Transitioning from Analysis to Design Using Interaction Diagrams

Objectives

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

- Explain the purpose and elements of the Design model
- Identify the essential elements of a UML Communication diagram
- Create a Communication diagram view of the Design model
- Identify the essential elements of a UML Sequence diagram
- Create a Sequence diagram view of the Design model

Additional Resources

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

- Jacobson, Ivar, Grady Booch, James Rumbaugh. *The Unified Software Development Process*. Addison-Wesley. Reading. 1999.
- Jacobson, Ivar. *Object-Oriented Software Engineering*. Harlow: Addison Wesley Longman, Inc., 1993.
- Knoernschild, Kirk. *Java Design (Objects, UML, and Process)*. Reading: Addison Wesley Longman, Inc., 2002.
- The Object Management Group. "Unified Modeling Language (UML), Version 2.2" [`http://www.omg.org/technology/documents/formal/uml.htm`].
- Rosenberg, Doug, Kendall Scott. *Use Case Driven Object Modeling with UML (A Practical Approach)*. Reading: Addison Wesley Longman, Inc., 1999.
- Rumbaugh, James, Jacobson Ivor, Booch Grady. *The Unified Modeling Language Reference Manual (2nd ed).* Addison-Wesley, 2004.
- The Object Management Group. "OMG Unified Modeling Language ™(OMG UML), Superstructure," [`http://www.omg.org/spec/UML/2.2/Superstructure/PDF/`], Version 2.2, February 2009.

Process Map

This module covers the first step in the Design workflow: creating the Design model to realize a use case. Figure 8-1 shows the activities and artifacts covered in this module. This module was placed before the Architecture workflow modules because you need to understand the purpose and structure of the Design model before creating the Architecture model.

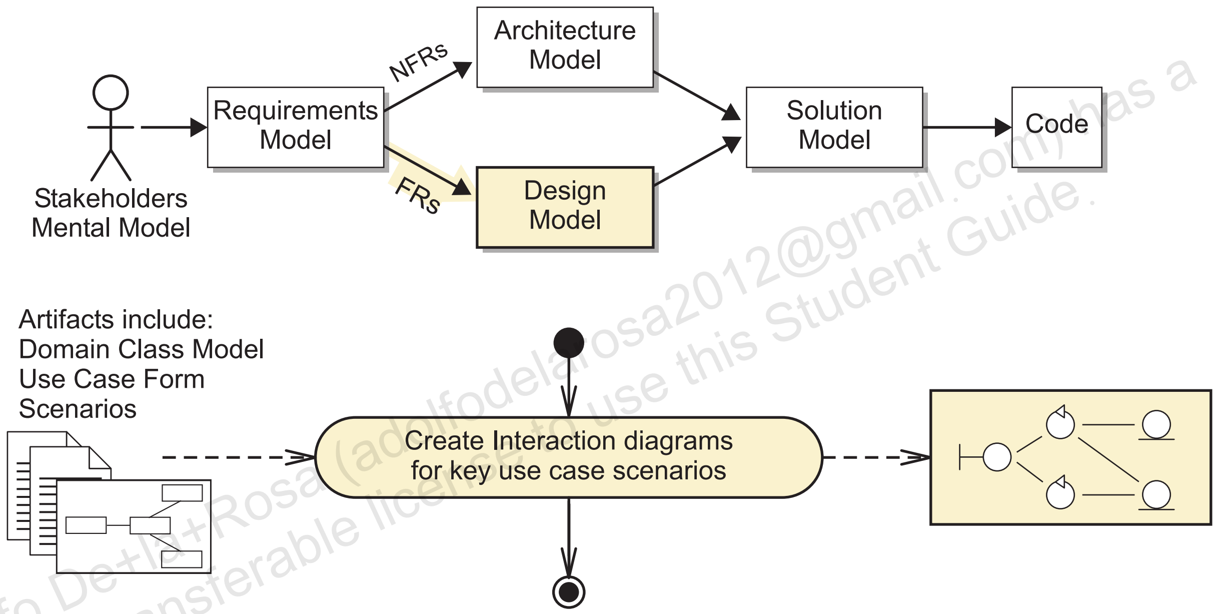


Figure 8-1 Interaction Diagrams Process Map

Introducing the Design Model

A Design model is created from the functional requirements of the Requirements model. The Design model is a realization of a use case; it describes the types of components required to perform the use case. The Design model is merged with the Architecture model to produce the Solution model. Figure 8-2 illustrates this.

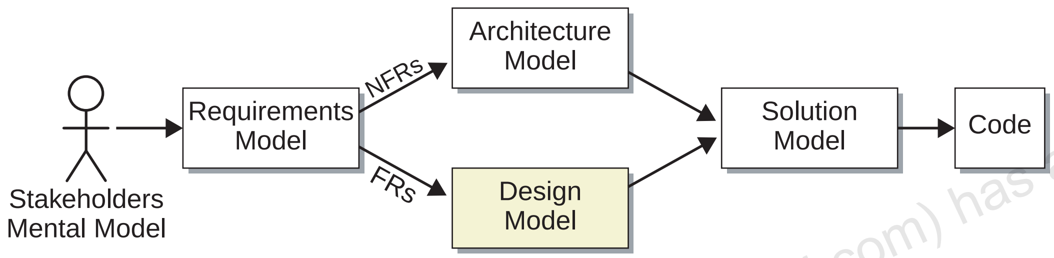


Figure 8-2 Introducing the Design Model

Interaction Diagrams

UML Interaction diagrams are the collective name for the following diagrams:

- Sequence diagrams
- Communication diagrams

  Formerly known as Collaboration diagrams
- Interaction Overview diagrams

  A combination of an Activity diagram and a Sequence diagram fragments

Each UML Interaction diagram is used to show the sequence of interactions that occur between objects during:

- One or two use case scenarios
- A fragment of one use case scenario

Interaction diagrams highlight the following:

- Which of the objects interact
- What messages are communicated between objects
- The sequence of the interactions

While creating an interaction diagram, you often discover the need for new classes which will assist in the control and co-ordination of the use case behavior. You may also discover new methods and attributes for the business domain classes, that were not discovered during the analysis phase.

UML Interaction diagrams might also be used to show the sequence of interactions that occur between:

- Systems
- Subsystems

  Interaction diagrams are primarily used to show interaction between specific objects, however they are useful for showing interactions between composite objects, that include subsystems and systems. For example, you can use Interaction diagrams to show interactions between an ATM, the Bank that owns the ATM, and the card users Bank.

Comparing Analysis and Design

Analysis helps you model *what* is known about a business process that the system must support:

- Use cases

  The behavior of the system relative to the user
- Domain model

  The entities involved in the business processes

Design helps you model *how* the system will support the business processes. The Design model consists of:

- Boundary (UI) components

  These components interact directly with the user. These components affect the system by accessing functions of the Service and Entity components.
- Service components

  These components perform business operations and use case workflow management.
- Entity components

  These components correspond to the entities in the Domain model.

Robustness Analysis

The components identified during robustness analysis are called design components.

Robustness analysis is a process that assists in identifying design components that would be required in the Design model. Figure 8-3 illustrates this.

