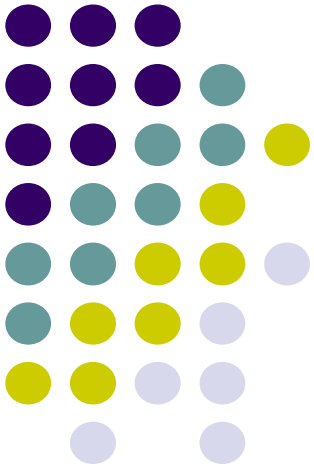
Interactions Diagrams

Object Interactions

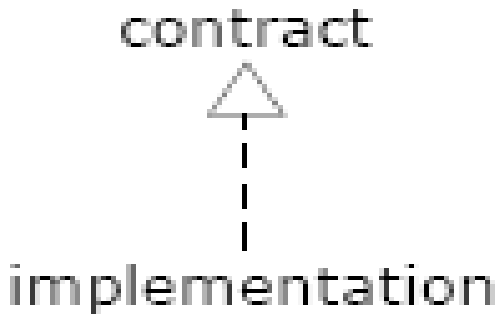
Sequence Diagrams

Communication (Collaboration) Diagrams

Examples

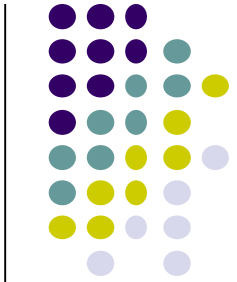


Retrospection - realization



- a semantic relationship between elements, wherein one element specifies a contract and another guarantees to carry out this contract
- relevant in two basic scenarios:
  - interfaces versus realizing classes/components
  - uses cases versus diagrams realizing collaborations
- graphically depicted as a dashed arrow with hollow head -> a cross between dependency and generalization

Use Case Realization

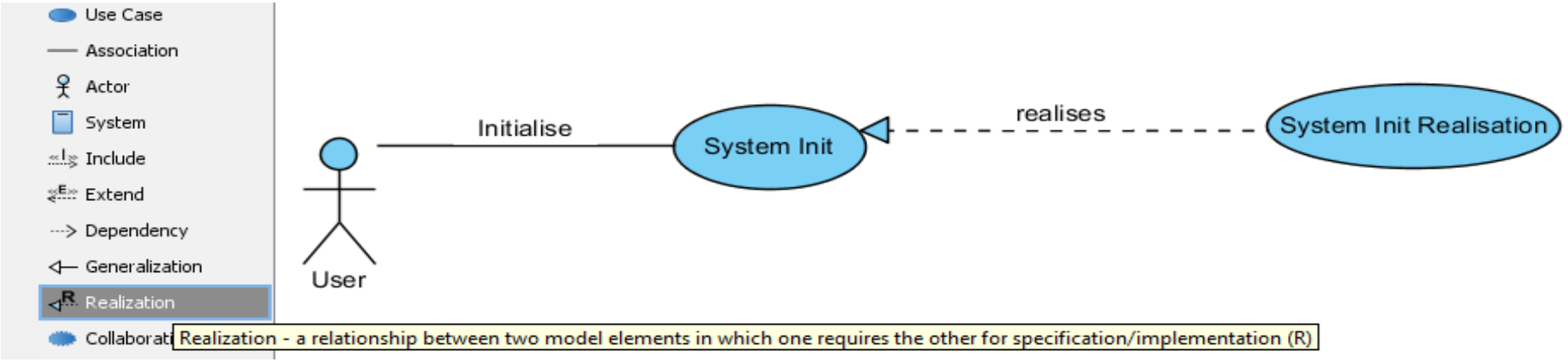


Use-Case Realization	A **use-case realization** describes how a particular use case is realized within the design model, in terms of collaborating objects.

A Use-Case Realization represents the Design perspective of a Use Case. The reason for separating the Use-Case Realization from its Use Case is that doing so allows the Use Cases to be managed separately from their realizations.

For each Use Case – One Realization

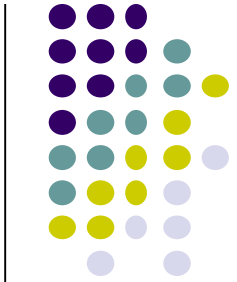
For each use case in the use-case model, there is a use-case realization in the design model with *Relization* relation to the use case.



Use case realization is an organization model element used to group a number of artifacts related to the design of a use case:

- ***Class diagrams*** of participating classes and subsystems, and
- ***Interaction diagrams*** which illustrate the flow of events of a use case, performed by a set of class and subsystem instances.

Interaction Diagrams

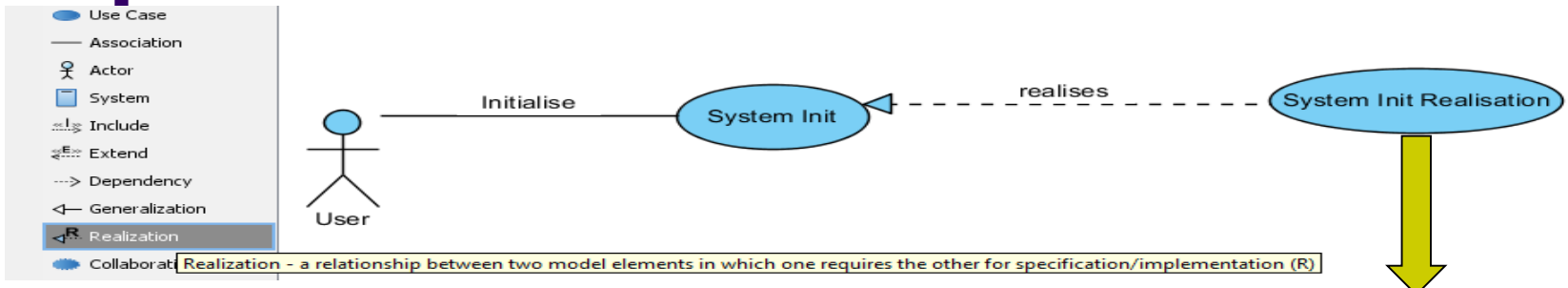


For each use-case realization there is one or more interaction diagrams depicting its participating objects and their interactions. There are two types of interaction diagrams:

1. ***Sequence diagrams*** - show the explicit sequence of messages and are better for real-time specifications and for complex scenarios;

2. ***Communications (prior UML 2.0: collaboration) diagrams*** - show the communication links between objects and are better for understanding all of the effects on a given object and for algorithm design.

Kept in a Picture



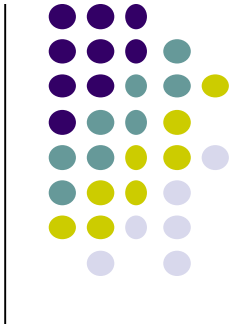
Use Case Realization

Class Diagrams

Interaction Diagrams

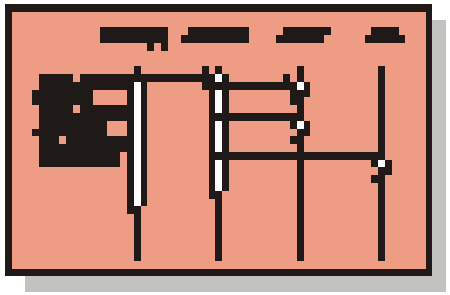
- Sequence Diagrams
- Communi-cation Diagrams

Models and views in UML 1.5

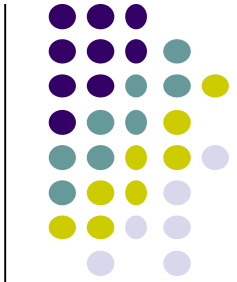


	Use case view	Logical view	Implementation view	Process view	Deployment view
Use case diagram	YES				
Class diagram		YES			
Sequence diagram	YES	YES			
Collaboration diagram	YES	YES		YES	
Statechart diagram	YES	YES		YES	
Component diagram			YES	YES	YES
Deployment diagram					YES

Sequence Diagram – Definition

Sequence Diagram	
	A **sequence diagram** describes a pattern of interaction among objects, arranged in a chronological order.

Properties of Sequence Diagrams

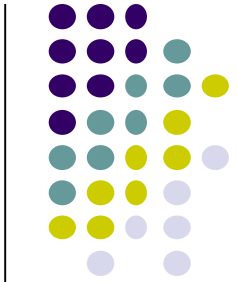


- Sequence diagrams show the objects participating in the interaction by their "lifelines" and the messages that they send to each other, i.e. how objects interact to perform the behavior of a use case.

- Sequence diagrams are particularly important to designers because they clarify the roles of objects in a flow and thus provide basic input for determining class responsibilities and interfaces.

- Unlike a communication (before UML 2.0 – collaboration) diagram, a sequence diagram includes chronological sequences (explicit sequence of messages), but does not include object relationships.

Contents of Sequence Diagrams



You can have:

objects (i.e. class instances)

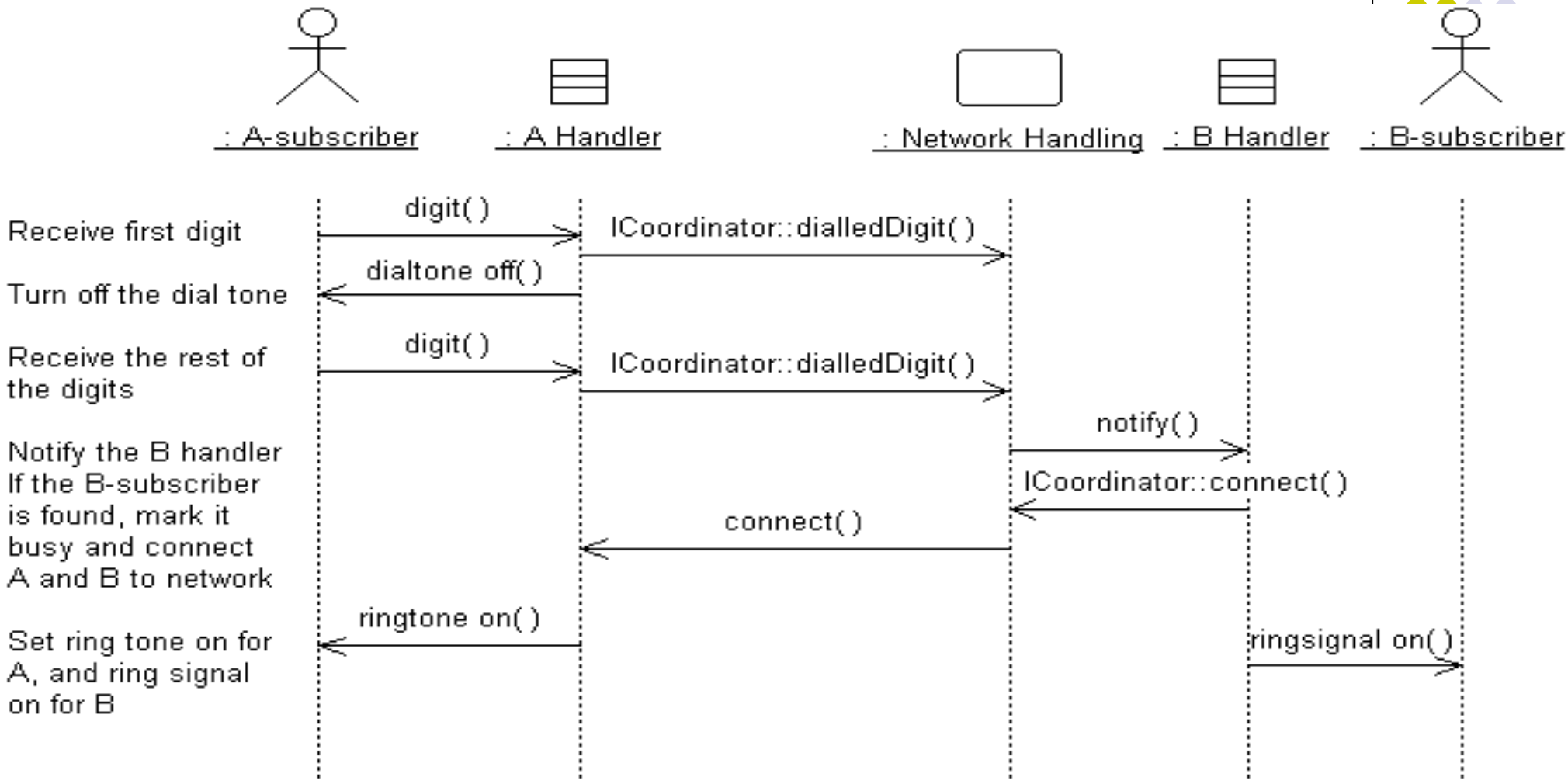
and

actor instances

in sequence diagrams, together with *messages* describing how they interact.

The diagram describes what takes place in the participating objects, in terms of activations, and how the objects communicate by sending messages.

Example of Sequence Diagram



A sequence diagram describing part of the flow of events of the use case **Place Local Call** in a simple **Phone Switch**.

Contents of Sequence Diagrams - 2

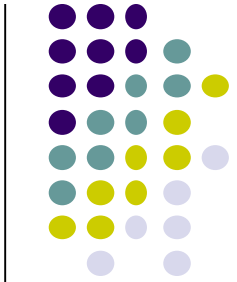
Objects - shown as a vertical dashed line called the "lifeline". The lifeline represents the existence of the object at a particular time. An object symbol shows the name of the object and its class underlined:

objectname [selector] : classname ref decomposition

selector – typically is an expression
decomposition – refers to another seq. diagram of decomposable sub-system

Actors - try keeping them either at the left-most, or the right-most lifelines

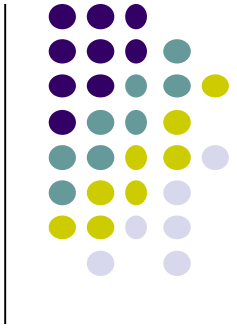
Contents of Sequence Diagrams - 3



Messages - communications between objects that conveys information with the expectation that activity will ensue; shown as a horizontal solid arrow from the lifeline of one object to the lifeline of another object. The arrow is labeled with the name of the message, and its parameters, or with a sequence number.

Scripts - describe the flow of events textually in a sequence diagram.

