Constructing the Problem Domain Model

Objectives

Upon completion of this module, you should be able to:

- Identify the essential elements in a UML Class diagram
- Construct a Domain model using a Class diagram
- Identify the essential elements in a UML Object diagram
- Validate the Domain model with one or more Object diagrams

Additional Resources

Additional resources – The following references provide additional information on the topics described in this module:

- Booch, Grady, James Rumbaugh, Ivar Jacobson. *The Unified Modeling Language User Guide*. Reading: Addison Wesley Longman, Inc., 1999.
- Booch, Grady. *Object Solutions (Managing the Object-Oriented Project).* Reading: Addison Wesley Longman, Inc., 1994.
- Folwer, Martin, Kendall Scott. *UML Distilled (2nd ed)*. Reading: Addison Wesley Longman, Inc., 2000.
- Jacobson, Ivar, Grady Booch, James Rumbaugh. *The Unified Software Development Process*. Addison-Wesley. Reading. 1999
- The Object Management Group. "Unified Modeling Language (UML), Version 2.2" [`http://www.omg.org/technology/documents/formal/uml.htm`].
- Jacobson, Ivar, Grady Booch, James Rumbaugh. *The Unified Modeling Language Reference Manual (2nd ed)*. Addison-Wesley, 2004.

Process Map

This module covers the next step in the Requirements Analysis workflow: creating the Domain model from the key abstractions. Figure 7-1 shows the activities and artifacts covered in this module.

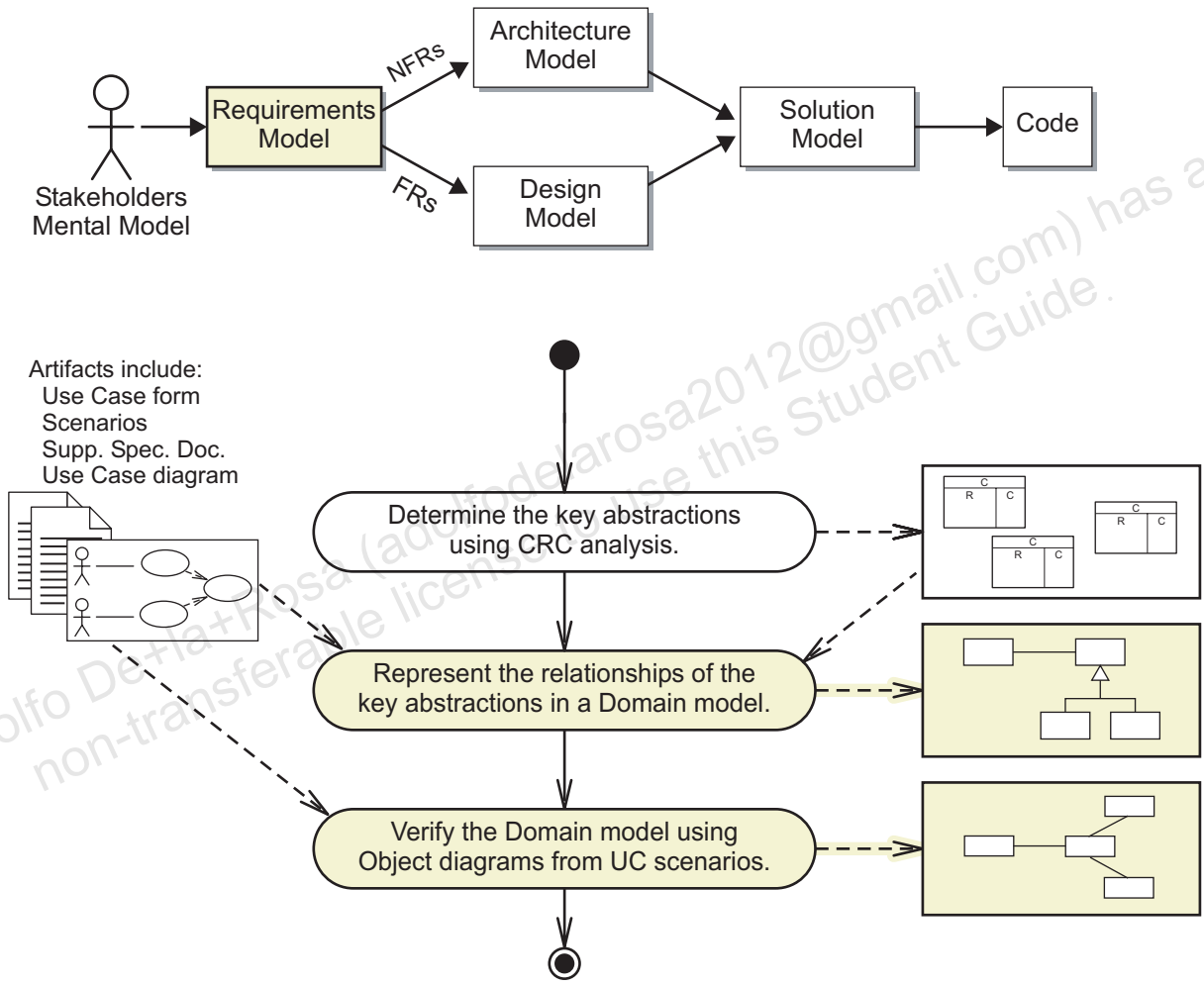

Figure 7-1 Domain Model Process Map

Introducing the Domain Model

> "The sea of classes in a system that serves to capture the vocabulary of the problem space; also known as a conceptual model." (Booch Object Solution page 304)

The *Domain model* is one of three major views of the Requirements model; the others are the Use Case model and the Supplementary Specification Document.

- The classes in the Domain model are the system's key abstractions.

  The Domain model is a visual representation (a UML Class diagram) of the key abstractions that are discovered during analysis. Each key abstraction becomes a class in the Class diagram.

