TOPIC 6

BEHAVIORAL MODELLING

(SEBAGIAN MATERI DIADOPSI DARI MATA KULIAH REKAYASA PERANGKAT LUNAK)

ANALISIS DAN PERANCANGAN SISTEM INFORMASI

CSIM603183

Learning Objectives

1. Able to explain the rules and component of sequence and communication diagram
2. Able to explain how to create sequence and communication diagram
3. Able to create sequence dan communication diagrams
4. Able to explain the correlation among behavioral, structural, and functional model

Outline

1. Sequence Diagram
2. Communication Diagram
3. Behavioral State Machine
4. CRUDE Analysis

1.1 BEHAVIORAL MODEL

Introduction

Remember that models in analysis phase can be classified into:

- **Functional Model**: represent system behavior
  - Use Case Diagram, Activity Diagram
- **Structural Model** (Conceptual Model): represent system objects and their relationships (People, Places, Things)
  - Class Diagram, Object Diagram
- **Behavioral Model**: represent internal behavior of a system
  - Interaction Diagram, Behavioral State Machines

We will focus on the Interaction Diagram (Sequence Diagram & Communication Diagram)

Introduction

- **Behavioral models**
  - describe the **internal dynamic aspects** of an information system that supports the business processes in an organization.
  - represent the **internal behavior or dynamic view** of an information system.
- During analysis, behavioral models describe what the **internal logic of the processes** is without specifying how the processes are to be implemented.
  - **Later**, in the design and implementation phases, the detailed design of the operations contained in the object is fully specified.

2 Types of Behavioural Model

- First, there are behavioral models used to represent the **underlying details of a business process portrayed by a use-case model.**
  - Interaction diagrams (**sequence and communication diagrams**) are used for this type of behavioral model.
  - Interaction diagrams to show how actors and objects collaborate to provide the functionality defined in a use case.
    - Interaction diagrams allow the analyst to model the distribution of the behavior of the system over the actors and objects in the system.
- Second, a behavioral model is used to represent **the changes that occur in the underlying data.**
  - **Behavioral state machines**.

Behavioural Models

- Analysts view the problem as a set of use cases (functional models) supported by a set of collaborating objects (structural models)
  - Behavioral models depict this view of the business processes:
    - How the objects interact and form a collaboration to support the use cases
    - An internal view of the business process described by a use case
- Creating behavioral models is an iterative process which may induce changes in other models

1.2 INTERACTION DIAGRAM

Class Diagrams VS Interaction Diagrams

One of the primary differences between class diagrams and interaction diagrams:

- Class diagram describes **structure** and interaction diagram describes **behavior**
- **A class diagram focus on the class level**, whereas the **interaction diagrams focus on the object level.**

Key definitions of Interaction Diagram Components

1. **An object** is an instantiation of a class, i.e. an actual person, place, or thing about which we want to capture the information.
   - Example of classes might include doctor, patient, and appointment.
   - The specific patients, such as Jim Maloney, Mary Wilson, and Theresa Marks are considered objects—i.e. instances of the patient class.
2. **Attributes**, describes information about the object, such as a patient's name, birth date, address, and phone number.
3. **Operations** describes the behaviors of an instance of a class or an action that an object can perform.
4. **Messages** represent information sent to objects to tell them to execute one of their behaviors

Types of Interaction Diagrams

1. Sequence Diagrams - emphasize message sequence
2. Communication Diagrams - emphasize message flow

1.3 SEQUENCE DIAGRAM

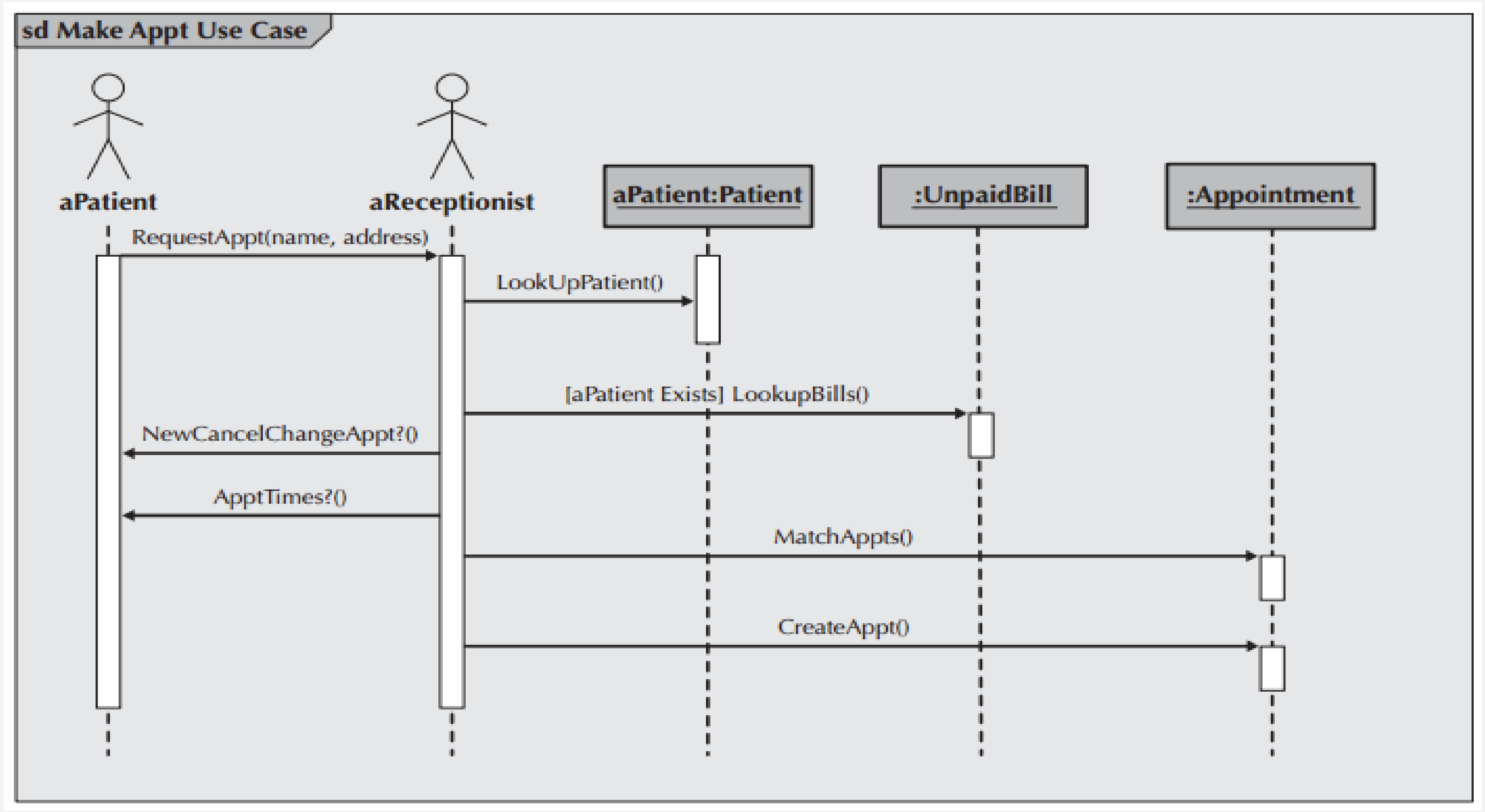
Sequence Diagram

- Illustrate the **objects that participate in a use case** and the **messages** that pass between them over time for one use case.
- Represent **dynamic model** that shows the **explicit sequence of messages** that are passed between objects in a defined interaction.
- Because sequence diagrams emphasize the **time-based ordering** of the activity that takes place among a set of objects, they are very helpful for understanding real-time specifications and complex use cases.

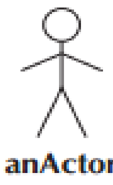
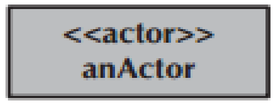
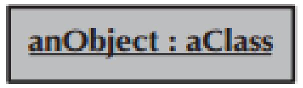
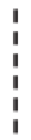
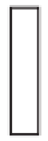
Sequence Diagram

- The sequence diagram can be a **generic** sequence diagram that shows all possible scenarios for a use case, but usually each analyst develops a set of instance sequence diagrams, each of which depicts **a single scenario within the use case**.
- If we are interested in understanding the flow of control of a scenario by time, we should use a sequence diagram to depict this information.
- The diagrams are used throughout the **analysis and design** phases.
- However, the design diagrams are very **implementation specific,** often including database objects or specific user interface components as the objects.

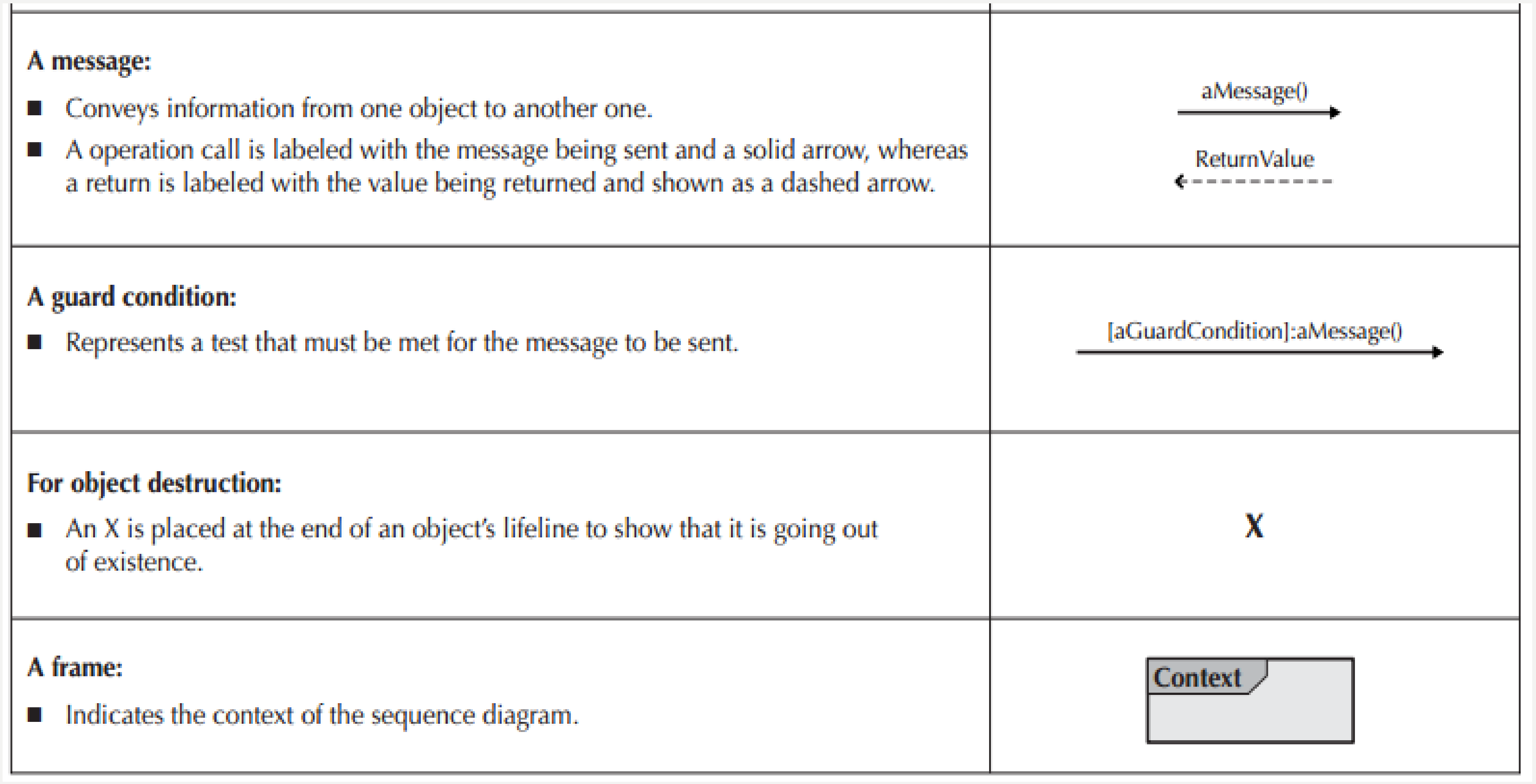
Sequence Diagram Example



Sequence Diagram Syntax

Term and Definition	Symbol
An actor: ■ Is a person or system that derives benefit from and is external to the system. ■ Participates in a sequence by sending and/or receiving messages. ■ Is placed across the top of the diagram. ■ Is depicted either as a stick figure (default) or, if a nonhuman actor is involved, as a rectangle with <<actor>> in it (alternative).	 
An object: ■ Participates in a sequence by sending and/or receiving messages. ■ Is placed across the top of the diagram.	
A lifeline: ■ Denotes the life of an object during a sequence. ■ Contains an X at the point at which the class no longer interacts.	
An execution occurrence: ■ Is a long narrow rectangle placed atop a lifeline. ■ Denotes when an object is sending or receiving messages.	

Sequence Diagram Syntax



A message: ■ Conveys information from one object to another one. ■ A operation call is labeled with the message being sent and a solid arrow, whereas a return is labeled with the value being returned and shown as a dashed arrow.	aMessage() ReturnValue
A guard condition: ■ Represents a test that must be met for the message to be sent.	[aGuardCondition]:aMessage()
For object destruction: ■ An X is placed at the end of an object's lifeline to show that it is going out of existence.	**X**
A frame: ■ Indicates the context of the sequence diagram.	Context

Actor and Object

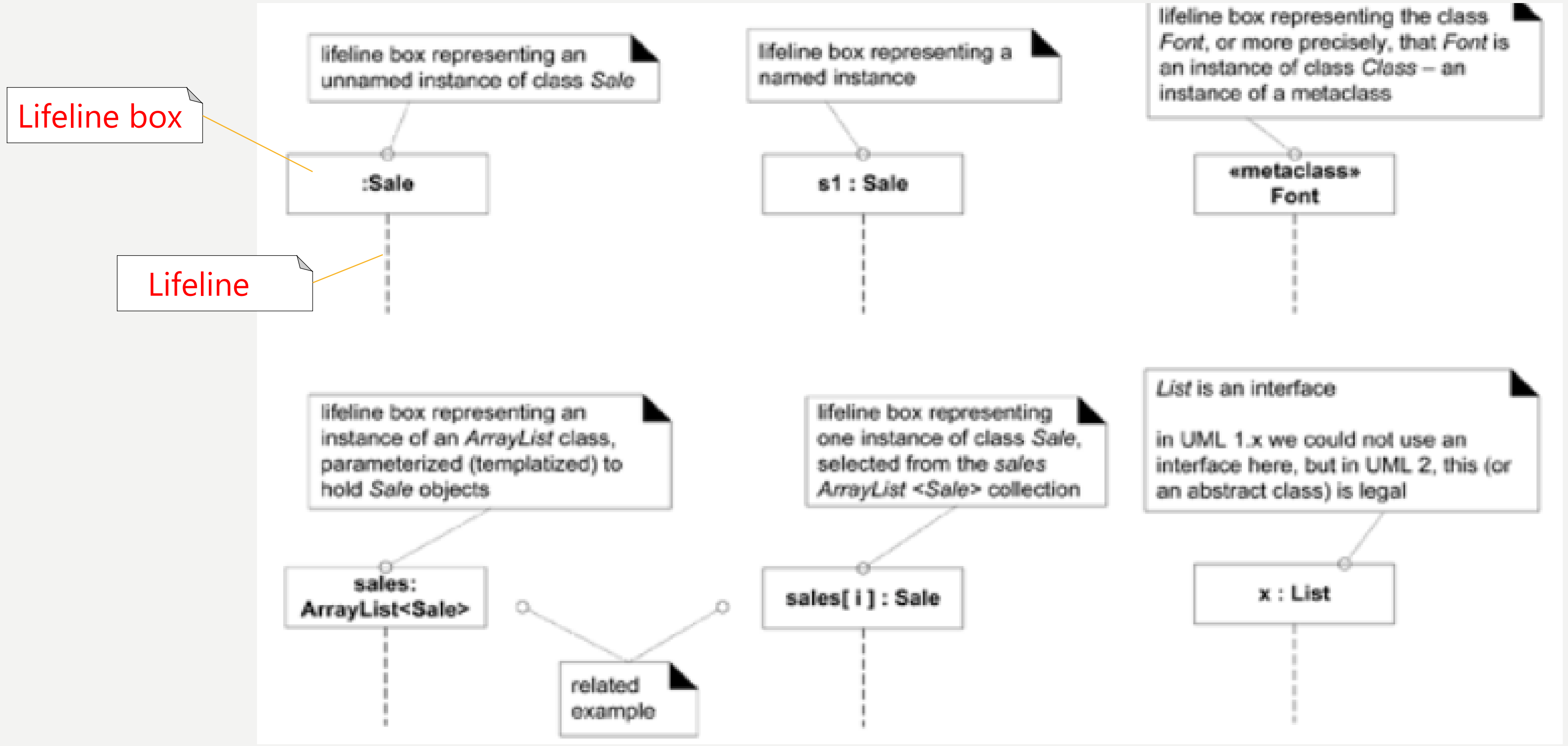
- Actors and objects that participate in the sequence are placed across the top of the diagram using actor symbols from the **use-case diagram** and object symbols from the **object diagram**
- Notice in the example before:
  - The actors are aPatient, and aReceptionist;
  - The objects are aPatient : Patient, aPatient : UnpaidBill, aPatient : Appointment.
- For **each of the objects**, the name of the class of which they are an instance is given after the object's name (e.g., aPatient means that aPatient is an instance of the Patient class).

A Lifeline

- A dotted line runs vertically below each actor and object to denote ***the lifeline*** of the actors and objects over time (see Figure 6-1).
- Sometimes an object **creates a temporary object**; in this case, an X is placed at the end of the lifeline at the point where the object is destroyed (not shown).
  - For example, think about a shopping cart object for a Web commerce application.
  - The shopping cart is used for temporarily capturing line items for an order, but once the order is confirmed, the shopping cart is no longer needed.
  - In this case, an X would be located at the point at which the shopping cart object is destroyed.
- When objects continue to exist in the system after they are used in the sequence diagram, then the lifeline continues to the bottom of the diagram.