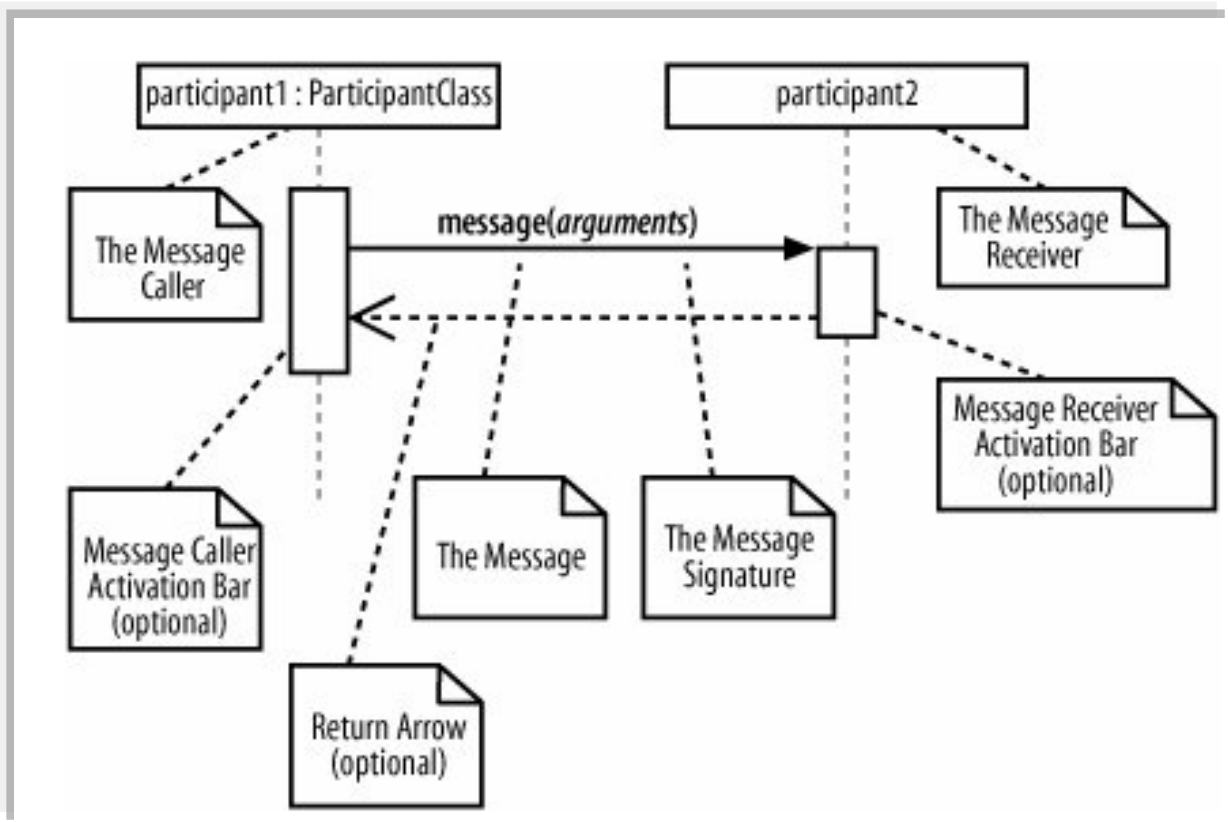
Object/Class Naming Convention



objectname [selector] : classname ref decomposition

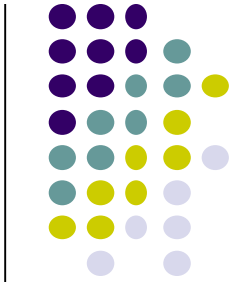
Example participant name	Description
admin	A part is named admin, but at this point in time the part has not been assigned a class.
: ContentManagementSystem	The class of the participant is ContentManagementSystem, but the part currently does not have its own name.
admin : Administrator	There is a part that has a name of admin and is of the class Administrator.
eventHandlers [2] : EventHandler	There is a part that is accessed within an array at element 2, and it is of the class EventHandler.
: ContentManagementSystem *ref cmsInteraction*	The participant is of the class ContentManagementSystem, and there is *another interaction diagram called cmsInteraction that shows how the participant* ContentManagementSystem *works internally*

Messages between Participants



Interactions on a sequence diagram are shown as messages between participants (from **Learning UML 2.0,** by K. Hamilton, R. Miles)

The Message Signature

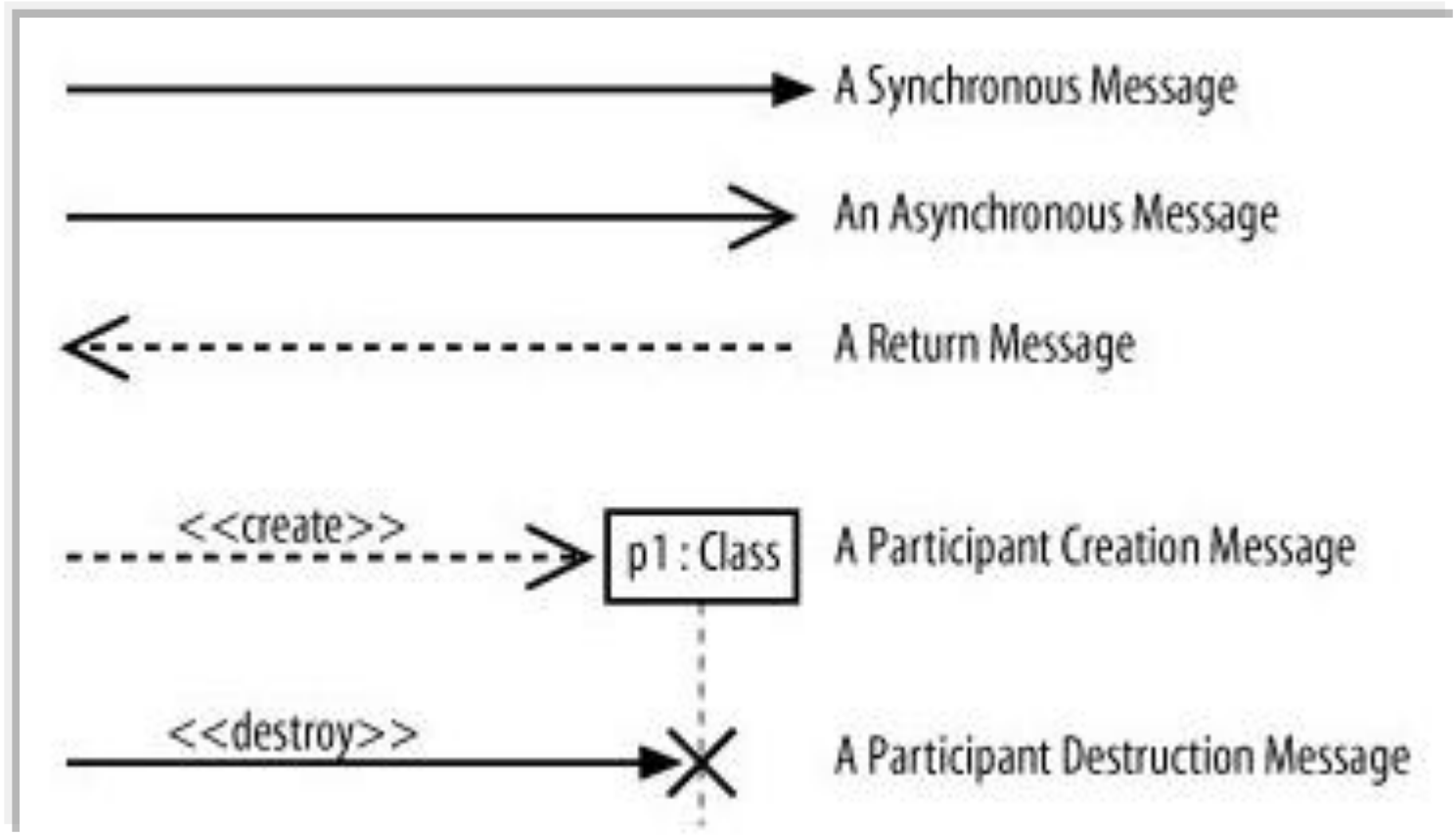


- Signature:
  attribute = signal_or_message_name (arguments) : return_type
- Arguments:
  name:type, …
- Only ***signal_or_message_name*** is not optional
- Example:

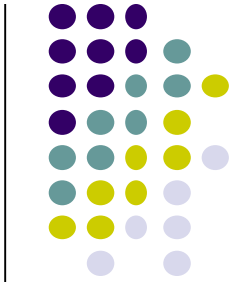
myVar = sendSignal() : ReturnClass

The message's name is **sendSignal**; no arguments; returns an object of class **ReturnClass** that is assigned to the **myVar** attribute of the *message caller*.

Five main types of messages



Message Synchronization

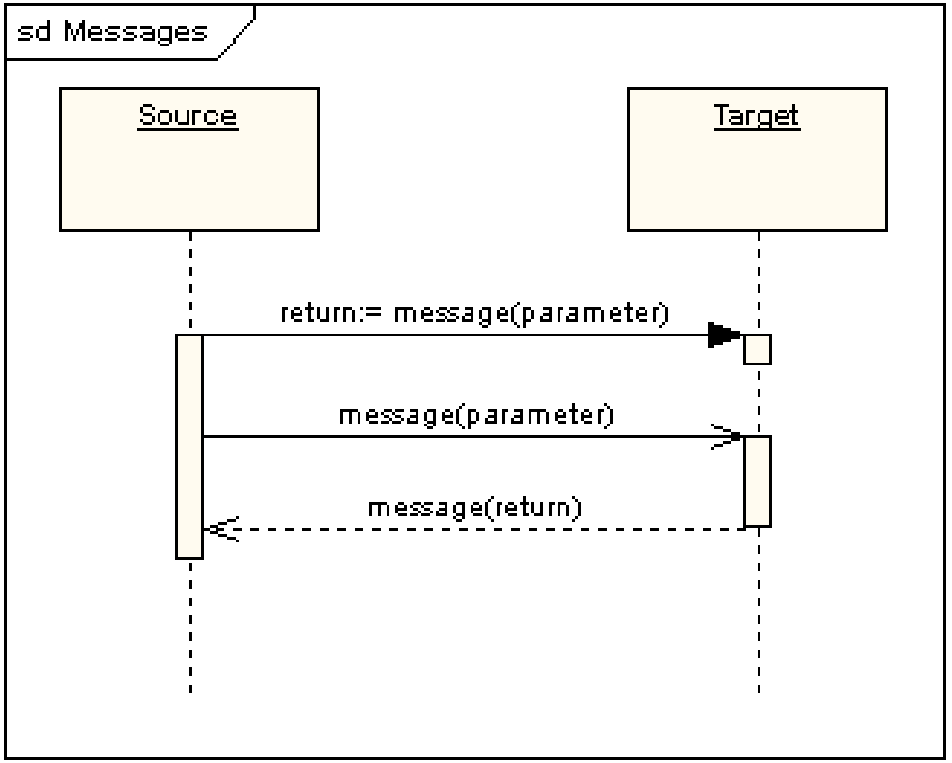


- **Synchronous message call -> method invocation**
- **Asynchronous message call -> method call in another thread:**

```
public void operation1( ) {
    // Receive the message and trigger off the thread
    Thread myWorker = new Thread(this);
    myWorker.start();
    // This call starts a new thread, calling the run() method of the thread
    // As soon as the thread has been started, the call returns.
 }
```

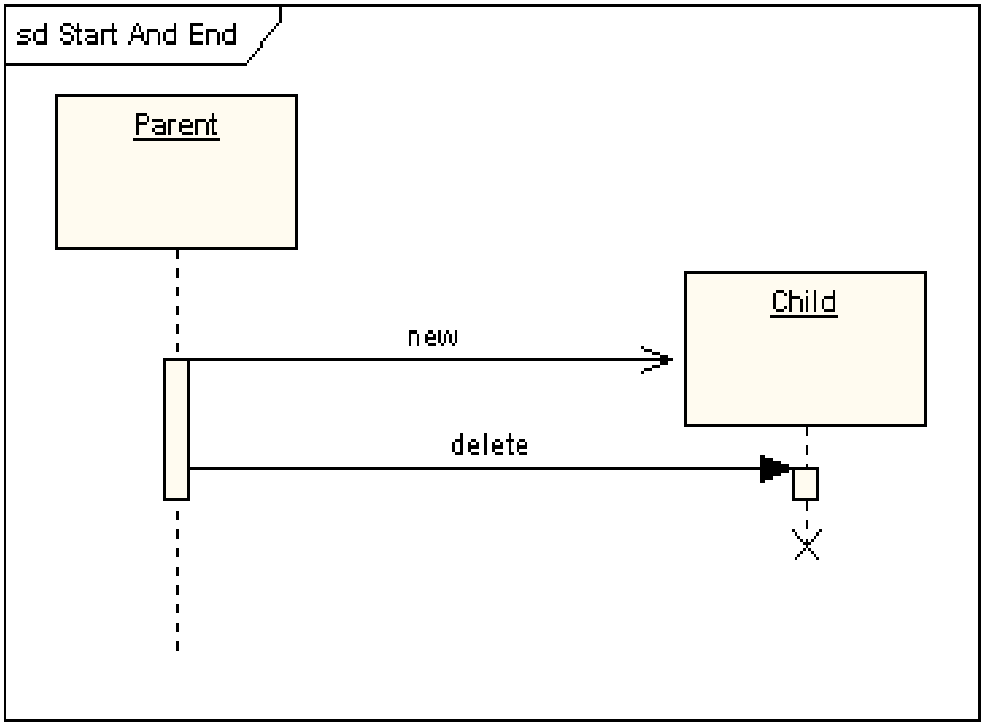
Example

- The first message is a **synchronous** message (denoted by the solid arrowhead) complete with an *implicit return message*;
- The second message is **asynchronous** (denoted by line arrowhead), and the third is the *asynchronous return message* (denoted by the dashed line).

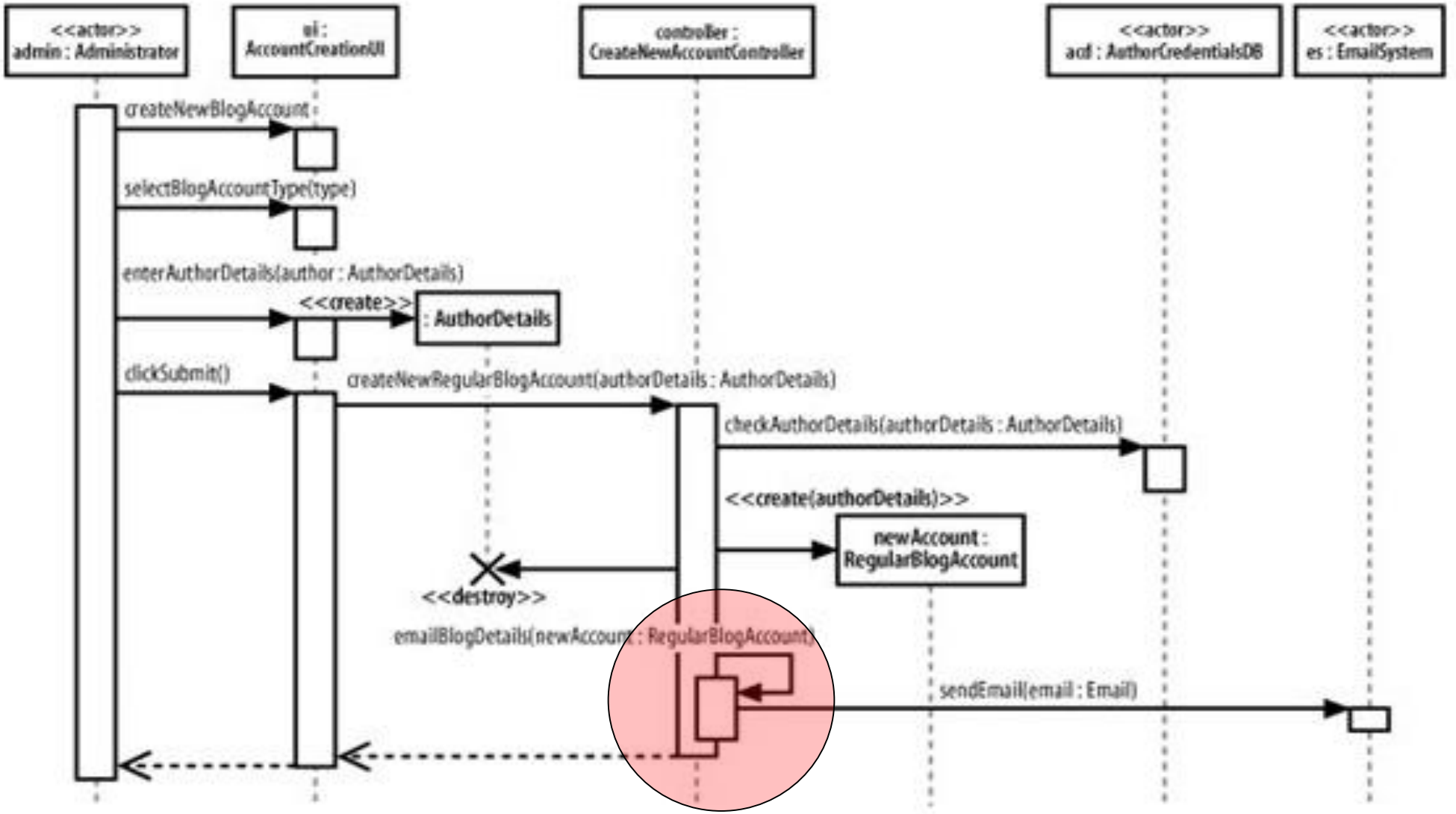


Lifeline Start and End

- A lifeline may be created or destroyed during the timescale represented by a sequence diagram.
- In this case, the lifeline is terminated by a stop symbol, represented as a cross.
- The symbol at the head of the lifeline is shown at a lower level down the page than the symbol of the object that caused the creation.