- The Domain model shows relationships (collaborators) between the key abstractions.

  CRC analysis identified the classes (key abstractions), responsibilities, and collaborations between classes. All of these features can be represented in the Domain model. It is the Domain model, not the CRC cards, that is a long-term artifact of the project.

This module describes the Domain model, how it is constructed, and the UML diagrams that represent it.

Identifying the Elements of a Class Diagram

The *Class diagram* in the UML is probably the most recognized. It is used to visually represent classes, the members of the class, and the relationships between classes (usually called associations). Figure 7-2 shows an example Class diagram with a few of the features labelled by callouts.

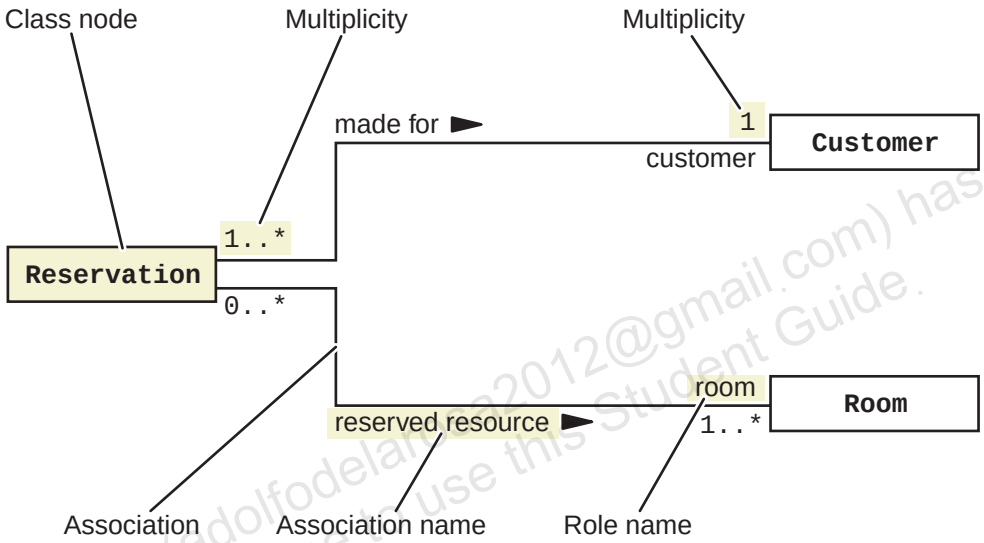

Figure 7-2 Elements of a UML Class Diagram

Class Nodes

Class nodes represent classes of objects within the model. Class nodes can take many visual forms in UML, however, a class node is represented by a rectangle with the name of the class in a bold, monospaced font. Several different representations of the `Customer` class is shown in Figure 7-3.

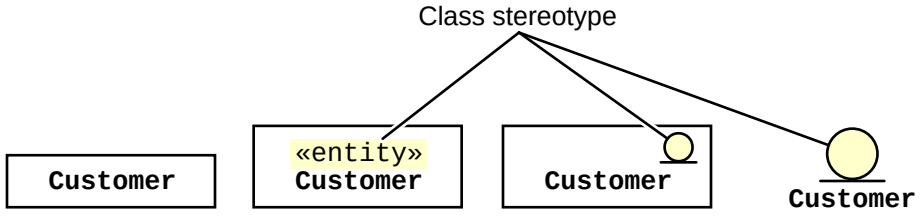

Figure 7-3 Different Ways to Represent a Class Node

A class node can represent:

- Conceptual entities, such as key abstractions

  During the Analysis workflow, class nodes in the Domain model represent key abstractions. These are conceptual entities that do not correspond to software components.

- Real software components

  During the Design workflow, class nodes usually represent a physical software component. The component is usually a class, but can also be other components. For example, a class node might represent an EJB technology entity bean.

A stereotype can help identify the type of the class node. A stereotype in UML can take one of two forms. The textual form uses a word or phrase surrounded by guillemet characters; for example, «Entity». A stereotype can also take an iconic form; for example, an Entity can also be symbolized by a circle with a line under it.

Class Node Compartments

A class node may be split into three or more compartments:

- The name compartment records the name of the class.

  This compartment is always the top of the node. The stereotype of the class must also be placed in the Name compartment.

- The attributes compartment records attributes of the class.

  This compartment is placed below the Name compartment. The attributes can be conceptual (see Figure 7-4) or detailed. You will see examples of detailed attributes later in the course.

- The operations compartment records operations of the class.

  This compartment is placed below the Attributes compartment. Methods can be conceptual or actual method signatures. You will see examples of method signatures later in the course.

- Additional compartments may be added.

  These compartments show additional information. These compartments are placed below the Methods compartment. You can use these compartments for any purpose, such as recording exceptions thrown by any of the methods or recording a textual description of the responsibilities of the class.

Figure 7-4 shows a class node for the `Reservation` class with three compartments visible.

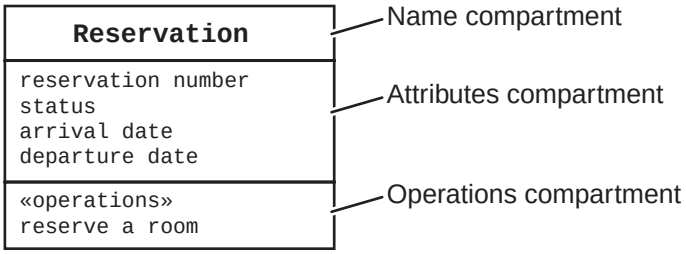

Figure 7-4 An Example Class Node With Compartments

Associations

Associations represent relationships between classes. Associations are manifested at runtime in which two objects are associated with each other, usually with an object reference.

It is important to understand that Class diagrams represent static information about the relationships between classes. A Class diagram *does not* represent the dynamic relationships of objects at runtime. It is better to think of a Class diagram as a set of constraints on the dynamic, runtime nature of objects.