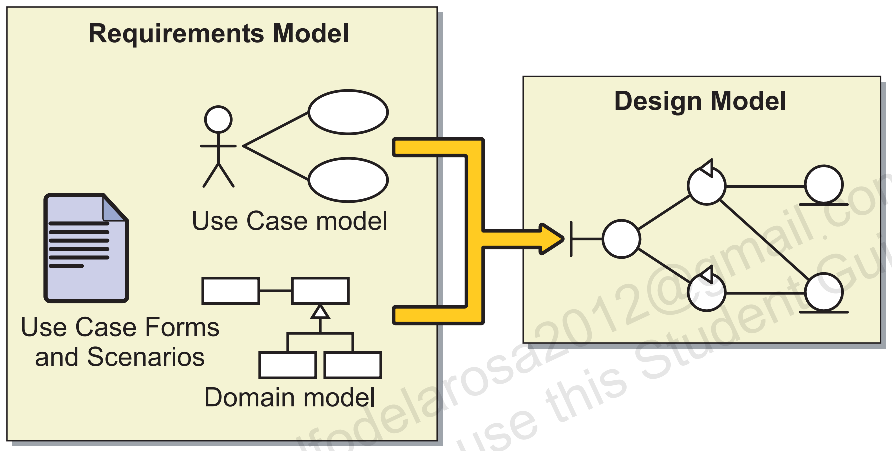


Figure 8-3 The Design Model Is Derived From the Requirements Model

Robustness analysis is the name given to the process that creates the Design model. This name is not widely used in the industry, but the process has deep roots starting with Jacobson's OOSE methodology.

Robustness analysis uses these inputs:

- A use case

  A use case is selected to be designed. You could perform Robustness analysis on multiple use cases at once, but that would lead to a rather large Design model. It is better to create multiple, small Design models; one for each use case of the system.

- The use case scenarios for that use case

  Robustness analysis is like CRC analysis in that you will perform a *walk through* of use case scenarios to construct the Design model.

- The use case Activity diagram (if available) for that use case

  You can use an Activity diagram for a use case instead of walking through each scenario.

- The Domain model

The Domain model is the source of entity components for the Design model. Not all Domain entities are used in every use case, therefore, not all of the Domain entities will appear in any given Design model. Only the entities that the use case being designed uses will appear in the Design model.

The Design model is usually captured in UML Interaction diagrams with design components such as Boundary, Service, and Entity components.

Boundary Components

"A boundary class (component) is used to model interaction between the system and its actors (that is, users and external systems)." (Jacobson, Booch, and Rumbaugh page 183)

A Boundary component is represented as a circle with a T-shaped line sticking to the side facing the actor. Figure 8-4 shows this.

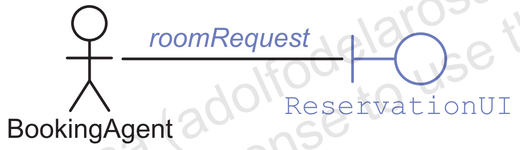


Figure 8-4 An Example Boundary Component

Characteristics of Boundary components include:

- Abstractions of UI screens, sensors, communication interfaces, and so on.

  Boundary components exist to interact with the outside world whether that is a human actor or a machine actor. These components can represent any contact to the outside of the system.
- High-level UI components.

  Boundary components must remain at a high-level. Do not decompose the UI into individual GUI widgets, such as text fields, labels, sliders, radio buttons, and so on.
- Every boundary component must be associated with at least one actor.

Boundary components exist to interact with actors. A Boundary component must be associated with at least one actor, but it can also interact with multiple actors. If a Boundary component is designed, yet is not associated with any actor, then that component serves no purpose.

Service Components

"Control (Service) classes (components) represent coordination, sequencing, transactions, and control of other objects and are often used to encapsulate control related to a specific use case." (Jacobson, Booch, and Rumbaugh page 185)

A Service component is represented as circle with an arrow on the edge of the circle. This is shown in Figure 8-5.

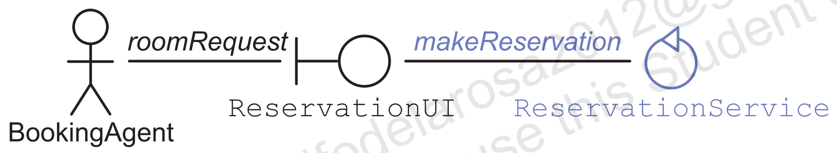


Figure 8-5 An Example Service Component

Jacobson called these components Control classes, but the term *control* can mean several different things. This course uses the term *service* to refer to these components as they relate back to the four fundamental application components discussed in Figure 12-2 "Generic Application Components" on page 12-6. The term *control* or *controller* usually refers to components that handle user actions.

Characteristics of Service components include:

- Coordinate control flow

  Service components have many purposes. However, at the top-level these components usually correspond to use cases of the system. That is to say that Service components manage use case workflows. Service components can also implement common functionality between multiple use cases. Service components can also implement complex algorithms.

- Isolate any changes in workflow from the boundary and entity components

You should separate UI behaviors from workflow behaviors, because the system will be more maintainable if these different behaviors are encapsulated in different components.

Entity Components

"An entity class (component) is used to model information that is long-lived and often persistent." (Jacobson, Booch, and Rumbaugh page 184)

An Entity component is represented as circle with line at the bottom of the circle. Figure 8-6 shows this.

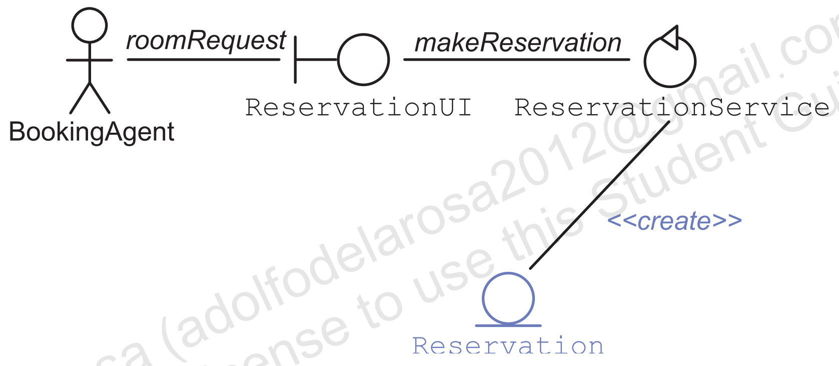


Figure 8-6 An Example Entity Component

Characteristics of Entity components are:

- Entities usually correspond to domain objects.

  There is usually a one-to-one mapping between the Domain model and the Entity components. However, auxiliary entities might be discovered during Robustness analysis.

- Most entities are persistent.

  Entities tend to be persistent objects. This is not a fixed rule; some entities might be created to hold results from other entities.

- Entities can have complex behavior.

  From this description you might think that entities only exist to hold attributes. However, entities can have complex behavior. There is a trade-off between putting this behavior in a Service component or in an Entity component. A general guideline is that if the behavior involves the interaction of multiple entities, then it should probably be in a Service component.

Service and Entity Components

An Entity component will often have a corresponding Service component.

- Each service object will often control its corresponding entity object.
- One service object can delegate control to another more coherent service object.

  Figure 8-7 shows that the `ReservationService` object delegates the task of finding rooms to the `RoomService` object, but manages the `Reservation` object directly.