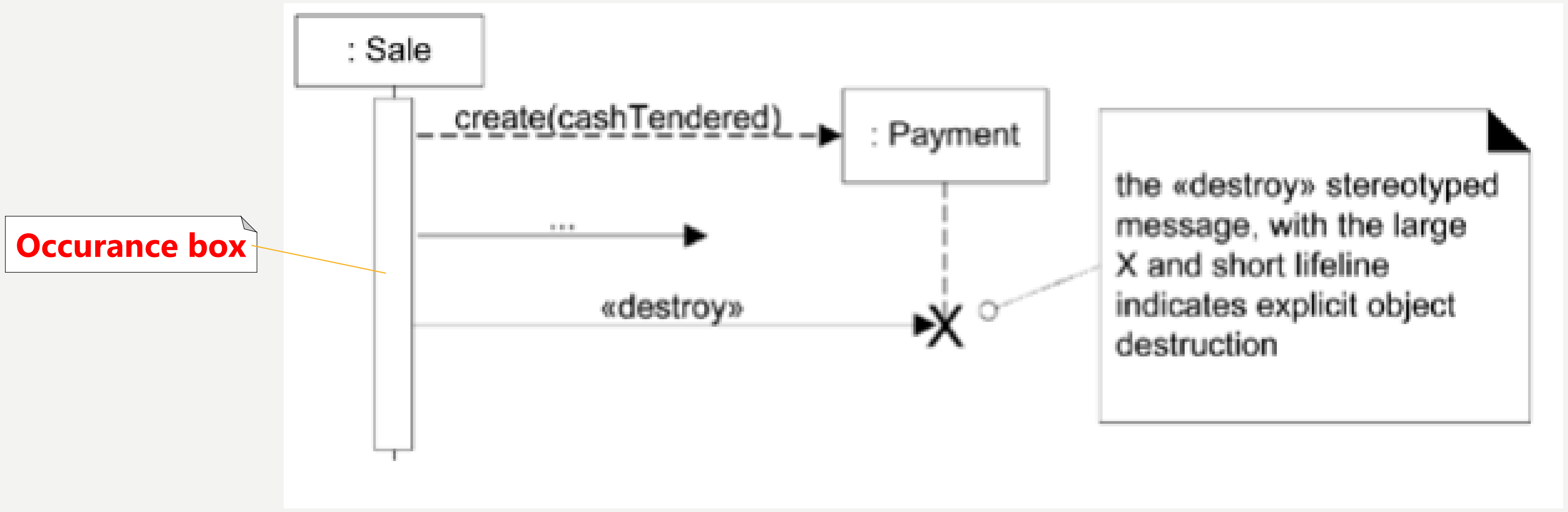
Lifeline boxes and lifelines

An object do not have a lifeline until they are created


Lifeline box
Lifeline
lifeline box representing an unnamed instance of class Sale
:Sale
lifeline box representing a named instance
s1 : Sale
lifeline box representing the class Font, or more precisely, that Font is an instance of class Class – an instance of a metaclass
«metaclass» Font
lifeline box representing an instance of an ArrayList class, parameterized (templatized) to hold Sale objects
sales: ArrayList<Sale>
related example
lifeline box representing one instance of class Sale, selected from the sales ArrayList <Sale> collection
sales[i] : Sale
List is an interface
in UML 1.x we could not use an interface here, but in UML 2, this (or an abstract class) is legal
x : List

Activation (Execution Occurrence) and Object Destruction

- The period of time an object is handling a message
- The end of an object's life is marked with an "X" at the end of the lifeline (Object Destruction)

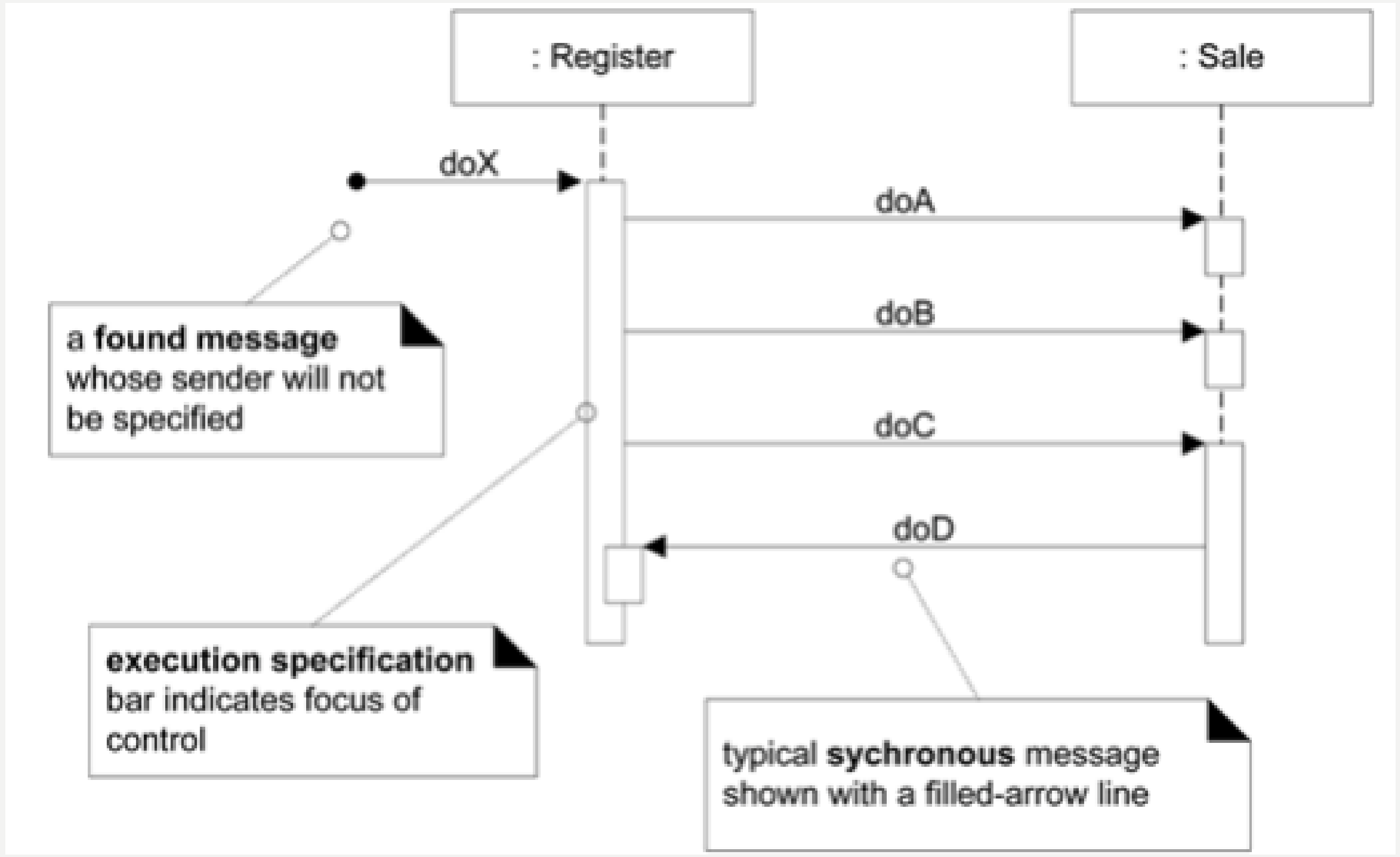

: Sale
create(cashTendered)
: Payment
...
«destroy»
the «destroy» stereotyped message, with the large X and short lifeline indicates explicit object destruction
Occurance box

Message

- A message is a communication between objects that conveys information with the expectation that activity will ensue. **Two types of messages are typically used: operation call and return.**
- **Operation call messages** passed between objects are shown using solid lines connecting two objects with an arrow on the line showing which way the message is being passed.
  - The order of messages goes from the top to the bottom of the page, so messages located higher on the diagram represent messages that occur earlier on in the sequence, versus the lower messages that occur later.
- **A return message** is depicted as a dashed line with an arrow on the end of the line portraying the direction of the return.
  - The information being returned is used to label the arrow.
  - However, because adding return messages tends to clutter the diagram, unless the return messages add a lot of information to the diagram, they can be omitted (esp. for void return value).

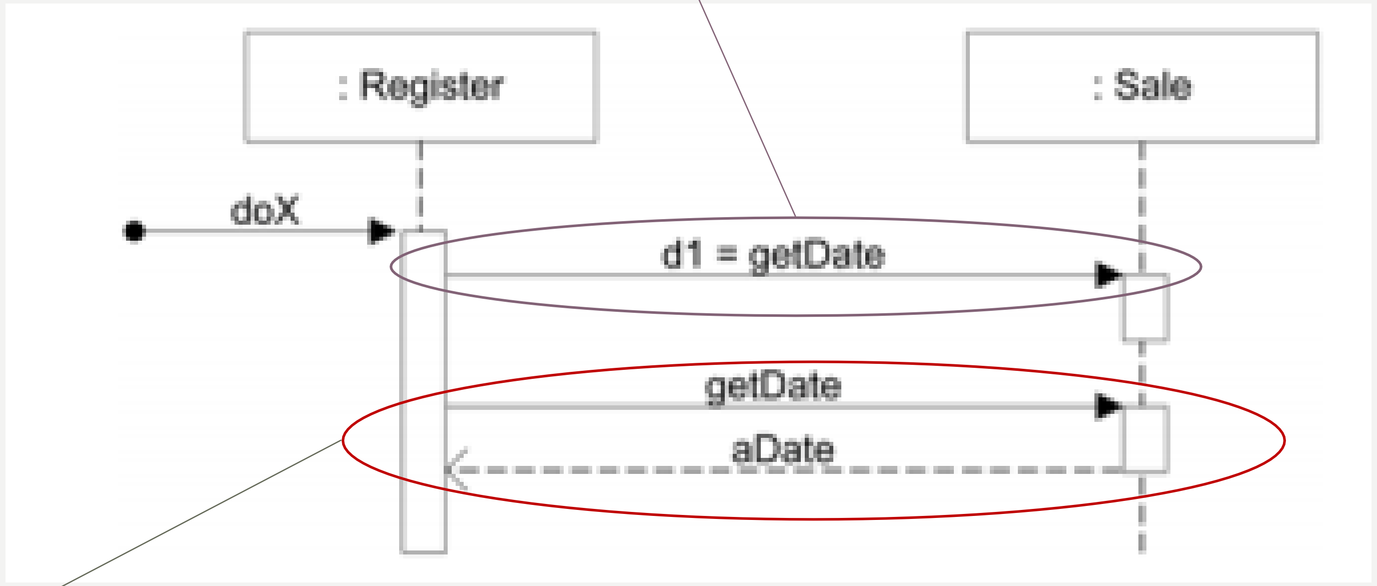
Message

- Represented with a message expression on an arrowed line between objects


: Register
: Sale
doX
doA
doB
doC
doD
a found message whose sender will not be specified
execution specification bar indicates focus of control
typical sychronous message shown with a filled-arrow line

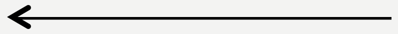
Reply or Return messages

If return message is not drawn, write the return variable in the sent message: *returnVar=message(parameter)*

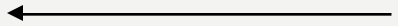


If return message is drawn (with the return variable written in it), draw it at the end of an activation bar

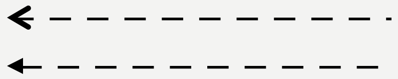
Synchronous v.s Asynchronous Message



- An **asynchronous message** has an open arrow head



- A **synchronous message** has a filled arrow head

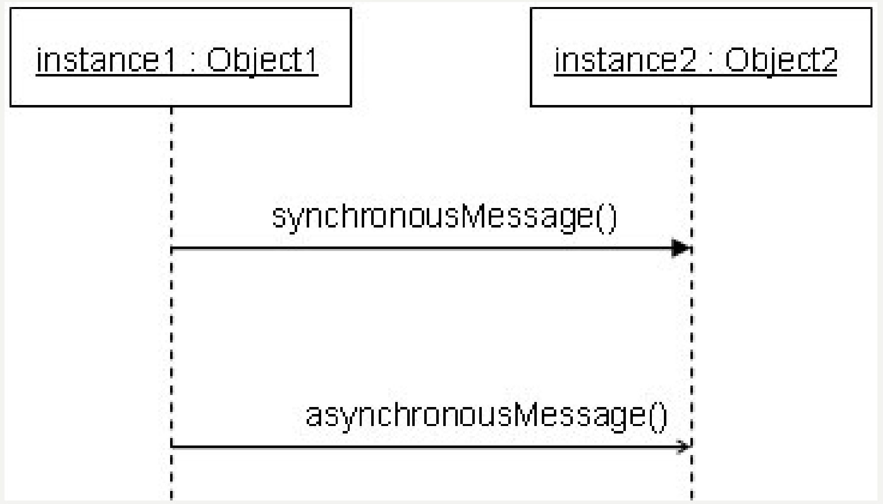


- A **reply message (return value)** has a dashed line with either an open or filled arrow head

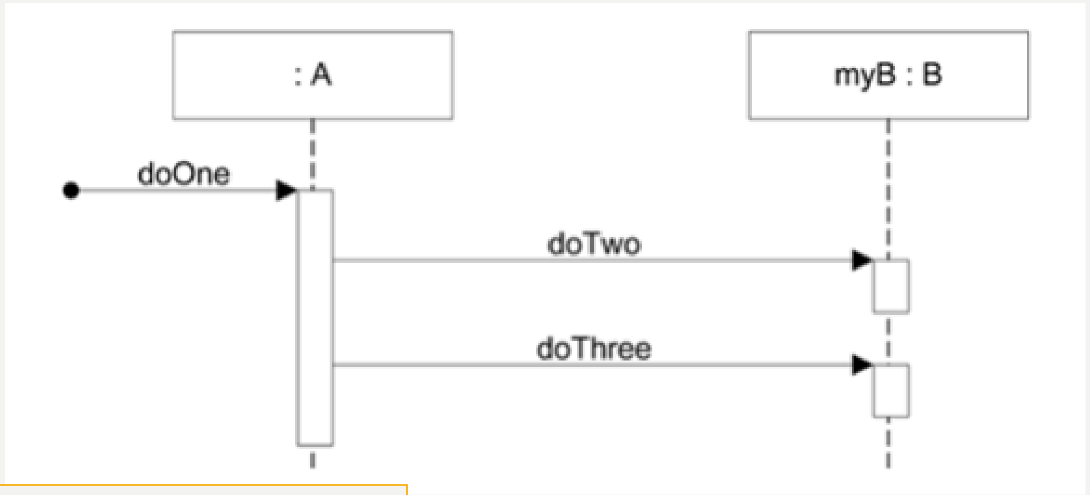
http://www.omg.org/spec/UML/2.3/

Synchronous v.s Asynchronous Message

- Use **asynchronous** messages when:
  - The sender does not wait for the receiver to finish processing the message, it continues immediately.
  - Messages sent to a receiver in another process or calls that start a new thread are examples of asynchronous messages.
  - message from a human actor to the user interface of the system (in MVC)
- Use **synchronous** messages for method calls
  - A synchronous message is used when the sender waits until the receiver has finished processing the message, only then does the caller continue



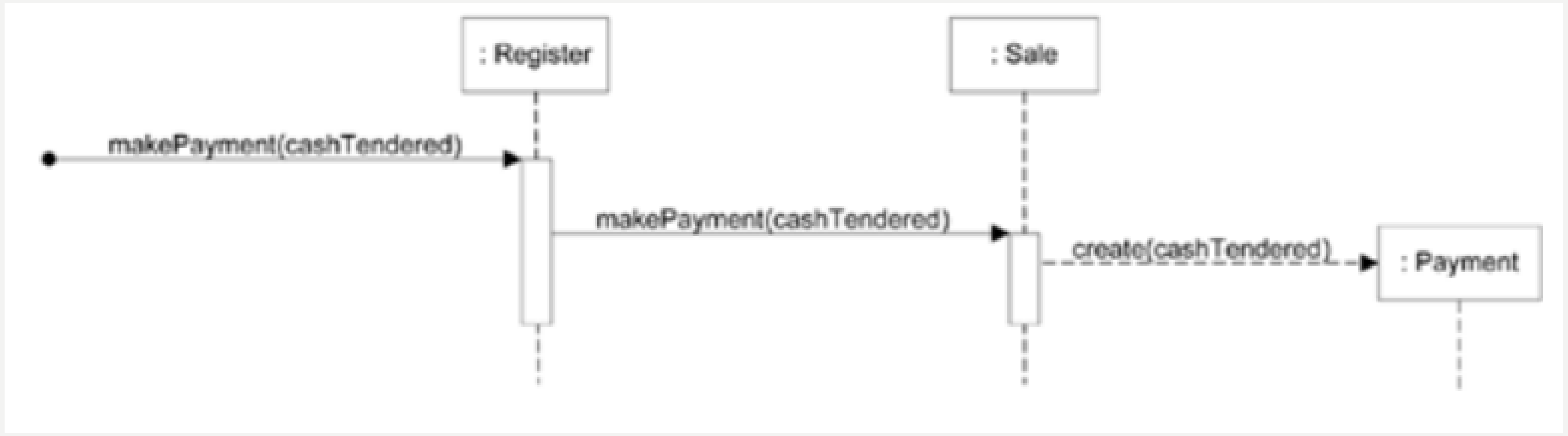
Example Sequence Diagram



```
public class A {
private B myB = new B();

public void doOne()
{
        myB.doTwo();
        myB.doThree();
}
// ...
}
```

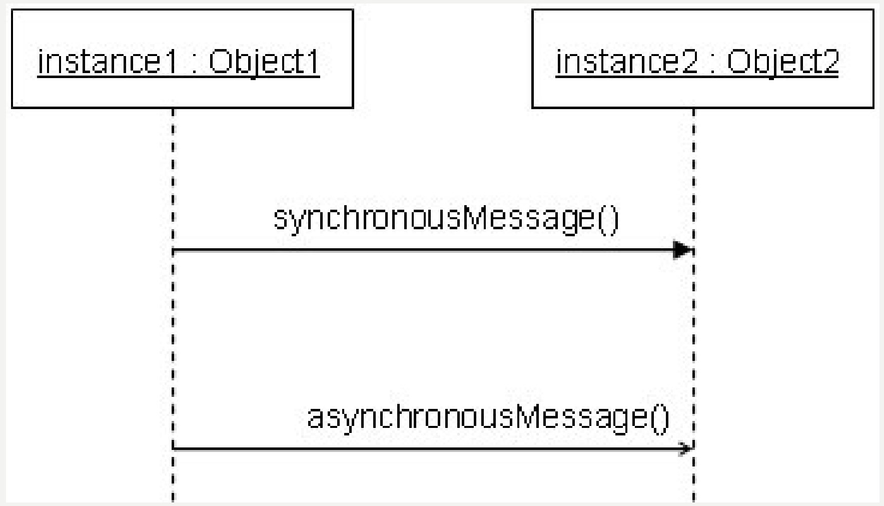
```
public class B {
public void doTwo()
{
}

public void doThree()
{
}
// ...
}
```

Example Sequence Diagram



More Sequence Diagram Notation

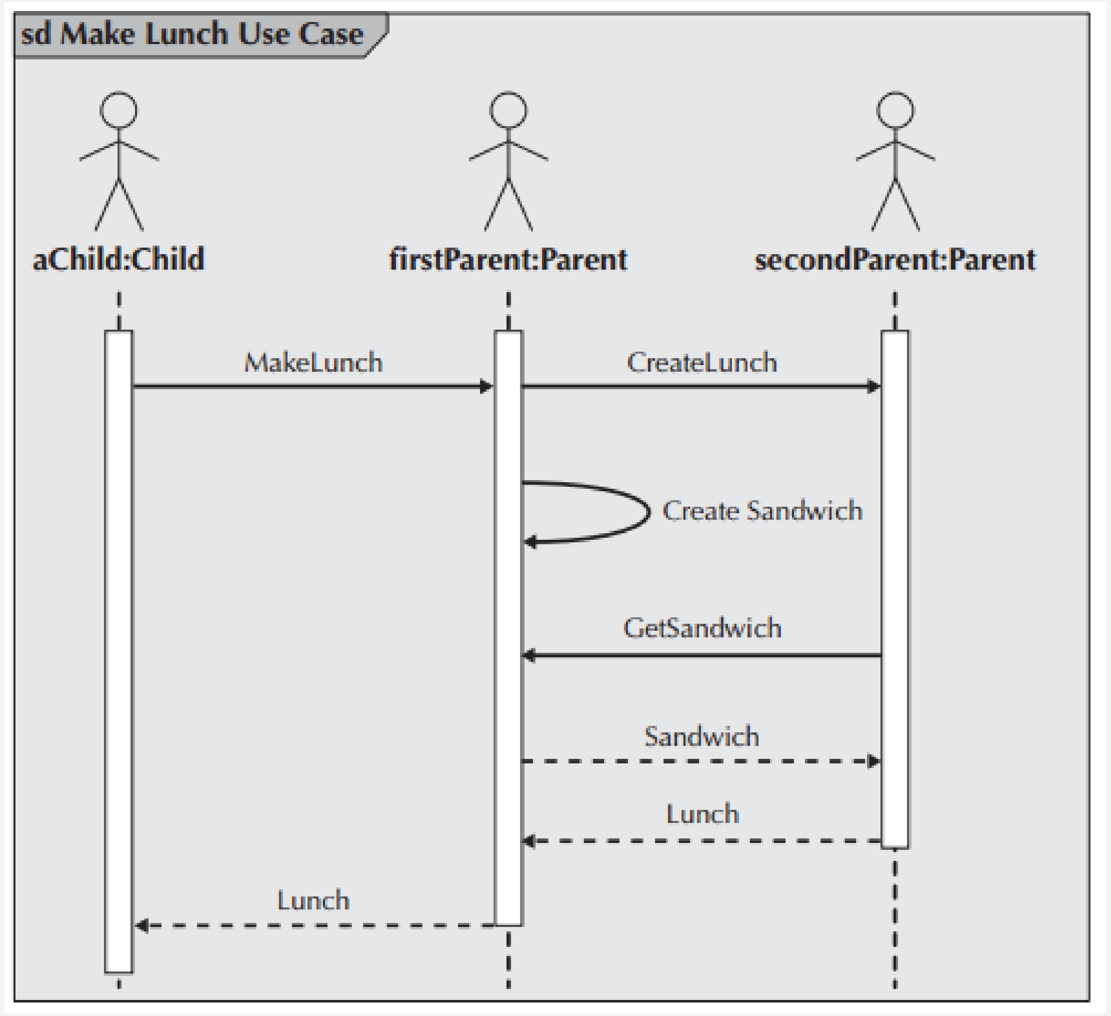
1. Messages to “self” or “this”
2. Creation of Instances
3. Diagram Frames
4. Conditional Message
5. Looping



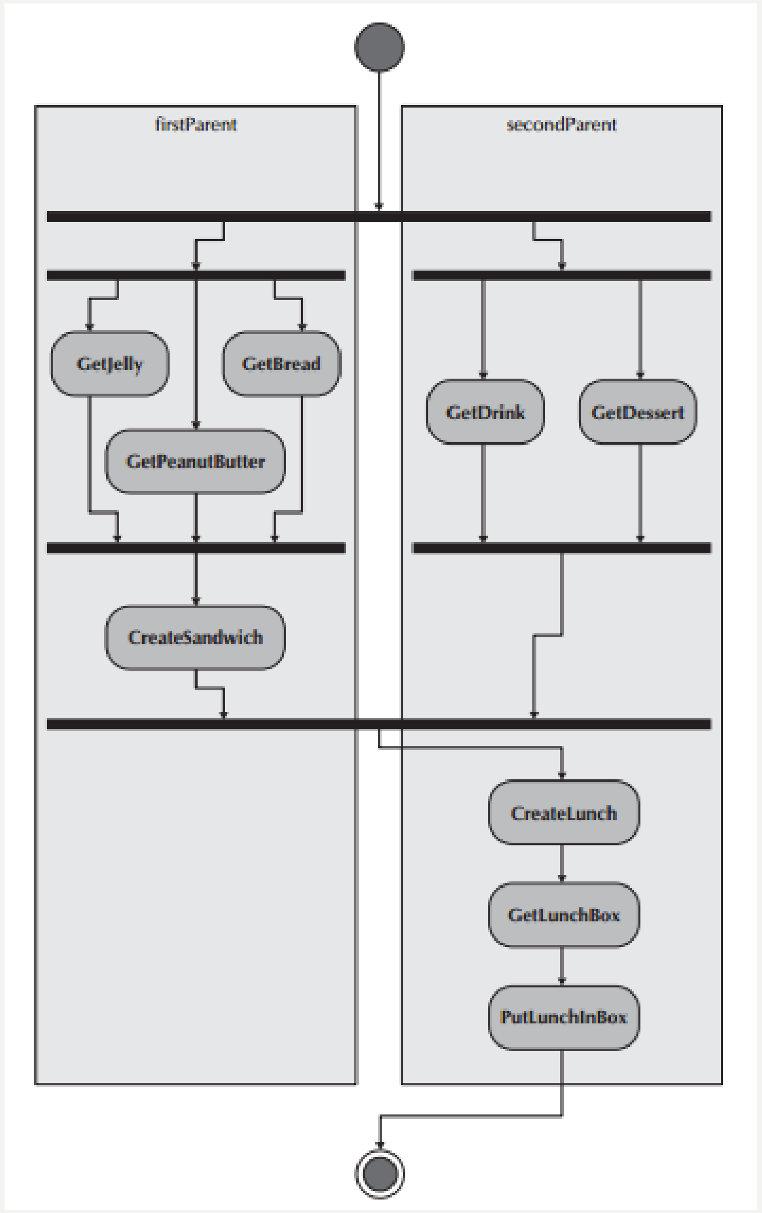
Messages to "self" or "this" (Self-Delegation)

- An object can send a message to itself. This is known as self-delegation. Sometimes, an object creates another object. This is shown by the message being sent directly to an object instead of its lifeline.