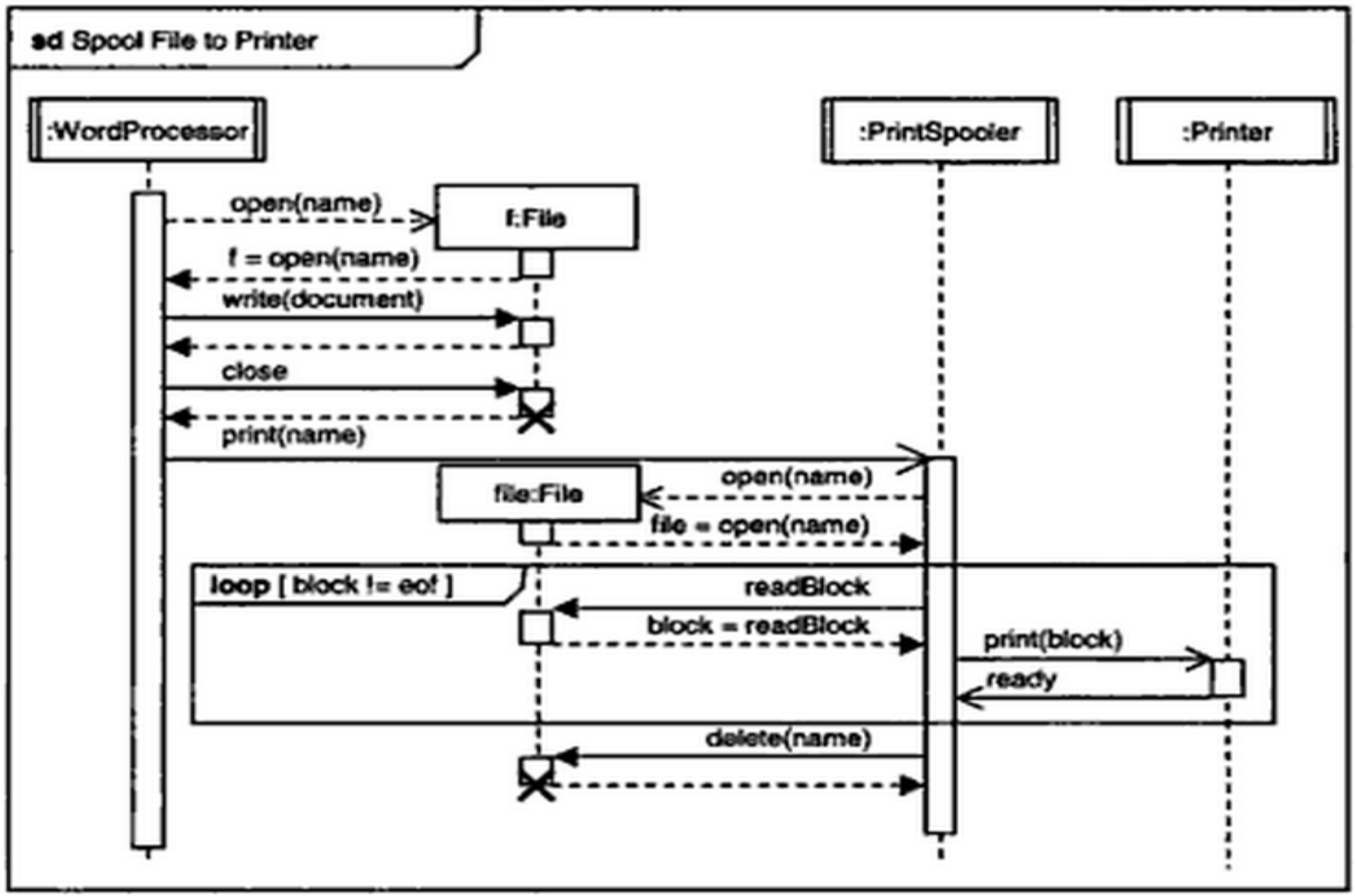


Create and Destroy Messages
(from Learning UML 2.0, by K. Hamilton, R. Miles)

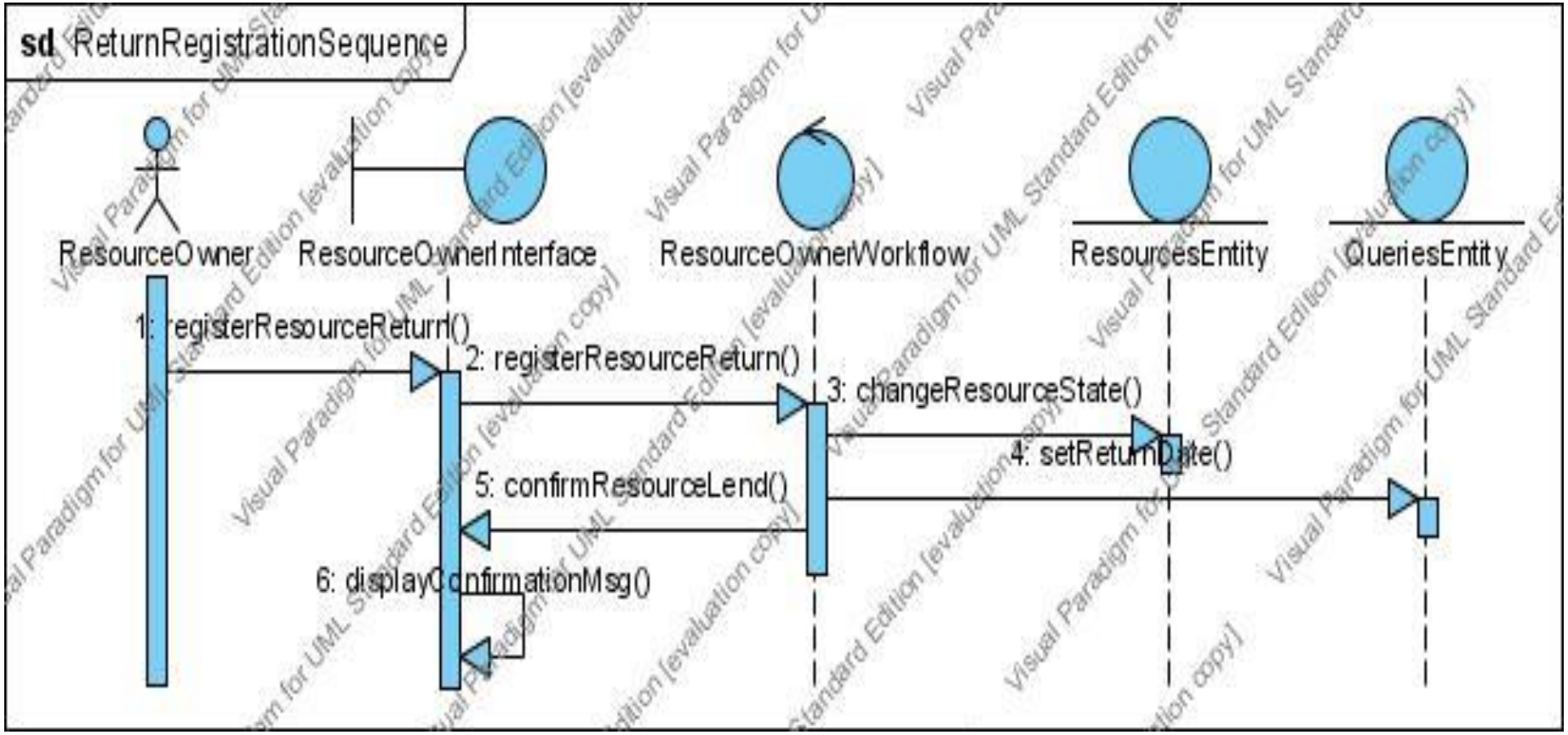


SD with Asynchronous Messages

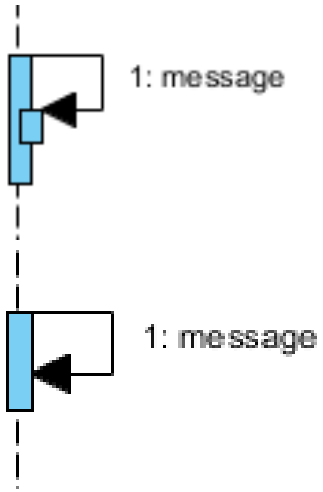
Source: UML, Second Edition, by S. Bennet, J. Skelton, K. Lunn



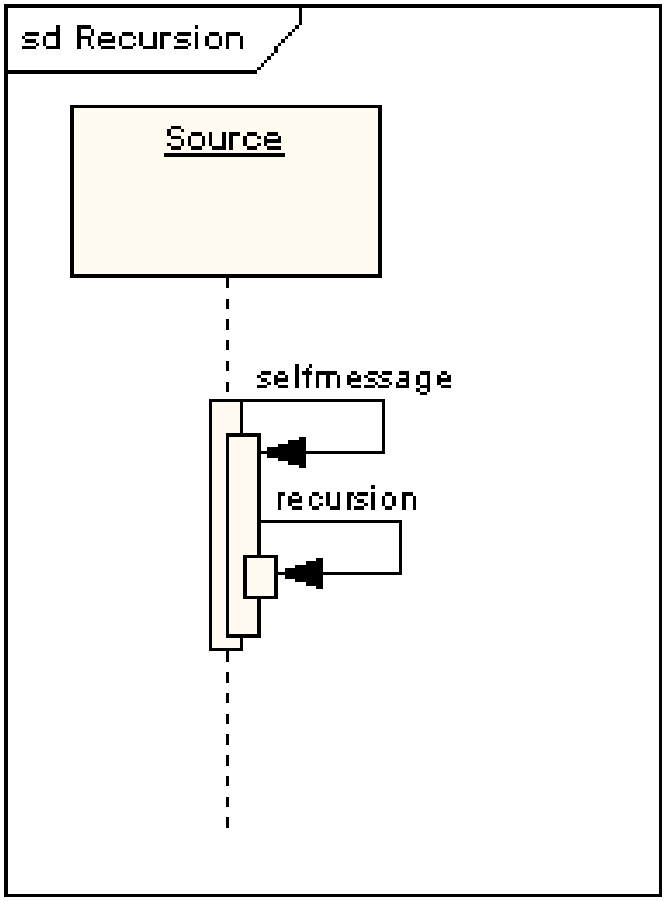
Nested and Self Messages



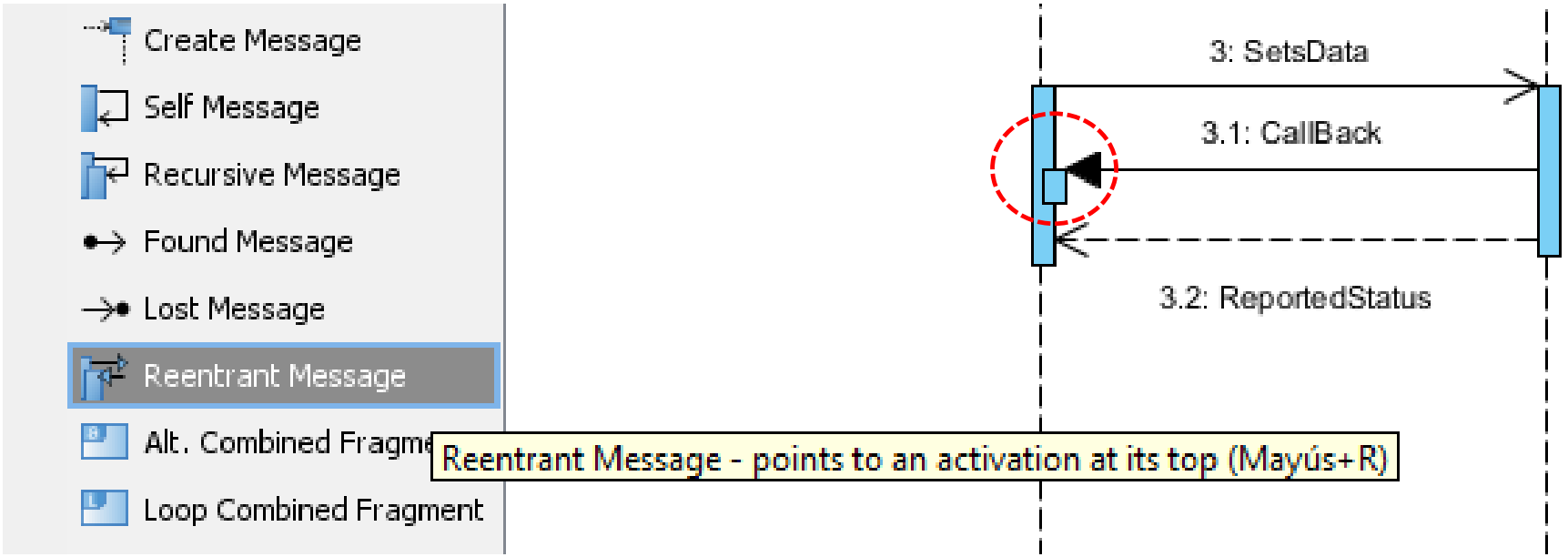
Self and Reculsive Messages



- Such messages can represent:
  - Either a recursive call of an operation,
  - Or one method calling another method belonging to the same object.
- It is shown as creating a nested focus of control in the lifeline's execution occurrence.

