Object Oriented Analysis & Design

Week 6

Interaction Diagram

Lecturer: Dr. Msagha J Mbogholi, PhD

Flashback from Lesson 5

- Kruchten (1995) described the architectural views of software development as the 5 views as follows: logical, process, physical, deployment, and use case.
- There is a 4+1 view of the OO model. For each view there are a number of diagrams that denote the respective view of the system's model. The views are process, deployment, logical and dynamic. The +1 view is the use case view.
- The Use case view – this is considered the pivot since it drives the development of all the other views from the requirements phase.
- The components or parts of the use case diagram are: actor, use case, system boundary, relationships (between actors, actors and use cases, and between use cases themselves).

Content

- Introduction
- CRC Cards
- Communication Diagrams
- Collaboration Diagrams
- Sequence Diagrams
- Examples

Part 1

Introduction

Introduction

- In lesson 5 we discussed the use case diagram and its role in the overall system development. The emphasis was that the use case drives the development of all the other views of the OO model. Further the use case starts being developed right from the requirements phase.
- The use case diagram presents an outside view of the system
- Interaction diagrams describe how use cases are realized as interactions among societies of objects.
- There are 3 types of interaction diagrams
  - Sequence diagrams
  - Collaboration diagrams
  - Communication diagrams.

Introduction (cont'd)

- Interaction diagrams are models that describe how groups of objects collaborate in some behavior
- There are 3 kinds of interaction diagrams
  - Sequence diagram
  - Collaboration diagram
  - Communication diagram
- Sequence diagrams are a temporal representation of objects and their interactions. This means that they represent objects and their interactions over a time element.
- Collaboration and communication diagrams are spatial representation of objects, links and interrelations. This means that they are not concerned with time; rather they are concerned with how objects interact in the greater dimension of space.

Introduction (cont'd)

- Before beginning the discussion on the individual interaction diagrams we introduce class – responsibility – collaboration (CRC) cards as they will be used in most of the representations (diagrams) going forward.
- Interaction diagrams, use case diagrams, state chart diagrams and activity diagrams all capture the behavioral aspects of the system in development. This behavioral view represents the dynamic view of the model.
- The major views of the system that UML supports are: 1) the user view, 2) the structural view, 3) the behavioral view, and 4) the implementation view. One or more diagrams for each view is defined by the UML and each provides a unique window into the system. (https://www.e-education.psu.edu/geog468/book/export/html/1799)

Part 2

CRC Cards

Introduction

- *"CRC (Class–Responsibility–Collaboration) cards* are used to document the responsibilities and collaborations of a class. In some object-oriented systems-development methodologies, CRC cards are seen to be an alternative competitor to the Unified Process employment of use cases and class diagrams. However, we see them as a useful, low-tech approach that can compliment a typical high-tech Unified Process approach that uses CASE tools. We use an extended form of the CRC card to capture all relevant information associated with a class. ((Dennis et al., 2015).
- Typically the CRC cards are not part of the UML. However, they are very useful in identifying classes and their associations during the initial stages of design.
- We describe the roles and responsibilities described in the card and how to go about creating them.

2.1 Responsibilities and collaborations

- The responsibilities of a class can be broken down into simply 2 areas:
- Identification of attributes and relationships
- Identification of its operations
- Recall from foundation knowledge that a class is a blueprint from which objects can be created. Applying this to this definition means that objects instantiated from the class must have attributes and relationships with other objects (of the class or other classes); objects must have operations that allow them to perform message passing among themselves.
- The structural model describes the objects necessary to support the business processes modeled by the use cases. The relationships between classes can be aptly described as collaborations since classes collaborate with each other to achieve the overall objective of the system. For example a student class can collaborate with a book class and a librarian class in a library system to fulfill the function (use case) of borrowing a book.
- Thus system analysts can use the concept of role playing to identify the classes, attributes and operations of each class. Dennis et al. (2015) suggest the use of anthropomorphism to aid in the identification.

2.1 Responsibilities and collaborations (cont'd)

- The describe it as follows: "Members of the development team can either ask questions of themselves or be asked questions by other members of the team. Typically the questions asked are of the form:
- Who or what are you?
- What do you know?
- What can you do?
- The answers to the questions are then used to add detail to the evolving CRC cards."
- By asking these questions the development team can come up with class names, their attributes and methods.

2.2 Contents

- A CRC card contains the following elements:
- Class name at the top. Class names should always be nouns. A class ID should also be added to uniquely identify the class.
- Class description – a general description of the class so that it is easily understood what this class is.
- Class type – the type of class it is.
- Use cases associated with the class.
- Class responsibilities on the left hand side
- Class collaborators on the right hand side, that is, those classes that help to carry out the responsibilities of this class.
- Fig 1 shows a sample CRC card.

Front:

Class Name: Old Patient	**ID:** 3	**Type:** Concrete, Domain
Description: An individual who needs to receive or has received medical attention		**Associated Use Cases:** 2

Responsibilities	**Collaborators**
Make appointment	Appointment
Calculate last visit	
Change status	
Provide medical history	Medical history

Back:

Attributes:

Amount (double)	
Insurance carrier (text)	

Relationships:

Generalization (a-kind-of): Person

Aggregation (has-parts): Medical History

Other Associations: Appointment

Fig 1. Sample CRC card (Dennis et al., 2015)

2.3 CRC card example

- Consider the following case study provided by Stevens and Pooley (2006):
- "You have been contracted to develop a computer system for a university library. The library currently uses a 1960s program, written in an obsolete language, for some simple bookkeeping tasks, and a card index for user browsing. You are asked to build an interactive system which handles both of these aspects online."
- In such a scenario some of the rather obvious classes that will come up include LibraryMember (the one who borrows books), copy(since members will borrow copies of books), and book (the details of the books to be borrowed). Of course there are other classes that will come into play but we use these ones to demonstrate how CRC cards can be made for them.
- Fig 2 shows sample CRC cards for these classes. Some fields are left blank unlike in fig 1, as these are just for demo purposes.

LibraryMember	
Responsibilities	**Collaborators**
Maintain data about copies currently borrowed Meet requests to borrow and return copies	 Copy

Copy	
Responsibilities	**Collaborators**
Maintain data about a particular copy of a book Inform corresponding Book when borrowed and returned	 Book

Book	
Responsibilities	**Collaborators**
Maintain data about one book Know whether there are borrowable copies	

Fig 2. Sample CRC cards (Steven and Pooley, 2006)

Part 3

Communication Diagrams

Introduction

- The term collaboration is used to describe all the interacting objects together with their links (or rather how they link with each other).
- A collaboration where there are no links (interactions) is most useful in the class diagram to identify objects, links and the actors. The syntax used is as follows:
- Object – each object name is shown in a rectangle. The format for displaying the name is *objectName: className*. In those instances where there is no need to mention an object directly using the class name alone will suffice; however, ensure that the colon mark : before the class name is present to differentiate class name from object name, for example, *:animal*.
- Link – it is normally shown as a straight line between interacting objects (classes). In some instances the line may have an arrowhead to emphasize a particular type of relationship.
- Actor – as we already know, actors are roles with respect to the system. Actors perform use cases; the notation for the actor in every case is the stickman. There are several actors but the one who intiates the use case is known as the initiator.
- Fig 3 shows a simple collaboration with no interaction.