(see the activity diagram on the next page)



Activity Diagram of Make Lunch



Creation of Instances

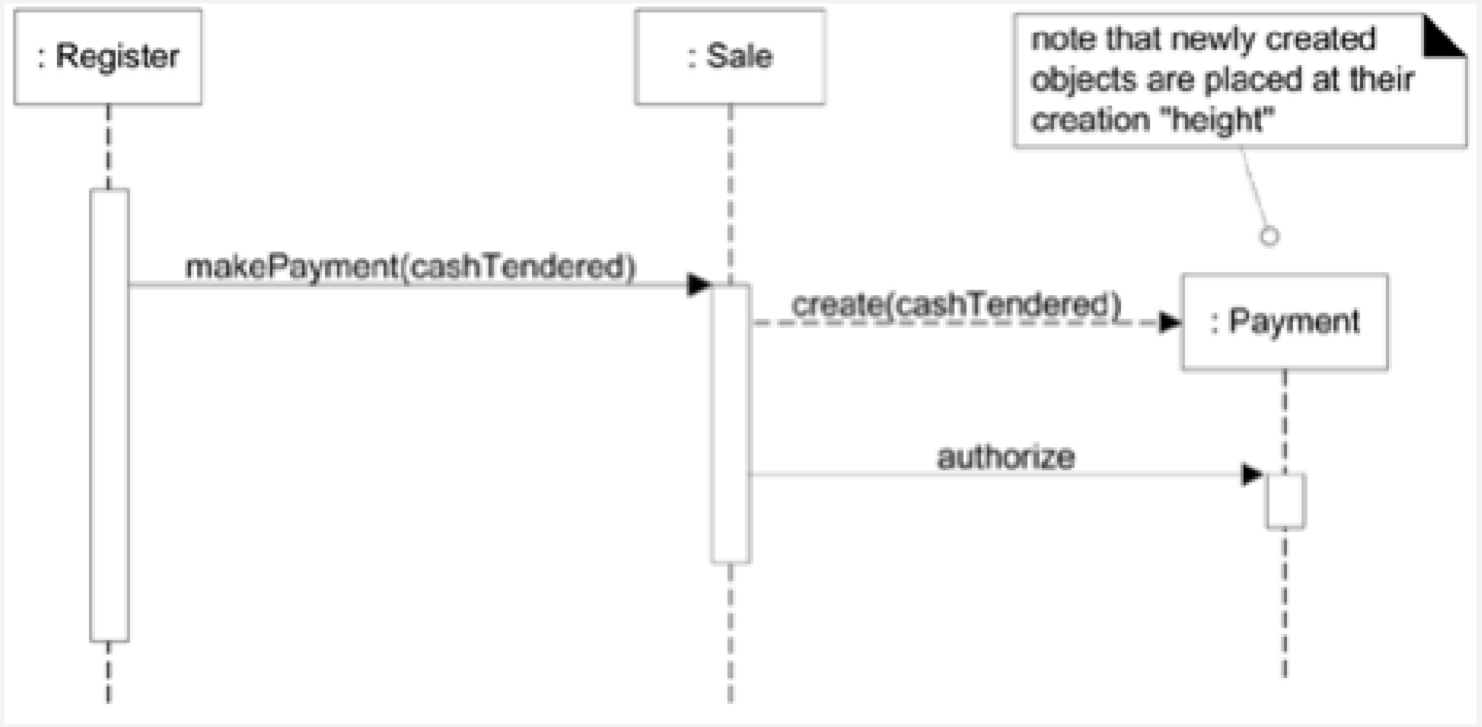

: Register
: Sale
note that newly created objects are placed at their creation "height"
makePayment(cashTendered)
create(cashTendered)
: Payment
authorize

Diagram Frames

- To support conditional and looping construct

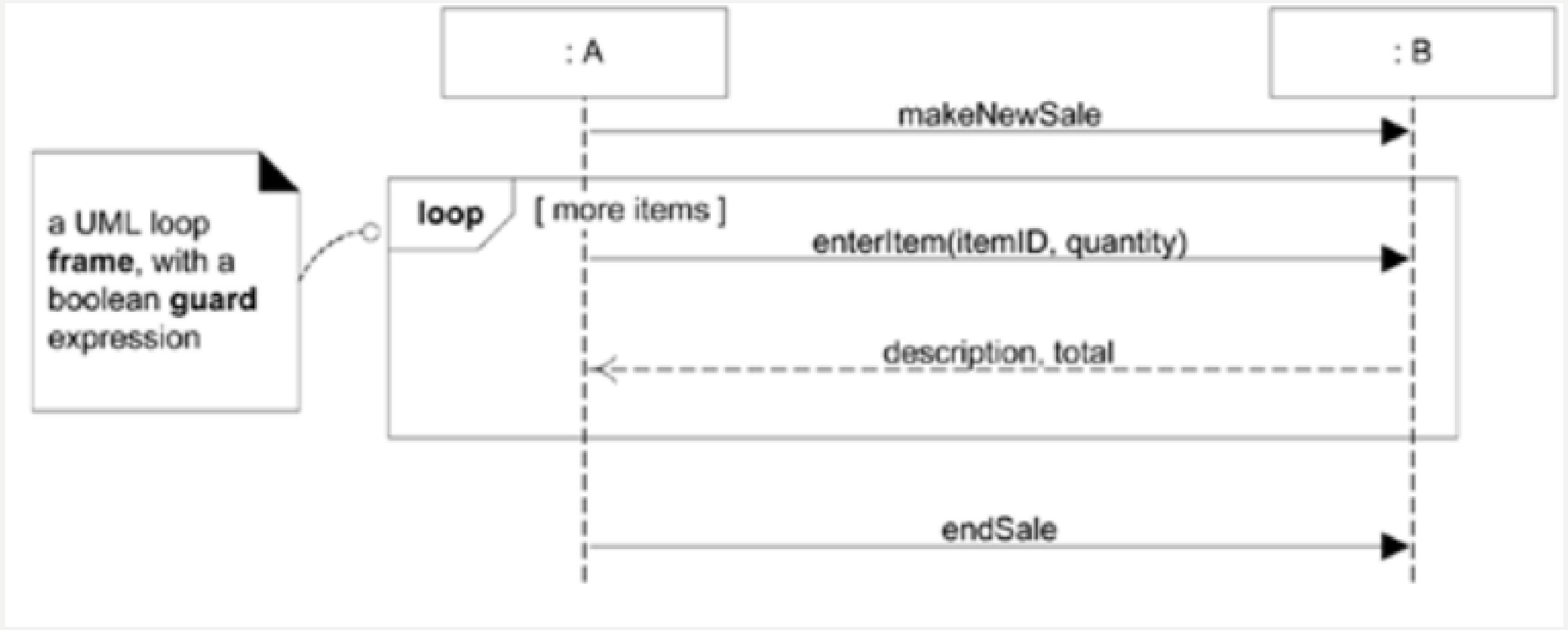
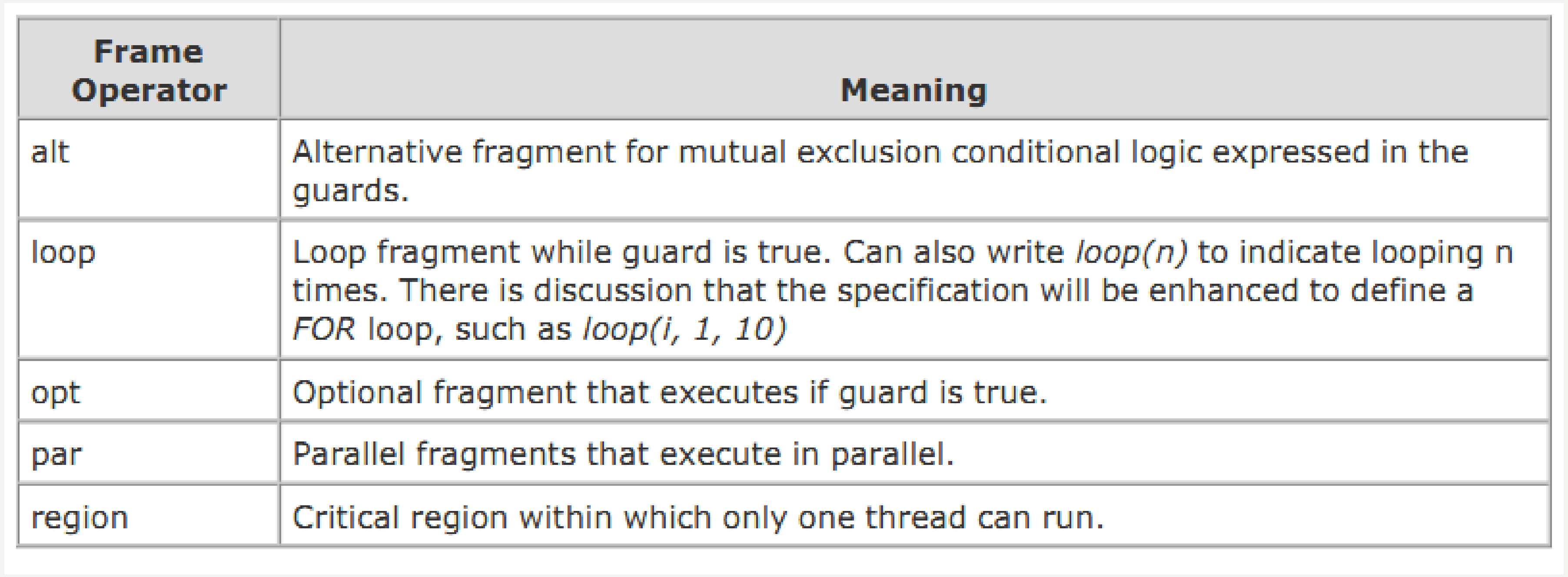

: A
: B
makeNewSale
a UML loop **frame**, with a boolean **guard** expression
loop
[more items]
enterItem(itemID, quantity)
description, total
endSale

Diagram Frames

- Frame Operator



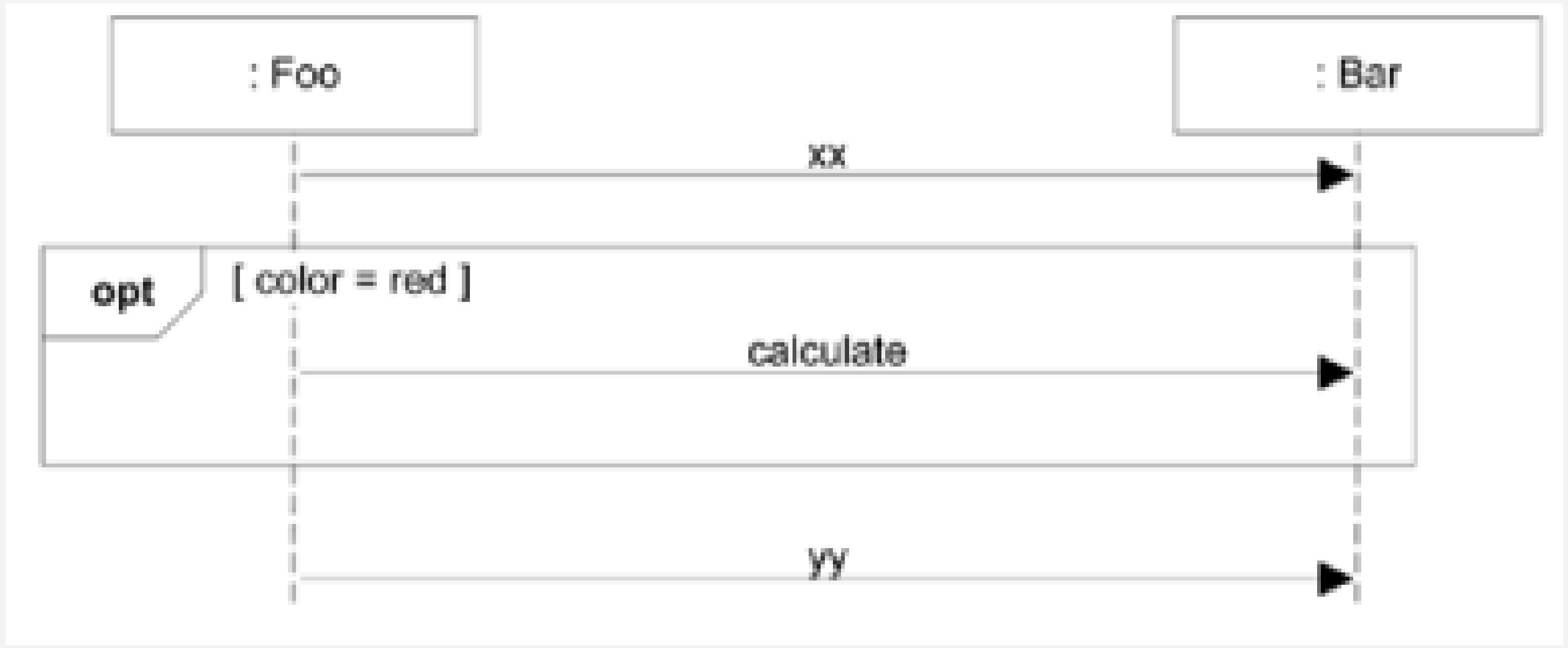
Frame Operator	Meaning
alt	Alternative fragment for mutual exclusion conditional logic expressed in the guards.
loop	Loop fragment while guard is true. Can also write *loop(n)* to indicate looping n times. There is discussion that the specification will be enhanced to define a *FOR* loop, such as *loop(i, 1, 10)*
opt	Optional fragment that executes if guard is true.
par	Parallel fragments that execute in parallel.
region	Critical region within which only one thread can run.

Diagram Frames

- At times a message is sent only if a condition is met. In those cases, the condition is placed between a set of brackets, [ ]—for example, [aPatient Exists] LookupBills().
- The condition is placed in front of the message name.
- However, when using a sequence diagram to model a specific scenario, conditions are typically not shown on any single sequence diagram.
- Instead, conditions are implied only through the existence of different sequence diagrams.