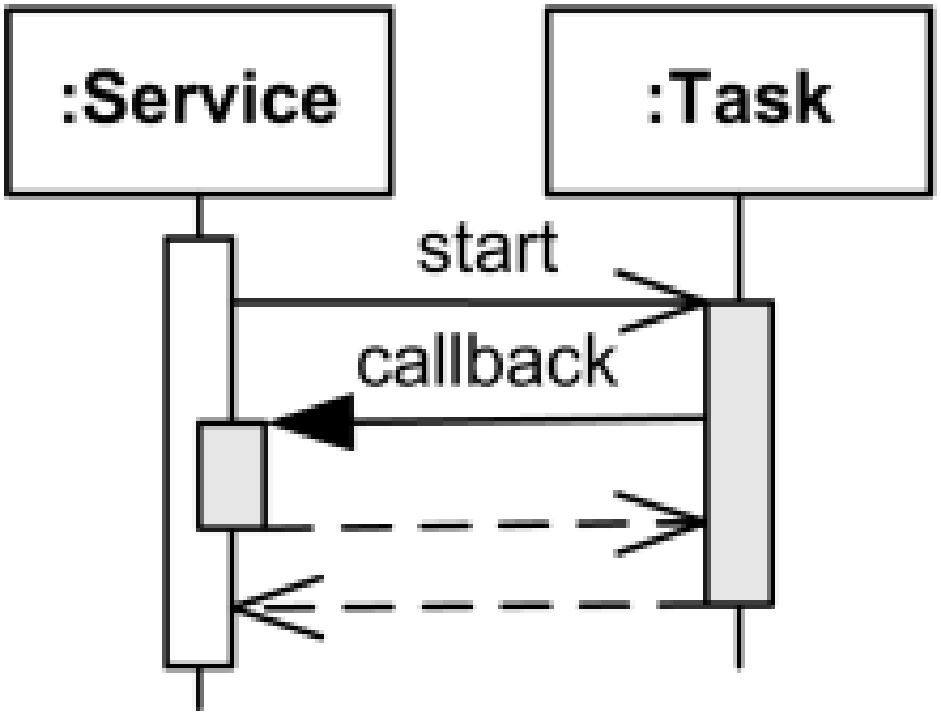


Reentrant (Callback) message

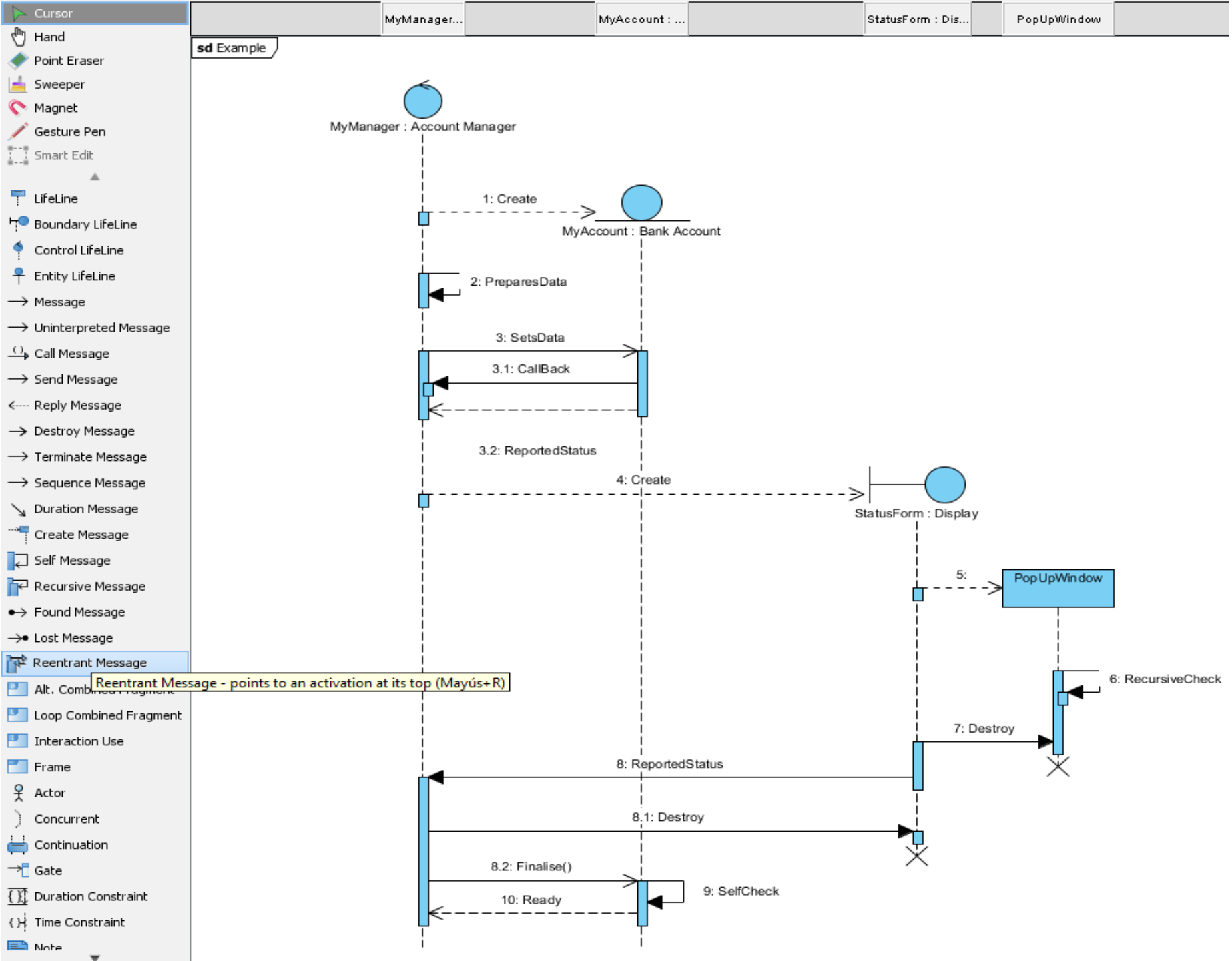


Callback message 2

- Overlapping execution specifications on the same lifeline - *callback message*

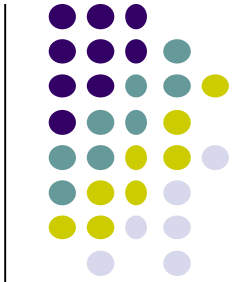


Source: http://www.uml-diagrams.org/sequence-diagrams-reference.html

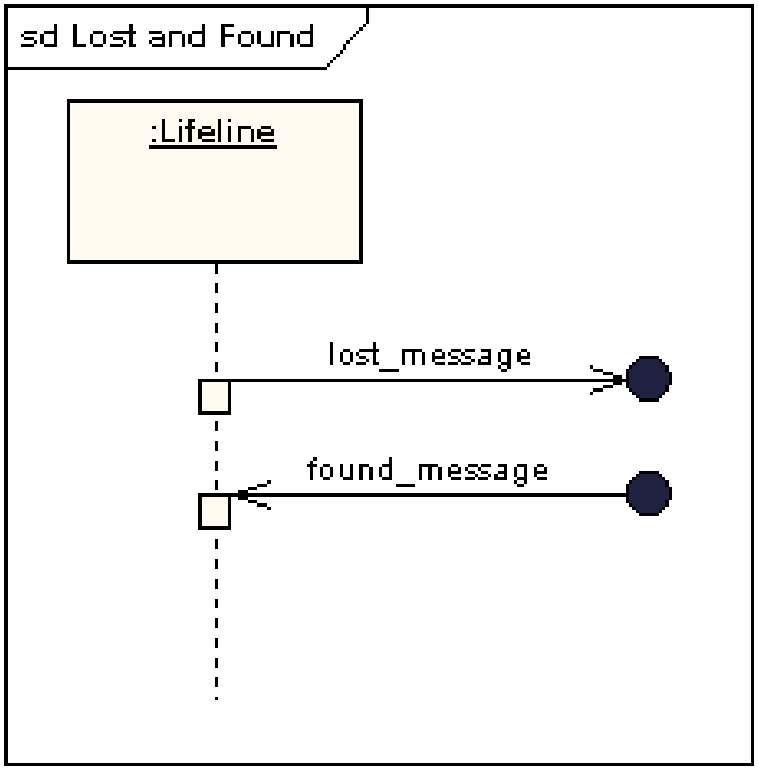

Cursor
Hand
Point Eraser
Sweeper
Magnet
Gesture Pen
Smart Edit
LifeLine
Boundary LifeLine
Control LifeLine
Entity LifeLine
Message
Uninterpreted Message
Call Message
Send Message
Reply Message
Destroy Message
Terminate Message
Sequence Message
Duration Message
Create Message
Self Message
Recursive Message
Found Message
Lost Message
Reentrant Message
Reentrant Message - points to an activation at its top (Mayús+R)
Loop Combined Fragment
Interaction Use
Frame
Actor
Concurrent
Continuation
Gate
Duration Constraint
Time Constraint
MyManager...
MyAccount : ...
StatusForm : Dis...
PopUpWindow
sd Example
MyManager : Account Manager
1: Create
MyAccount : Bank Account
2: PreparesData
3: SetsData
3.1: CallBack
3.2: ReportedStatus
4: Create
StatusForm : Display
5:
PopUpWindow
6: RecursiveCheck
7: Destroy
8: ReportedStatus
8.1: Destroy
8.2: Finalise()
9: SelfCheck
10: Ready

SD in VP

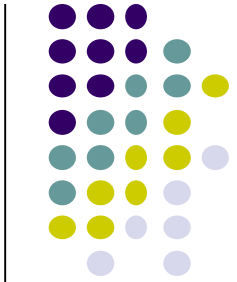
Lost and Found Messages (UML 2.0)



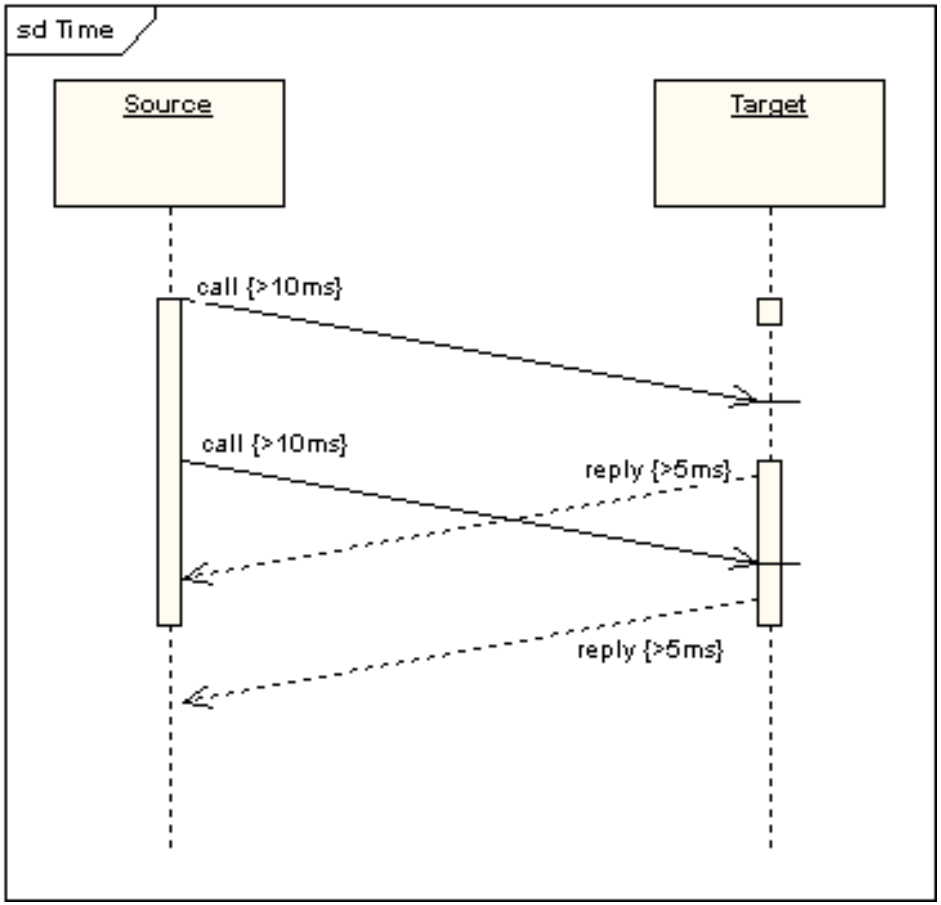
- **Lost messages** are those that are *either sent but do not arrive* at the intended recipient, or which *go to a recipient not shown* on the current diagram.
- **Found messages** are those that *arrive from an unknown sender*, or *from a sender not shown* on the current diagram. They are denoted going to or coming from an endpoint element.



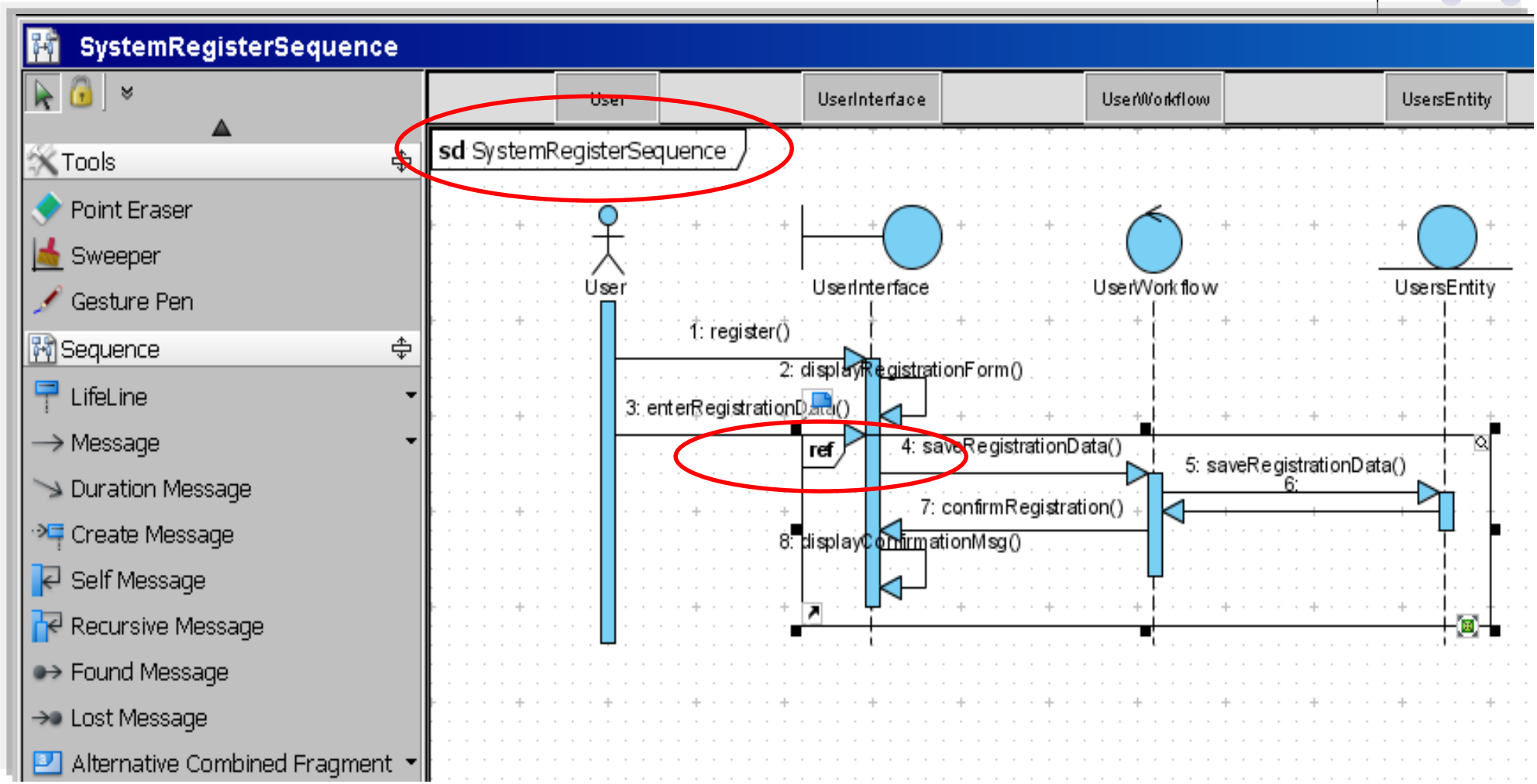
Duration and Time Constraints (UML 2.0)