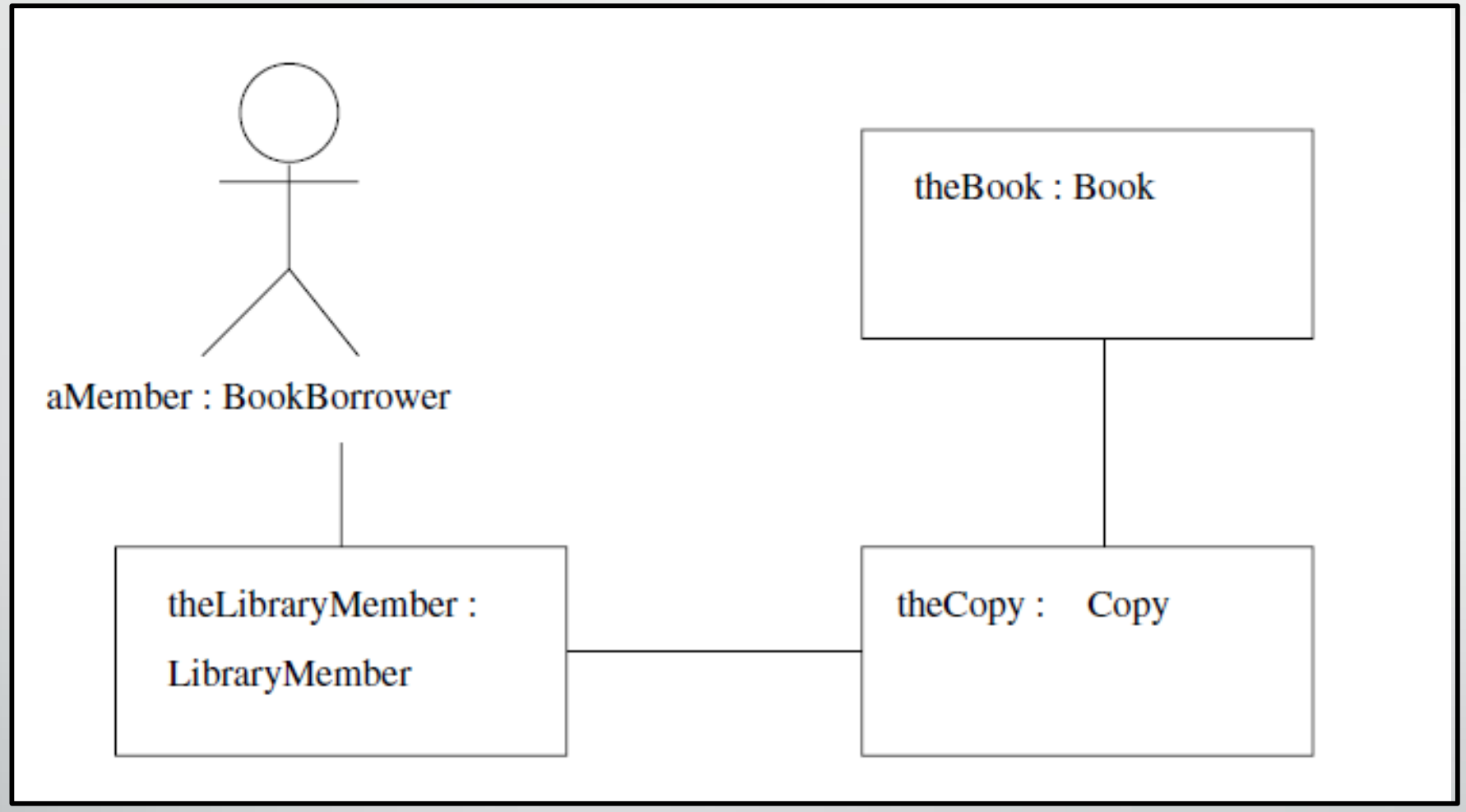


Fig 3. Simple collaboration with no interaction (Stevens and Pooley, 2006)

3.1 Communication Diagram

- Communication diagrams represent part of the dynamic view of the system. They show how objects collaborate to implement a use case or use case scenario.
- The emphasis of a communication diagram is to show the messages that pass between objects in the use case scenario; thus communication diagrams are more concerned with the spatial rather than temporal representation of the interactions.
- The members are the actors, objects and communication links between them.
- Table 1 shows the syntax used in a communication diagram.

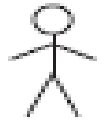
Term and Definition	Symbol
An actor: ■ Is a person or system that derives benefit from and is external to the system. ■ Participates in a collaboration by sending and/or receiving messages. ■ Is depicted either as a stick figure (default) or, if a nonhuman actor is involved, as a rectangle with <<actor>> in it (alternative).	 anActor <<actor>> anActor
An object: ■ Participates in a collaboration by sending and/or receiving messages.	anObject : aClass
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.	Context

Table 1. Communication diagram syntax (Dennis et al., 2015)