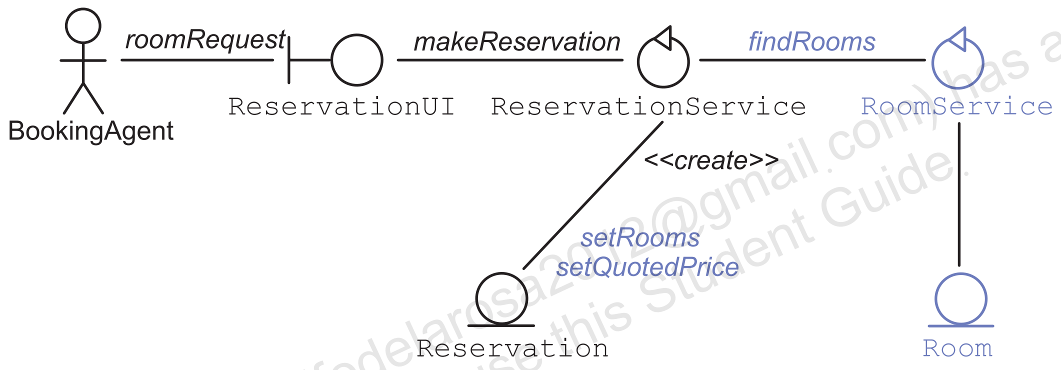


Figure 8-7 An example of coherent services

Boundary And Entity Components

A Boundary component can often retrieve the attributes of an Entity component. Figure 8-8 shows an example of a Boundary object accessing the details from the `Reservation` entity object.

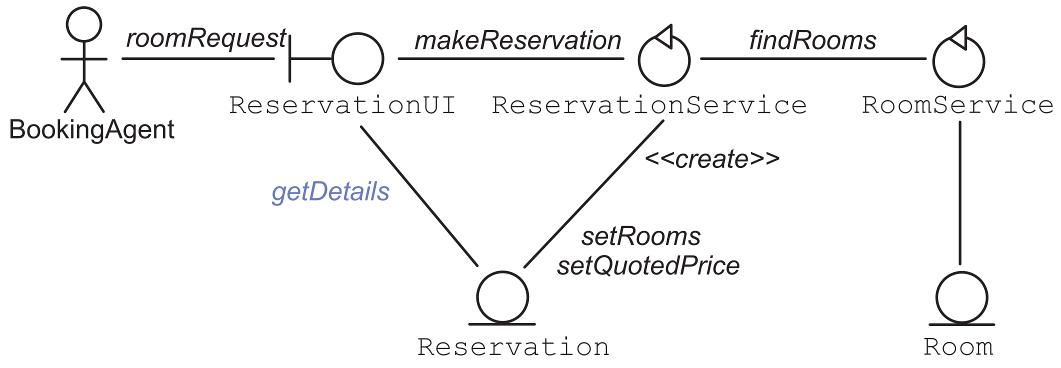


Figure 8-8 An example of a Boundary object accessing data from an Entity object

Describing the Robustness Analysis Process

The Robustness analysis process is as follows:

1. Select a use case.

   Robustness analysis is driven by a use case. Select a single use case to develop into a Design model.

2. Construct a Communication diagram or Sequence diagram that satisfies the activities of the use case.

   Using the use case Activity diagram (or scenarios), analyze each action in the Activity diagram, and perform the following steps:

   a. Identify design components that support the activities of the use case.

      If the action refers to user interaction, create a Boundary component to handle this interaction. If the action refers to some action taken by the system, use a Service component. If the action refers to a responsibility of a Domain object, use an Entity component.

   b. Draw the associations between these components.

      Identify the associations between components to perform the action described in the Activity diagram.

   c. Label the associations with messages.

      Create labels on the associations to describe the messages sent between components for performing the action.

3. Convert the Communication diagram into a Sequence diagram, or vice versa, for an alternate view (optional).

   This step enables you to transform the Communication diagram into a Sequence diagram, which provides a time-ordered view of the messages between the collaborating objects.

This is a very high-level description of the Robustness analysis process. The following sections in this module describe this process in more detail. The next section describes the features of the UML Communication diagram.

Identifying the Elements of a Communication Diagram

A *Communication diagram* represents the objects of a system, their relationships, and the messages sent between objects. Figure 8-9 shows a Communication diagram.

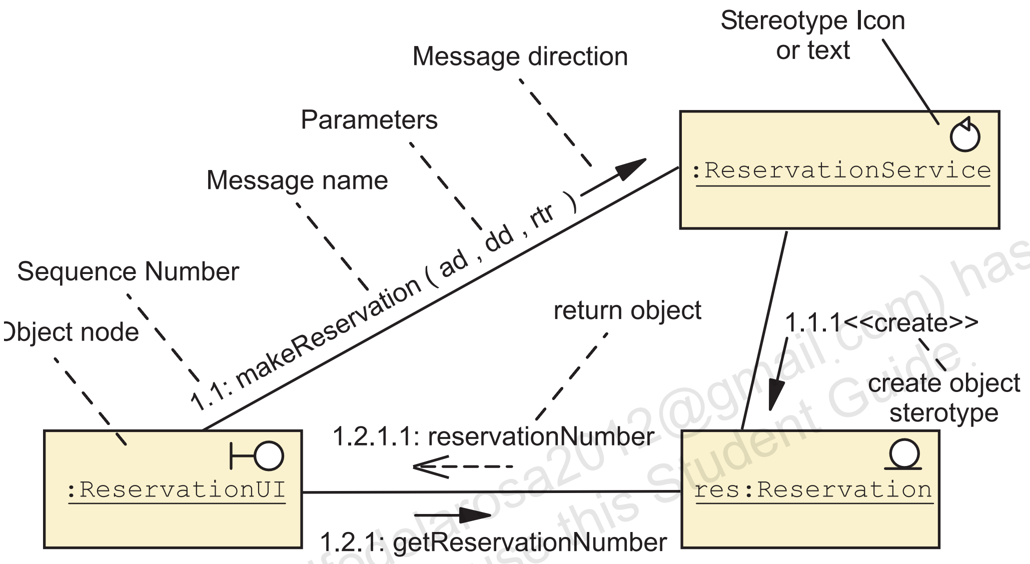


Figure 8-9 An Example of a Communication Diagram

Message arrows show the direction in which the objects collaborate. For example, the arrow labeled "`1.1:makeReservation`" indicates that the `ReservationUI` object sends the `makeReservation` message to the `ReservationService` object.

A Message arrow can indicate:

- A method call – Signified by a solid line and filled arrowhead
- A return from a synchronous method call – Signified by a dashed line and an open arrowhead
- An asynchronous messages – Signified by a solid line and an open arrowhead

A Sequence label indicates the following:

- The order of the message

  The first portion of a label indicates a hierarchical order in which the message is issued in the communication. For example, in the label "`1.1.1:«create»`", "`1.1.1`" indicates the order in which this message is sent. The «`create`» message is sent after the `makeReservation` message, but before the `getReservationNumber` message.

- The activity the message will invoke

  The second portion of a label indicates the actual message that is being invoked on the receiving object. For example, in the label "`1.1:makeReservation(ad,dd,rtr)`", "`makeReservation`" indicates that the `makeReservation` method is invoked on the `ReservationService` object.

In Robustness analysis, you will see a variation in the Communication diagrams such that the object node stereotypes are used as icons for the node. Figure 8-10 shows an example; this diagram is similar to the diagram in Figure 8-9 on page 8-12.