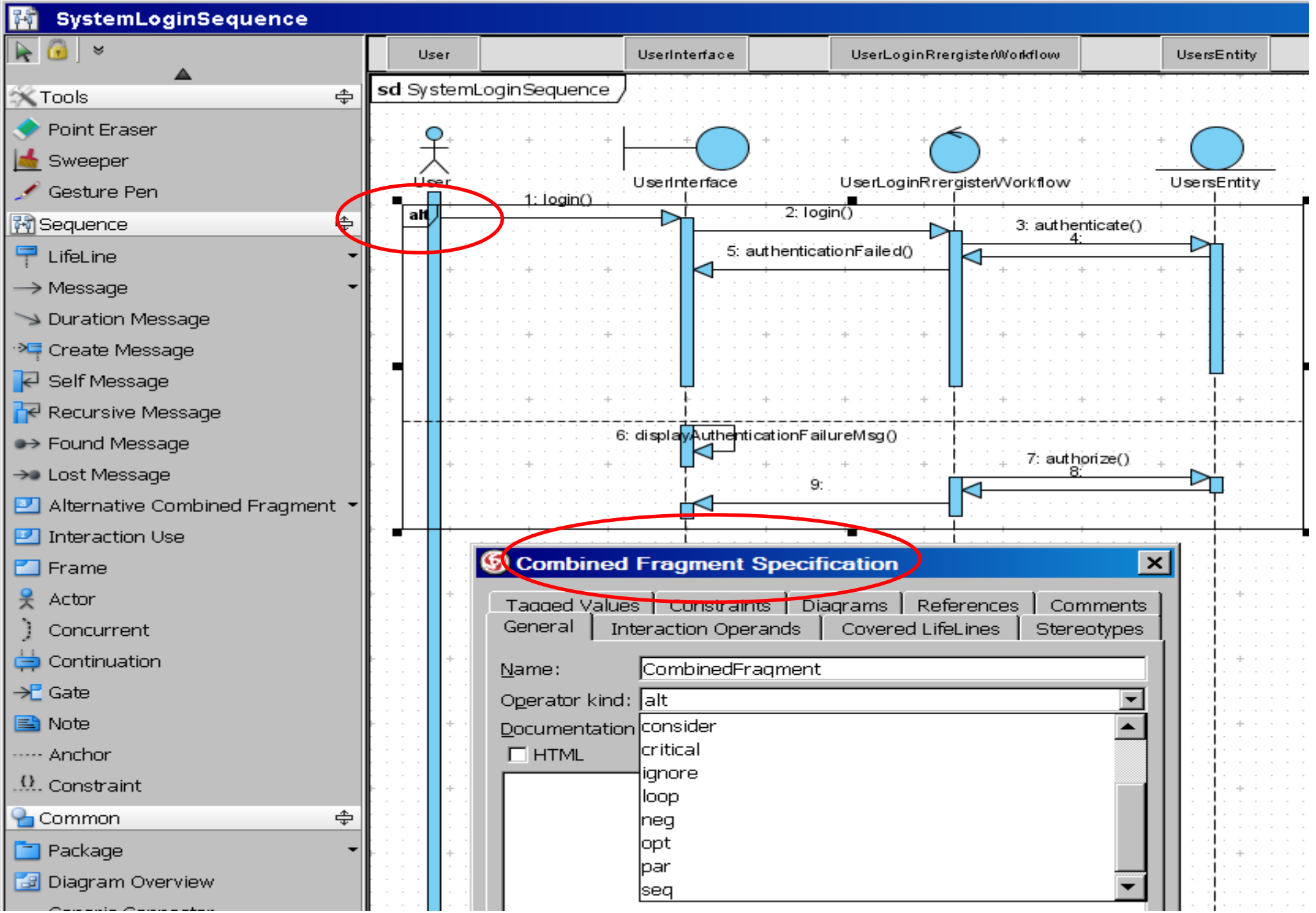


- By default, a message is shown as a horizontal line. Since the lifeline represents the passage of time down the screen, when modelling a real-time system, or even a time-bound business process, it can be important to consider the length of time it takes to perform actions.
- By setting a duration constraint for a message, the message will be shown as a sloping line.

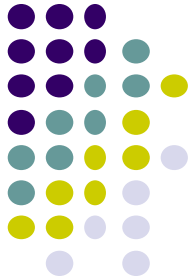


UML 2.0 Fragments




SystemLoginSequence
Tools
Point Eraser
Sweeper
Gesture Pen
Sequence
LifeLine
Message
Duration Message
Create Message
Self Message
Recursive Message
Found Message
Lost Message
Alternative Combined Fragment
Interaction Use
Frame
Actor
Concurrent
Continuation
Gate
Note
Anchor
Constraint
Common
Package
Diagram Overview
User
UserInterface
UserLoginRrergisterWorkflow
UsersEntity
sd SystemLoginSequence
alt
1: login()
2: login()
3: authenticate()
4:
5: authenticationFailed()
6: displayAuthenticationFailureMsg()
7: authorize()
8:
9:
Combined Fragment Specification
Tagged Values
Constraints
Diagrams
References
Comments
General
Interaction Operands
Covered LifeLines
Stereotypes
Name:
CombinedFragment
Operator kind:
alt
Documentation
HTML
consider
critical
ignore
loop
neg
opt
par
seq

About fragments



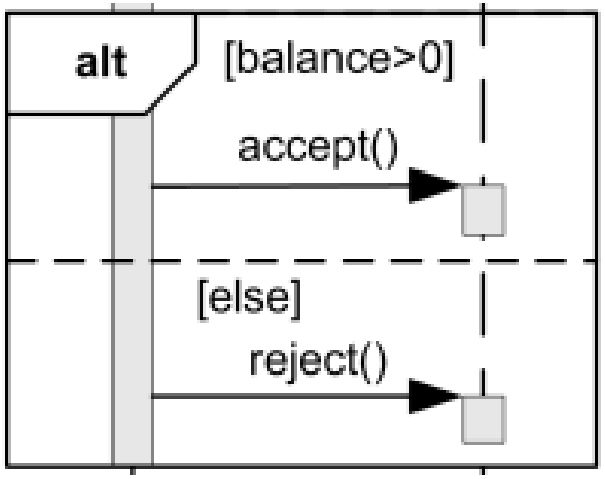
- Fragments allow for *adding a degree of procedural logic* to diagrams and which come under the heading of combined fragments.
- A combined fragment is *one or more processing sequence enclosed in a frame and executed under specific named circumstances*. The fragments available are:

  - alt - alternatives
  - opt - option
  - loop - iteration
  - break - break
  - par - parallel
  - strict - strict sequencing
  - seq - weak sequencing
  - critical - critical region
  - ignore - ignore
  - consider - consider
  - assert - assertion
  - neg – negative

Alt

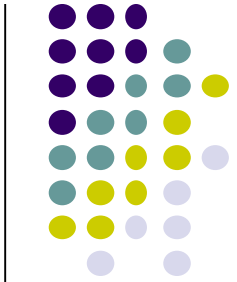
- Alternative fragment (denoted "**alt**") models if…then…else constructs, i.e. a choice or alternatives of behavior
- At most one of the operands will be chosen.
- The chosen operand must have an explicit or implicit guard expression that evaluates to true at this point in the interaction.

Call accept()
if balance > 0,
call reject() otherwise.



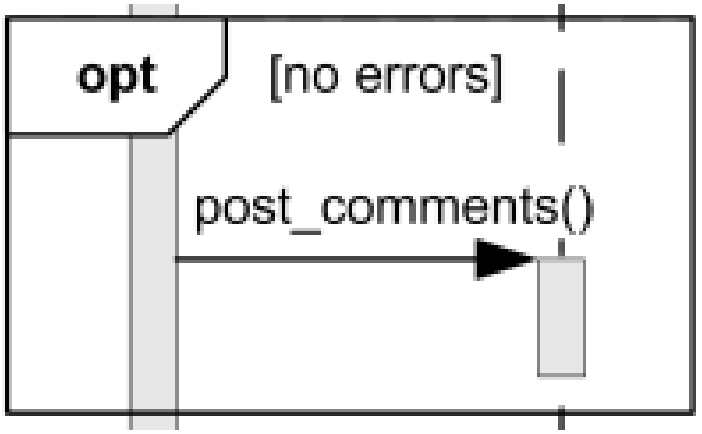
Source: http://www.uml-diagrams.org/sequence-diagrams-reference.html

Opt



- Option fragment (denoted “**opt**”) models switch constructs – a choice to execute or not the fragment depends on a condition.
- Here, the combined fragment represents a choice of behavior where either the (sole) operand happens or nothing happens.

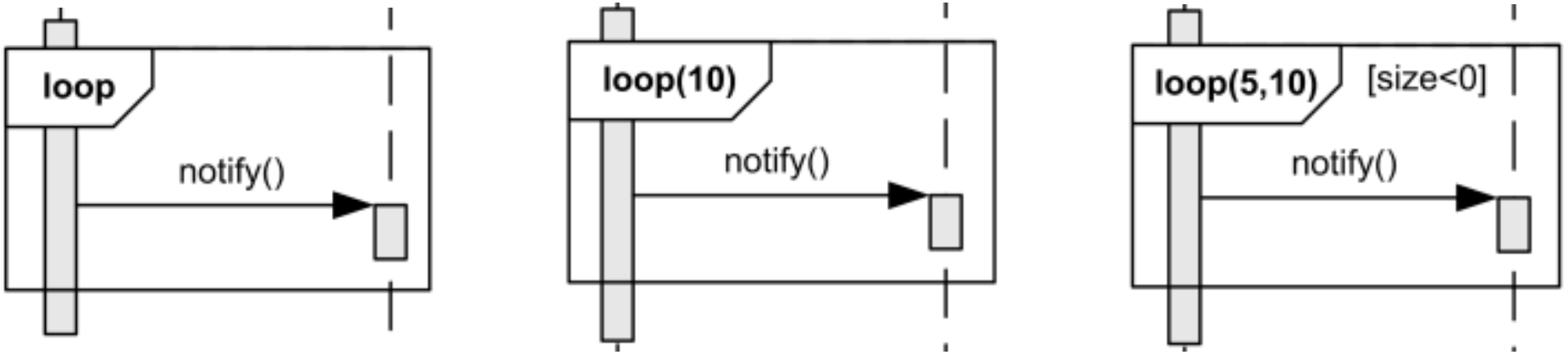
Post comments
if there were no errors.



Source: http://www.uml-diagrams.org/sequence-diagrams-reference.html

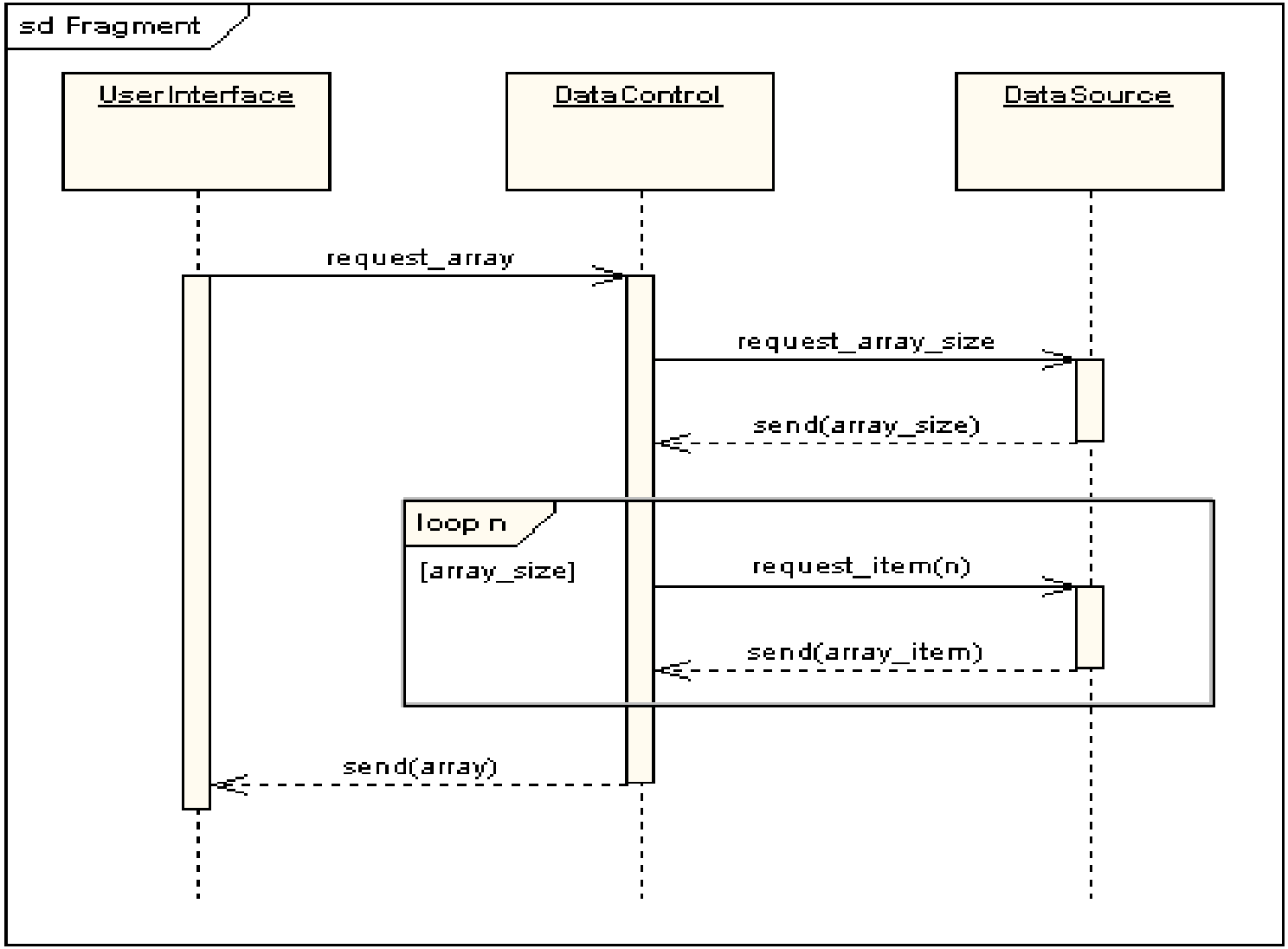
Loop

- Potentially infinite loop
- Loop with exact number of occurrences – if both bounds are specified, loop will iterate minimum the *min-int* number of times and at most the *max-int* number of times.
- Iteration bounds loop with possible interaction constraint (guard condition)

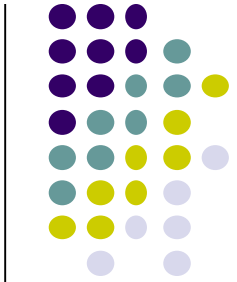


Source: http://www.uml-diagrams.org/sequence-diagrams-reference.html

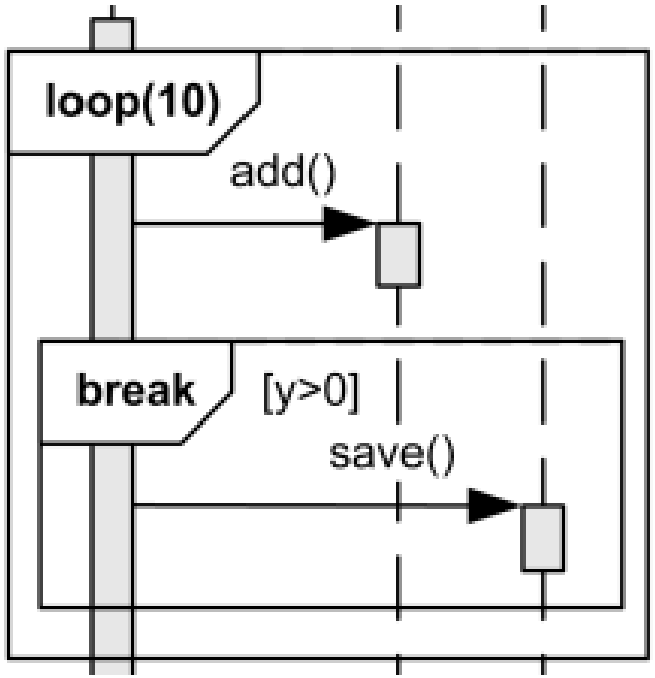
A Loop Fragment Enclosing a Series of Repeated Messages



Break



- **Break** fragment models an exceptional sequence of events that is processed instead of the whole of the rest of the diagram
- UML allows only breaking one level (!)