Relationship and Roles

Figure 7-5 shows an example association. This association would be read as "A reservation is made for a customer."

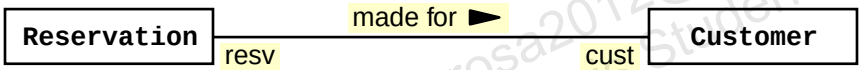

Figure 7-5 Example Association With Relationship and Role Labels

The direction arrow indicates which direction to read the association. Role names indicate what role that a particular class is taking in the association. In Figure 7-5, customer is the role of the association between `Reservation` and `Customer`. However, if the role name is the same as the class name, then the role name can be eliminated. In this example, both role names could be eliminated.

Multiplicity

Multiplicity determines how many objects might participate in the relationship.

Figure 7-6 shows an example association with multiplicity labels. This association would be read as "A reservation is made for one and only one customer." Reading it in the other direction is "A customer can make one or more reservations."

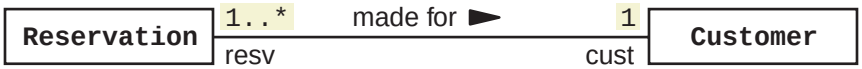

Figure 7-6 Example Association With Multiplicity Labels

The default multiplicity for any side of the association is one. However, it is a good practice to explicitly show the "one" multiplicity labels. The multiplicity label can take several syntactic forms. These are listed in Table 7-1.

Table 7-1 Different Types of Multiplicity

Multiplicity Syntax	Meaning
`1`	One and only one object.
`n..m`	At least n objects, but no more than m. This is an inclusive range.
`0..1`	Zero or one object. This multiplicity value is used to represent that an association to an object is optional.
`n,m`	There are either exactly n objects or exactly m objects. This syntax can be used in combinations with the double-dot notation. For example, `1..2,4,6..10` is valid syntax.
`0..* or *`	There may be zero or more objects.
`1..*`	There is at least one object, but there might be more.

Navigation

Navigation arrows on the association determine what direction an association can be traversed at runtime. This additional information is most important during design, but it can also be useful in analysis. An association without navigation arrows means that you can navigate from one object to the other and the other way around as well. For example, Figure 7-6 on page 7-8 shows the association implies that from a `Reservation` object you can retrieve (navigate to) the `Customer` object. It also implies that from a `Customer` object you can retrieve the set of reservations for that customer.

This is very flexible, but in some situations it might not be meaningful to the problem domain. For example, the functional requirements of the Hotel Reservation System do not require navigation from the room to the set of reservations. This constraint is illustrated in Figure 7-7. This association would be read as "From a reservation the system *can directly* retrieve the room, and from a room the system *cannot directly* retrieve the reservation for that room."

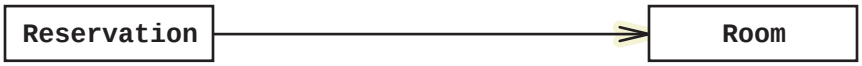

Figure 7-7 Example Association With Navigation Arrows

Association Classes

Sometimes information is included in the association between two classes. For example, a person works for many employers. If you need to record details about each term of employment, this information belongs to the association between the person and the employer. You can show this using an association class. Figure 7-8 shows this.

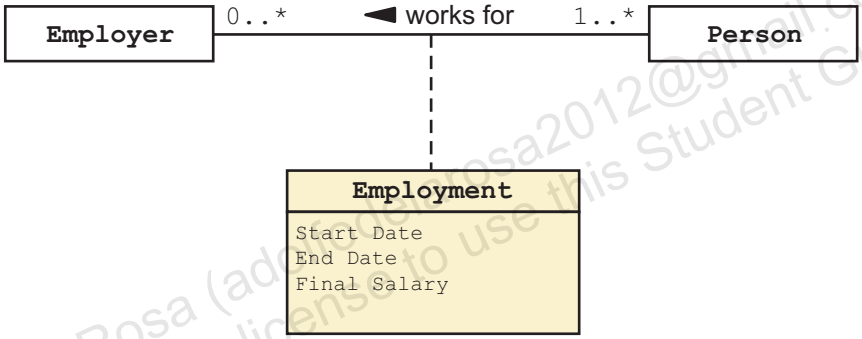

Figure 7-8 Example Association Class

Note – Association classes are used to specify the need to record the attributes shown in the association class. In the Design workflow, the designer will decide the best class to record these attributes.

Creating a Domain Model

Starting with the key abstractions, you can create a Domain model using these steps:

1. Draw a class node for each key abstraction, and:
   a. List known attributes.
   b. List known operations.
2. Draw associations between collaborating classes.
3. Identify and document relationship and role names.
4. Identify and document association multiplicity.
5. Optionally, identify and document association navigation.

The following figures demonstrate the major steps of creating a Domain model from the set of key abstractions discovered in the previous analysis activity.

Note – The example used includes only a subset of classes. The example uses business rules that a hotel must have at least one room and a reservation must have at least one room. In addition, room names are used instead of room numbers.

Step 1 – Draw the Class Nodes

The first step is to draw the class nodes for each key abstraction. Show all attributes and responsibilities, if you want more detail in the Domain model. Figure 7-9 illustrates this.

Figure 7-9 Step 1 – Draw the Class Nodes

These diagrams have been drawn with precise orientation of each class node to facilitate showing a progression of steps where the class nodes stay in place. In practice, you should start with one class node (usually the most prominent key abstraction) and place it in the center of your drawing space. Add other class nodes around it as you draw the associations. As the diagram expands you will discover that you need to associate two classes that end up on different sides of the diagram. Because of this issue, you will have to draw the Domain model several times before you find the right visual structure.

Step 2 – Draw the Associations

The next step is to draw association lines between each collaborating class. This information comes from the Collaboration column of the CRC card for each class. Figure 7-10 shows the associations for the key abstractions of the Hotel Reservation System.