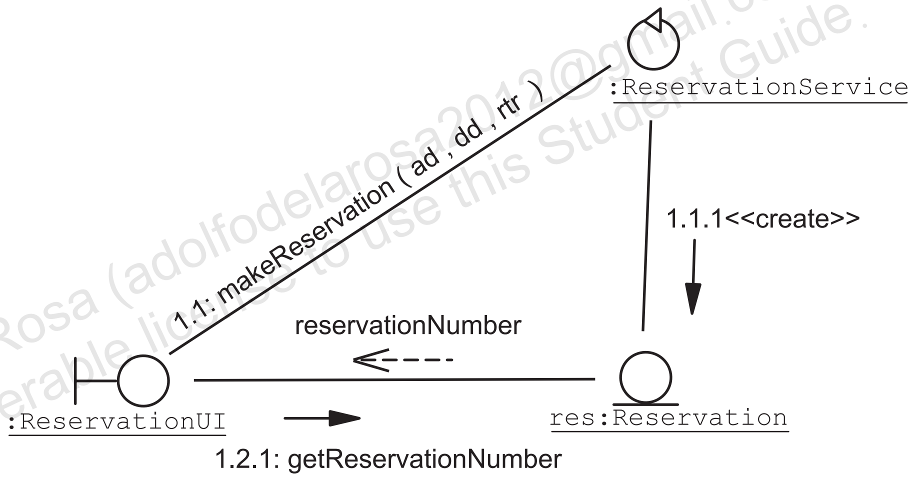


Figure 8-10 A Variation Using Stereotype Icons

Creating a Communication Diagram

Robustness analysis can be performed by completing the following steps on a selected use case.

1. Place the actor in the Communication diagram.
2. Analyze the Use Case form or the Activity diagram for the use case. For every action in the use case:
   a. Identify and add a Boundary component.
   b. Identify and add a Service component.
   c. Identify and add an Entity component.
   d. Identify and add further Methods in the Service and Entity components to satisfy the interactions in the use case.

The following sections describe the steps to create a Communication diagram for the "Create a Reservation" use case.

Step 1– Place the Actor in the Diagram

Every use case starts with the actor that uses the use case. For example, in the "Create Reservation" use case the BookingAgent is one of the actors for this use case. Figure 8-11 shows that you should place the BookingAgent actor icon at the far left of the Communication diagram.

BookingAgent

Figure 8-11 Step 1 – Place the Actor in the Communication Diagram

Step 2a – Identify Boundary Components

By analyzing an action in the Flow of Events, you must identify the three types of design components. Boundary components are fairly easy to identify. These components enable the actor to interact with the system. Actions that have the actor do something with the system requires one or more Boundary components. For example, the "BookingAgent enters the arrival date, departure date, requested types of rooms, and then submits these to the boundary in 'roomRequest' message. Figure 8-12 shows this.

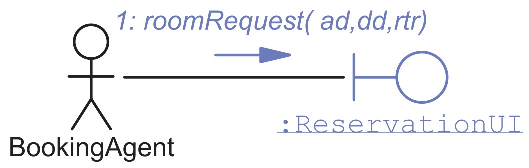


Figure 8-12 Step 2a – BookingAgent Interacts With the ReservationUI Component

This diagram shows an association between the actor and the ReservationUI component. The message arrow and label over the association indicates the action and its sequence within the Flow of Events. Note that these messages are not specified by the use case Flow of Events. You must identify how the system satisfies the actions in the use case. This requires imagination and a deep understanding of the purpose of the use case. A User Interface prototype can also be helpful during Robustness analysis because the Boundary components are already defined by the screens of the UI prototype.

Sanity checking of the data is usually performed in the boundary component. For example the dates will be checked for invalid days, months, and years.

Step 2b – Identify Service Components

The ReservationUI components need to delegate the request to business methods which are modeled with a Service component. For example, the roomRequest method in ReservationUI calls the `makeReservation` method in the ReservationService component, as Figure 8-13 shows.

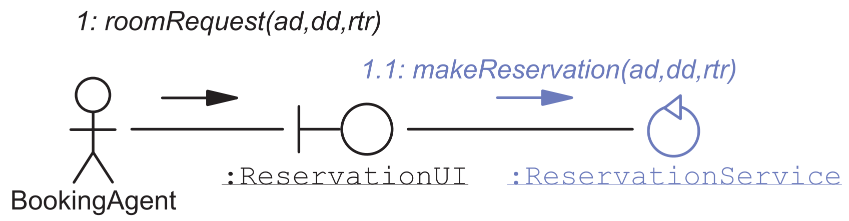


Figure 8-13 Step 2b – The ReservationUI Component delegates some of its behavior to a business method in the `ReservationService` Component

This ReservationService component is fairly generic. It will support most operations for the “Create a Reservation” use case. However, you can also have ReservationUI component interact with other services as necessary.

Step 2c – Identify Entity Components

Eventually, the system needs to create and manipulate entities. These entities are modeled with Entity components. For example, the ReservationService component will create a Reservation Entity component and passes the arrival date, departure date, and requested type of rooms, as Figure 8-14 shows.

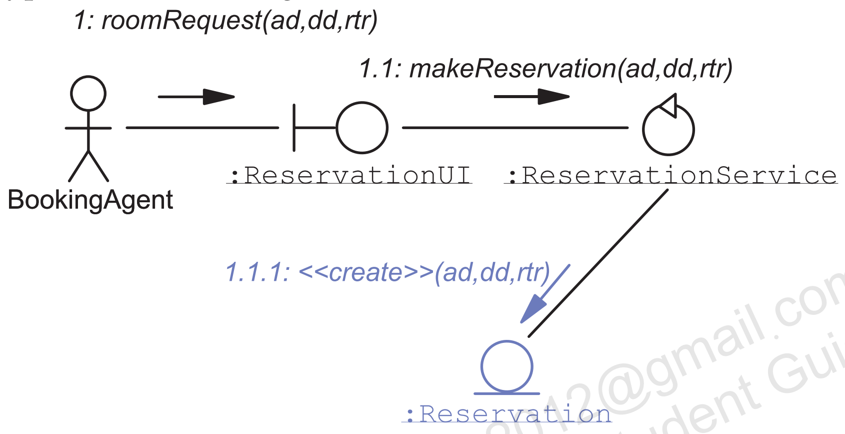

1: roomRequest(ad,dd,rtr)
1.1: makeReservation(ad,dd,rtr)
BookingAgent
:ReservationUI
:ReservationService
1.1.1: <<create>>(ad,dd,rtr)
:Reservation

Figure 8-14 Step 2c – The ReservationService Component creates the Reservation Entity Component

Note – At this conceptual stage, the Design model does not effectively represent how the system will interact with a database. For now, the service components will appear to perform all database operations.

Step 2d – Identify Additional Interactions

The previous pages demonstrated Robustness analysis for a single action in the Flow of Events for the use case. Continue this process for every action in the Flow of Events. This diagram will elaborate the Design model to completely handle the use case.

The makeReservation method uses the findRoom method to find rooms of the required type. After finding and choosing free rooms (not shown in Figure 8-15), the setRooms method assigns the rooms to the Reservation entity. The calculated price (not shown in Figure 8-15) is assigned to the Reservation as shown in Figure 8-15

