and skeleton (.cpp) files.

In addition, for Visual Basic 6.0, System Architect provides a synchronization wizard (select **Tools, Synchronization Wizard**) to synchronize a class diagram with an open Visual Basic project. The wizard provides a direct tie-in between System Architect and the IDE. ***The Synchronization Wizard does not apply to the Student Version. This is only available with System Architect's commercial release.***

END OF TUTORIAL