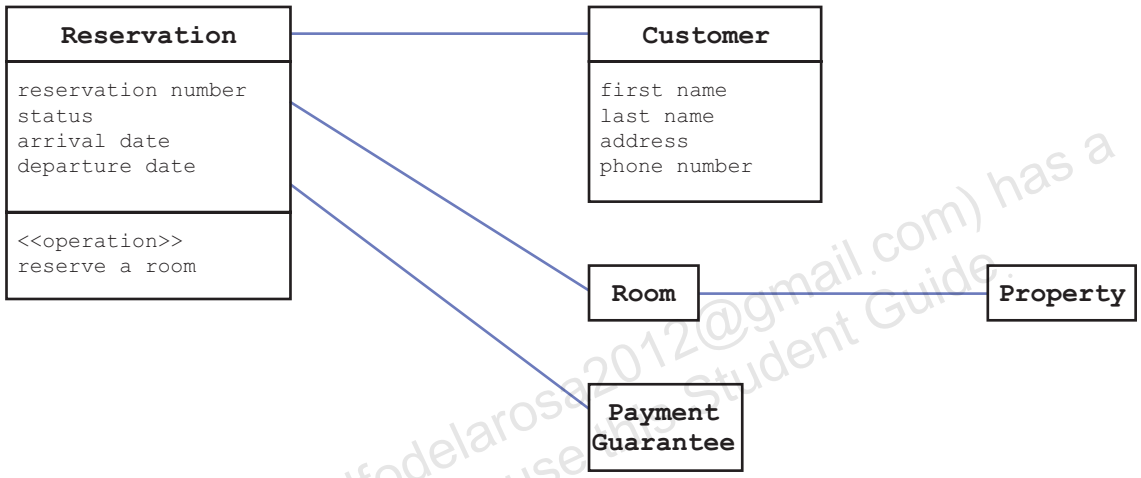

Figure 7-10 Step 2 – Draw Associations Between Collaborating Classes

Step 3 – Label the Association and Role Names

The next step is to label the associations with association names and role names. Figure 7-11 illustrates this.

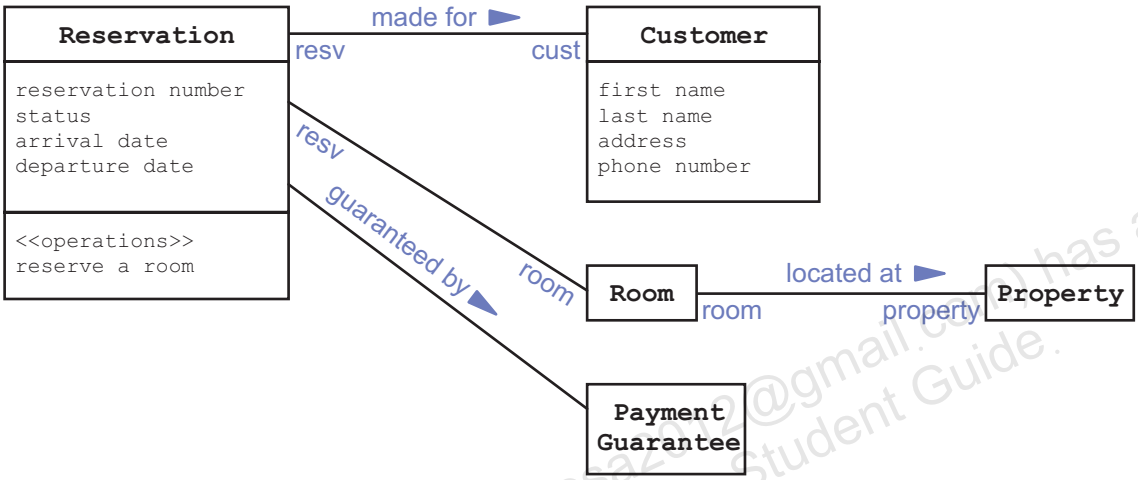

Figure 7-11 Step 3 – Record the Relationship and Role Names

Association and role names tend to be obvious, so many analysts ignore this step.

Step 4 – Label the Association Multiplicity

The next step is to add the multiplicity labels to the associations. The multiplicity information is usually not identified during CRC analysis. This information can be identified from the Use Case scenarios, the Use Case form, or a domain expert. Figure 7-12 illustrates this step.

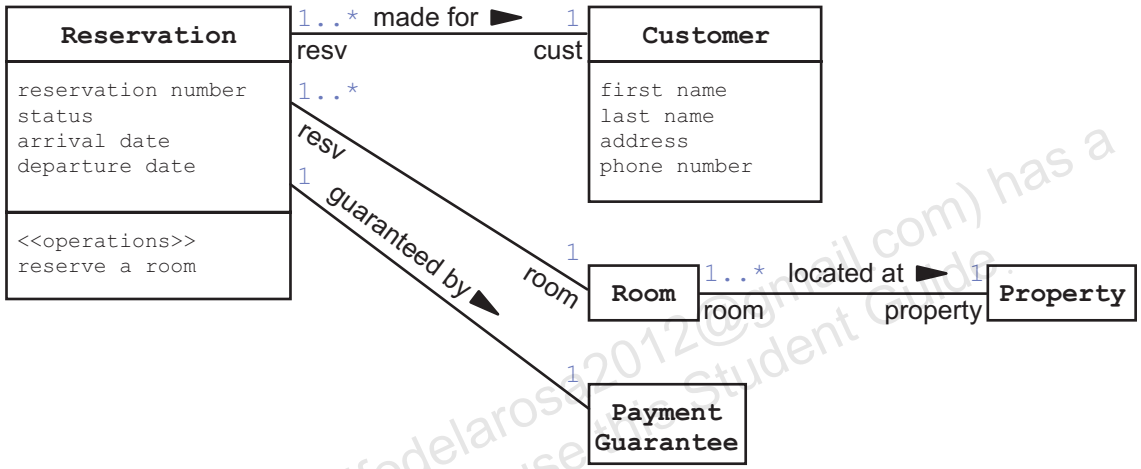

Figure 7-12 Step 4 – Record the Association Multiplicity

It is common to make mistakes about the multiplicity of associations during analysis. In "Validating the Domain Model Using Object Diagrams" on page 7-20, you will learn a technique for identifying these problems.