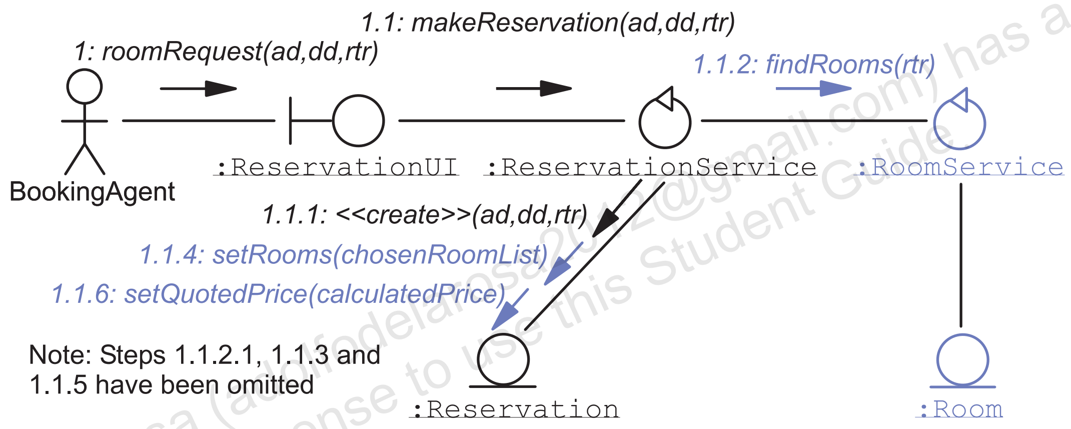
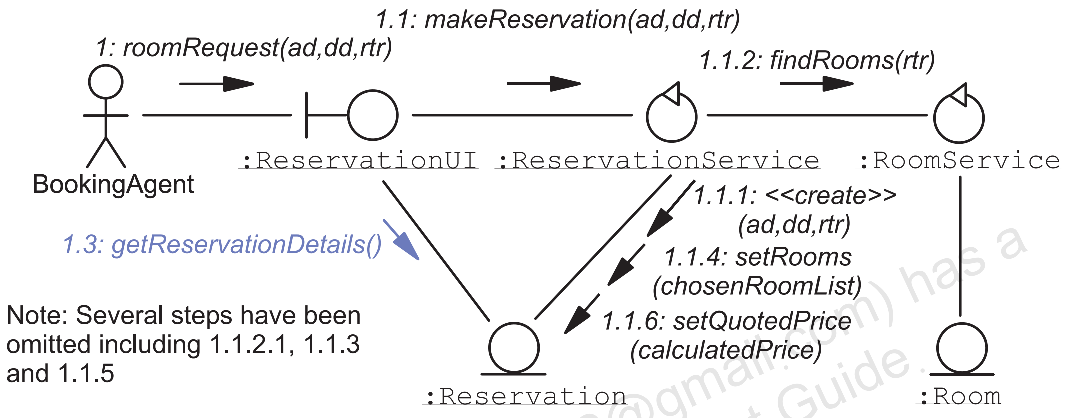

1.1: makeReservation(ad,dd,rtr)
1: roomRequest(ad,dd,rtr)
1.1.2: findRooms(rtr)
BookingAgent
:ReservationUI
:ReservationService
:RoomService
1.1.1: <<create>>(ad,dd,rtr)
1.1.4: setRooms(chosenRoomList)
1.1.6: setQuotedPrice(calculatedPrice)
Note: Steps 1.1.2.1, 1.1.3 and 1.1.5 have been omitted
:Reservation
:Room

Figure 8-15 Additional Interactions between Services and Entity Components

The makeReservation method of the ReservationService object returns the Reservation object (not shown in Figure 8-16) to the ReservationUI. The roomRequest method of ReservationUI then calls the getReservationDetails method of the Reservation object, which returns the details of the reservation. These will then be notified to the booking agent (not shown in Figure 8-16), as shown in Figure 8-16.



1.1: makeReservation(ad,dd,rtr)
1: roomRequest(ad,dd,rtr)
1.1.2: findRooms(rtr)
BookingAgent
:ReservationUI
:ReservationService
:RoomService
1.1.1: <<create>> (ad,dd,rtr)
1.3: getReservationDetails()
1.1.4: setRooms (chosenRoomList)
Note: Several steps have been omitted including 1.1.2.1, 1.1.3 and 1.1.5
1.1.6: setQuotedPrice (calculatedPrice)
:Reservation
:Room

Figure 8-16 Additional Interactions between Boundary and Entity Components

Communication Diagram Examples

The following two Communication diagrams show a more detailed view of the CreateReservation:

1. Figure 8-17 shows a Primary (successful) scenario.

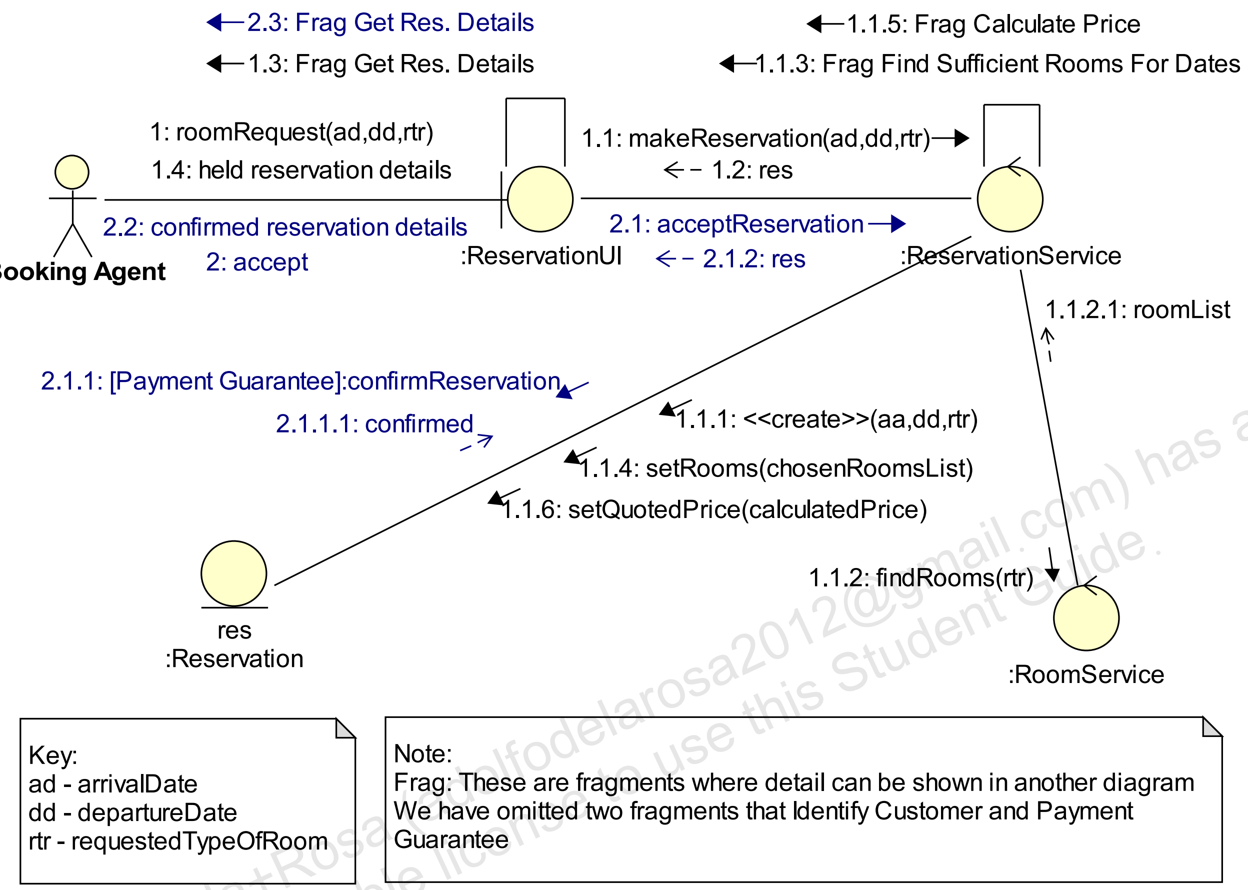

2.3: Frag Get Res. Details
1.3: Frag Get Res. Details
1.1.5: Frag Calculate Price
1.1.3: Frag Find Sufficient Rooms For Dates
1: roomRequest(ad,dd,rtr)
1.4: held reservation details
2.2: confirmed reservation details
2: accept
Booking Agent
:ReservationUI
1.1: makeReservation(ad,dd,rtr)
1.2: res
2.1: acceptReservation
2.1.2: res
:ReservationService
1.1.2.1: roomList
2.1.1: [Payment Guarantee]:confirmReservation
2.1.1.1: confirmed
1.1.1: <<create>>(aa,dd,rtr)
1.1.4: setRooms(chosenRoomsList)
1.1.6: setQuotedPrice(calculatedPrice)
1.1.2: findRooms(rtr)
res
:Reservation
:RoomService
Key:
ad - arrivalDate
dd - departureDate
rtr - requestedTypeOfRoom
Note:
Frag: These are fragments where detail can be shown in another diagram
We have omitted two fragments that Identify Customer and Payment Guarantee