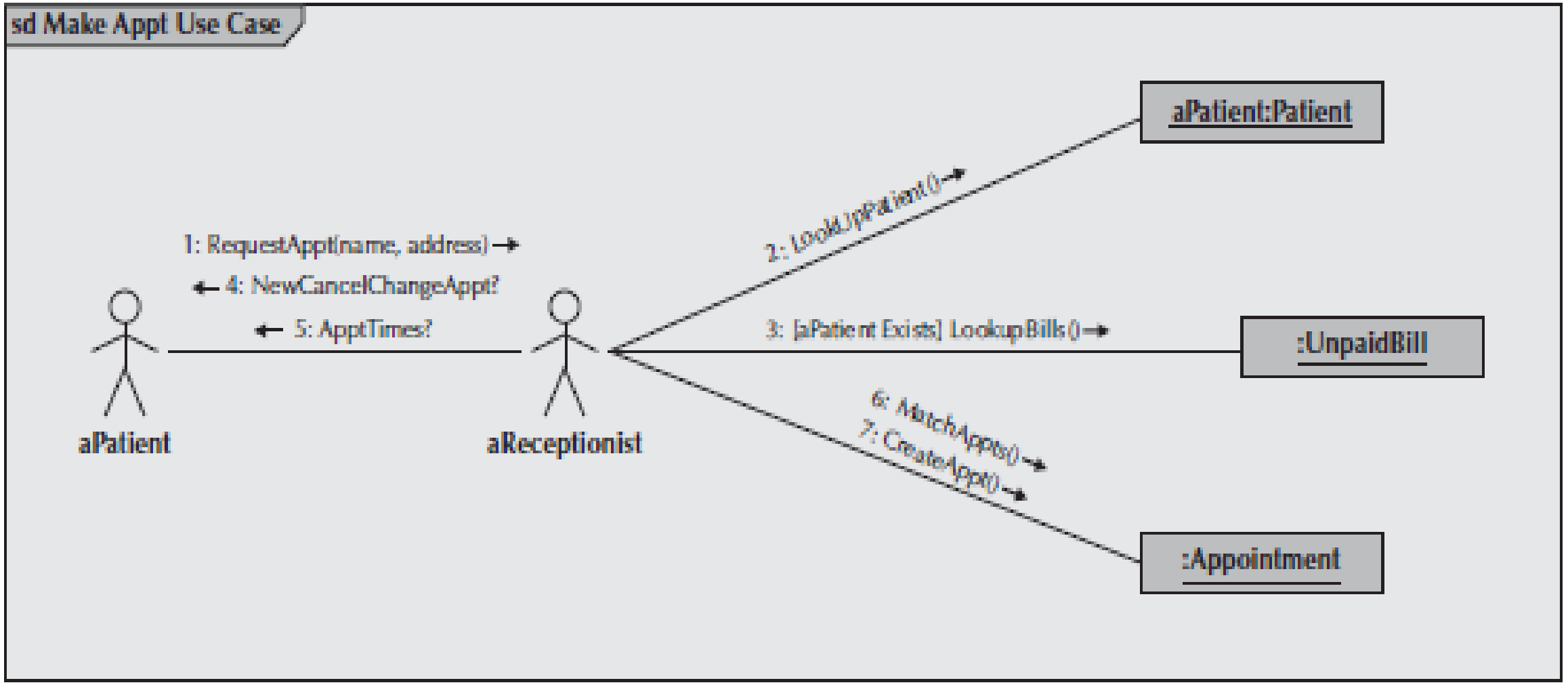


Fig 4. Sample communication diagram (Dennis et al., 2015)

3.1 Communication Diagram (cont'd)

- Fig 4 shows a sample communication diagram for the 'make appointment' use case. Notice how easy it is to understand for a non technical user even if you are not familiar with the whole case study? Lets give it a go.
- A patient requests for an appointment giving their name and address [1].
- The receptionist looks up the patient details in the patient class for this particular patient (aPatient: Patient), that is, following the syntax (*objectName: className)* [2]
- After getting the details and confirming that the patient exists, the receptionist then checks if the patient has any pending/unpaid bills [3].
- The receptionist then confirms from the patient whether they wish to make a new appointment, cancel an existing one, or change the time for an existing one [4].
- After getting this information the receptionist gives the patient the appointment time [5].
- The receptionist then checks the appointment class for details of existing appointments to match this one in [6].
- The receptionist adds this appointment [7].

3.2 Communication Diagram guidelines

- Ambler (as cited in Dennis et al., 2015) provides the following 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.
- 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.
- These guidelines should be used in conjunction with those for sequence diagrams (covered later in this lesson).

3.3 Steps in creating a communication diagram

- Dennis et al. (2015) provide a 5 step process in building a communication diagram:"
- Step 1 (set context): determine the context of the communication diagram. Like a sequence diagram, the context of the diagram can be a system, a use case, or a scenario of a use case. The context of the diagram is depicted as a labeled frame around the diagram (see table 1).
- Step 2 (identify objects, actors and associations): identify the objects (actors) and the associations that link the objects (actors) that participate in the collaboration together.
- Step 3 (lay out the diagram): lay out the objects (actors) and their associations on the communication diagram by placing them together based on the associations that they have with the other objects in the collaboration.
- Step 4 (add messages): add the messages to the associations between the objects. We do this by adding the name of the message(s) to the association link between the objects and an arrow showing the direction of the message being sent. Each message has a sequence number associated with it to portray the time-based ordering of the message. This is not to be confused with the sequence diagram as shall be demonstrated later.
- Step 5 (validate): to validate the communication diagram. The purpose of this step is to guarantee that the communication diagram faithfully portrays the underlying process(es). This is done by ensuring that all steps in the process are depicted on the diagram."

Part 4

Collaboration Diagrams

Introduction

- Strictly speaking a collaboration diagram is a form of communication diagram and the differences between the two lie in how we represent them using UML notation.
- Collaboration diagrams illustrate the interaction between the objects, using spatial structure; this structure is not concerned with time order of the messages but rather the sequence in which they occur.
- Unlike sequence diagram the time is not explicitly represented in these diagrams
- In collaboration diagram the sequence of messages is indicated by numbering the messages. The UML uses the decimal numbering scheme, that is the sequence of messages begins from 1, 2, 3, ….and so on.
- In these diagrams, an actor can be displayed in order to represent the triggering of interaction by an element external to the system.

4.1 Components

- There are 3 components of the collaboration diagram:
- Objects – these are represented using clear distinguished names. The format is the same as for other communication diagrams, that is, *objectName: className.*
- Links – these are represented by a continuous line between the communicating objects.
- Messages – a message is a communication between objects (though an object can send a message to itself called a self message) and will have the following attributes:
  - thread name (or thread within a thread as described by Stevens and Pooley, 2006),
  - Sequence number which shows the order of the message among all other messages
  - Message label which is a description of the message being sent from one object to another
  - Message direction (showing the sender and recipient of the message, that is, an arrow from sender to receiver)

4.2 Ordering and representation

- A message in a collaboration diagram will have the following details:
- Role names – the name of the actor sending the message
- Message qualifiers include:
  - Iteration expression – whether the message will be repeated any number of times (usually denoted with an asterisk *)
  - Parameters – the applicable parameters of the message
  - Return values – whether the message has any return values (this is usually optional unless the return value is not obvious)
  - Guard – any guards should also be carried in the message
  - Concurrent thread sequencing – this happens in non procedural environments. In a procedural environment only one object is computing at any given time. However, object oriented environment we have more than one object computing at any given time. In order to represent this nesting and sub nesting is used. For example if object A is communicating with object B, the latter (B) may have to perform other actions that will enable it to send a response to A. If the message sequence is 2, then the actions to enable 2 to happen will be labeled 2.1, 2.1.1, and so on, indicating the different levels of threading.
  - Thread dependencies – thread dependencies are also implemented using the nesting technique.

4.3 Representation

- A common scenario used in object oriented analysis and design to explain different concepts is the withdrawal of cash use case. There are very many literatures that refer to this use case to explain the use case, interaction diagrams, and even the class diagram; the learner is encouraged to do some online searches to find this use case with accompanying diagrams.
- Fig 5 shows the license renewal collaboration diagram. Notice the sub threading and numbering used in the whole process.
- Collaboration diagrams should be used together with sequence and communication diagrams to enable the developers get a clear picture of the dynamic view (behavioral view) of the system.