Step 5 – Draw the Navigation Arrows

The next step is to draw the navigation arrows on the associations, if they are known during analysis. Figure 7-13 illustrates this.

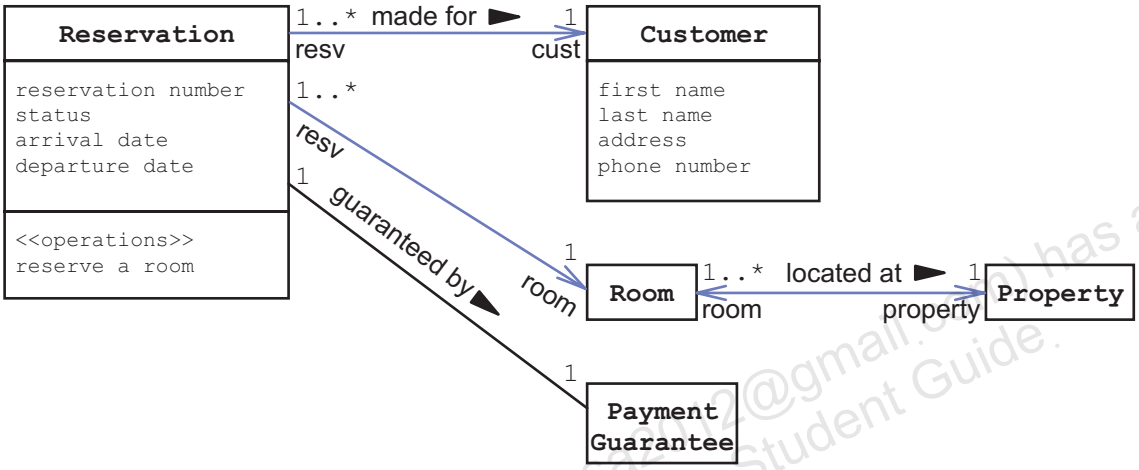

Figure 7-13 Step 5 – Record the Association Navigation

This is done only when an FR constrains the Domain model or when the business analysts decides that the navigation of a particular association is meaningless for the domain. For example, it is not necessary to be able to navigate from a `Room` object to a reservation, so this association has a navigation arrow pointing from the Reservation to the Room class, but not in the other direction.

Notice that the association between `Room` and `Property` has a navigation arrow on both ends. This makes that association explicitly bidirectional. Associations without navigation arrows are implicitly bidirectional.

Validating the Domain Model

In this section, you have seen how to construct a Domain model. How do you know that this diagram accurately models the problem domain? One technique is to build Object diagrams from the Use Case scenarios and check that the Object diagrams fit the Class diagram. This technique will be discussed in "Validating the Domain Model Using Object Diagrams" on page 7-20, but first this module describes the UML Object diagram syntax.

Identifying the Elements of an Object Diagram

> "A static object diagram is an instance of a class diagram; it shows a snapshot of the detailed state of a system at a point in time." (UML v1.4 page 3-35)

The *Object diagram* in the UML visually represents objects and their runtime links. Figure 7-14 shows an example Object diagram with a few of the features labelled by callouts.

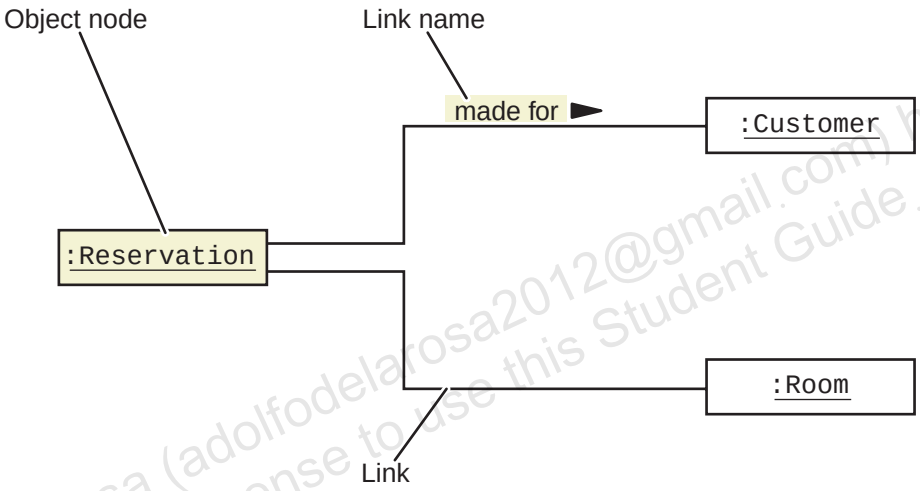

Figure 7-14 Elements of a UML Object Diagram

Note – Link names are usually ignored in Object diagrams.

Object Nodes

An object node usually includes some form of name and data type. Figure 7-15 shows this.

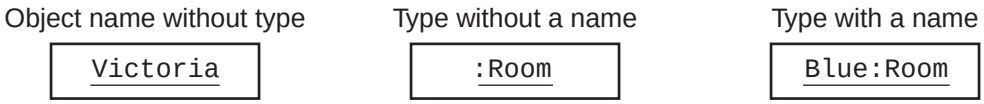

Figure 7-15 Object Node Types

The first node, `Victoria`, does not include a data type. This means that you have an object, but you do not yet know what type of the object it is. This situation is rare. The second node shows an object with a type, but no name. This is very common. The third node shown an object with both a name and a data type. The name usually refers to a programming variable that refers to the object; also, the name could represent a name attribute of the object.