Figure 8-17 Example of a Primary (successful) Scenario Communication Diagram

2. Figure 8-18 shows a Secondary (unsuccessful) scenario, where Rooms offered are rejected by the booking agent:

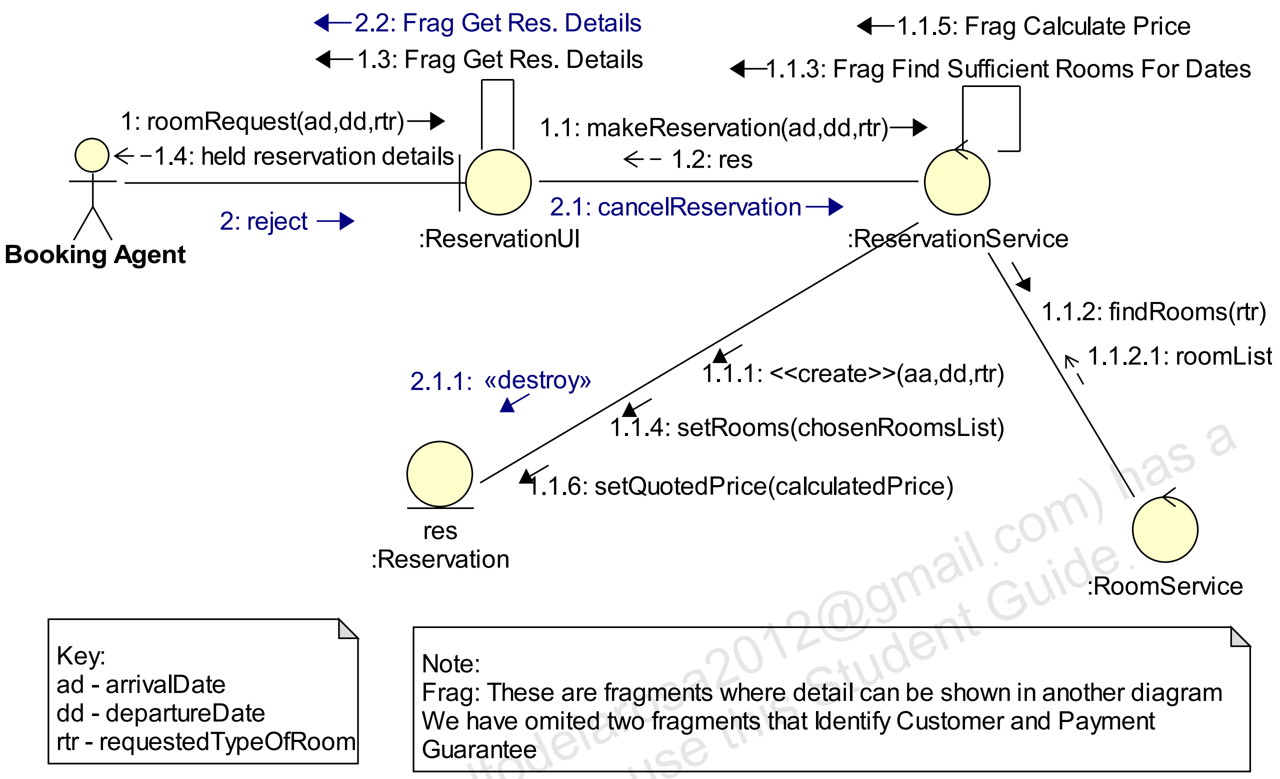

2.2: Frag Get Res. Details
1.3: Frag Get Res. Details
1.1.5: Frag Calculate Price
1.1.3: Frag Find Sufficient Rooms For Dates
1: roomRequest(ad,dd,rtr)
1.4: held reservation details
1.1: makeReservation(ad,dd,rtr)
1.2: res
Booking Agent
2: reject
:ReservationUI
2.1: cancelReservation
:ReservationService
1.1.2: findRooms(rtr)
1.1.2.1: roomList
2.1.1: «destroy»
1.1.1: <<create>>(aa,dd,rtr)
1.1.4: setRooms(chosenRoomsList)
1.1.6: setQuotedPrice(calculatedPrice)
res
:Reservation
:RoomService
Key:
ad - arrivalDate
dd - departureDate
rtr - requestedTypeOfRoom
Note:
Frag: These are fragments where detail can be shown in another diagram
We have omited two fragments that Identify Customer and Payment Guarantee

Figure 8-18 An Example of a Secondary (unsuccessful) Scenario Communication Diagram

Sequence Diagrams

Sequence diagrams are UML diagrams that:

- Provide a different perspective on the interaction between objects
- Can be used instead of Communication diagrams
- Can be converted to or from a Communication diagram
- Prove to be more useful for developers
- Highlight the time ordering of the interactions

The next section describes UML Sequence diagrams.

Identifying the Elements of a Sequence Diagram

A Sequence diagram is "A diagram that shows object interactions arranged in time sequence." (UML v1.4, page B-17)

A *Sequence diagram* represents the objects of a system and the messages sent between objects organized in a time-ordered fashion. Figure 8-19 shows an example Sequence diagram.

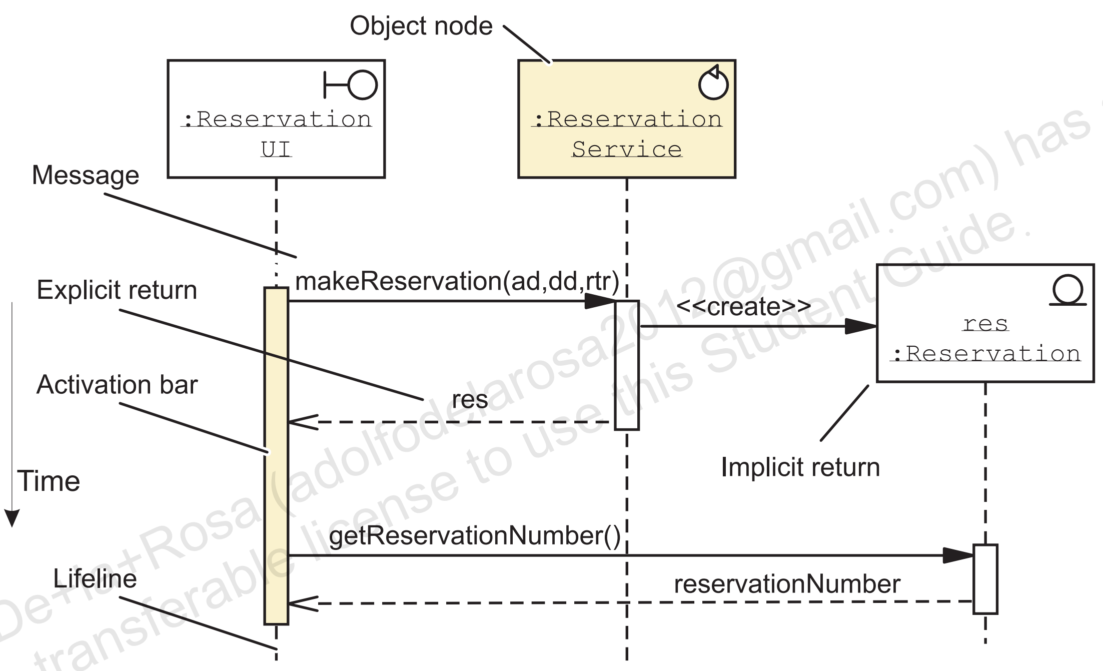


Figure 8-19 An Example Sequence Diagram

The collaborating objects are organized along the top of the diagram (creating columns). Time is organized from the to the bottom of the diagram.