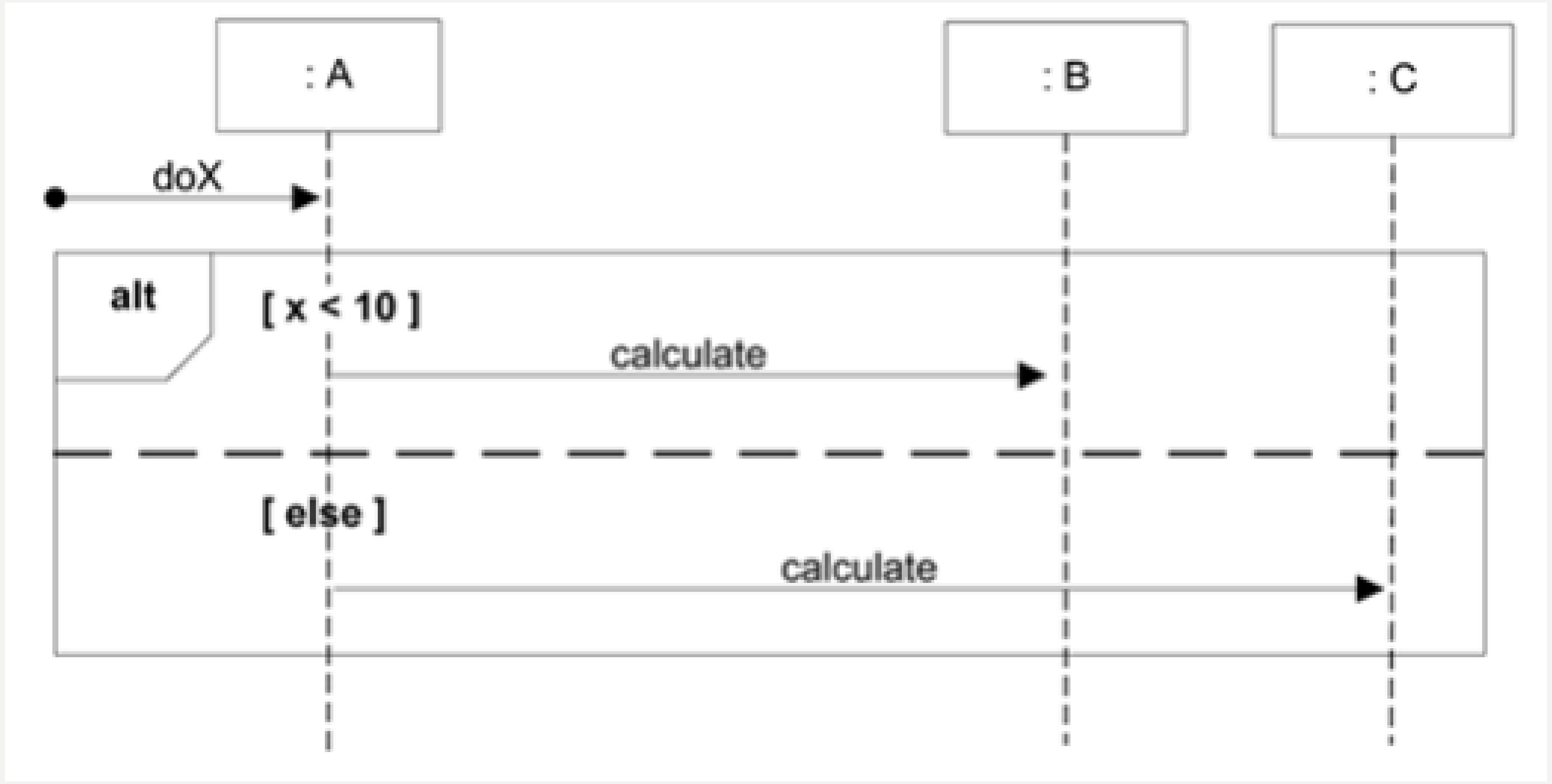
Conditional Message

To describe the “if” syntax



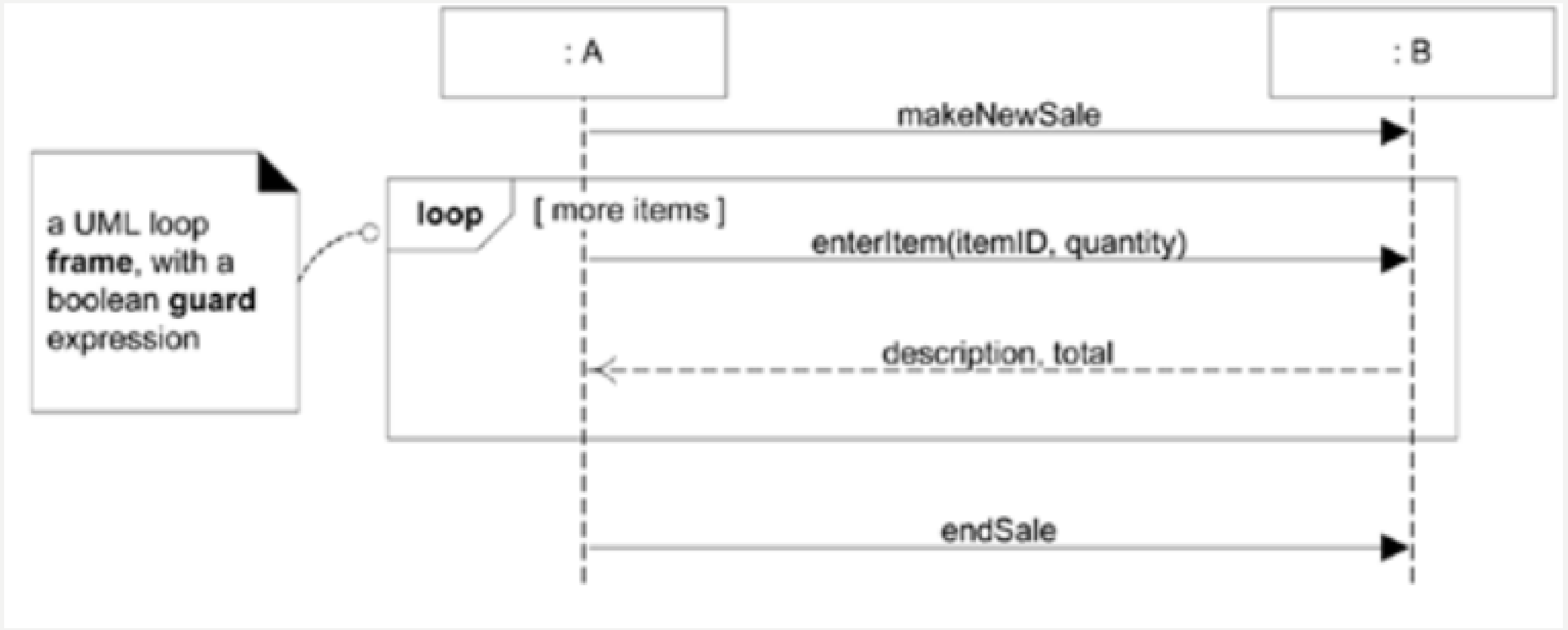
Conditional Message

To describe a Mutually Exclusive condition



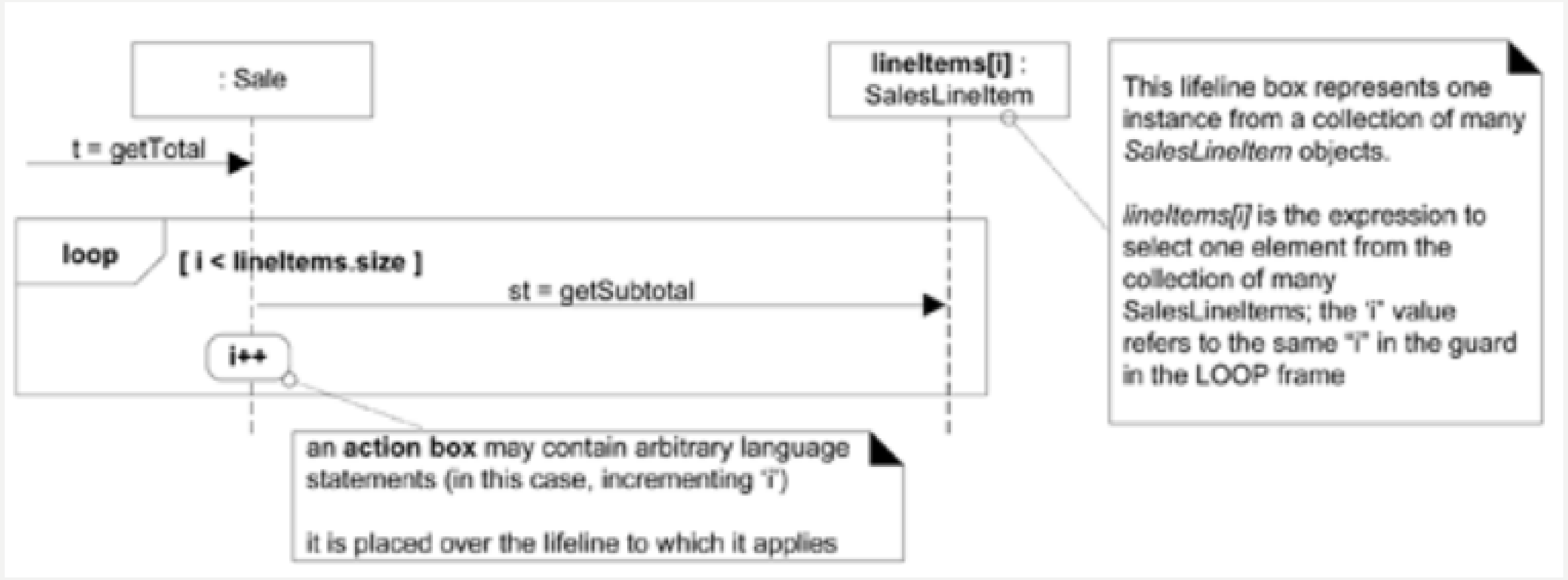
Looping

To describe the “Loop” syntax


: A
: B
makeNewSale
a UML loop frame, with a boolean guard expression
loop
[more items]
enterItem(itemID, quantity)
description, total
endSale

Looping

Iteration over a collection (1) – relatively explicit notation



This lifeline box represents one instance from a collection of many *SalesLineItem* objects.

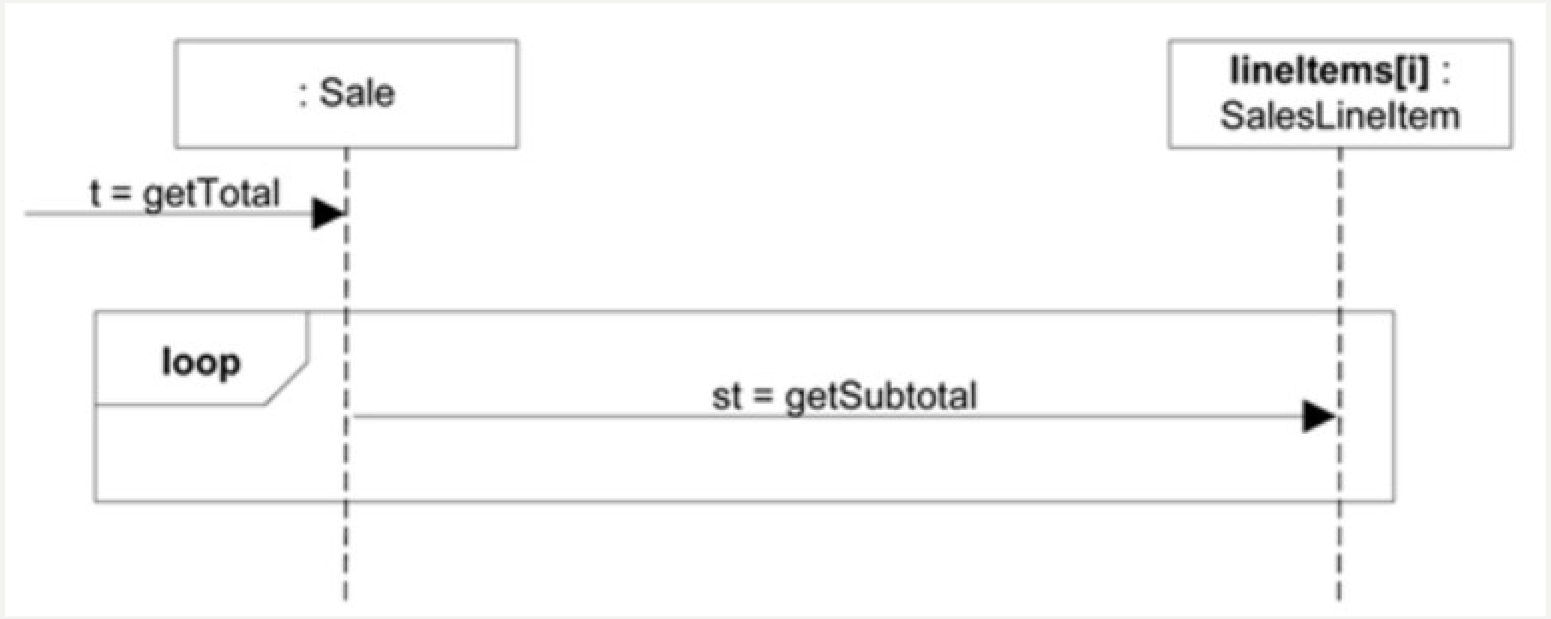
lineItems[i] is the expression to select one element from the collection of many SalesLineItems; the 'i" value refers to the same "i" in the guard in the LOOP frame

an **action box** may contain arbitrary language statements (in this case, incrementing 'i')

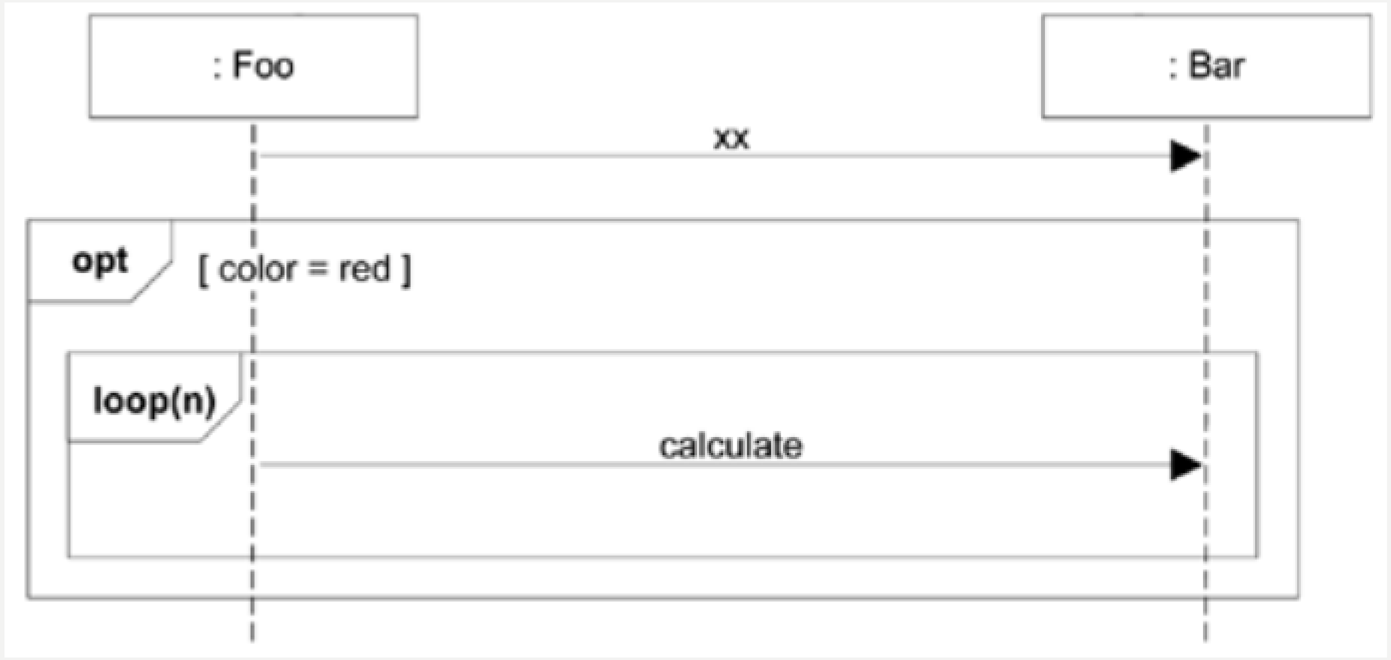
it is placed over the lifeline to which it applies

Looping

Iteration over a collection (2) – leaving things more implicit



Nested Frame



Guidelines in Creating Sequence Diagrams

1. Try to have the messages **not only in a top-to-bottom order** but also, when possible, in **a left-to-right order.**
   - ➢ Given that Western cultures tend to read left to right and top to bottom, a sequence diagram is much easier to interpret if the messages are ordered as much as possible in the same way.
   - ➢ To accomplish this, order the actors and objects along the top of the diagram in the order that they participate in the scenario of the use case.

Guidelines in Creating Sequence Diagrams

2. If an **actor and an object** conceptually represent the same idea, one inside of the software and the other outside, **label them with the same name**.
   - ➢ In fact, this implies that they exist in both the use-case diagram (as an actor) and in the class diagram (as a class).
   - ➢ At first glance, this might seem to lead to confusion.
   - ➢ However, if they do indeed represent the same idea, then they should have the same name.
   - ➢ For example, ***a customer actor interacts with the system and the system stores information about the customer.*** In this case, they do indeed represent the same conceptual idea.

Guidelines in Creating Sequence Diagrams

3. The **initiator** of the scenario—actor or object—should be the drawn as the farthest left item in the diagram.
    - This guideline is essentially a specialization of the first guideline.
    - In this case, it relates specifically to the actor or object that triggers the scenario.

Guidelines in Creating Sequence Diagrams

4. When there are **multiple objects of the same type**, be sure to **include a name for the object** in addition to the class of the object.

   For example, in the making a lunch example, there are **two objects of type Parent**. As such, **they should be named.** Otherwise, you can simply use the class name. This will simplify the diagram. In this case, the Child object did not have to be named. We could have simply placed a colon in front of the class name instead.

Guidelines in Creating Sequence Diagrams

5. Show return values only when they are not obvious. Showing all of the returns tends to make a sequence diagram more complex and potentially difficult to comprehend. In many cases, less is more. Only show the returns that actually add information for the reader of the diagram.
6. Justify message names and return values near the arrowhead of the message and return arrows, respectively. This makes it much easier to interpret the messages and their return values.

Steps to Build Sequence Diagrams

1. Set the context
2. Identify which actors and objects will participate
3. Set the lifeline for each object
4. Lay out the messages from top to bottom of the diagram based on the order in which they are sent
5. Add execution occurrence to each object's lifeline
6. Validate the sequence diagram

1.4 COMMUNICATION DIAGRAM

Communication Diagrams

- A **communication** diagram in the Unified Modeling Language (UML) 2.0, is a simplified version of the UML 1.x **collaboration diagram**
- Like sequence diagram, it shows the messages that pass between objects for a particular use-case
- It is an object diagram that shows message-passing relationships instead of aggregation or generalization associations

Communication Diagrams

- Communication diagrams, like sequence diagrams, essentially provide a view of the dynamic aspects of an object-oriented system.
  - show how the members of a set of objects collaborate to implement a use case or a use-case scenario.
  - model all the interactions among a set of collaborating objects, in other words, a collaboration (see CRC cards in Chapter 5). In this case, a communication diagram can portray how dependent the different objects are on one another.
- Communication diagrams are very useful to **show process patterns** (i.e., patterns of activity that occur over a set of collaborating classes).
- Communication diagrams are **equivalent to sequence diagrams**, but they **emphasize the flow of messages** through a set of objects, whereas the sequence diagrams focus on the time ordering of the messages being passed.

Communication Diagrams

- Communication diagrams, like sequence diagrams, essentially provide a view of the dynamic aspects of an object-oriented system.
  - show how the members of a set of objects collaborate to implement a use case or a use-case scenario.
  - model all the interactions among a set of collaborating objects, in other words, a collaboration (see CRC cards in Chapter 5). In this case, a communication diagram can portray how dependent the different objects are on one another.
- Communication diagrams are very useful to **show process patterns** (i.e., patterns of activity that occur over a set of collaborating classes).
- Communication diagrams are **equivalent to sequence diagrams**, but they **emphasize the flow of messages** through a set of objects, whereas the sequence diagrams focus on the time ordering of the messages being passed.

The Differences

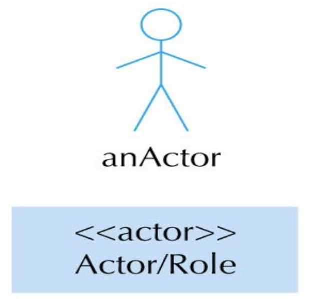
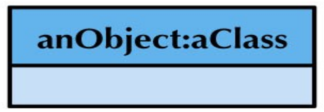
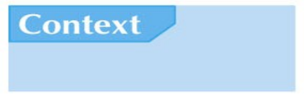
Sequence Diagram

- Emphasize the **time ordering** of the messages being passed among objects
- Can optionally show returns from message
- Association between objects are not shown

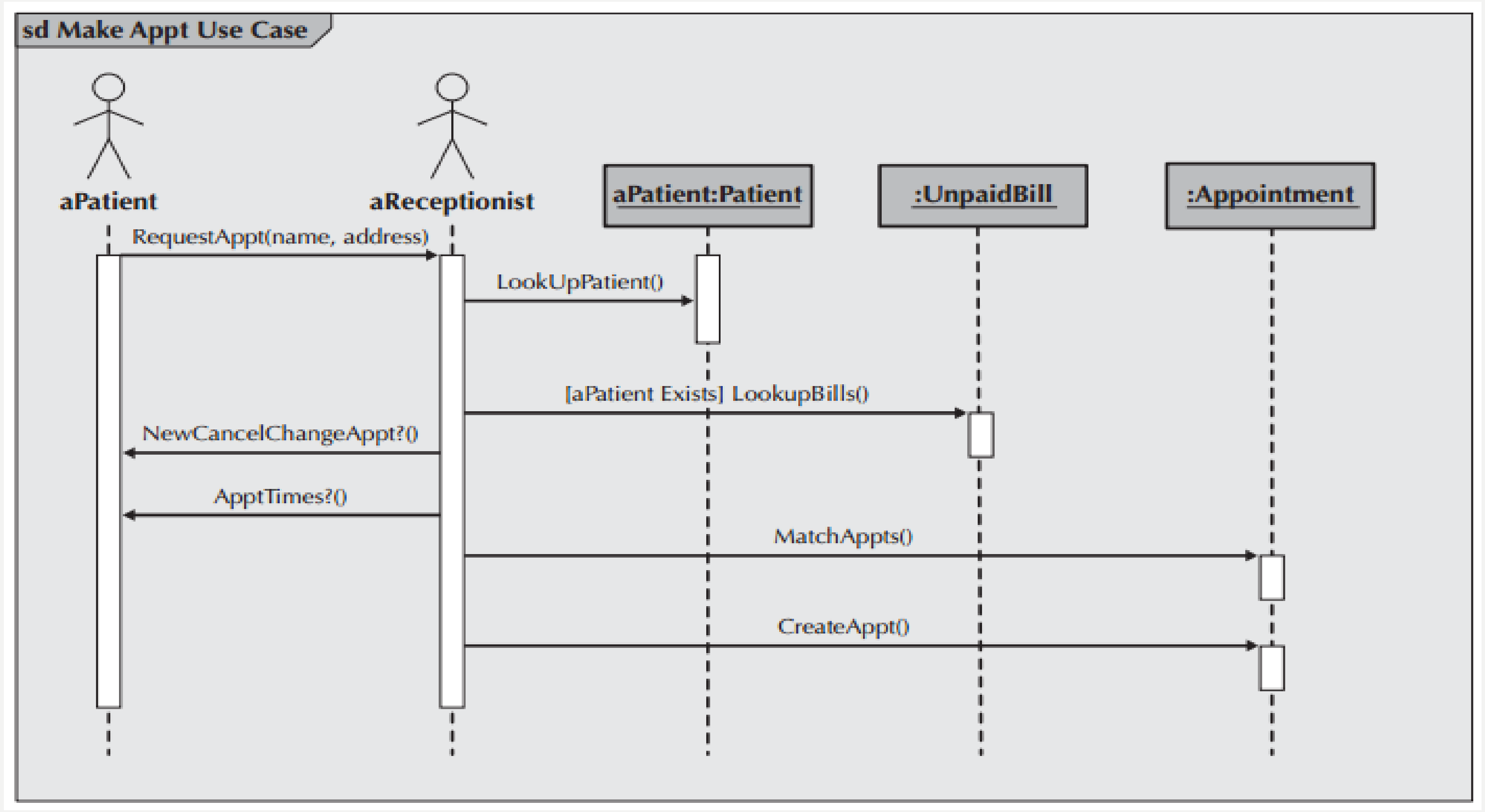
Communication Diagram

- Emphasize the **flow** of messages through a set of objects
- Never shows returns from message
- Association between objects are shown