An object node might also include attributes. Object nodes can include an attributes compartment. Each attribute is listed on a separate line with an equal sign (=) to separate the name from the value. Figure 7-16 shows this notation.

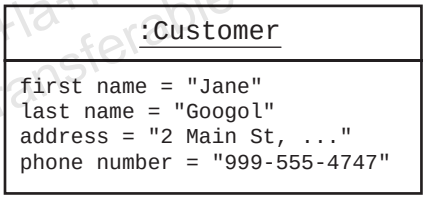

Figure 7-16 An Object Node With Attributes

Links

In Object diagrams, each link is unique and is one-to-one with respect to the participants. A link is a unique instance of a class association. That means that each link connects only two objects together. There is no multiplicity on these links because everything is one-to-one. For example, in Figure 7-17, the `Property` object is associated with three different rooms; each association has its own association link.

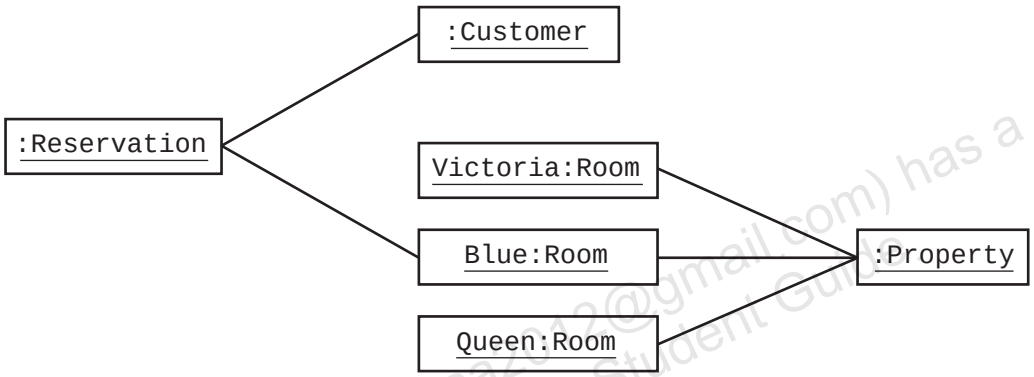

Figure 7-17 An Object Diagram With Links

Validating the Domain Model Using Object Diagrams

To validate the Domain model:

1. Pick one or more use cases that exercise the Domain model.

   Ideally, you should create Object diagrams that represent scenarios from every use case. However, this would be extremely time-consuming. Pick use cases that exercise significant portions of the Domain model.

   For example, in the Hotel Reservation System, the use case "Create a Reservation" has a very broad coverage of the Domain model.

2. Pick one or more use case scenarios for the selected use cases.

   Again, you should create Object diagrams that represent all scenarios for a given use case. Pick scenarios that explore a variety of situations. You want to find scenarios that push the constraints of the Domain model.

   For example, in the "Create a Reservation" use case you should explore a scenario that is simple, such as reserving one room, and others that are more complex, such as reserving multiple rooms.

3. Walk through each scenario (separately), and construct the objects (with data) mentioned in the scenario.

   Because scenarios are specific, they include a lot of detail. This detail is very useful. In "Creating a Scenario Object Diagram" on page 7-20, you will see how to construct an Object diagram for a use case scenario.

4. Compare each Object diagram against the Domain model to see if any association constraints are violated.

   By comparing Object diagrams from a variety of use case scenarios you will see if the Domain model has been incorrectly constrained. In "Comparing Object Diagrams to Validate the Domain Model" on page 7-25, you will see how to perform this validation.

Creating a Scenario Object Diagram

The following figures demonstrate the process of creating an Object diagram from a use case scenario. Each paragraph in the scenario represents a *time step* in the scenario. You can build the Object diagram as a sequence, one for each time step.

The use case begins when the booking agent receives a request to make a reservation for rooms in the hotel. The booking agent enters the arrival date, the departure date, and the quantity of each type of room that is required.

At the beginning of this scenario, the only thing known is the `Property` object of hotel that the customer wishes to book a room. Figure 7-18 shows this rudimentary Object diagram.

Figure 7-18 Step 1 – Create Reservation Scenario 1

The booking agent then submits the entered details.The system finds rooms that will be available during the period of the reservation.

In this step of the scenario, a new `Reservation` object is created in the system with the arrival and departure dates filled in. Also, the booking agent has performed a search for rooms that would be available at the property. This search returned three rooms: Victoria, Blue, and Queen. The Object diagram for the scenario is expanded to include these objects. Figure 7-19 shows this. Note that this is not the same Object diagram as the previous diagram (Figure 7-18 on page 7-21) because each diagram is a snapshot in time; these represent snapshots at two different times.

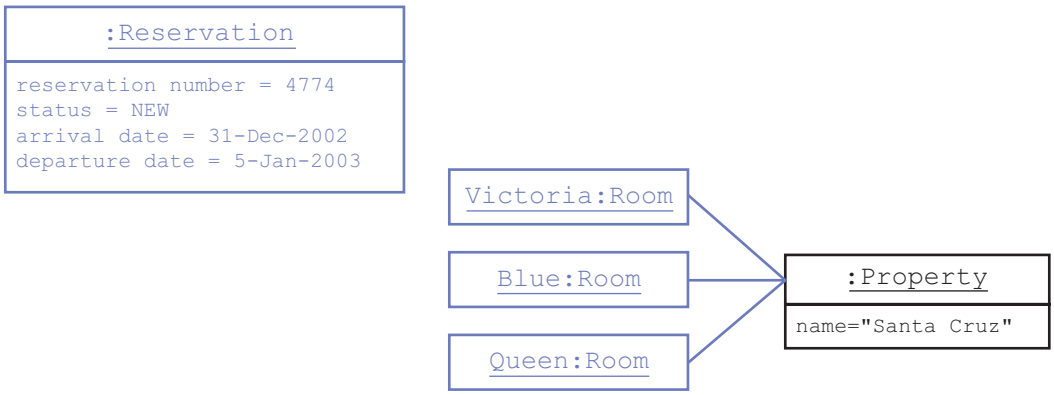

Figure 7-19 Step 2 – Create Reservation Scenario 1

The system allocates the required number and type of rooms from the available rooms. The system responds that the specified rooms are available, returns the provisional reservation number, and marks the reservation as “held”.