Under each object is a dashed line that indicates the duration or the *life* of the object. If the object exists throughout the duration of the activity represented by the Sequence diagram, then the lifeline should extend through the bottom of the diagram.

The activation bar indicates that the object is engaged in some activity. This usually occurs when a message is sent to the object. For example, the `makeReservation` message is sent to the `ReservationService` object. An activation bar starts at the point where the message arrow arrives at the object and extends down (in time) to the point when the `ReservationService` object has completed processing that message. A dashed line with a stick arrow provides an explicit indication that a message returns a value. Alternatively, you can leave out the return arrow; in this case the end of the activation bar indicates that the message processing is complete.

Fragments

Sequence diagrams support a Fragment notation. The uses of fragments include:

- Showing sequence loops
- Showing alternative paths
- Allowing two or more scenarios to be shown on one diagram
- Showing a reference to another detailed Sequence diagram fragment
- Allowing you to break up a large diagram into several smaller diagrams

Figure 8-20 shows examples of the Fragment notation.

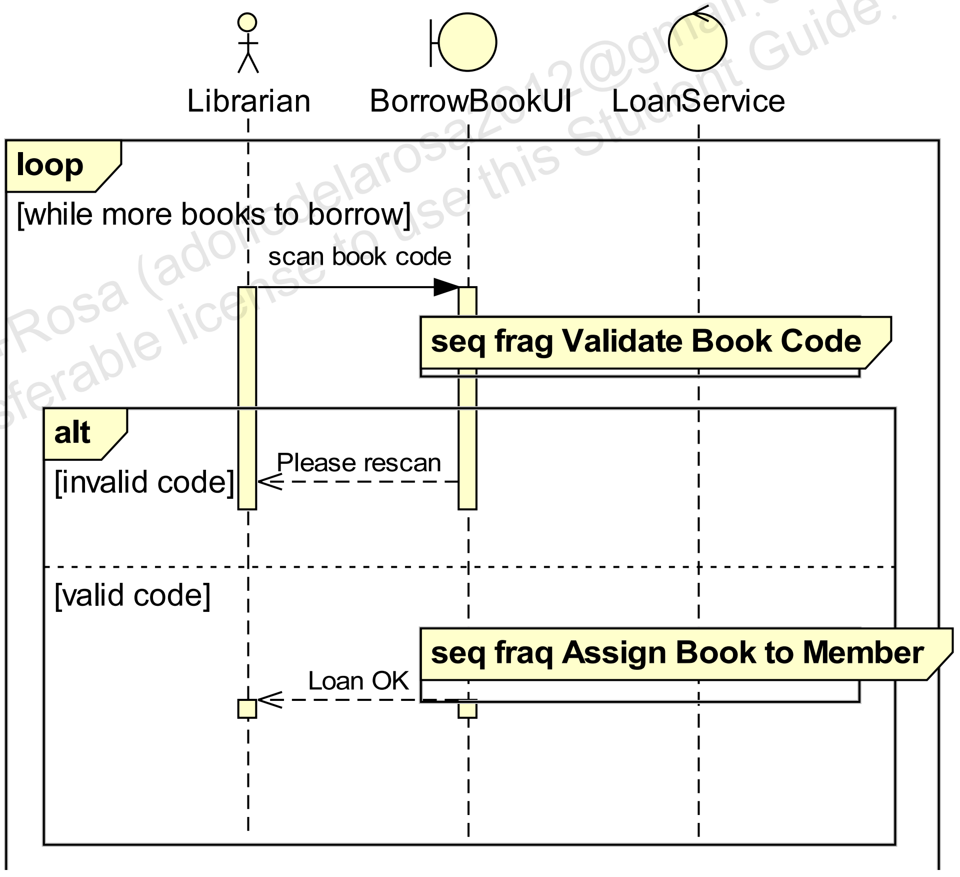


Figure 8-20 An Example Showing a Loop, an Alt, and a Reference Fragment

Sequence Diagram Examples

The following three Sequence diagrams show a more detailed view of the CreateReservation:

1. Figure 8-21 shows the primary (successful) scenario and an secondary (unsuccessful) scenario in one diagram using an *alt* fragment.