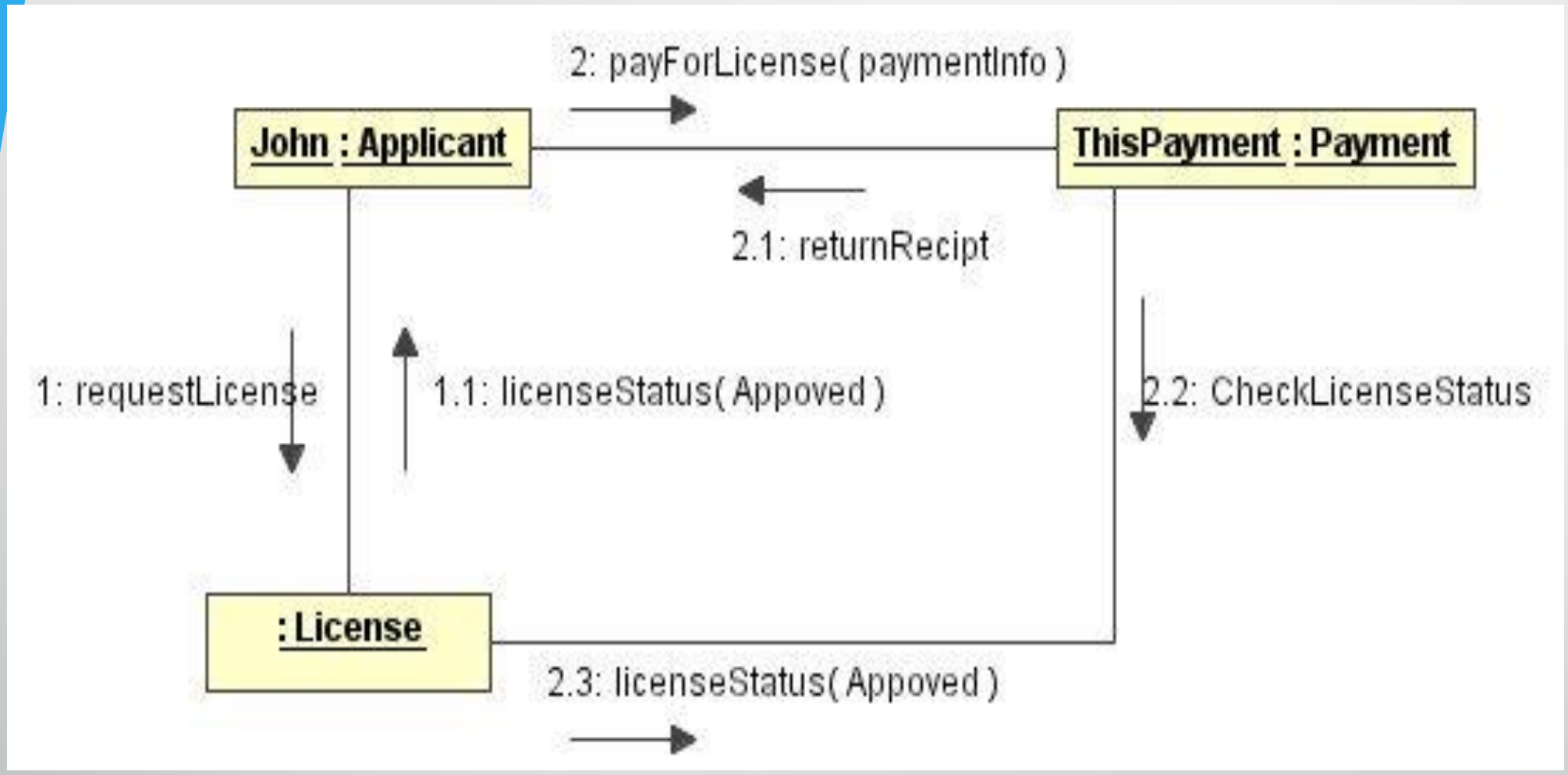


Fig 5. License renewal collaboration diagram (Ojo and Estevez, 2005)

Part 5

Sequence Diagrams

Introduction

- The sequence diagram is arguably the most used interaction diagram; so in a way we have saved the best for last.
- The sequence diagram shows a temporal representation of the interactions between objects. It is temporal since the interest is in capturing the exchange of messages using time as the reference.
- In the communication and collaboration diagram we were concerned with simply using a message number to show which message comes first and subsequently.
- In the sequence diagram we not only get to have an idea of the ordering of events but we also get to know which messages take longer to execute due to the amount of time they take. This helps the developers in making judgements regarding certain aspects of the design and implementation of the system.

Introduction (cont'd)

- The sequence diagram is thus 2 dimensional: on the x axis there is a representation of the objects while the y – axis represents time component.
- The main components of the sequence diagram are:
- Objects
- Lifeline
- Message

5.1 Objects

- Objects are the main players in the sequence diagram. The syntax for naming an object is *objectName: className.* All objects must have clear unique names.
- In those instances where an object does not need to be referenced directly the className will suffice.
- Include instance names when objects are referred to in messages or when several objects of the same type exist in the diagram.
- Objects send messages to each other (and they can also create self messages).
- An object in a sequence diagram can be created or destroyed (via it's lifeline).
- In the UML objects in a sequence diagram are represented by rectangles with the object name inside the rectangle.
- Objects and classes should be named consistently with the class diagram.

5.2 Lifeline

- Lifelines represent how long an object takes part in an interaction.
- They are represented by a dotted line perpendicular to the rectangle representing the object.
- When an object has played its role in an interaction it can be destroyed by "killing" its lifeline.
- Activation box – this is box drawn on an object's lifeline to show how long the message takes to be executed. The length of the activation box is directly proportional to the execution time of the message. The activation box means either that object is running its code, or it is on the stack waiting for another object's method to finish.

5.3 Messages

- A message is represented by an arrow between the life lines of two objects.
  - Self calls are also allowed
  - The time required by the receiver object to process the message is denoted by an *activation-box.*
- A message is labeled at minimum with the message name.
  - Arguments and control information (conditions, iteration) may be included.
- Messages may be synchronous or asynchronous
- You may indicate a return value using a dashed arrow in some circumstances; for example, when the return value is not obvious or if it is needed in another part of the interaction.