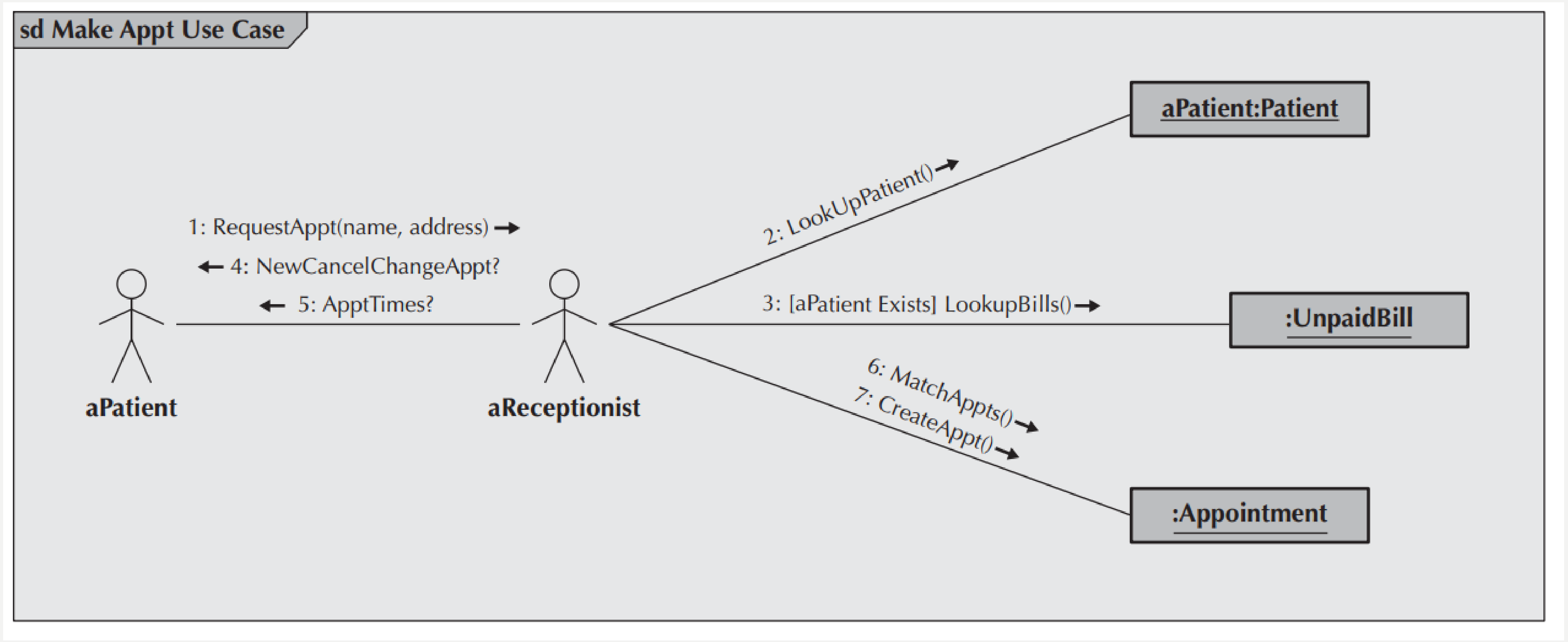
The Syntax

Term and Definition	Symbol
An actor: ■ Is a person or system that derives benefit from and is external to the system. ■ Participates in a collaboration by sending and/or receiving messages. ■ Is depicted either as a stick figure (default) or, if a nonhuman actor is involved, as a rectangle with <<actor>> in it (alternative).	
An object: ■ Participates in a collaboration by sending and/or receiving messages. ■ Is placed across the top of the diagram.	
An association: ■ Shows an association between actors and/or objects. ■ Is used to send messages.	————————
A message: ■ Conveys information from one object to another one. ■ Has direction shown using an arrowhead. ■ Has sequence shown by a sequence number.	SeqNumber: aMessage →
A guard condition: ■ Represents a test that must be met for the message to be sent.	SeqNumber: [aGuardCondition]: aMessage →
A frame: ■ Indicates the context of the communication diagram.	

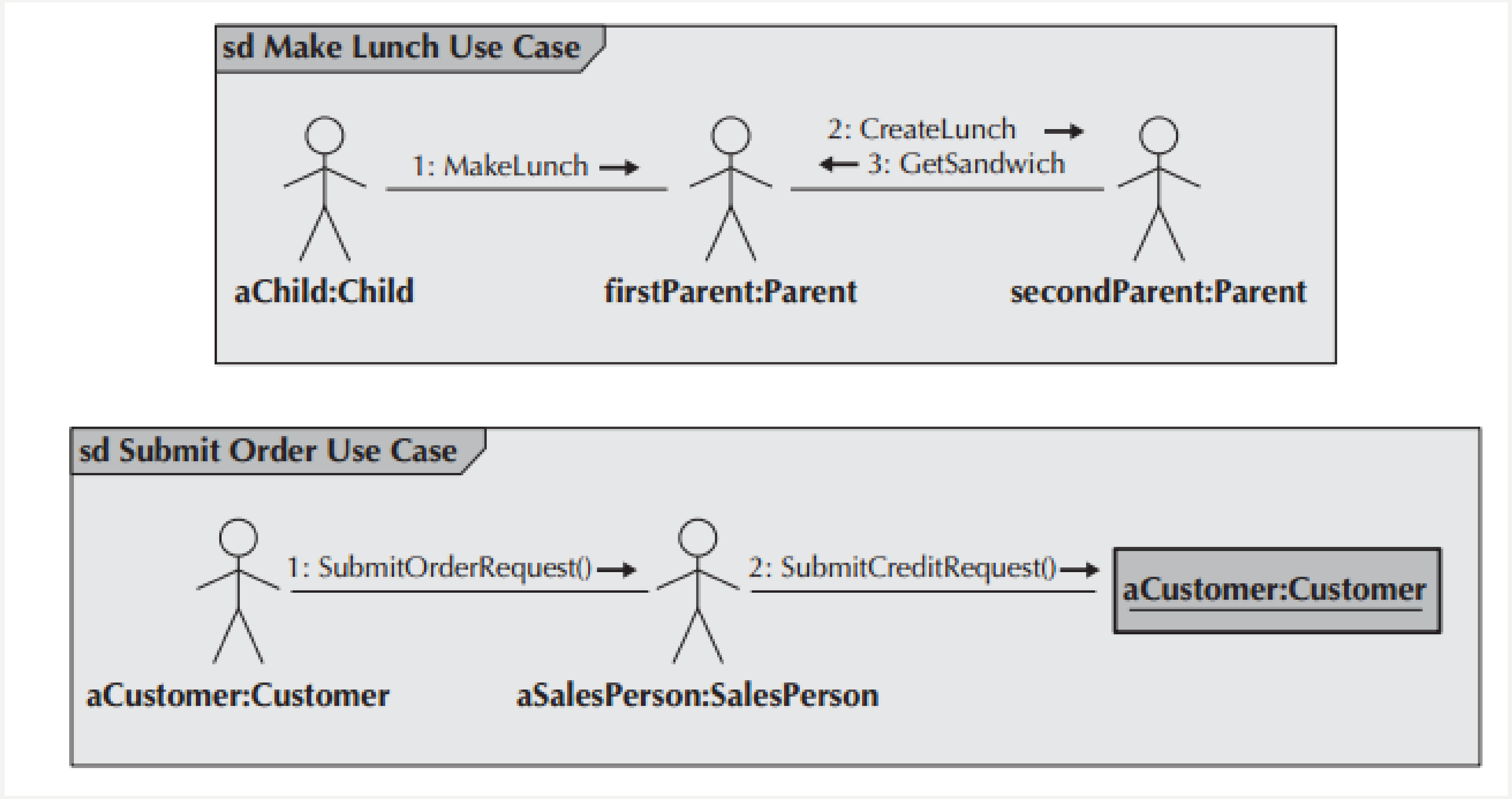
Sample Sequence Diagram



Sample Communication Diagram



Sample Sequence Diagram



Actor and Object

Actors and objects that collaborate to execute the use case are placed on the communication diagram in a manner to **emphasize the message passing** that takes place between them.

- Notice that the actors and objects in example before, are the same ones: aPatient, aReceptionist, aPatient, UnpaidBill, and Appointment.
- Again, as with the sequence diagram, for each of the objects, the name of the class of which they are an instance is given after the object's name (e.g., aPatient: Patient).

Actor and Object

Unlike the sequence diagram, the communication diagram does **not have a means to explicitly show an object being deleted or created.**

- It is assumed that when a delete, destroy, or remove message is sent to an object, it will go out of existence, and a create or new message will cause a new object to come into existence.
- Another difference between the two interaction diagrams is that the communication diagram never shows returns from message sends, whereas the sequence diagram can optionally show them.

Association

- An association is shown between actors and objects with an **undirected line.** For example, an association is shown between the aPatient and aReceptionist actors.
- Messages are shown as labels on the associations. Included with the labels are lines with arrows showing the direction of the message being sent.
  - For example, the aPatient actor sends the RequestAppt() message to the aReceptionist actor, and the aReceptionist actor sends the NewCancelChangeAppt?() and the ApptTimes?() messages to the aPatient actor. The sequence of the message sends is designated with a sequence number. According to the example before, the RequestAppt() message is the first message sent, whereas the NewCancelChangeAppt?() and the ApptTimes?() messages are the fourth and fifth message sent, respectively.

Conditional Messages

- Like the sequence diagram, the communication diagram can represent conditional messages.
- For previous example, the LookupBills() message is sent only if the [aPatient exists] condition is met. If a message is repeatedly sent, an asterisk is placed after the sequence number. Finally, an association that loops onto an object shows self-delegation.
- The message is shown as the label of the association

Guidelines for Creating Communication Diagrams

- Use the correct diagram for the information you are interested in communicating with the user. Communication diagrams allow the team to easily identify a set of objects that are intertwined. Do not use communication diagrams to model process flow. Instead, you should use an activity diagram with swimlanes that represent objects (see Chapter 4). On the other hand, it would be very difficult to "see" how the objects collaborated in an activity diagram.
- When trying to understand the sequencing of messages, a sequence diagram should be used instead of a communication diagram. As in the previous guideline, this guideline essentially suggests that you should use the diagram that was designed to deal with the issue at hand. Even though communication diagrams can show sequencing of messages, this was never meant to be their primary purpose.

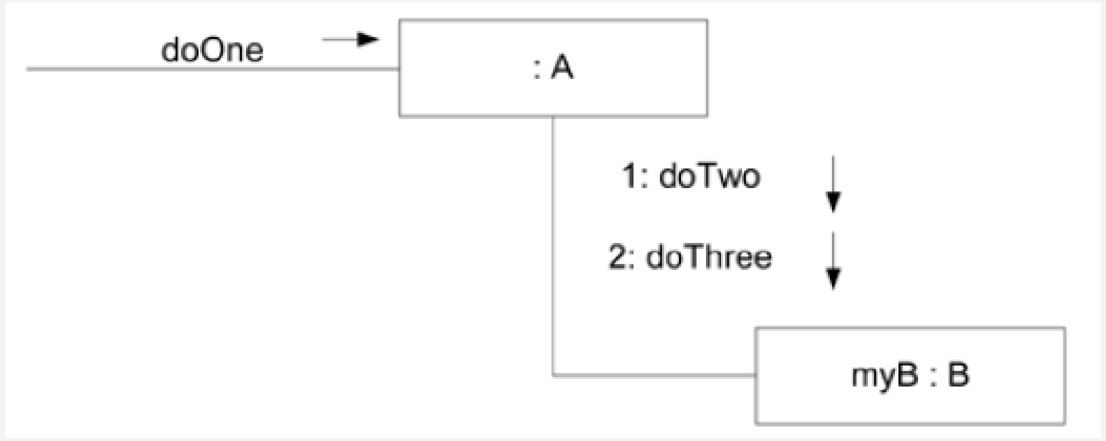
Simplifying Communication Diagram

- When a communication diagram is fully populated with all the objects, it can become very complex and difficult to understand. When this occurs, it is necessary to simplify the diagram. One approach to simplifying a communication diagram, like use-case diagrams(see Chapter 4) and class diagrams (see Chapter 5), is through the use of packages(i.e., logical groups of classes).
- In the case of communication diagrams, its objects are grouped together based on the messages sent to and received from the other objects.

Steps to Build Communication Diagrams

1. Set the context
2. Identify objects (actors)
3. Layout the associations between the objects
4. Add messages
5. Validate the communication diagram

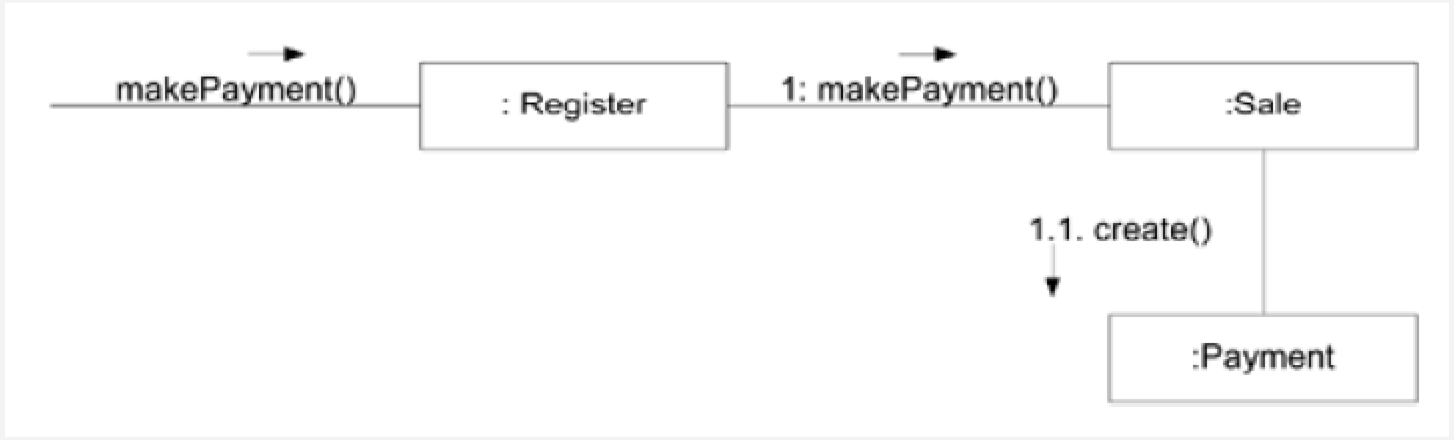
Example



```
public class A {
private B myB = new B();

public void doOne()
{
      myB.doTwo();
      myB.doThree();
}

// ...
}
```

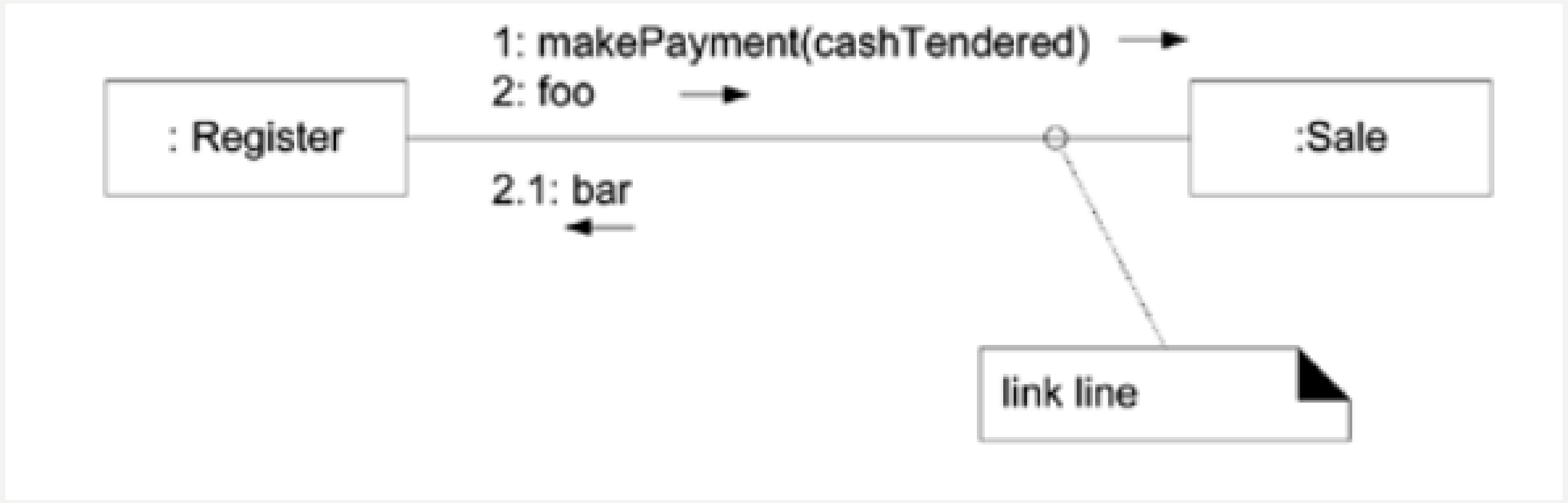
```
public class B {
public void doTwo()
{
}

public void doThree()
{
}
// ...

}
```

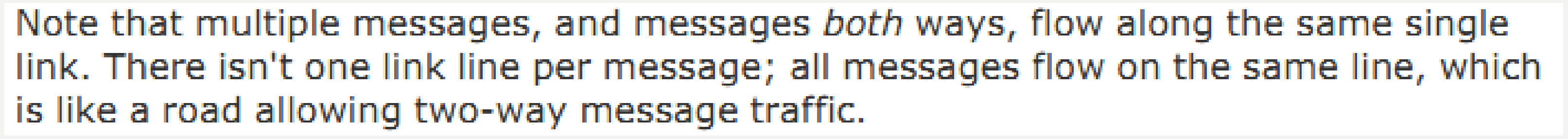
Example Communication Diagram: makePayment



Link

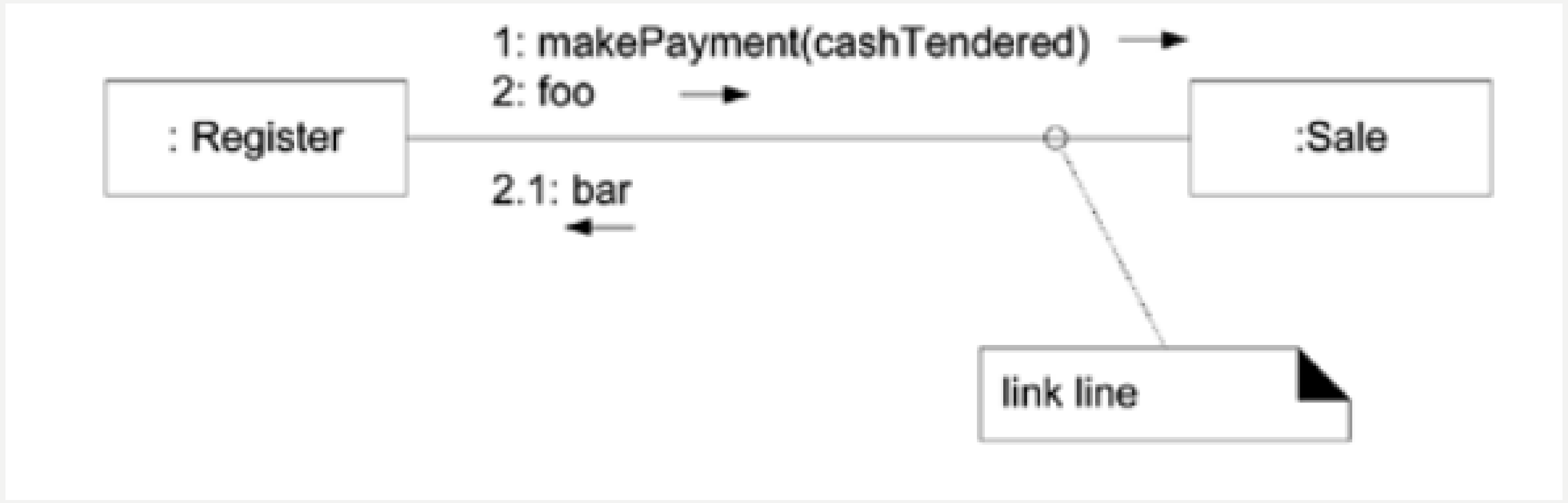
A connection path between two objects (an instance of an association)

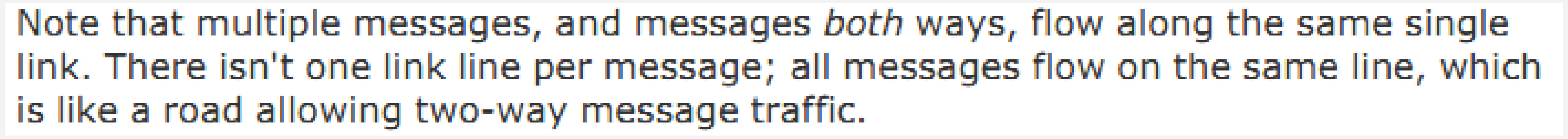



Note that multiple messages, and messages *both* ways, flow along the same single link. There isn't one link line per message; all messages flow on the same line, which is like a road allowing two-way message traffic.