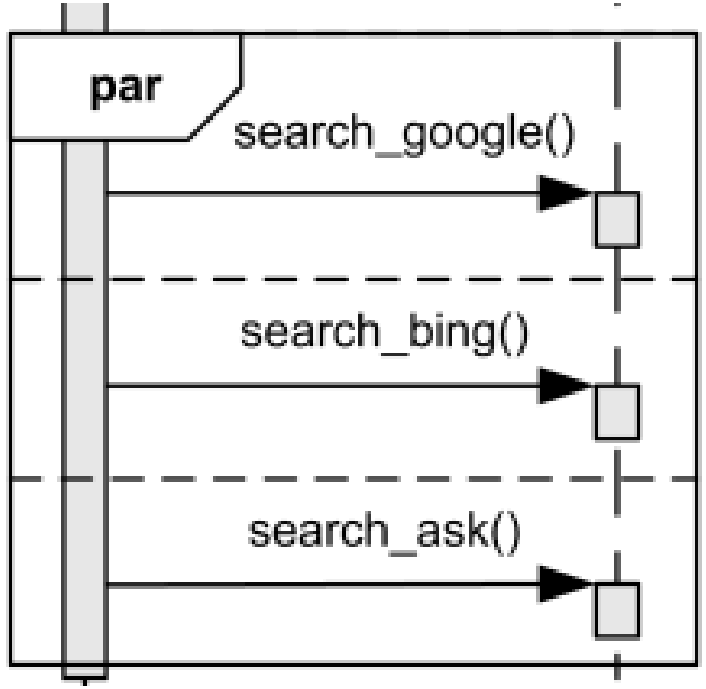


Source: http://www.uml-diagrams.org/sequence-diagrams-reference.html

Par

- **Par** defines *potentially/possible* parallel execution of behaviors of the operands of the combined fragment

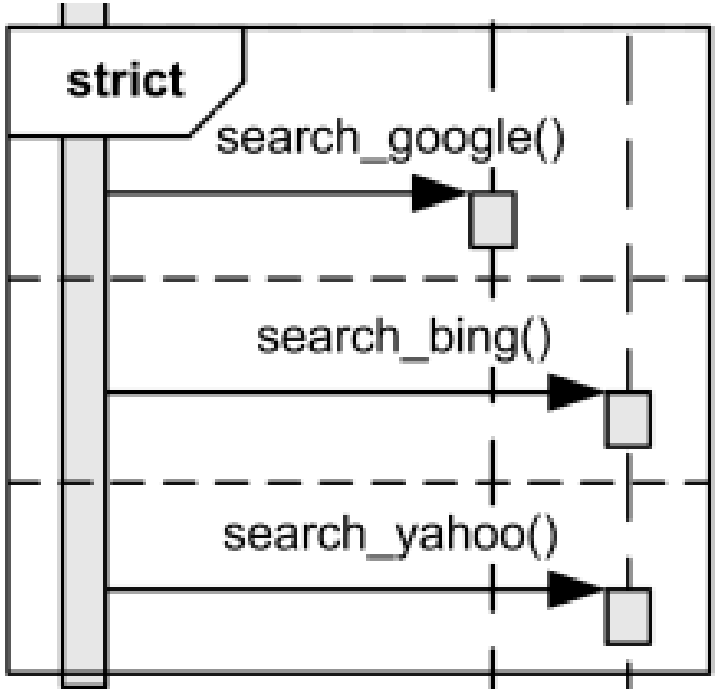
Search Google, Bing and Yahoo in possibly in parallel.



Source: http://www.uml-diagrams.org/sequence-diagrams-reference.html

Strict

- **strict** requires a strict sequencing (order) of the operands on the first level within the combined fragment.



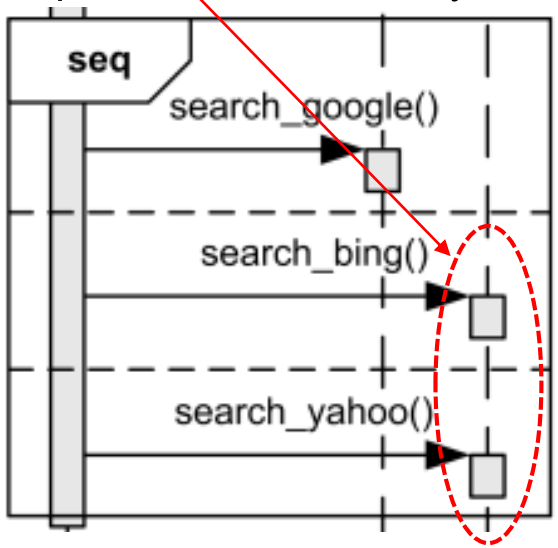
Search Google, Bing and Yahoo in the strict sequential order.

Source: http://www.uml-diagrams.org/sequence-diagrams-reference.html

Seq (weak sequencing)

- Occurrence specifications on different lifelines from different operands may come in any order.
- Occurrence specifications *on the same lifeline* from different operands are *ordered* such that an occurrence specification of the first operand comes before that of the second operand.
- Weak sequencing **seq** reduces to a **par** when the operands are on disjoint sets of participants.
- Weak sequencing reduces to **strict** sequencing when the operands work on only one participant.

Search Google possibly parallel with Bing and Yahoo,
but search Bing before Yahoo.

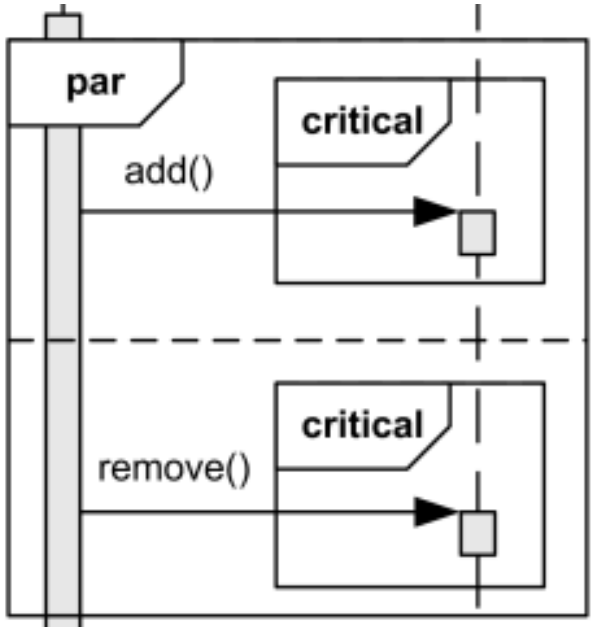


Source: http://www.uml-diarams.org/sequence-diagrams-reference.html

Critical

- **critical** defines that the combined fragment represents a critical region.
- A critical region is a region with traces that cannot be interleaved by other occurrence specifications (on the lifelines covered by the region).

Add() or remove() could be called in parallel, but each one should run as a critical region.

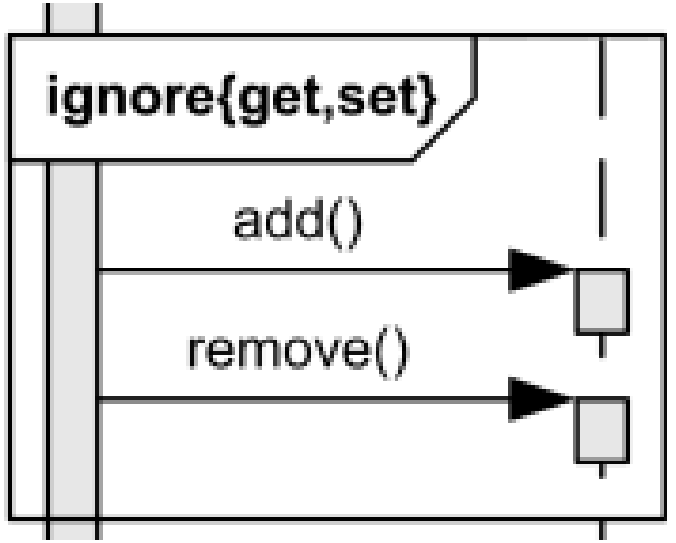


Source: http://www.uml-diagrams.org/sequence-diagrams-reference.html

Ignore

- **ignore** means that there are some insignificant messages that are not shown within this combined fragment.
- The list of ignored messages follows the operand enclosed in a pair of curly braces "{" and "}".

Ignore get() and set() messages, if any.

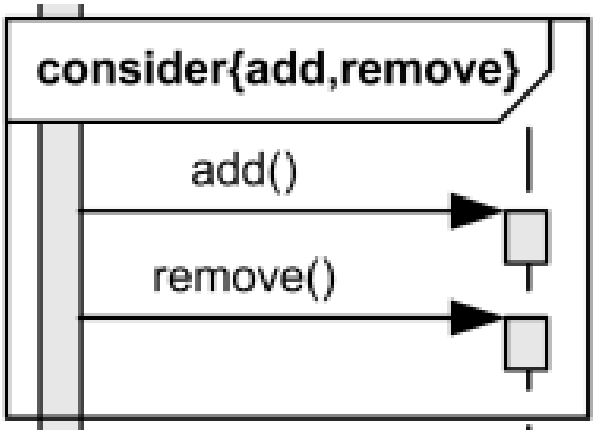


Source: http://www.uml-diagrams.org/sequence-diagrams-reference.html

Consider

- **consider** defines which messages should be considered within this combined fragment, meaning that any other message will be ignored.
- The list of considered messages follows the operand enclosed in a pair of curly braces "{" and "}".

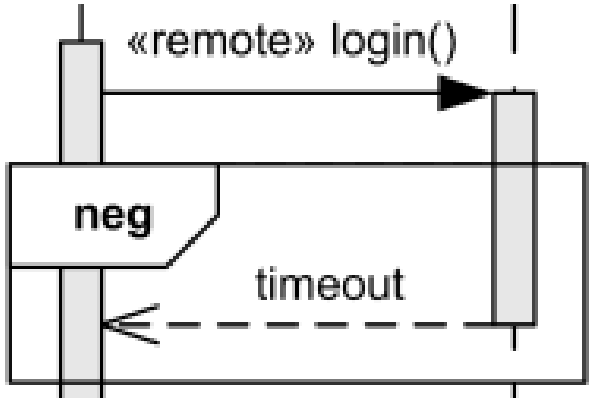
Consider only add() or remove() messages, ignore any other.



Source: http://www.uml-diagrams.org/sequence-diagrams-reference.html

Neg

- neg describes combined fragment of traces that are defined to be negative (invalid).
- Negative traces are the traces which occur when the system has failed.
- All interaction fragments that are different from the negative are considered positive, meaning that they describe traces that are valid and should be possible.

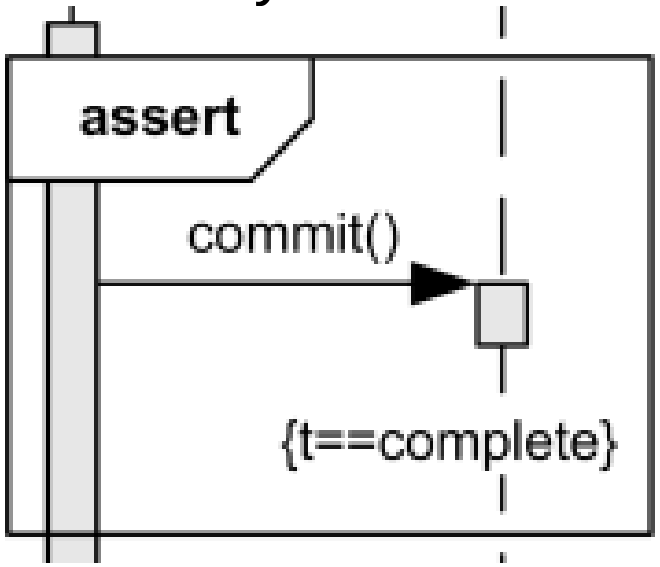


Source: http://www.uml-diagrams.org/sequence-diagrams-reference.html

Assert

- assert means that the combined fragment represents the assertion that the sequences of the assert operand are the only valid continuations
- must be satisfied by a correct design of the system

Commit() message should occur at this point, following with evaluation of state invariant.

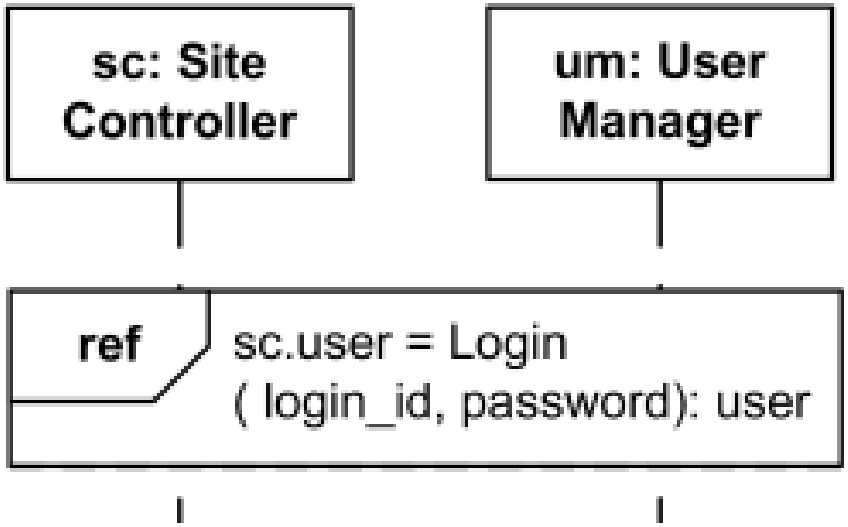


Source: http://www.uml-diagrams.org/sequence-diagrams-reference.html

Ref

- **Ref** defines interaction fragment which allows to use (or call) another interaction.