5.3.1 Synchronous and asynchronous messages

- Synchronous messages are those which make the assumption that a return value(message) is required. The sender waits for the return value before proceeding with any other action. The usual notation applies (full arrow for sending and dashed arrow for reply). This is shown in fig 6(a)
- Asynchronous messages are those which a response is not needed in order for the sender to continue; the sender's role is simply to send the message. The usual notation for such a message is a half arrowhead as demonstrated in fig 6(b)

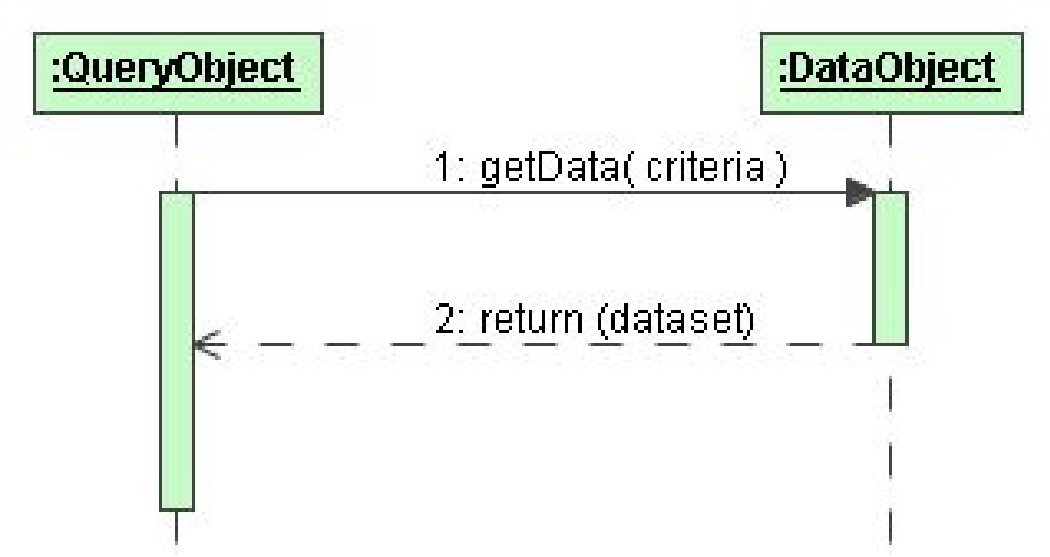


Fig 6(a) Synchronous message (Ojo and Estevez, 2005)

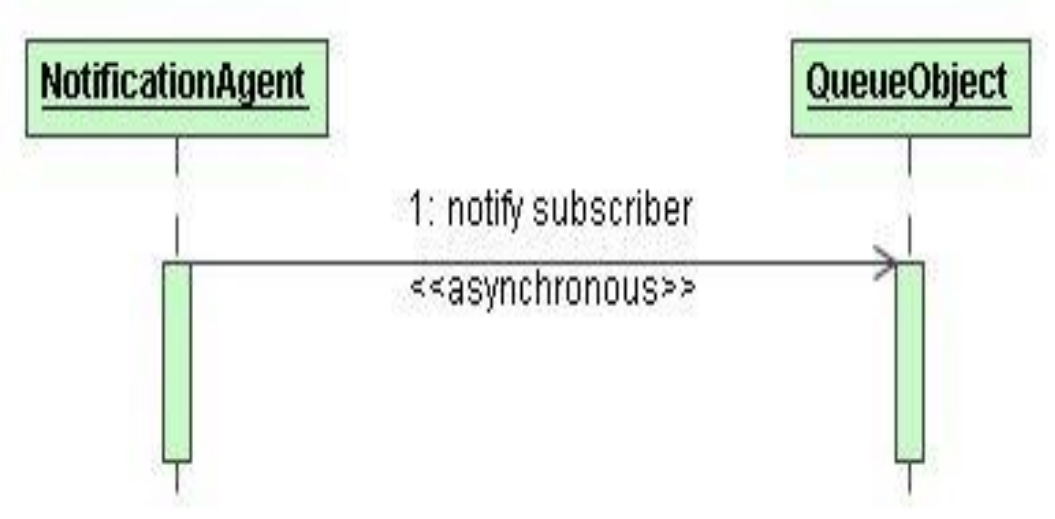


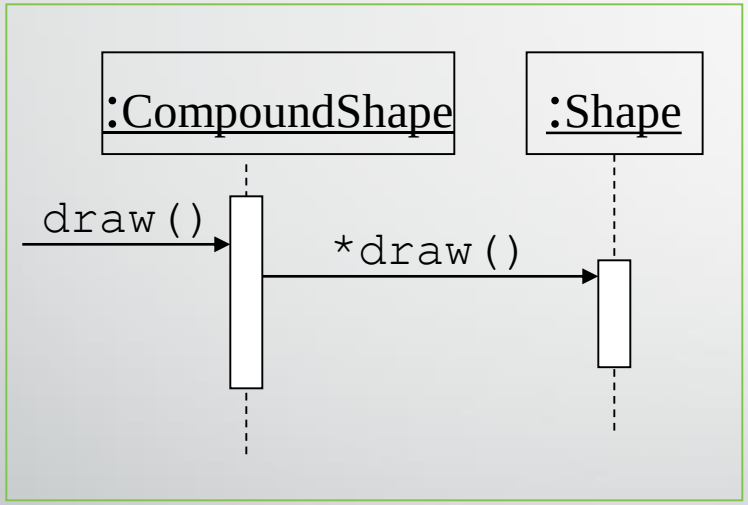
Fig 6(b) Asynchronous message (Ojo and Estevez, 2005)

5.4 Control information

- Conditions are expressed in sequence diagrams as follows:
  - syntax: ‘[‘ expression ‘]’ message-label
  - The message is sent only if the condition is true
  - example: [ok] borrow(member)
- Iteration is expressed in sequence diagrams as follows:
  - syntax: * [‘[‘ expression ‘]’] message-label
  - The message is sent many times to possibly multiple receiver objects.

5.4 Control Information (Cont.)

- Fig 7 (a) and 7(b) show some iteration examples:



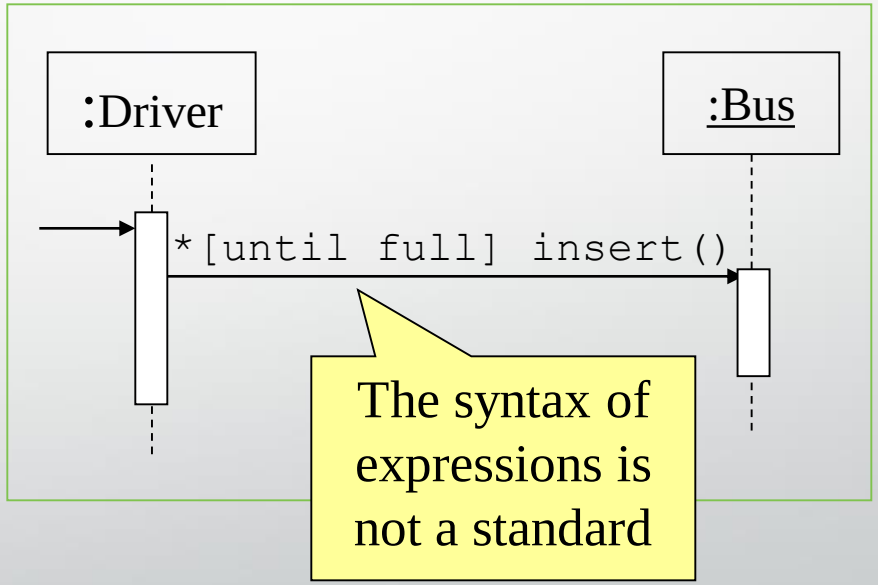


Fig 7(a) Iteration example 1 and Fig 7 (b) Iteration 2 (Gutterman and Karmi, n.d.)