Message

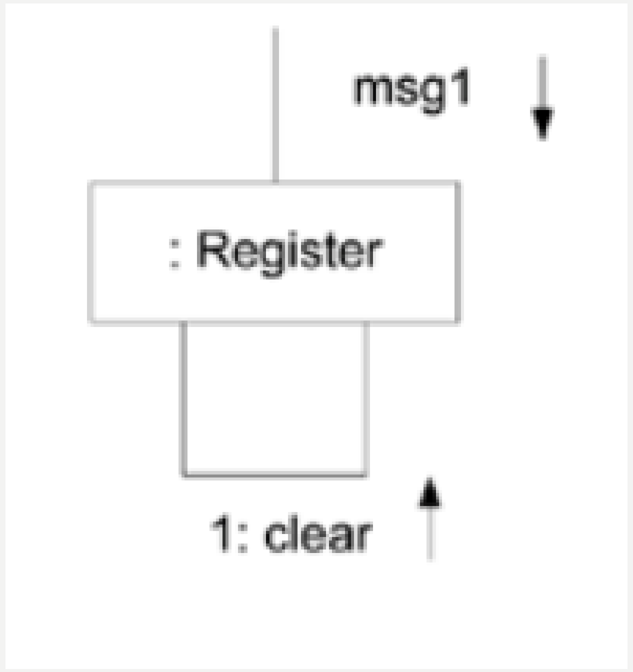
Represented with a message expression on an arrowed line between objects




Note that multiple messages, and messages *both* ways, flow along the same single link. There isn't one link line per message; all messages flow on the same line, which is like a road allowing two-way message traffic.

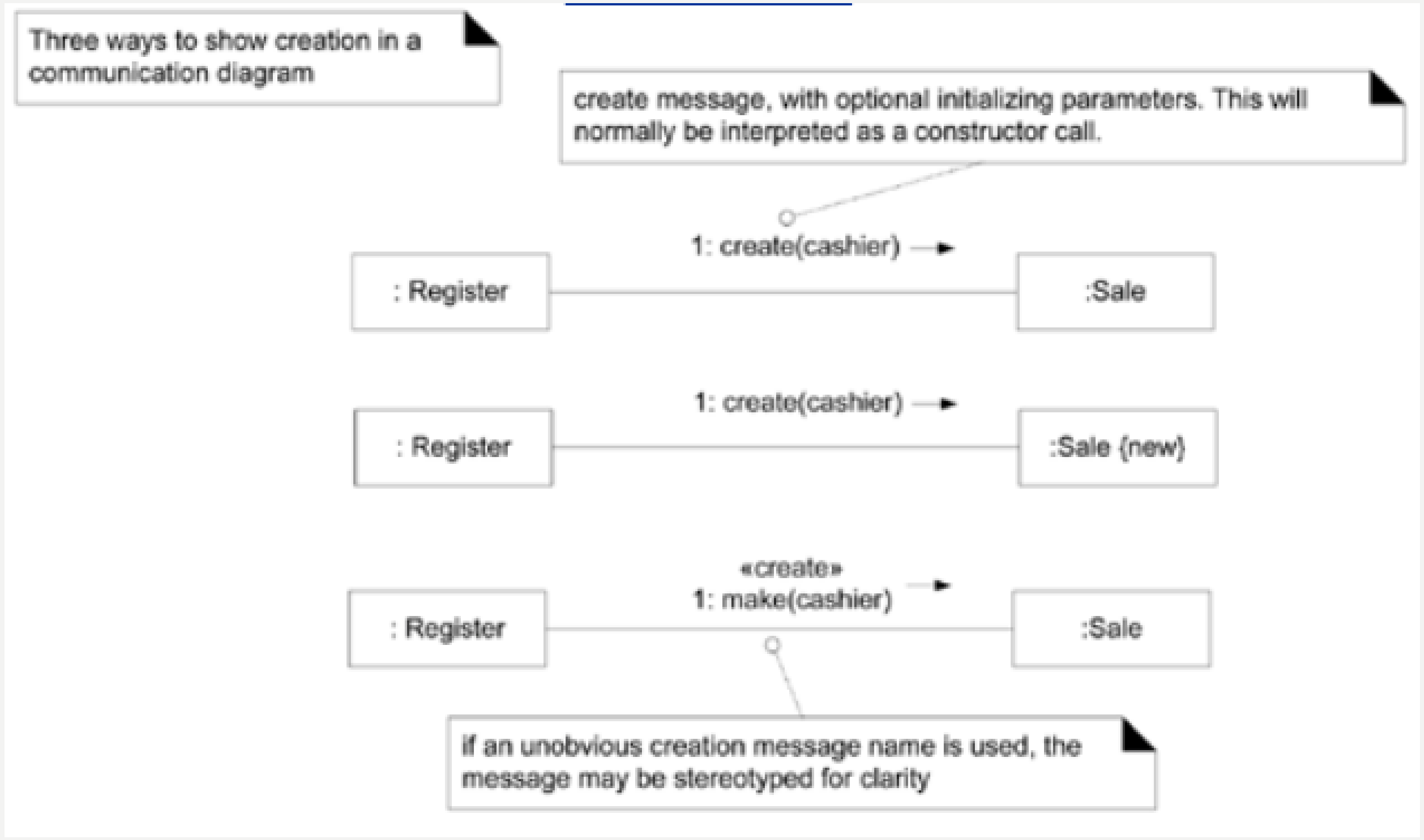
Message

Message to “self” or “this”



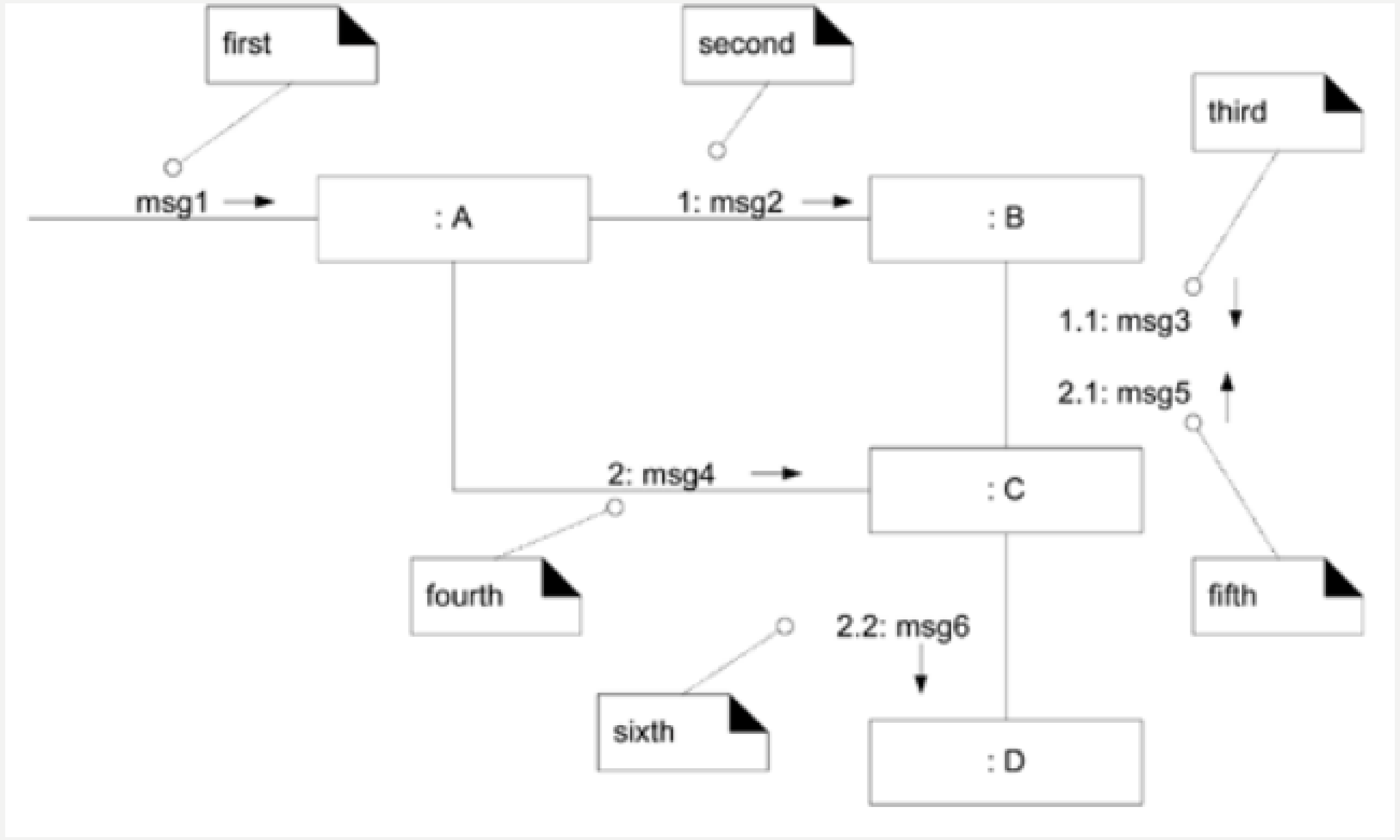
Message

Instance Creation


Three ways to show creation in a communication diagram
create message, with optional initializing parameters. This will normally be interpreted as a constructor call.
1: create(cashier)
: Register
:Sale
1: create(cashier)
: Register
:Sale {new}
«create»
1: make(cashier)
: Register
:Sale
if an unobvious creation message name is used, the message may be stereotyped for clarity

Sequence Number

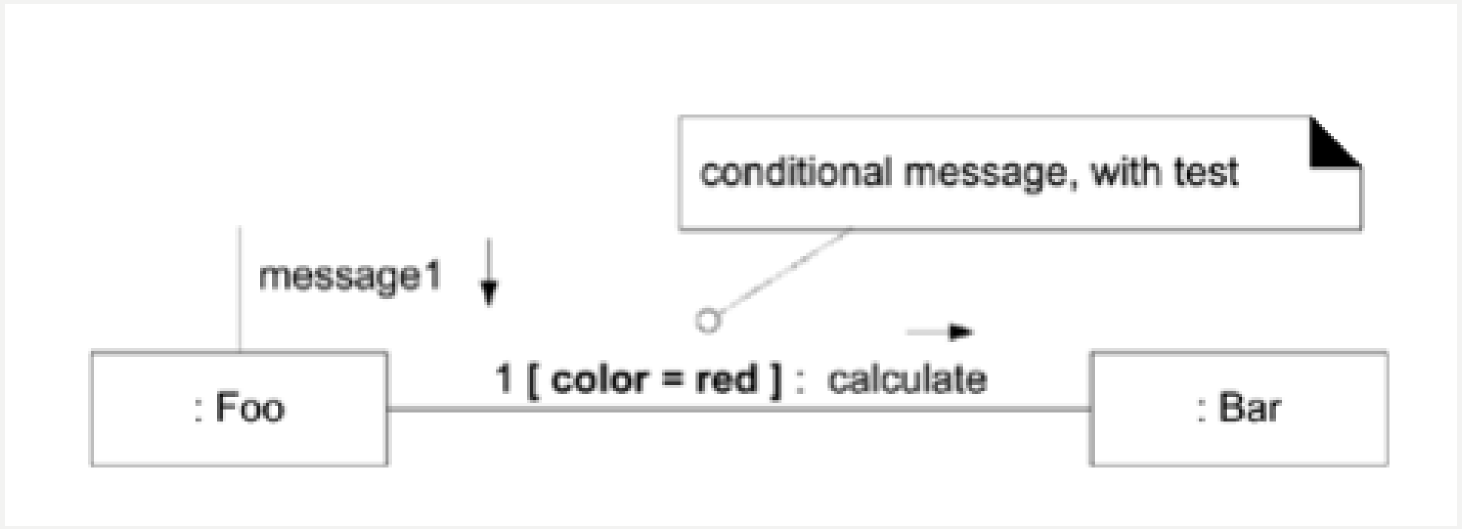
Represents the order in which the flows are used



Guard

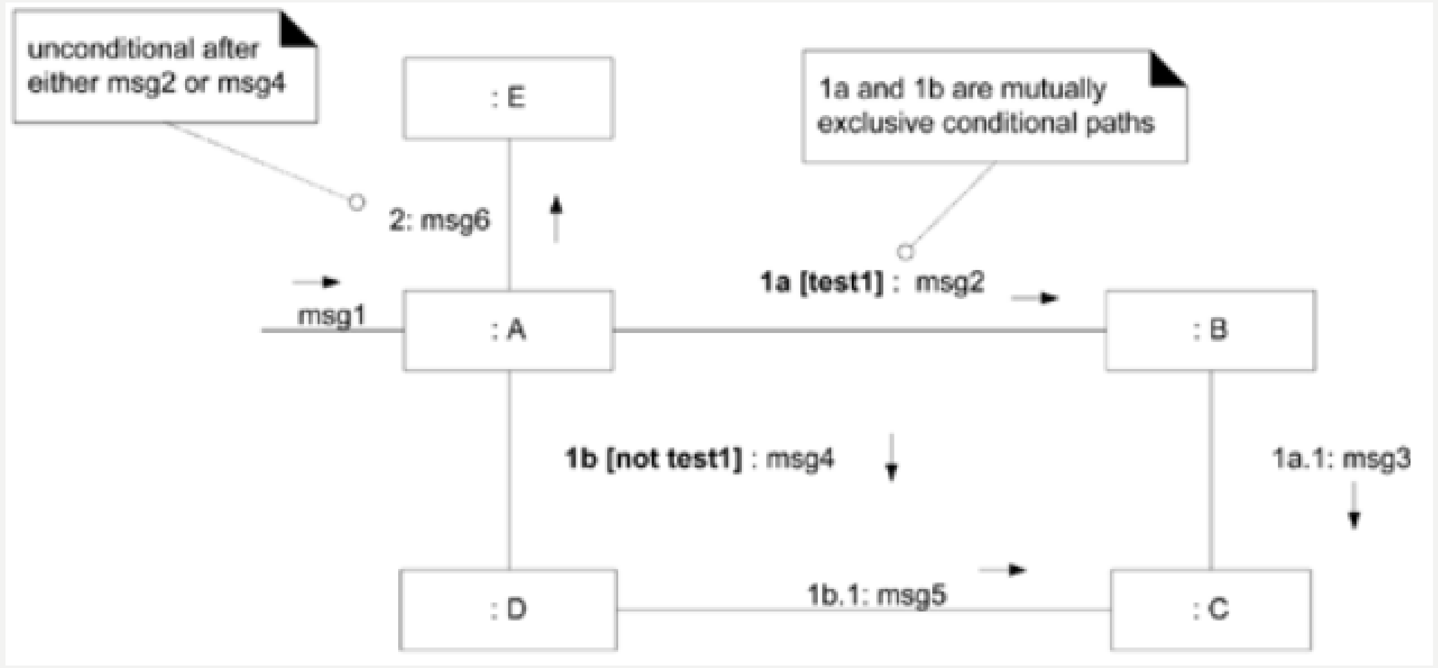
(Conditional Message)

- Seq. Number *[variable = value]* : message()
- Message is sent only if clause evaluates to *true*



Guard

To describe Mutually Exclusive condition

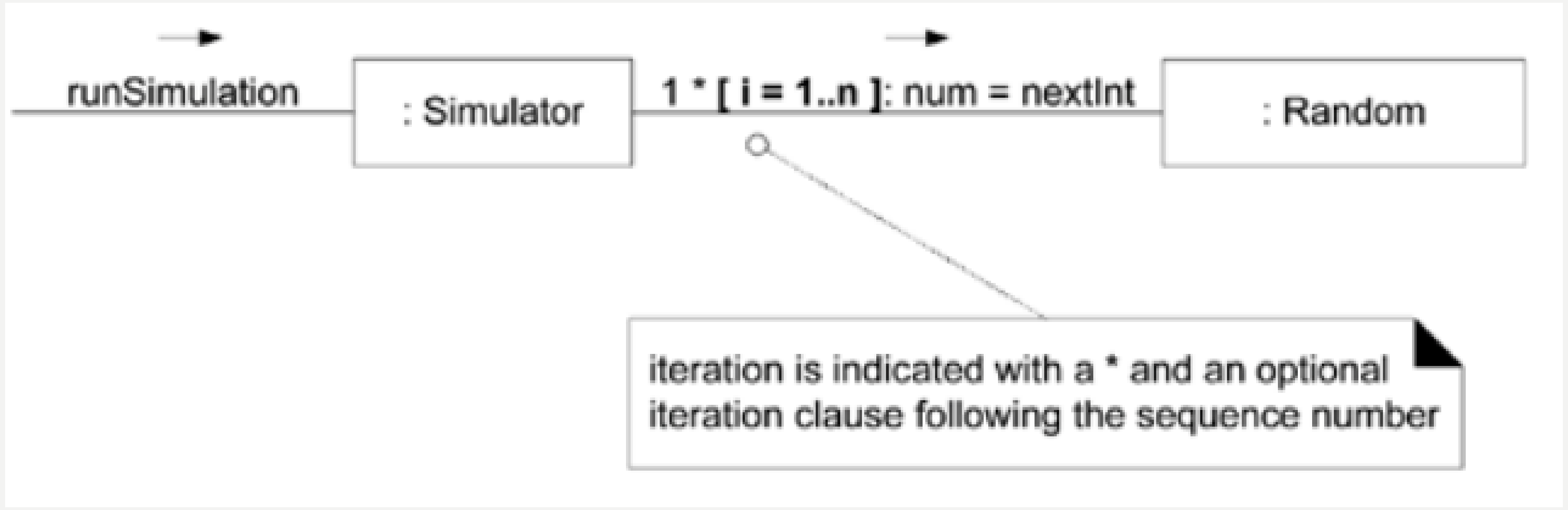

unconditional after either msg2 or msg4
: E
1a and 1b are mutually exclusive conditional paths
2: msg6
1a [test1] : msg2
msg1
: A
: B
1b [not test1] : msg4
1a.1: msg3
: D
1b.1: msg5
: C

Either 1a or 1b but not both should be executed after msg1

Loop

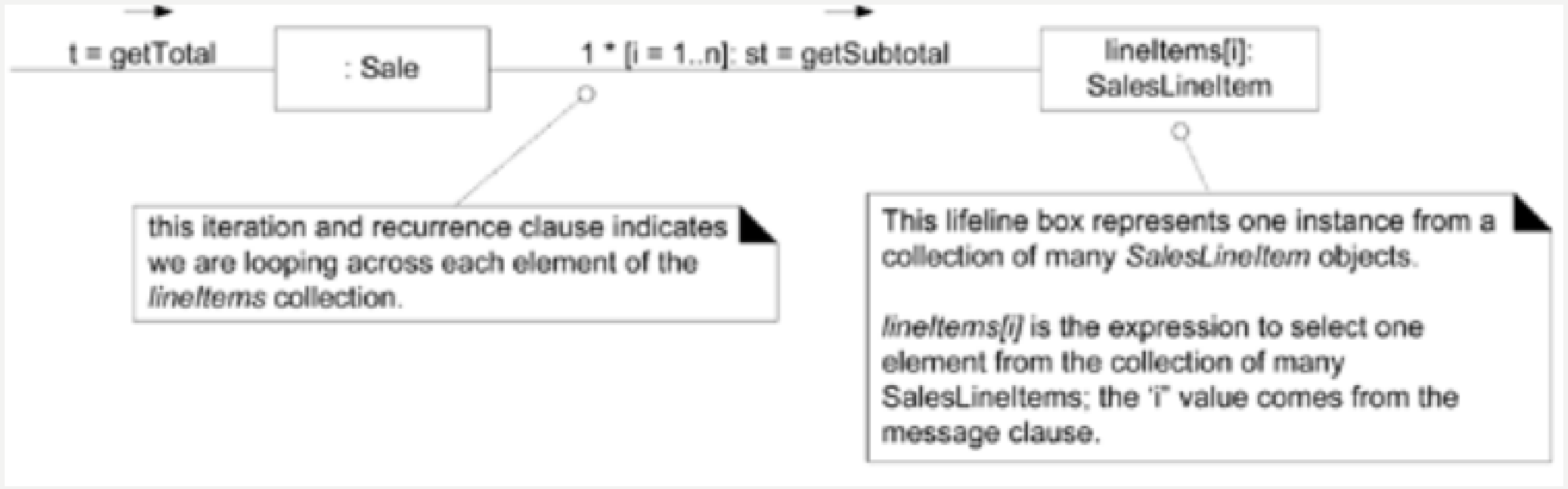
(Iteration)