In this step, the system allocates one of the available rooms. An association is added to connect the reservation to the room. Figure 7-20 shows this Object diagram.

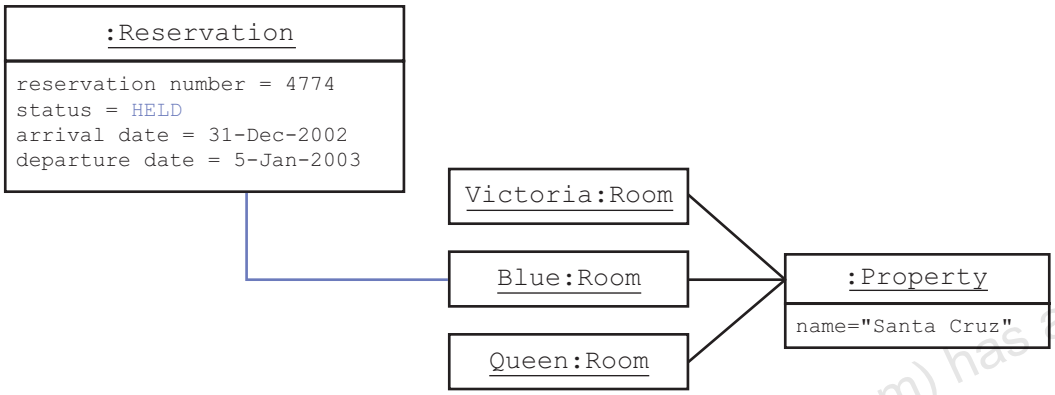

Figure 7-20 Step 3 – Create Reservation Scenario 1

The booking agent accepts the room offered. The booking agent selects that the customer has visited one of the hotels in this group before, and enters the zip code and customer name. The system finds and returns a list of matching customers with full address details. The booking agent selects one of the customers as being the valid customer. The system assigns this customer to the reservation.

In this step, the `Customer` object is retrieved and associated with the reservation. Figure 7-21 shows this diagram.

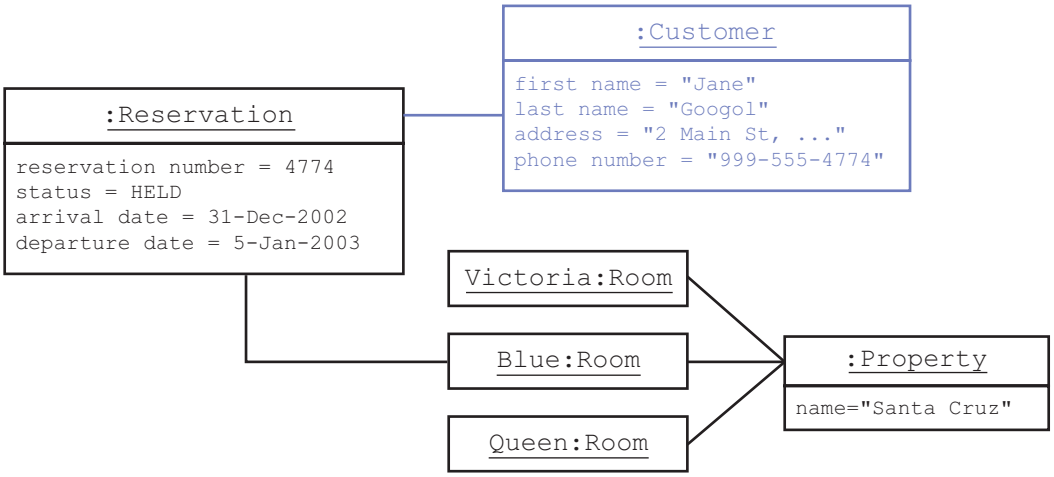

Figure 7-21 Step 4 – Create Reservation Scenario 1

The booking agent performs a payment guarantee check. This check is successful. The system assigns the payment guarantee to the reservation and changes the state of the reservation to “confirmed”. The system returns the reservation ID and booking details.

In this step, a sub-use case is invoked to ensure that the payment is guaranteed. The details of this sub-use case can be handled in separate scenarios. A `Payment Guarantee` object is created and associated with the reservation. The status of the reservation is changed to `CONFIRMED`. Figure 7-22 shows this diagram.

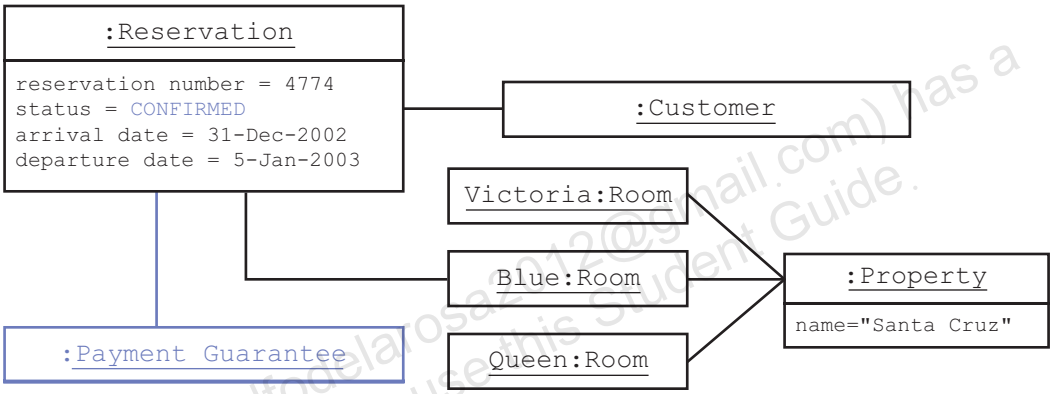

Figure 7-22 Step 5 – Create Reservation Scenario 1

This Use Case scenario is now complete.