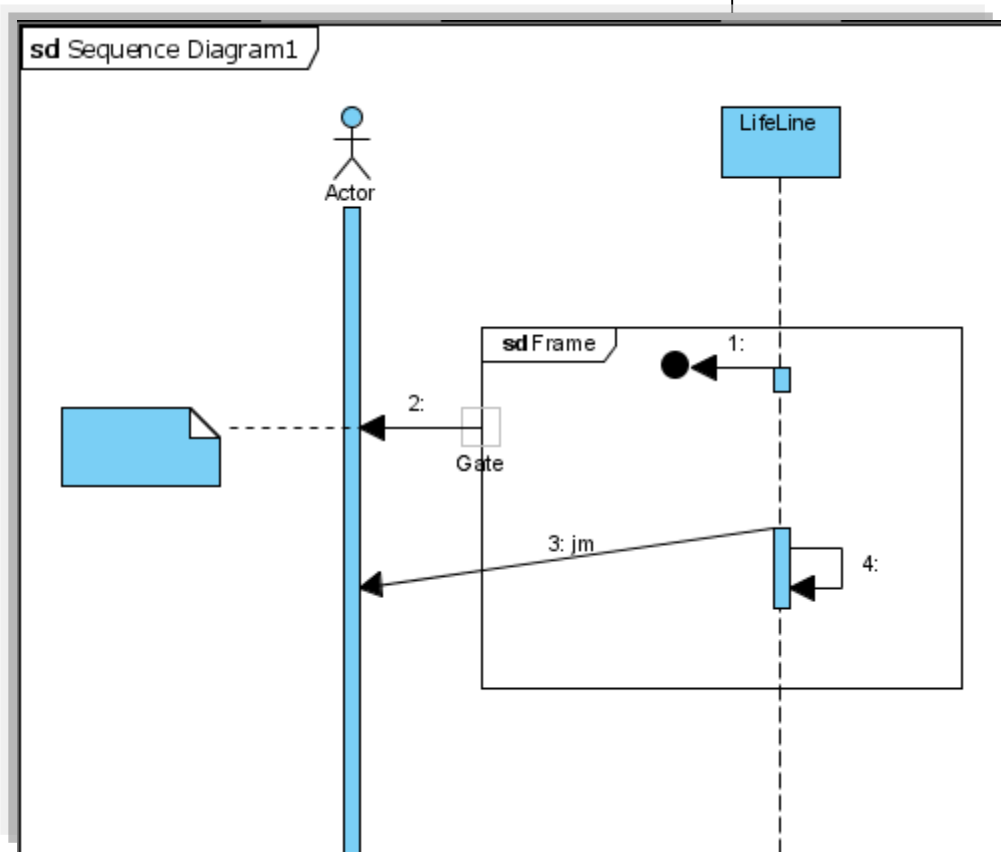
Use Login interaction to authenticate user and assign result back to the user attribute of Site Controller.



Source: http://www.uml-diagrams.org/sequence-diagrams-reference.html

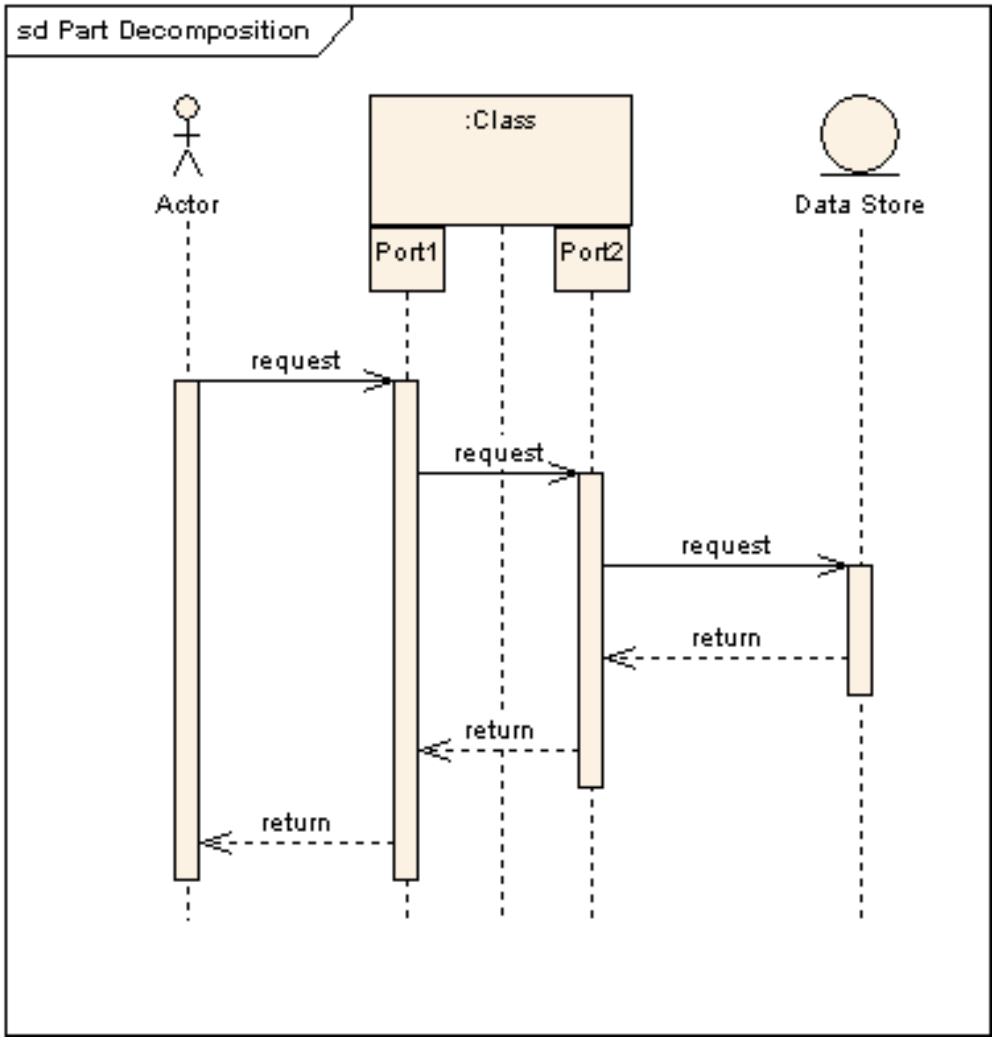
Gate (UML 2.0)

- A gate is a connection point for connecting a message inside a fragment with a message outside a fragment. SD show a gate as a small square on a fragment frame.
- Diagram gates act as off-page connectors for sequence diagrams, representing the source of incoming messages or the target of outgoing messages.



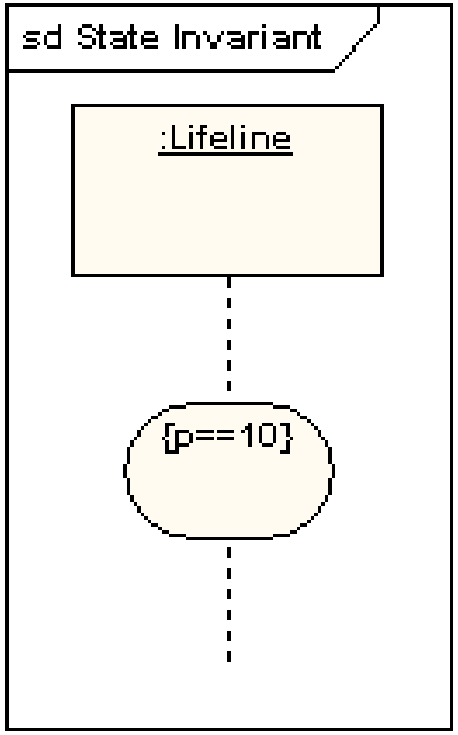
Part Decomposition (UML 2.0)

- An object can have more than one lifeline coming from it.
- This allows for inter- and intra-object messages to be displayed on the same diagram.



State Invariant (UML 2.0)

- A state invariant is a constraint placed on a lifeline that must be true at run-time.
- It is shown as a rectangle with semi-circular ends.



Sequence diagram for ReportEmergency [Bruege&Dutoit]

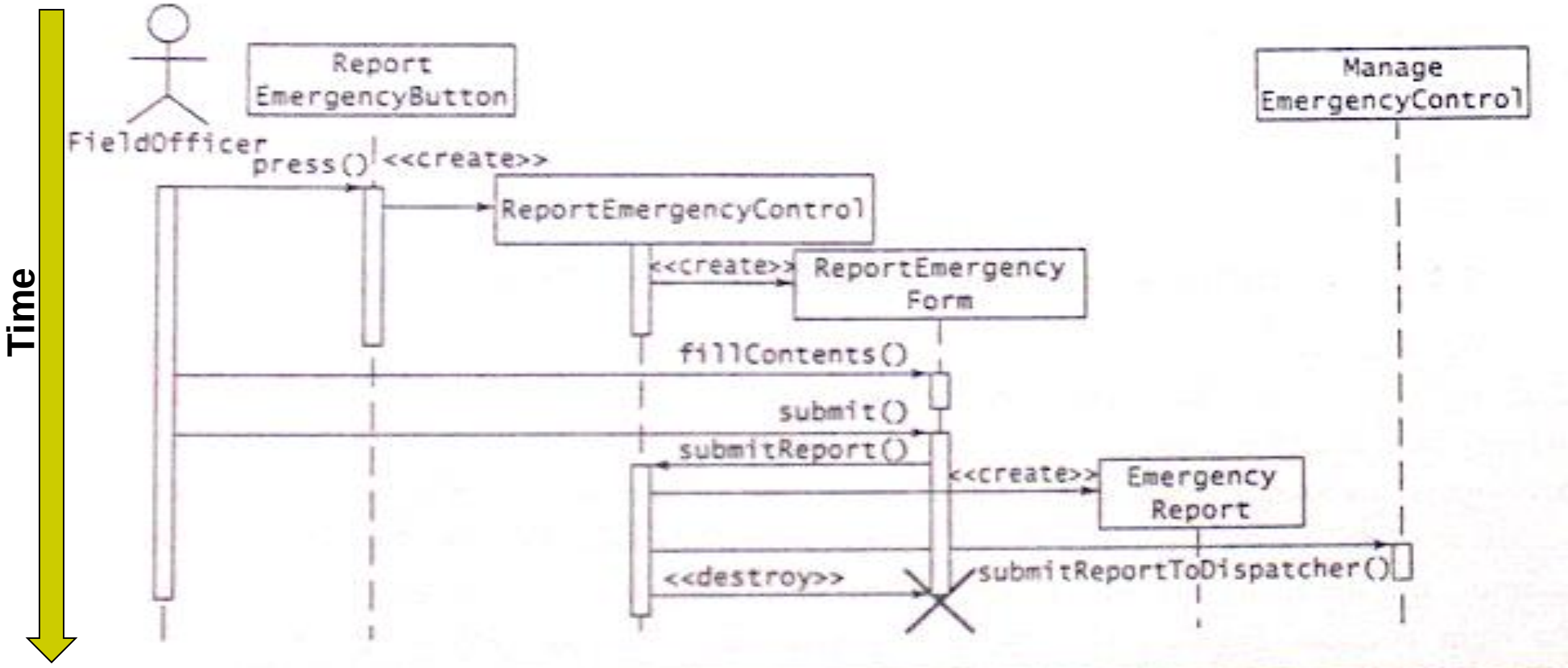


Figure 5-8 Sequence diagram for the ReportEmergency use case.

Sequence diagram for ReportEmergency-2 [Bruege&Dutoit]

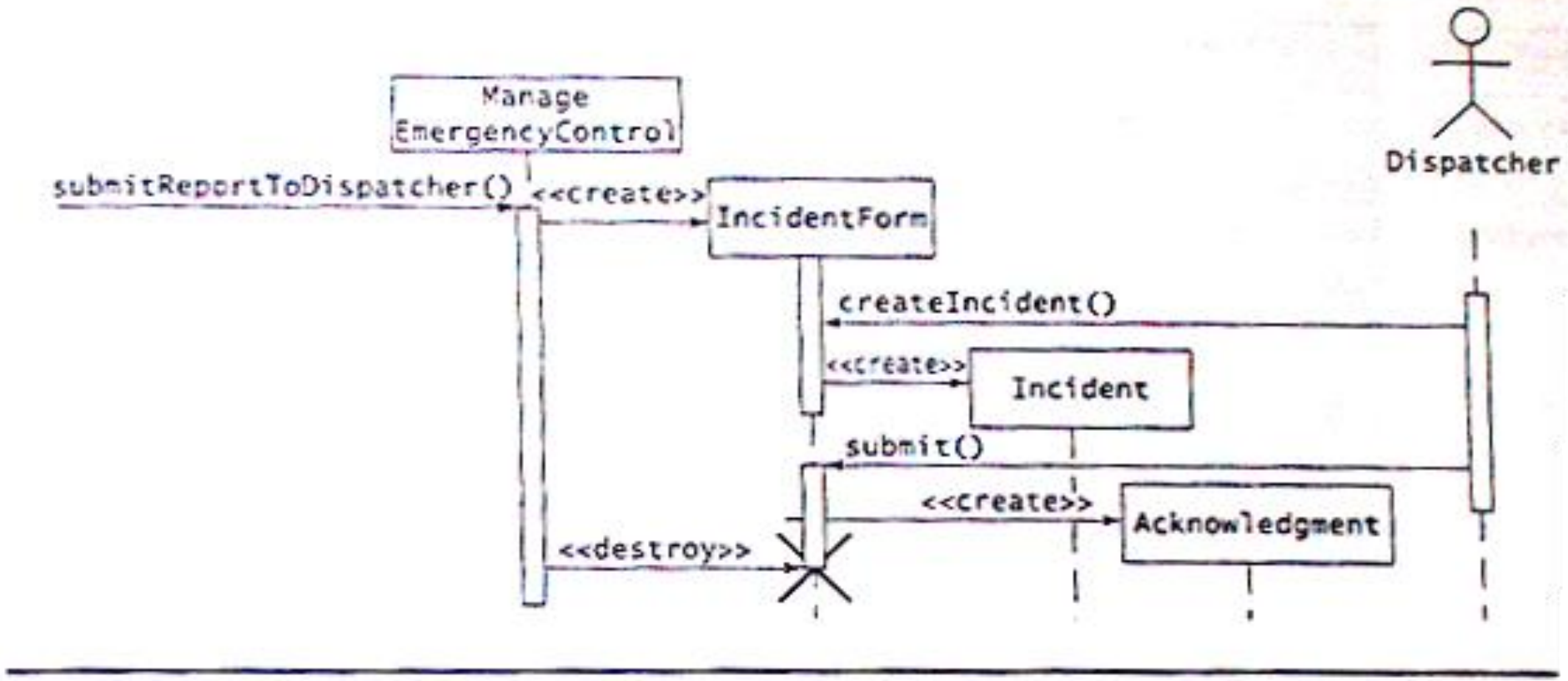


Figure 5-9 Sequence diagram for the ReportEmergency use case (continued from Figure 5-8).