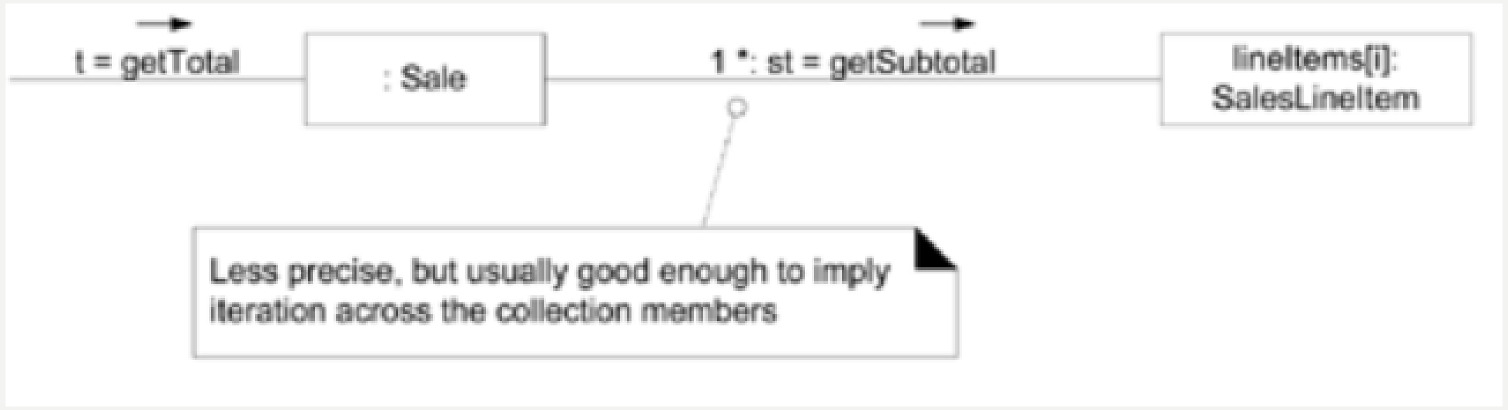
- Seq. Number * *[i := 1..N]*: message()
- "*" is required; […] clause Is optional


runSimulation
: Simulator
1 * [i = 1..n]: num = nextInt
: Random
iteration is indicated with a * and an optional iteration clause following the sequence number

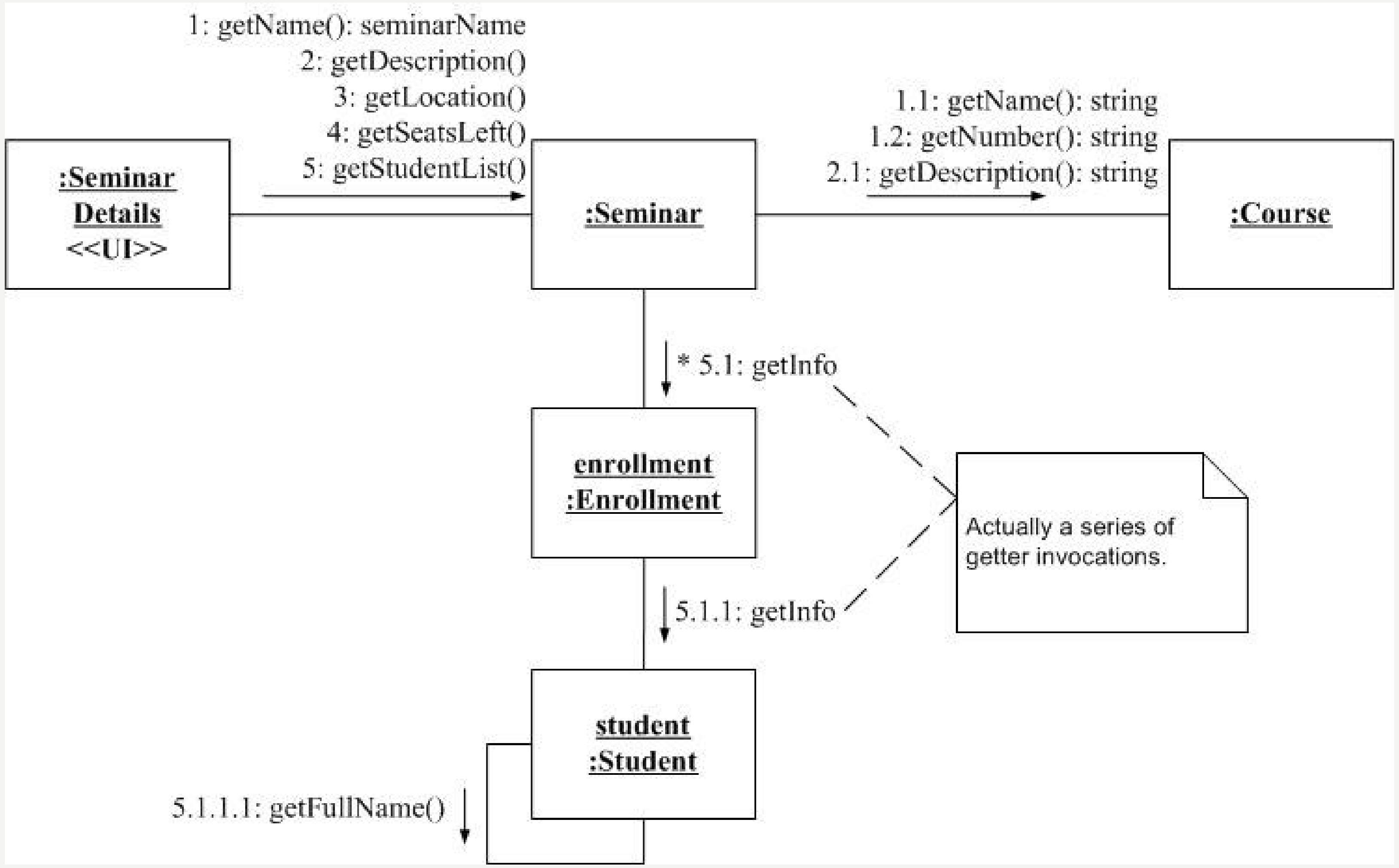
Loop

Iteration over a collection


t = getTotal
: Sale
1 * [i = 1..n]: st = getSubtotal
lineItems[i]:
SalesLineItem
this iteration and recurrence clause indicates we are looping across each element of the *lineItems* collection.
This lifeline box represents one instance from a collection of many *SalesLineItem* objects.
lineItems[i] is the expression to select one element from the collection of many SalesLineItems; the 'i" value comes from the message clause.


t = getTotal
: Sale
1 *: st = getSubtotal
lineItems[i]:
SalesLineItem
Less precise, but usually good enough to imply iteration across the collection members

Example


1: getName(): seminarName
2: getDescription()
3: getLocation()
4: getSeatsLeft()
5: getStudentList()
:Seminar Details <<UI>>
:Seminar
1.1: getName(): string
1.2: getNumber(): string
2.1: getDescription(): string
:Course
* 5.1: getInfo
enrollment :Enrollment
Actually a series of getter invocations.
5.1.1: getInfo
student :Student
5.1.1.1: getFullName()

1.5 BEHAVIORAL STATE MACHINE DIAGRAM

Behavioral State Machine

- Some of the classes in the class diagrams represent a set of objects that are quite dynamic in that they pass through a variety of states over the course of their existence.
  - For example, a patient can change over time from being new to current to former based on his or her status with the doctor's office.
- **A behavioral state machine** is a dynamic model that shows the different states through which a single object passes during its life in response to events, along with its responses and actions.
- Typically, behavioral state machines are not used for all objects; rather, behavioral state machines are used with complex objects to further define them and to help simplify the design of algorithms for their methods.

Behavioral State Machine

- The behavioral state machine shows the different states of the object and what events cause the object to change from one state to another.
- Behavioral state machines should be used to help understand the dynamic aspects of a single class and how its instances evolve over time unlike interaction diagrams that show how a particular use case or use-case scenario is executed over a set of classes.

The Components of Behavioral State Machine

1. **The state of an object** is defined by the value of its attributes and its relationships with other objects at a particular point in time.
   - For example, a patient might have a state of new, current, or former. The attributes or properties of an object affect the state that it is in; however, **not all attributes or attribute changes will make a difference.**
   - For example, think about a patient's address. Those attributes make very little difference to changes in a patient's state. However, if states were based on a patient's geographic location (e.g., in-town patients were treated differently than out-of-town patients), changes to the patient's address would influence state changes.

The Components of Behavioral State Machine

2. **An event** is something that takes place at a certain point in time and changes a value or values that describe an object, which, in turn, changes the object’s state.
   - It can be a designated condition becoming true, the receipt of the call for a method by an object, or the passage of a designated period of time. The state of the object determines exactly what the response will be.

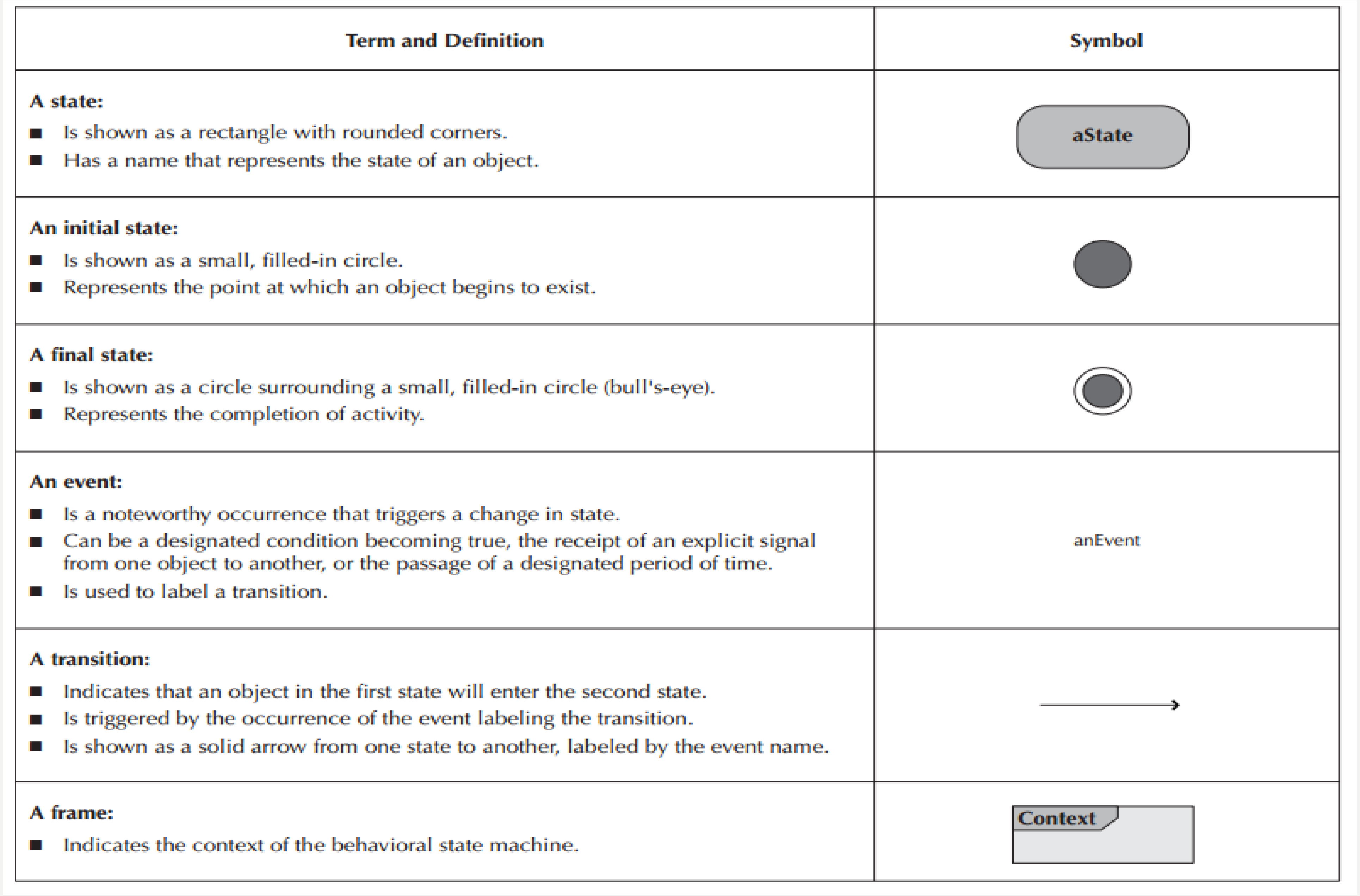
The Components of Behavioral State Machine

3. **A transition** is a relationship that represents the movement of an object from one state to another state. Some transitions have a guard condition.
    - A guard condition is a Boolean expression that includes attribute values, which allows a transition to occur only if the condition is true. An object typically moves from one state to another based on the outcome of an action triggered by an event.

The Components of Behavioral State Machine

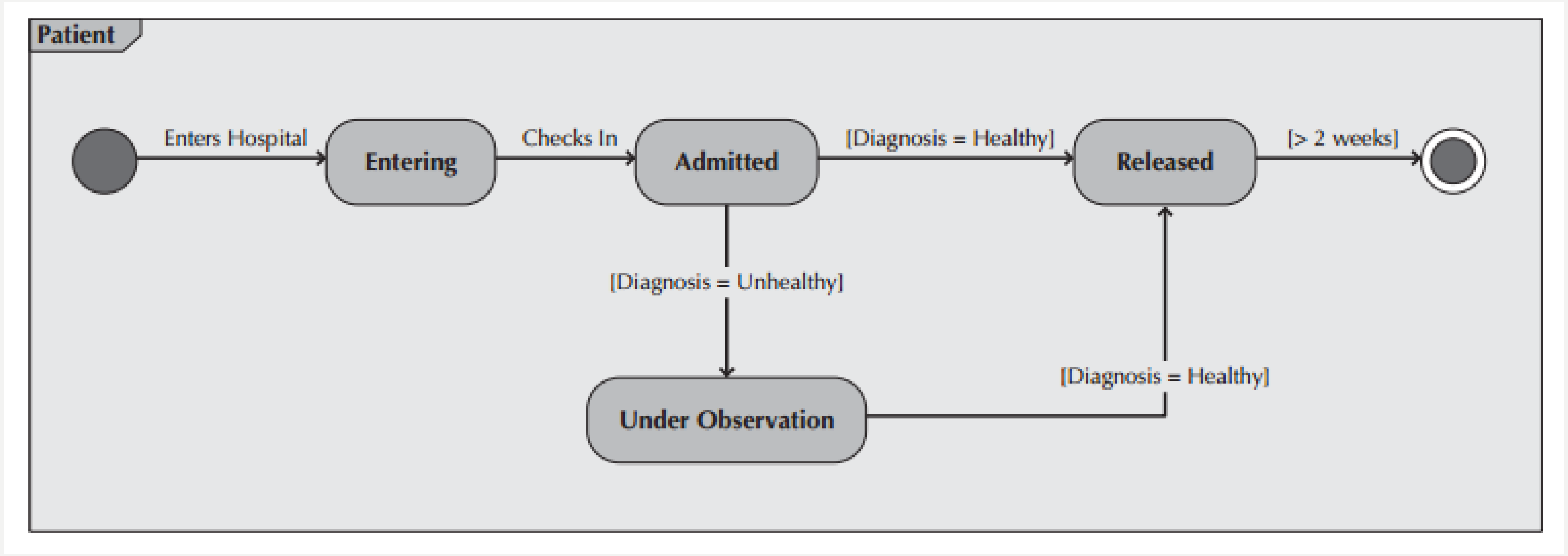
4. **An action** is an atomic, non decomposable process that cannot be interrupted. From a practical perspective, actions take zero time, and they are associated with a transition.
5. In contrast, **an activity** is a non atomic, decomposable process that can be interrupted. Activities take a long period of time to complete, and they can be started and stopped by an action.

Behavioral State Machine Diagram Syntax

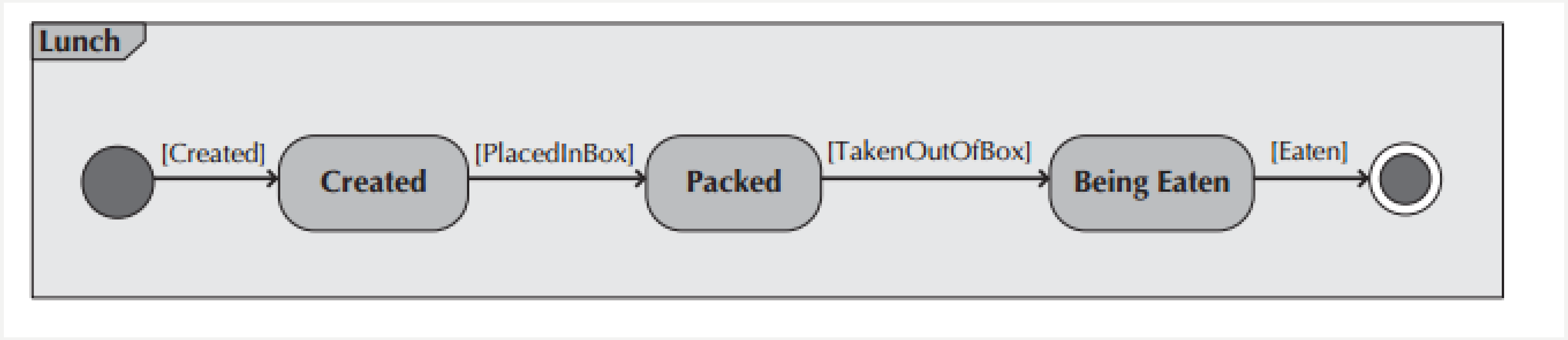


Term and Definition	Symbol
A state: ■ Is shown as a rectangle with rounded corners. ■ Has a name that represents the state of an object.	aState
An initial state: ■ Is shown as a small, filled-in circle. ■ Represents the point at which an object begins to exist.	●
A final state: ■ Is shown as a circle surrounding a small, filled-in circle (bull's-eye). ■ Represents the completion of activity.	◉
An event: ■ Is a noteworthy occurrence that triggers a change in state. ■ Can be a designated condition becoming true, the receipt of an explicit signal from one object to another, or the passage of a designated period of time. ■ Is used to label a transition.	anEvent
A transition: ■ Indicates that an object in the first state will enter the second state. ■ Is triggered by the occurrence of the event labeling the transition. ■ Is shown as a solid arrow from one state to another, labeled by the event name.	→
A frame: ■ Indicates the context of the behavioral state machine.	Context