5.4 Control Information (Cont.)

- An object can also send a message to itself called a self message. This means that the object calls itself in the message. Further messages may have user defined time attributes. When showing a timed message it may be slanted downwards (as opposed to straight horizontally across) to show it is timed.
- Fig 8 shows most of the components of a sequence diagram while fig 9 shows a timed message.
- Table 2 also summarizes all the notation used in sequence diagrams.
- Note that the control mechanisms of sequence diagrams suffice only for modeling simple alternatives.
  - Consider drawing several diagrams for modeling complex scenarios.
  - Don't use sequence diagrams for detailed modeling of algorithms (this is better done using *activity diagrams*, *pseudo-code* or *state-charts*).

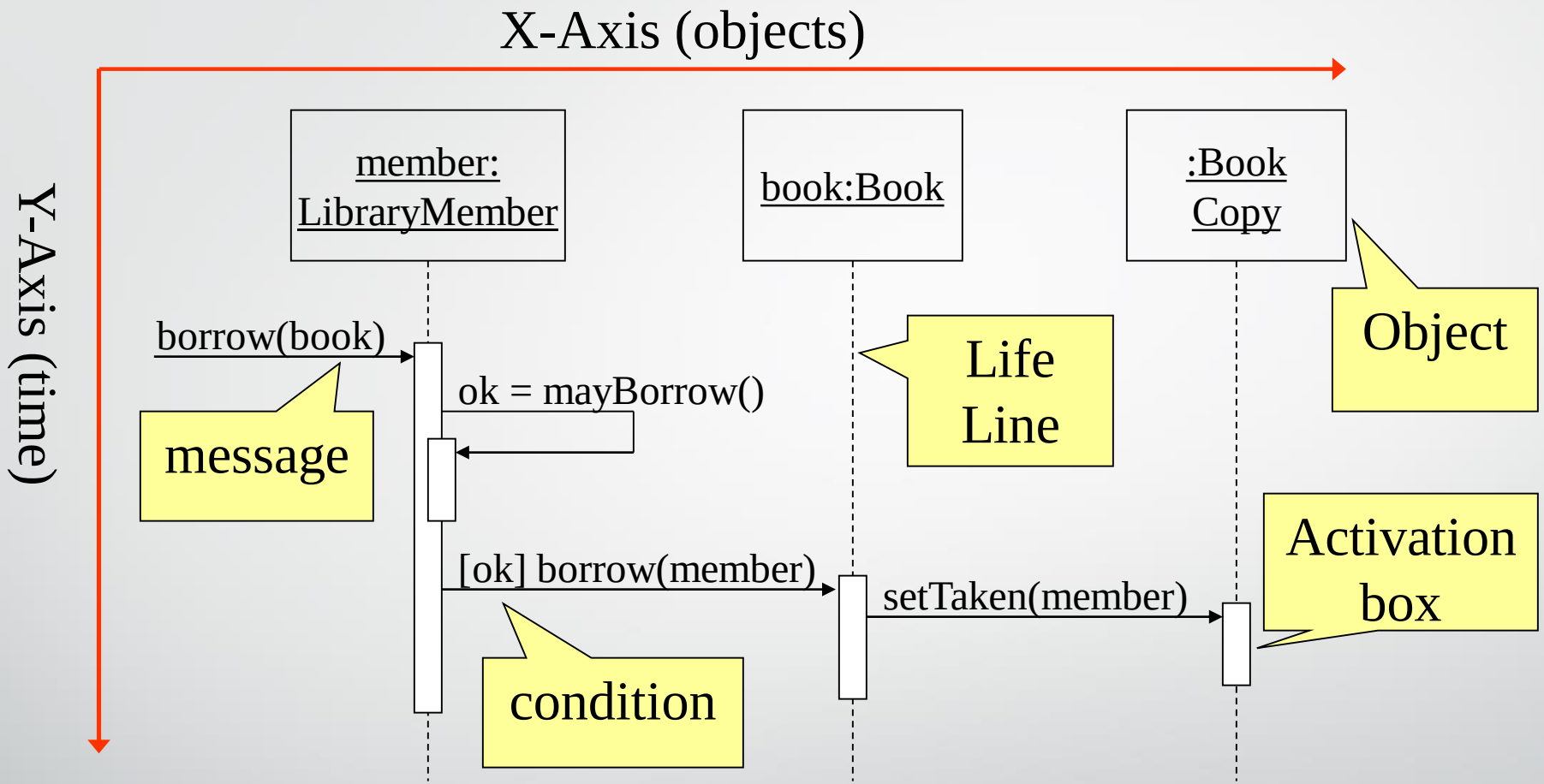


Fig 8. Sequence diagram components (Gutterman and Carmi, n.d.)