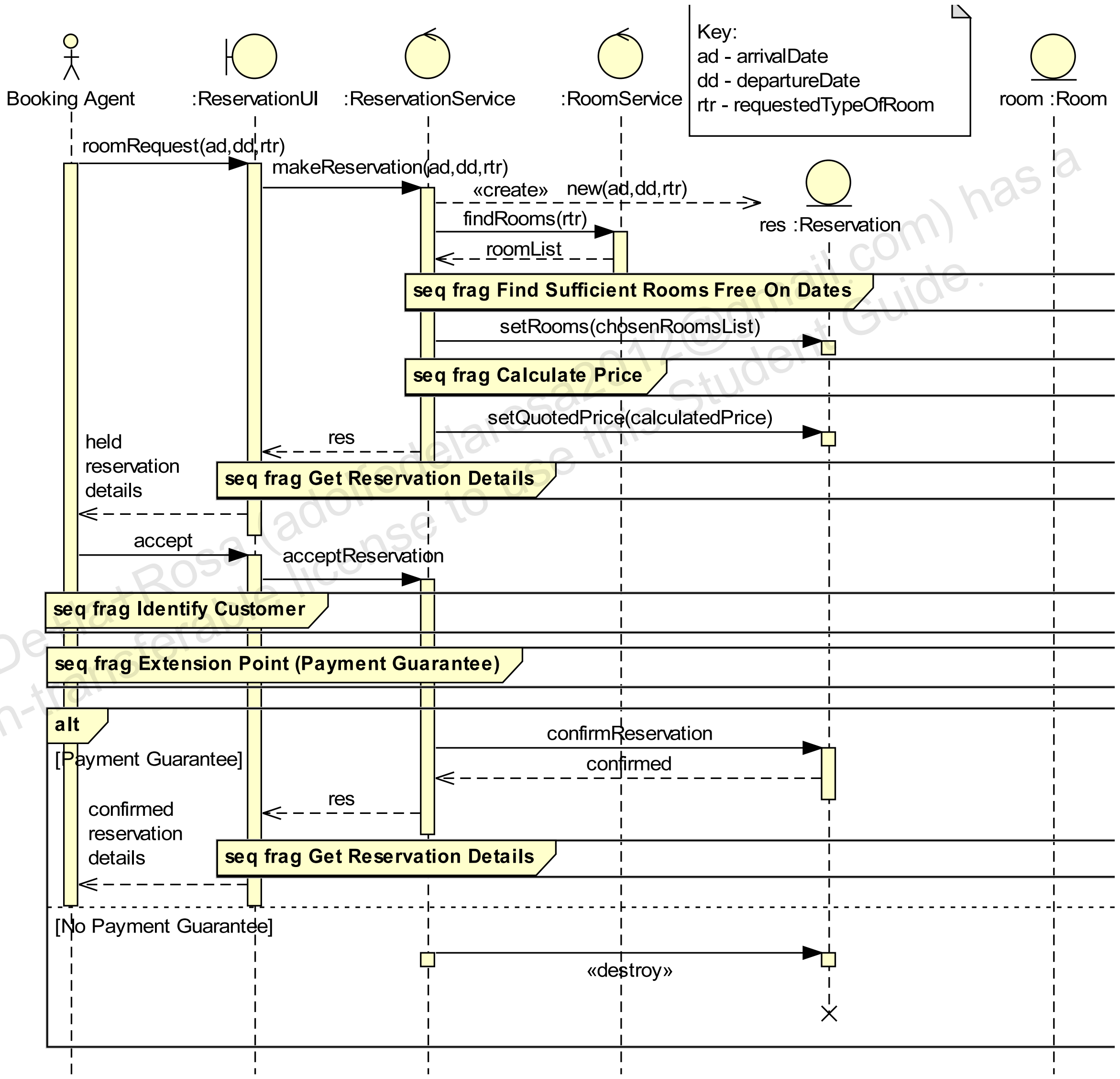


Figure 8-21 Example of a Primary (successful) and Secondary (unsuccessful) scenarios on the same diagram

2. Figure 8-22 shows a secondary (unsuccessful) scenario, where the rooms offered are rejected by the booking agent.

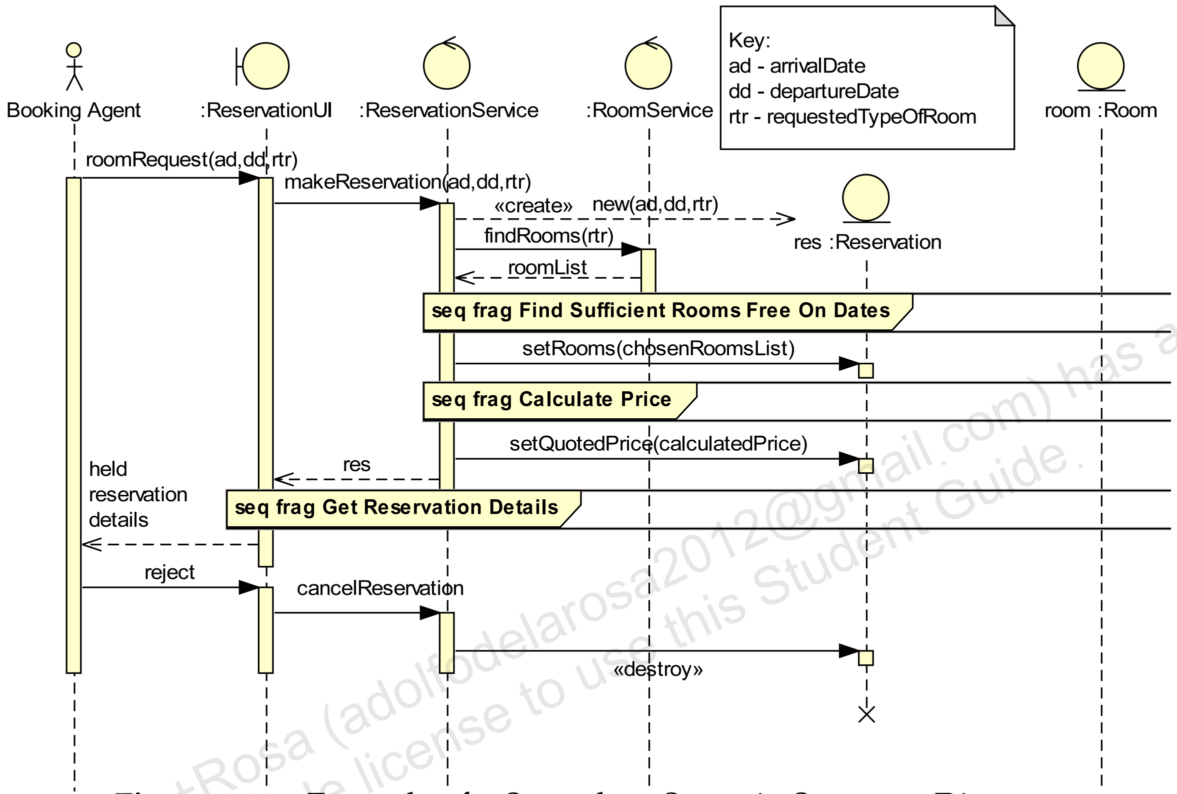


Figure 8-22 Example of a Secondary Scenario Sequence Diagram

3. Figure 8-23 shows a Fragment Sequence diagram that is referenced from both Figure 8-21 and Figure 8-22. It shows the finer details of the GetReservationDetails fragment and includes a *loop* fragment.

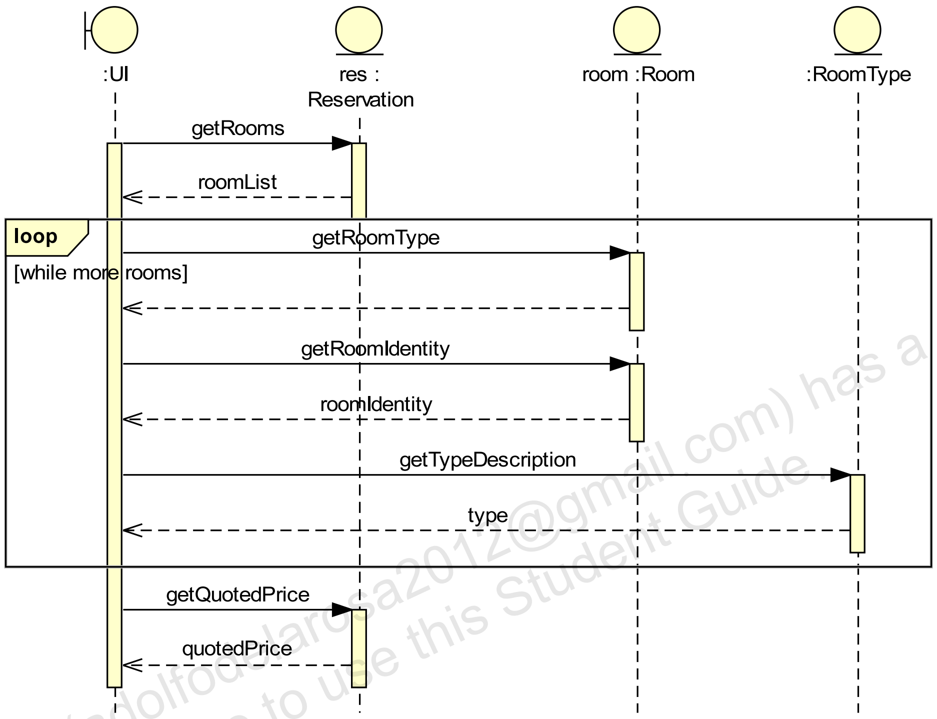


Figure 8-23 Example of a Sequence Diagram Fragment

Summary

In this module, you were introduced to Robustness analysis and the Design model. The following lists a few important concepts that were covered:

- Interaction diagrams are used to identify design components that satisfy a use case.
- Object interactions can be visualized with a UML:
  - Communication diagram
  - Sequence diagram.