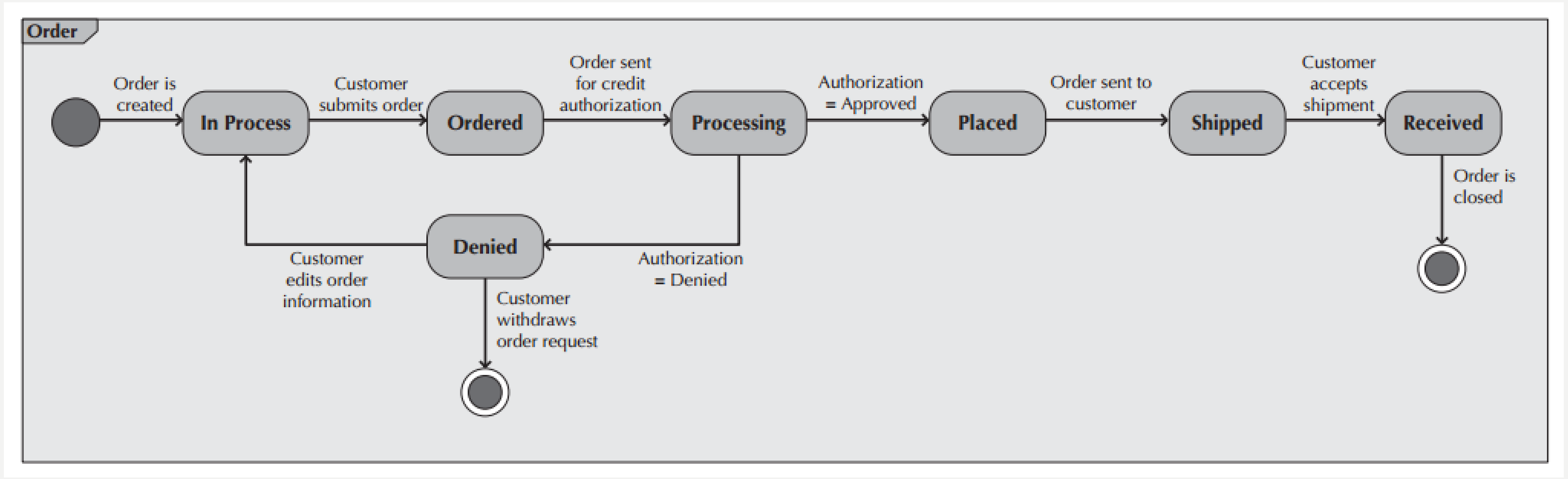
Example of Behavioral State Machine



Example of Behavioral State Machine



Example of Behavioral State Machine



States vs Subclasses

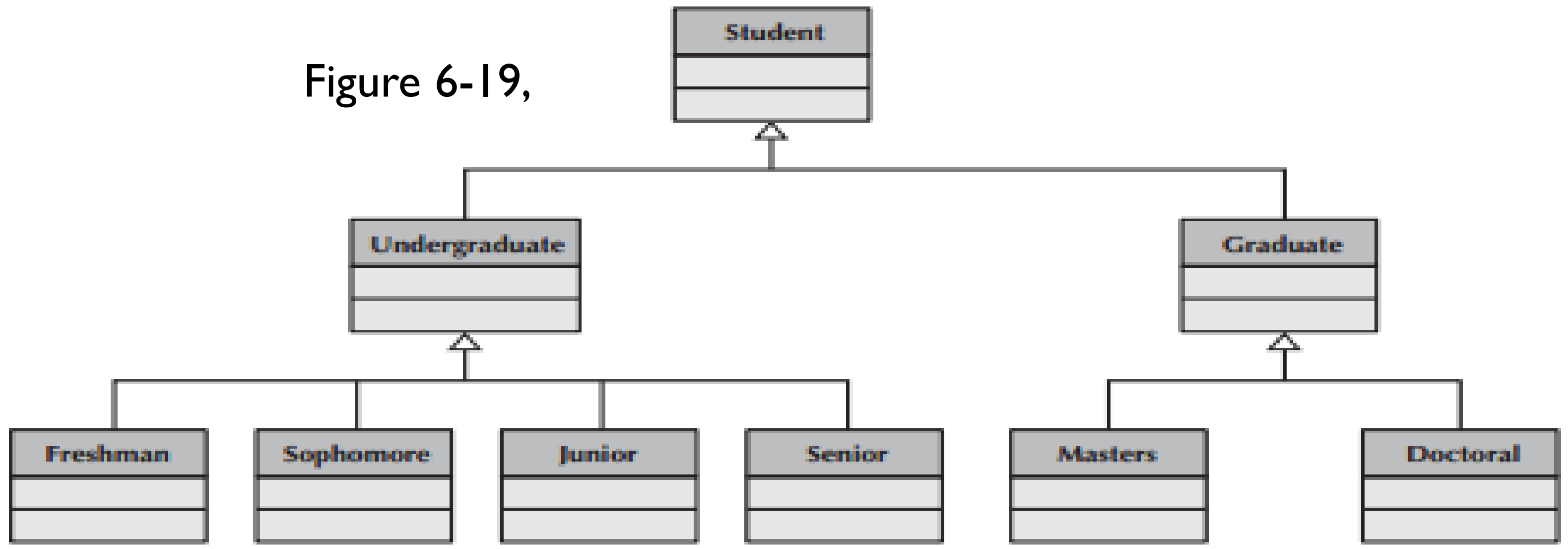
- Sometimes, states and subclasses can be confused.
  - For example, in Figure 6-19, are the classes Freshman, Sophomore, Junior, and Senior subclasses of the class Undergraduate or are they states that an instance of the Undergraduate class goes through during its lifetime?
  - In this case, the latter is the better answer.
- When trying to identify all potential classes during structural modeling (see Chapter 5), you might actually identify states of the relevant superclass instead of subclasses. This is another example of how tightly intertwined the functional, structural, and behavioral models can be.

States vs Subclasses

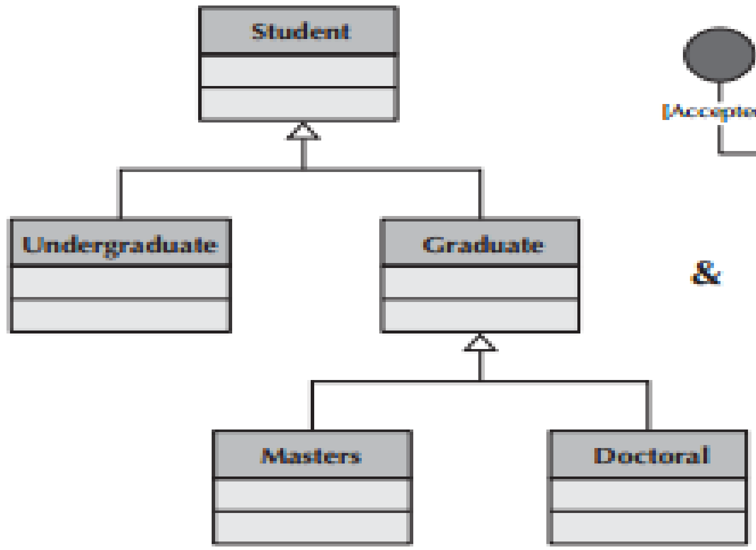
- From a modeling perspective, although we eventually removed the Freshman, Sophomore, Junior, and Senior subclasses from the structural model, capturing that information during structural modeling and removing it based on discoveries made during behavioral modeling were preferable to omitting it and taking a chance of missing a crucial piece of information about the problem domain.
- Remember, object-oriented development is iterative and incremental. As we progress to a correct model of the problem domain, we will make many mistakes.

States vs Subclasses

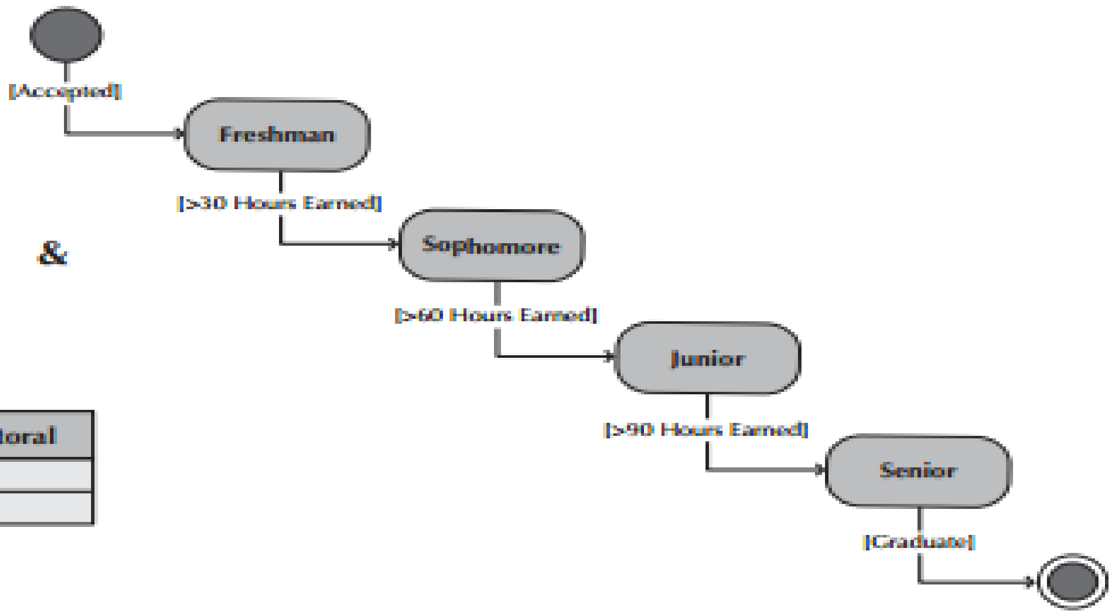
Figure 6-19,



vs.



&



Guidelines for Creating Behavioral State Machines

1. Create a behavioral state machine for objects whose behavior changes based on the state of the object. In other words, do not create a behavioral state machine for an object whose behavior is always the same regardless of its state. These objects are too simple.
2. To adhere to the left-to-right and top-to-bottom reading conventions of Western cultures, the initial state should be drawn in the top left corner of the diagram and the final state should be drawn in the bottom right of the diagram.

Guidelines for Creating Behavioral State Machines

3. Make sure that the names of the states are simple, intuitively obvious, and descriptive. For example in Figure 6-16, the state names of the patient object are Entering, Admitted, Under Observation, and Released.
4. Question black hole and miracle states. These types of states are problematic for the same reason black hole and miracle activities are a problem for activity diagrams (see Chapter 4). Black hole states, states that an object goes into and never comes out of, most likely are actually final states. Miracle states, states that an object comes out of but never went into, most likely are initial states

Guidelines for Creating Behavioral State Machines

5. Be sure that all guard conditions are mutually exclusive (not overlapping). For example, in Figure 6-16, the guard condition [Diagnosis =Healthy] and the guard condition [Diagnosis = Unhealthy] do not overlap. However, if you created a guard condition of [x >=0] and a second guard condition [x <=0], the guard conditions overlap when x =0, and it is not clear to which state the object would transition. Th is would obviously cause confusion.
6. All transitions should be associated with a message and operation. Otherwise, the state of the object could never change. Even though this may be stating the obvious, there have been numerous times that analysts forget to go back and ensure that this is indeed true.

Steps for Creating A Behavioral State Machines

1. Set Context
   - Examine your class diagram to identify which classes undergo a complex series of state changes and draw a diagram for each of them
   - The context of a behavioral state machine is usually a class. However, it also could be a set of classes, a subsystem, or an entire system.
2. Identify object states
3. Layout Diagram
4. Add transition
5. Validation

1.6 CRUDE ANALYSIS

CRUDE Analysis

- CRUDE analysis uses a CRUDE matrix, in which each interaction among objects is labeled with a letter for the type of interaction: C for create, R for read or reference, U for update, D for delete, and E for execute.
- In an object-oriented approach, a class/actor-by-class/actor matrix is used.
- Each cell in the matrix represents the interaction between instances of the classes.

CRUDE Analysis

- Unlike the interaction diagrams and behavioral state machines, a CRUDE matrix is most useful as a system-wide representation.
- Once a CRUDE matrix is completed for the entire system, the matrix can be scanned quickly to ensure that every class can be instantiated.
- Each type of interaction can be validated for each class.
    - For example, if a class represents only temporary objects, then the column in the matrix should have a D in it somewhere. Otherwise, the instances of the class will never be deleted. Because a data warehouse contains historical data, objects that are to be stored in one should not have any U or D entries in their associated columns.

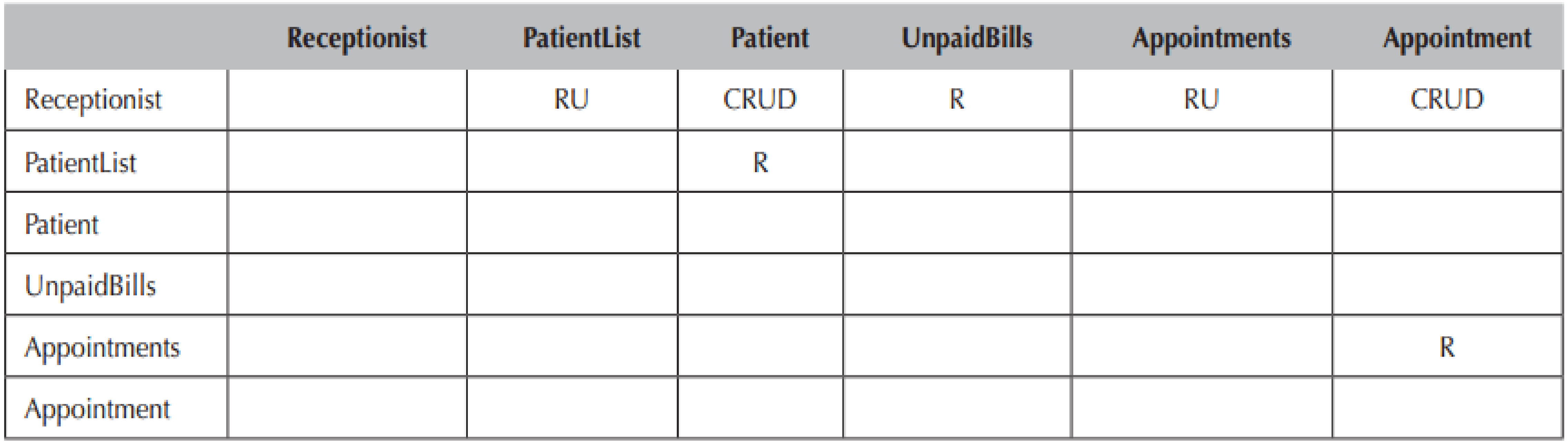
CRUDE Analysis

- Finally, the more interactions among a set of classes, the more likely they should be clustered together in a collaboration.
- However, the number and type of interactions are only an estimate at this point in the development of the system. Care should be taken when using this technique to cluster classes to identify collaborations. We return to this subject in the next chapter when we deal with partitions and collaborations.
- CRUDE analysis also can be used to identify complex objects. The more (C)reate, (U)pdate, or (D)elete entries in the column associated with a class, the more likely the instances of the class have a complex life cycle.
- As such, these objects are candidates for state modeling with a behavioral state machine.

CRUDE Analysis

- In this way, CRUDE analysis can be used as a way to partially validate the interactions among the objects in an object-oriented system.
- It could be the case, by creating a CRUDE matrix, we discovered an additional requirements that had previously been **overlooked**. Consequently, we need to **go back** and add additional the associated use cases, activity diagrams, sequence diagrams, communication diagrams and review the class diagrams, CRC cards and behavioral state machines to ensure that they are still correct.

CRUDE Matrix for the Make Old Patient Apt Use Case



	Receptionist	PatientList	Patient	UnpaidBills	Appointments	Appointment
Receptionist		RU	CRUD	R	RU	CRUD
PatientList			R			
Patient						
UnpaidBills						
Appointments						R
Appointment						

FIGURE 6-23 CRUDE Matrix for the Make Old Patient Apt Use Case

1.6 VERIFYING AND VALIDATING THE BEHAVIORAL MODEL

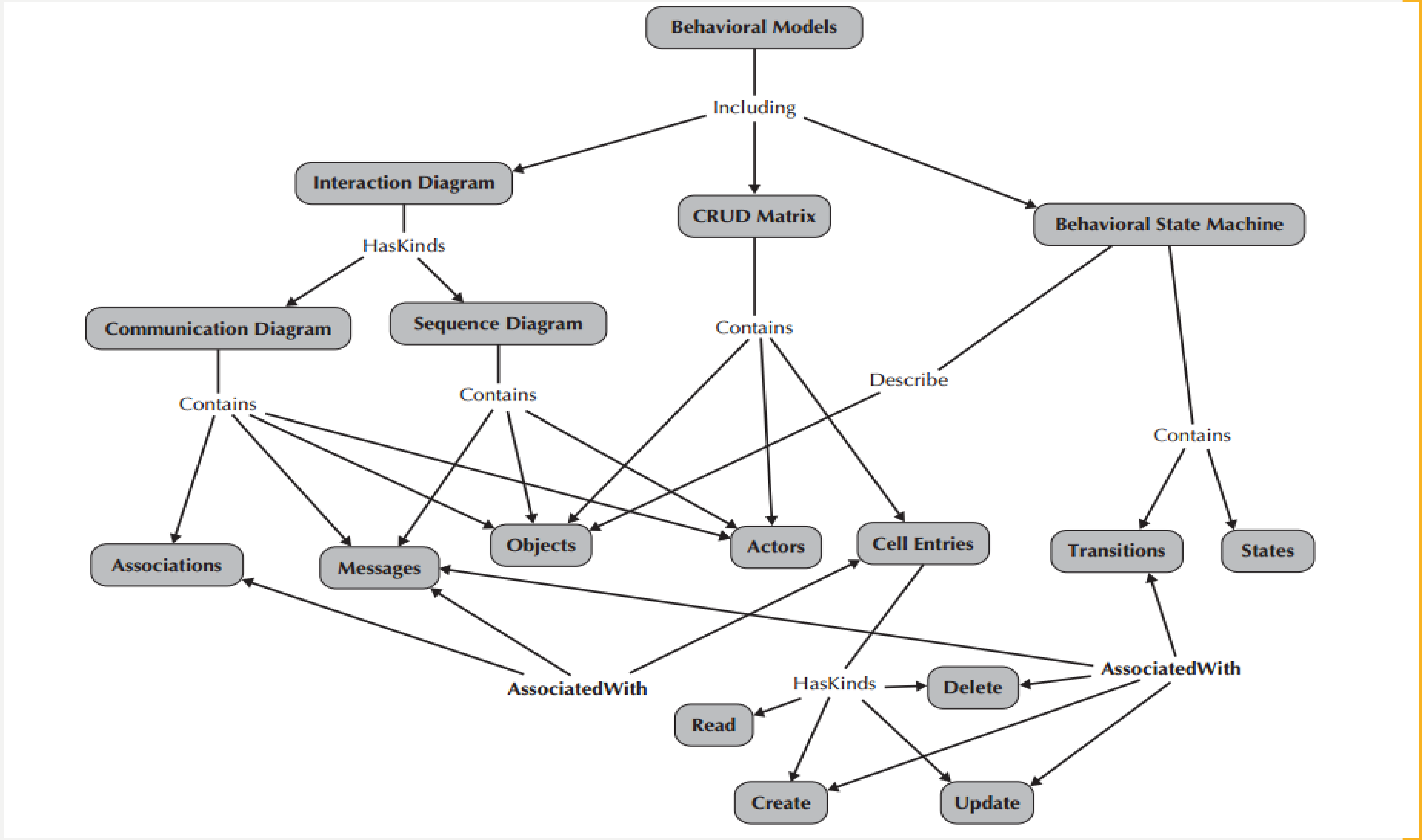
Verifying and Validating The Behavioral Model

- First, every actor and object included on a sequence diagram must be included as an actor and an object on a communication diagram, and vice versa.
- Second, if there is a message on the sequence diagram, there must be an association on the communications diagram, and vice versa.
- Third, every message that is included on a sequence diagram must appear as a message on an association in the corresponding communication diagram, and vice versa.
- Fourth, if a guard condition appears on a message in the sequence diagram, there must be an equivalent guard condition on the corresponding communication diagram, and vice versa.

Verifying and Validating The Behavioral Model

- Fifth, the sequence number included as part of a message label in a communications diagram implies the sequential order in which the message will be sent. Therefore, it must correspond to the top-down ordering of the messages being sent on the sequence diagram.
- Sixth, all transitions contained in a behavior state machine must be associated with a message being sent on a sequence and communication diagram, and it must be classified as a (C)reate, (U)pdate, or (D)elete message in a CRUDE matrix.
- Seventh, all entries in a CRUDE matrix imply a message being sent from an actor or object to another actor or object. If the entry is a (C)reate, (U)pdate, or (D)elete, then there must be an associated transition in a behavioral state machine that represents the instances of the receiving class.

Interrelationships Among Behavioral Models



Your Turn ☺ - Case in Previous Chapter

The borrowing activities are built around checking books out and returning books by borrowers. There are three types of borrowers: students, faculty or staff, and guests. Regardless of the type of borrower, the borrower must have a valid ID card. If the borrower is a student, having the system check with the registrar's student database validates the ID card. If the borrower is a faculty or staff members, having the system check with the personnel office's employee database validates the ID card. If the borrower is a guest, the ID card is checked against the library's own borrower database. If the ID card is valid, the system must also check to determine whether the borrower has any overdue books or unpaid fines. If the ID card is invalid, the borrower has overdue books, or the borrower has unpaid fines, the system must reject the borrower's request to check out a book, otherwise the borrower's request should be honored.

Create an Sequence Diagram and Communication Diagram for Borrow Books (Use Case) for the Students with a valid ID, no fines, and available book scenario.

References

- Dennis, Alan, et. al., System Analysis and Design with UML 5rd Edition, John Wiley & Sons, 2015.
- Larman, Craig. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, 3rd Edition, Pearson Education International, USA, 2005.
- Unified Modeling Language. Available at: http://www.omg.org/spec/UML/ last accessed April 20th 2015.
- Communication Diagram. Available at: http://en.wikipedia.org/wiki/Communication_diagram last accessed April 23rd 2015.