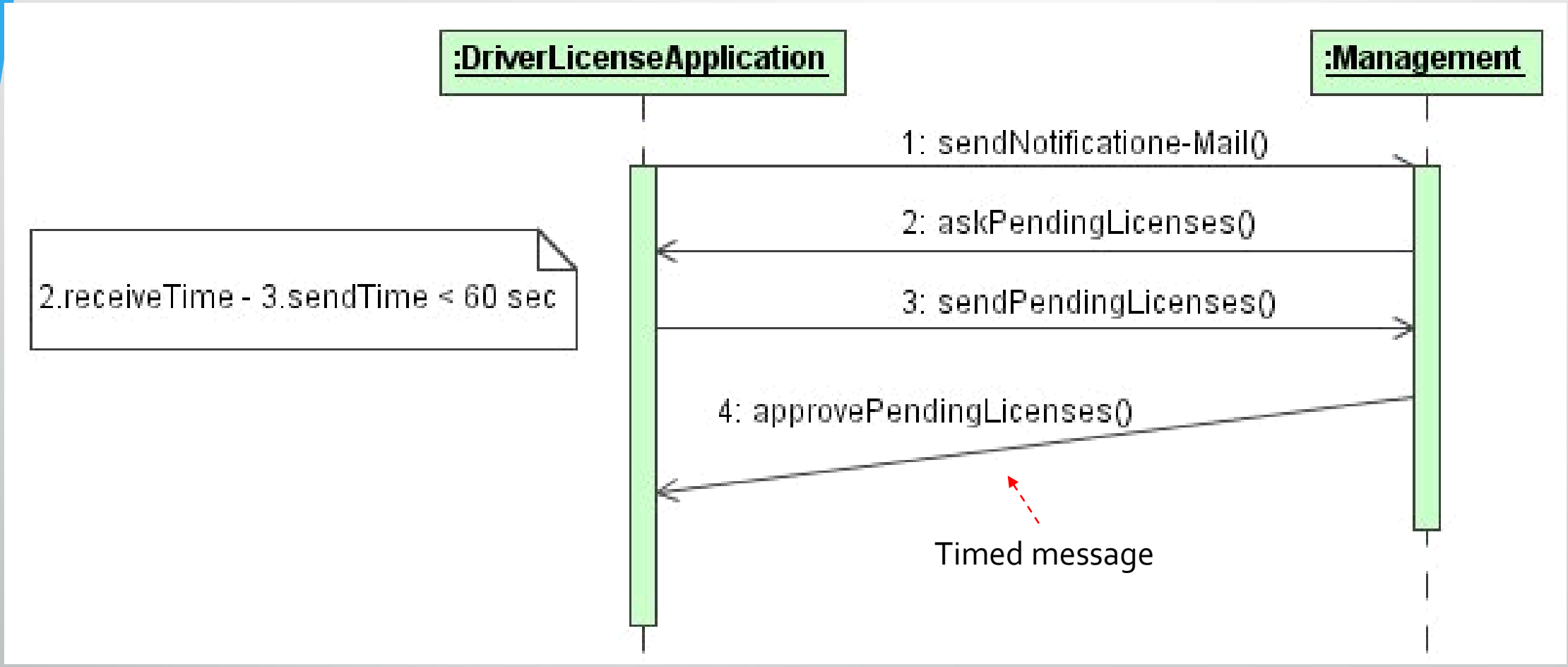


Fig 9. Timed message (Ojo and Estevez, 2005)

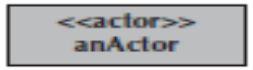
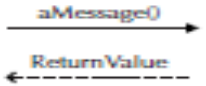
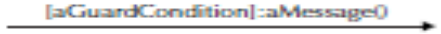
Term and Definition	Symbol
An actor: ■ Is a person or system that derives benefit from and is external to the system. ■ Participates in a sequence by sending and/or receiving messages. ■ Is placed across the top of the diagram. ■ Is depicted either as a stick figure (default) or, if a nonhuman actor is involved, as a rectangle with <<actor>> in it (alternative).	 anActor 
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.	
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.	
A guard condition: ■ Represents a test that must be met for the message to be sent.	
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.	

Table 2. Sequence diagram notation (Dennis et al., 2015)

5.5 Guidelines for creating sequence diagrams

- Ambler (as cited in Daniel et al., 2015) provides the following guidelines to create a sequence diagram:"
- 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.
- 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).
- 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.

5.5 Guidelines for creating sequence diagrams (cont'd)

- 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.
- 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.
- 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."

5.6 Steps in creating a sequence diagram

- Dennis et al. (2015) provide the following 6 steps to create a sequence diagram:
- Step 1 (set context) – set the context for the sequence diagram, be it a system, a use case, or a scenario of a use case.
- Step 2 – identify the actors and objects
- Step 3 – set the lifelines for each object in the sequence diagram
- Step 4 – determine and add the messages that will be sent among and between objects.
- Step 5 – show all the activation boxes for each object lifeline
- Step 6 – validate your diagram, ensuring every object, lifeline and message are incorporated in the right sequence along the timeline (y-axis).

Part 6

Examples

Example 1

- Design the Sequence diagram for our Courseware Management System case study application. Because a Sequence diagram represents the dynamic flows in an application, we will aim to represent one of the flows using a Sequence diagram. In a previous analysis, the following use cases for the Courseware Management System were defined:
- View courses
- Manage topics for a course
- Manage course information
- View course calendar
- View tutors
- Manage tutor information
- Assign courses to tutors
- We wish to draw the sequence diagram for the “manage course information “ use case.

Solution 1

- The sequence of steps carried out in the "Manage course information" flow are:
- A user who is a course administrator invokes the manage course functionality.
- The manage course functionality of the course administrator invokes either the course creation or course modification functionality of a course.
- After the course is either created or modified, the manage topic functionality of the course administrator calls the topic creation or modification functionality of a topic.
- Finally, the user invokes the assign tutor to course functionality of the course administrator to assign a tutor to the selected course.
- Now, let us model these steps into a Sequence diagram for the "Manage course information" functionality. The sequence diagram is represented in fig 10.