Create Reservation Scenario 2

You have seen a walk through of a Use Case scenario to generate an Object diagram. Another "Create a Reservation" scenario has the Actor making a reservation for a small family reunion in which three rooms are booked. Figure 7-23 shows the Object diagram this scenario creates.

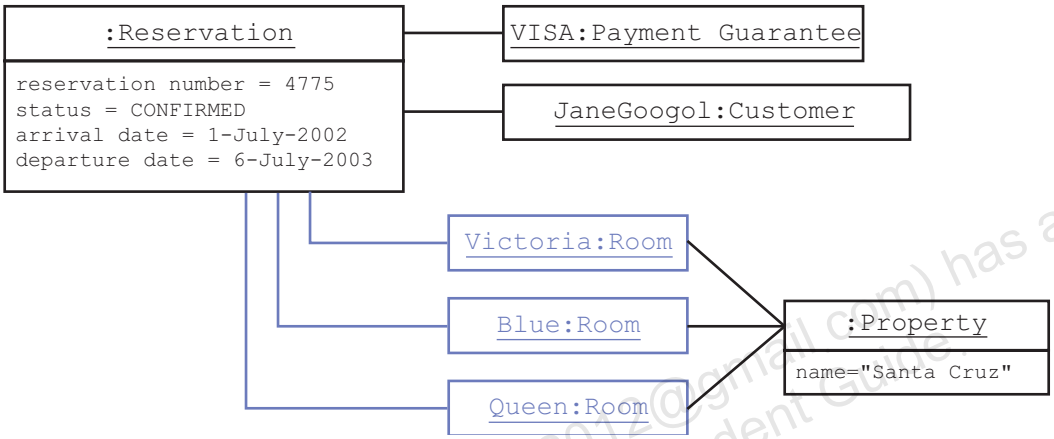

Figure 7-23 Create Reservation Scenario 2

By comparing these two scenarios, you will see how to validate the Domain model and finds errors in the model.

Comparing Object Diagrams to Validate the Domain Model

To validate the Domain model, compare the Class diagram with the scenario Object diagrams.

- Are there attributes or responsibilities mentioned in a scenario that are not listed in the Domain model?

  Data in a scenario should be reflected in some object of the scenario Object diagram. Verify that the data attributes in the objects have corresponding data attributes in the Class diagram.

- Are there associations in the Object diagrams that do not exist in the Domain model?

  Every association in the scenario Object diagrams must have a corresponding association in the Domain model. If you find an association that does not exist in the Domain model, ask the domain expert which diagram is correct, and make the necessary changes to the inaccurate diagram.

- Are there scenarios in which the multiplicity of a relationship is wrong?

  For every Object diagram, count the number of associations for a given type of association, and verify that this number is valid for the multiplicity constraints specified in the Domain model.

The first scenario Object diagram verifies the Domain model, but the second scenario identifies a problem with the Domain model. The multiplicity of "Reservation reserves a Room" currently states that only one room can be reserved per reservation (see Figure 7-12 on page 7-15). The second scenario shows multiple rooms reserved for a single reservation. Therefore, the multiplicity label next to the `Room` class must be changed to `1..*`. Figure 7-24 shows this change.

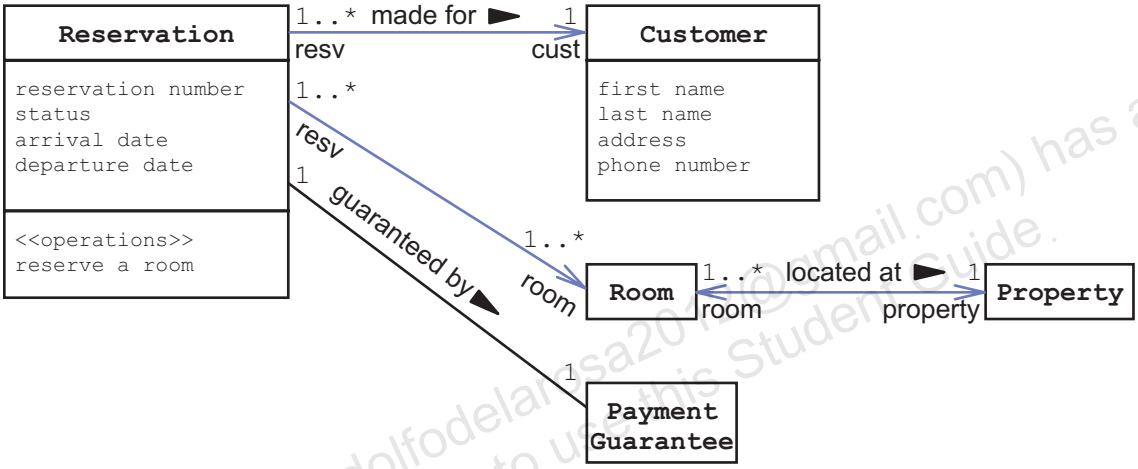

Figure 7-24 Revised Domain Model for the Hotel Reservation System

Summary

In this module, you were introduced to the process of creating a Domain model. Here are a few important concepts:

- Use the Domain model to provide a static view of the key abstractions for the problem domain.
- Use the UML Class diagram to represent the Domain model.
- Validate the Domain model by creating Object diagrams from use case scenarios to see if the network of objects fits the association constraints specified by the Domain model.