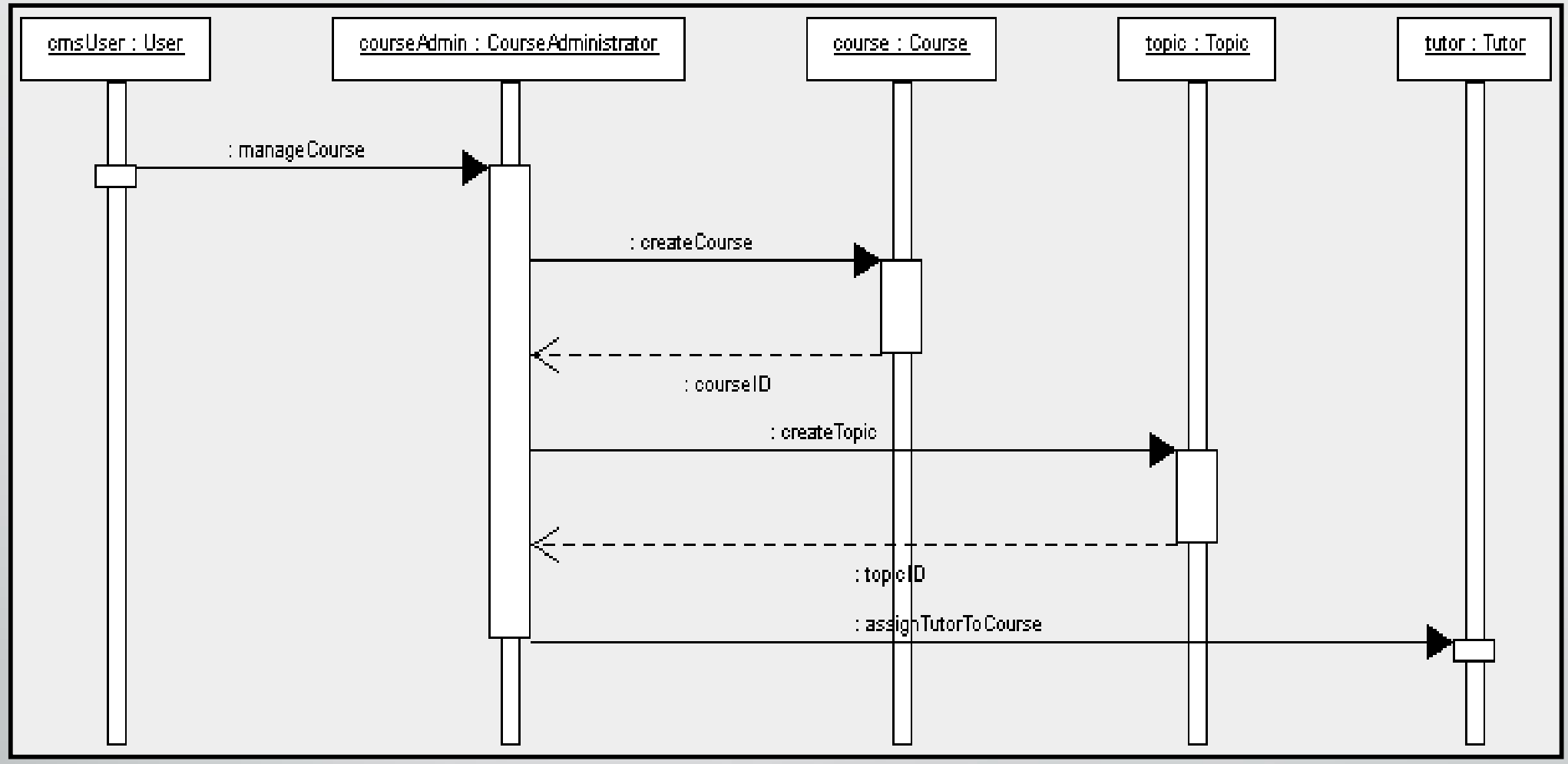


Fig 10. Sequence diagram for example 1

Example 2

- Sketch a sequence diagram for the "normal" operation of an ATM (i.e. a customer inserts the card, enters his/her PIN, enters the amount, takes the card, and takes the money). The sequence diagram should show the interaction between the different components of the ATM. The ATM consists of the following components:
- screen
- keyboard
- card reader
- money tray
- money output device
- interface to banking server (for modifications of the account data of the customer)
- Fig 11 shows the solution to this problem.

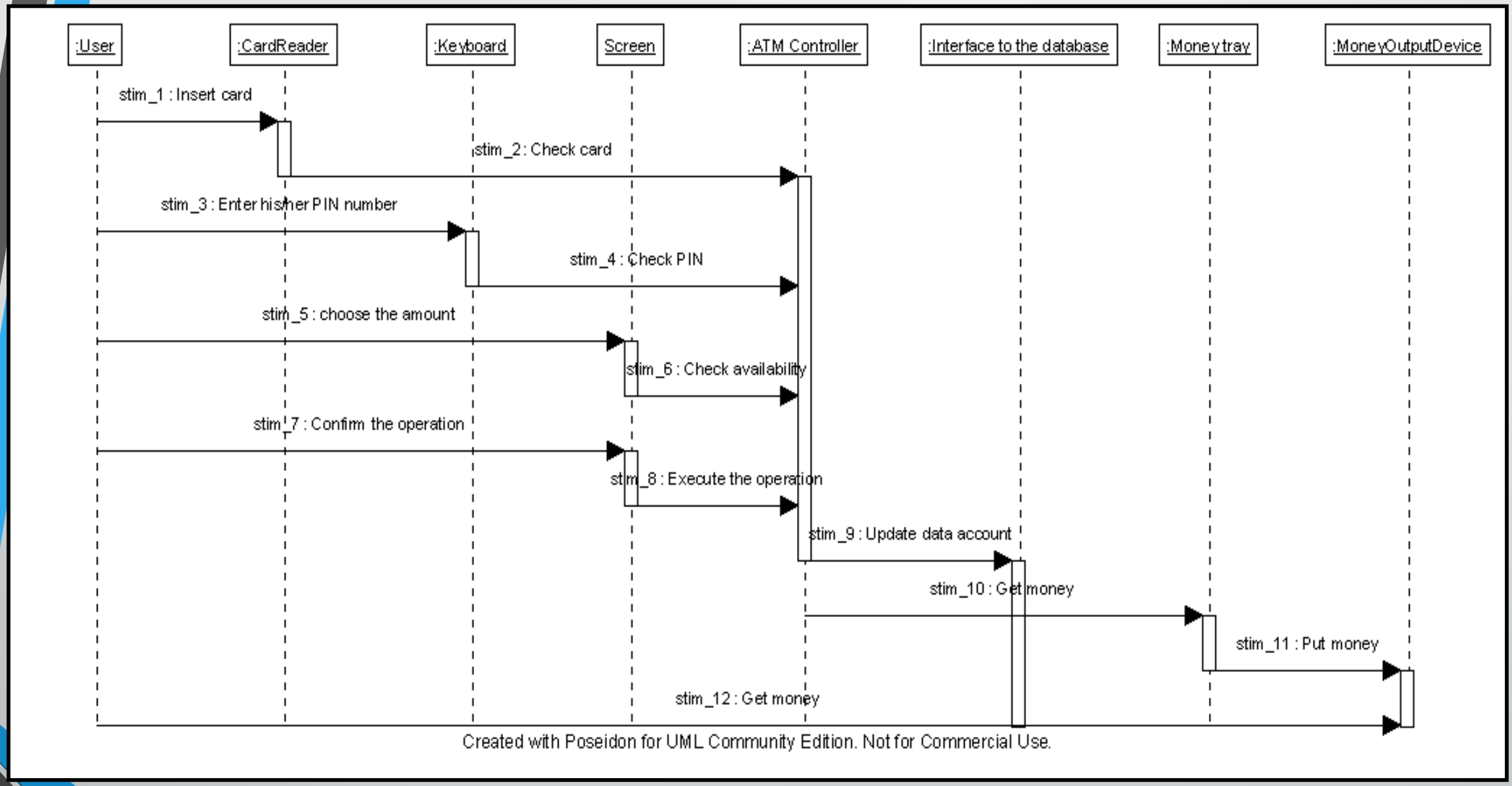


Fig 11. Solution to example 2

Summary

- Interaction diagrams describe how use cases are realized as interactions among societies of objects.
- Sequence diagrams are a temporal representation of objects and their interactions. This means that they represent objects and their interactions over a time element.
- Collaboration and communication diagrams are spatial representation of objects, links and interrelations. This means that they are not concerned with time; rather they are concerned with how objects interact in the greater dimension of space.

References

- Ambler, S.W.(2005). *The Elements of UML 2.0 Style* (Cambridge, England: Cambridge University Press, 2005).
- Dennis, A., Wixom, B. H., Tegarden, D. P., & Seeman, E. (2015). *System analysis & design: An object-oriented approach with Uml*. Wiley.
- Gutterman, Z., & Carmi, A. (n.d.). *Sequence Diagrams*. *class notes*.
- Ojo, A., & Estevez, E. (2005). (rep.). *Object-Oriented Analysis and Design with UML - Training Course* (Vol. 1).
- Stevens, P., & Pooley, R. (2006). *Using Uml: Software engineering with objects and components* (2nd ed.). Pearson Educaion Limited.