Sequence diagram for ReportEmergency-3 [Bruege&Dutoit]

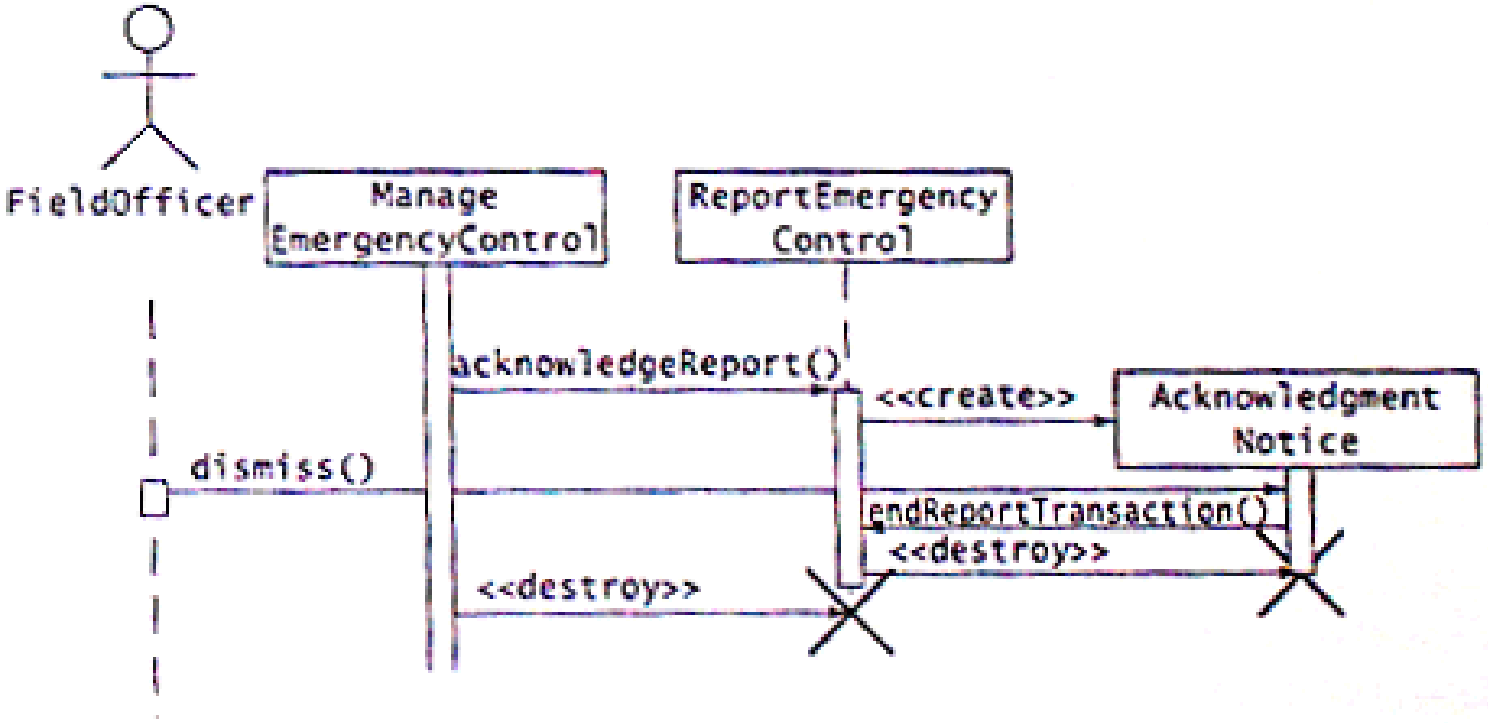
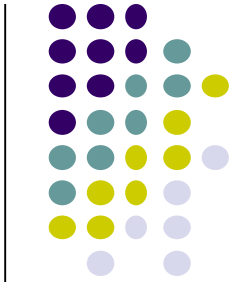


Figure 5-10 Sequence diagram for the ReportEmergency use case (continued from Figure 5-9).

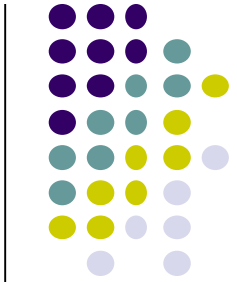
Centralizing Control Flow in Sequence Diagrams



Centralized control of a flow of events means that **a few objects steer the flow by sending messages to, and receiving messages from other objects**. These controlling objects decide the order in which other objects will be activated in the use case. Interaction among the rest of the objects is very minor or does not exist.

Main advantage: **each object does not have to keep track of the next object's tally**. To change the order of the sub-event phases, you merely make the change in the control object. Another advantage to this structure is that **you can easily reuse the various sub-event phases in other use cases** because the order of behavior is not built into the objects.

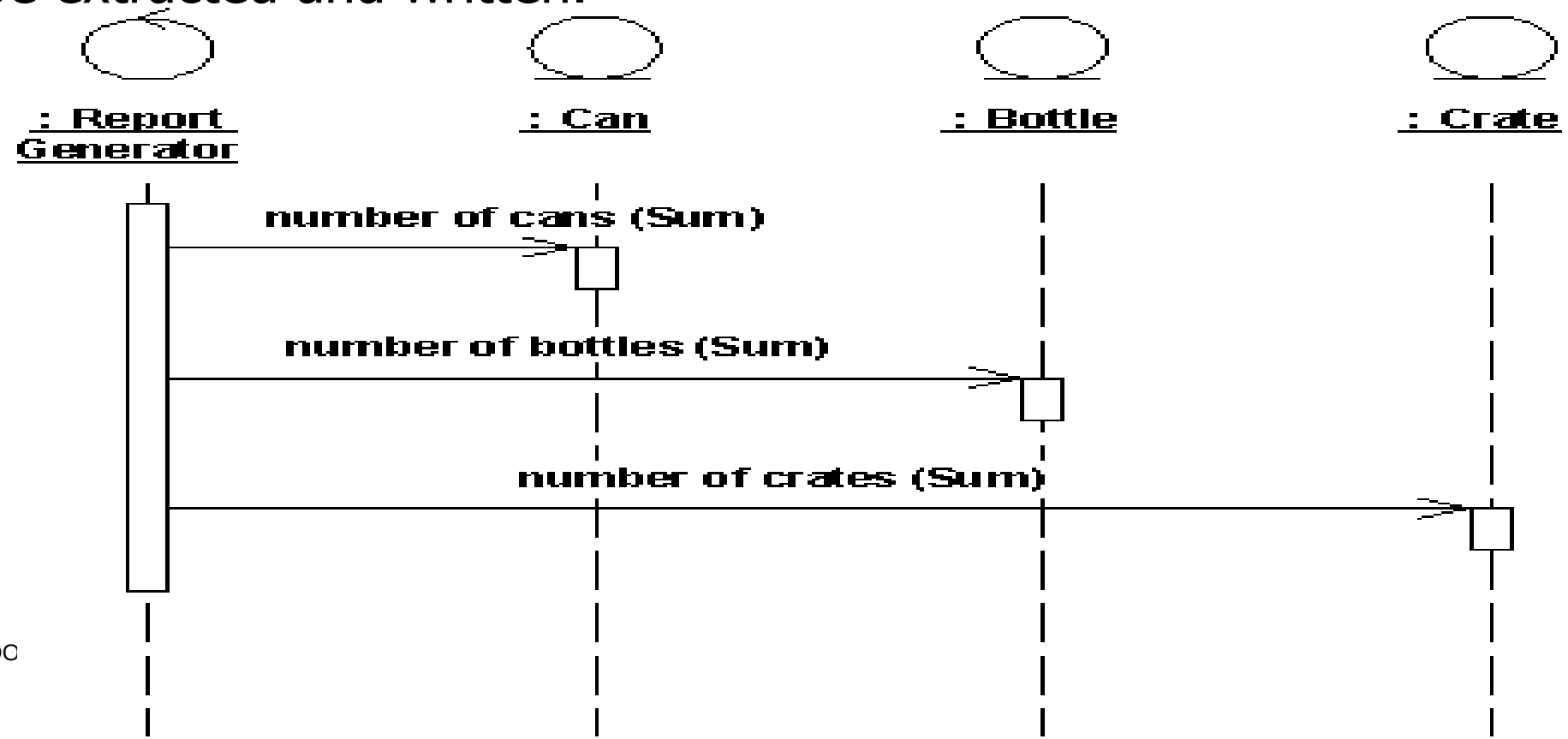
Distributing Control Flow in Sequence Diagrams



Decentralized control arises when the participating objects communicate directly with one another, not through one or more controlling objects.

Centralized Control - Example

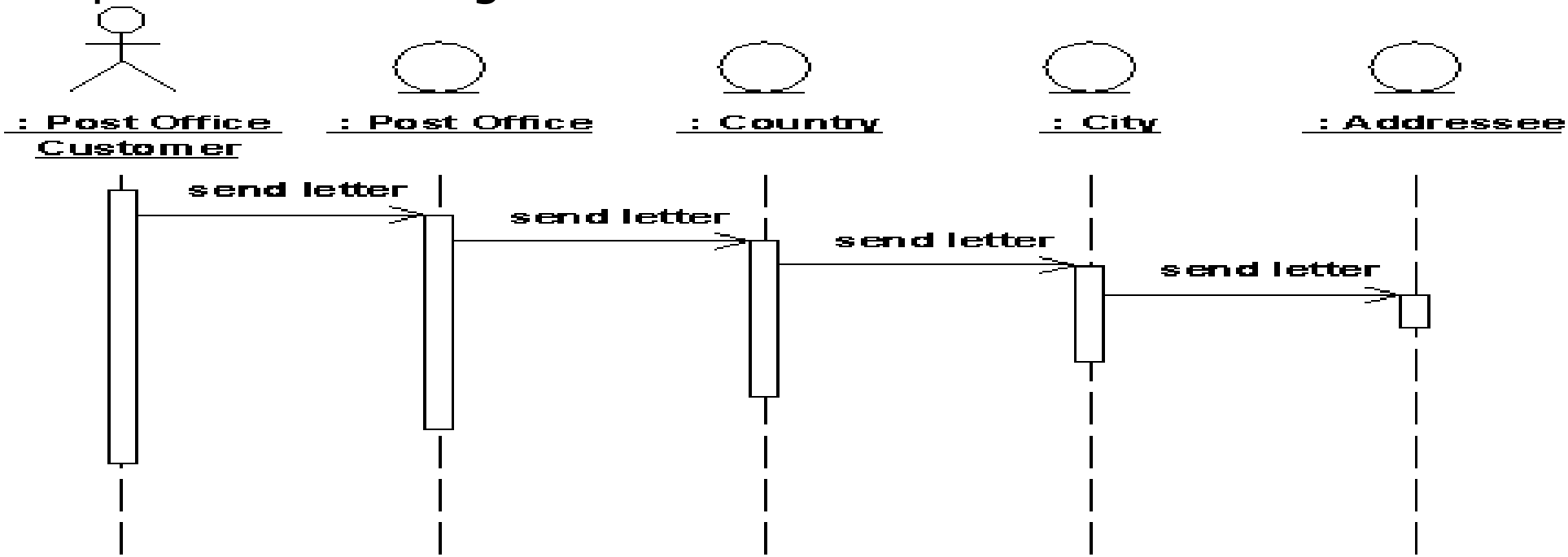
In the **Recycling-Machine System,** the use case **Print Daily Report** keeps track of - among other things - the number and type of returned objects, and writes the tally on a receipt. The **Report Generator** control object decides the order in which the sums will be extracted and written.



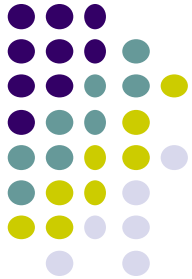
Decentralized Control - Example

In the use case **Send Letter** someone mails a letter to another country through a post office. The letter is first sent to the country of the addressee. In the country, the letter is sent to a specific city. The city, in turn, sends the letter to a specific city. The city, in turn, sends the letter to the home of the addressee.

The sub-event phases belong together. The sender of the letter speaks of "sending a letter to someone."



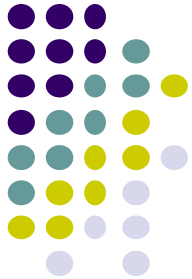
Control Flow – How to be used?



*A **decentralized** structure is appropriate:*

- If the sub-event phases are tightly coupled:
  - **Form a part-of or consists-of hierarchy, such as Country - State - City;**
  - **Form an information hierarchy, such as CEO - Division Manager - Section Manager;**
  - **Represent a fixed chronological progression (the sequence of sub-event phases will always be performed in the same order), such as Advertisement - Order - Invoice - Delivery - Payment; or**
  - **Form a conceptual inheritance hierarchy, such as Animal - Mammal - Cat.**
- If you want to encapsulate, and thereby make abstractions of, functionality.

Control Flow – How to be used? (2)

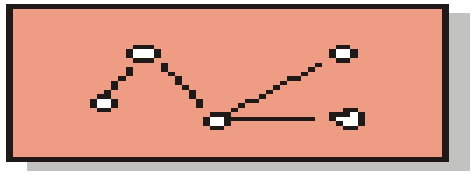


*A **centralized** structure is appropriate:*

- If the order in which the sub-event phases will be performed is likely to change.
- If you expect to insert new sub-event phases.
- If you want to keep parts of the functionality reusable as separate pieces.

Communication (collaboration) UML 2.0 Diagrams

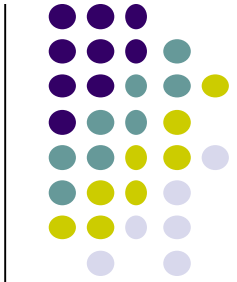
Collaboration Diagram	A **communication (collaboration) diagram** describes a pattern of objects interaction; it shows the objects participating in the interaction by their links to each other and the messages sent to each other.



Unlike a sequence diagram, a collaboration diagram shows the *relationships* among the objects.

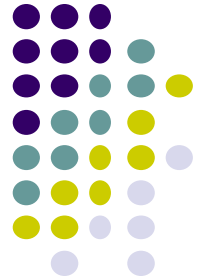
You can have ***objects*** and ***actor instances*** in communication (collaboration) diagrams, together with links and ***messages*** describing how they are related and how they interact. The diagram describes what takes place in the participating objects, in terms of how the objects communicate by sending messages to one another.

Communication vs Sequence Diagrams



- Communication diagrams are especially good at showing which links are needed between participants to pass an interaction's messages.
- On a sequence diagram, the links between participants are implied by the fact that a message is passed between them. Communication diagrams provide an intuitive way to show the links between participants that are required for the events that make up an interaction (the order of the events involved in an interaction is secondary).
- Sequence and communication diagrams are so similar that most UML tools can automatically convert from one diagram type to the other.

Contents of Communication (Collaboration) Diagrams - Links

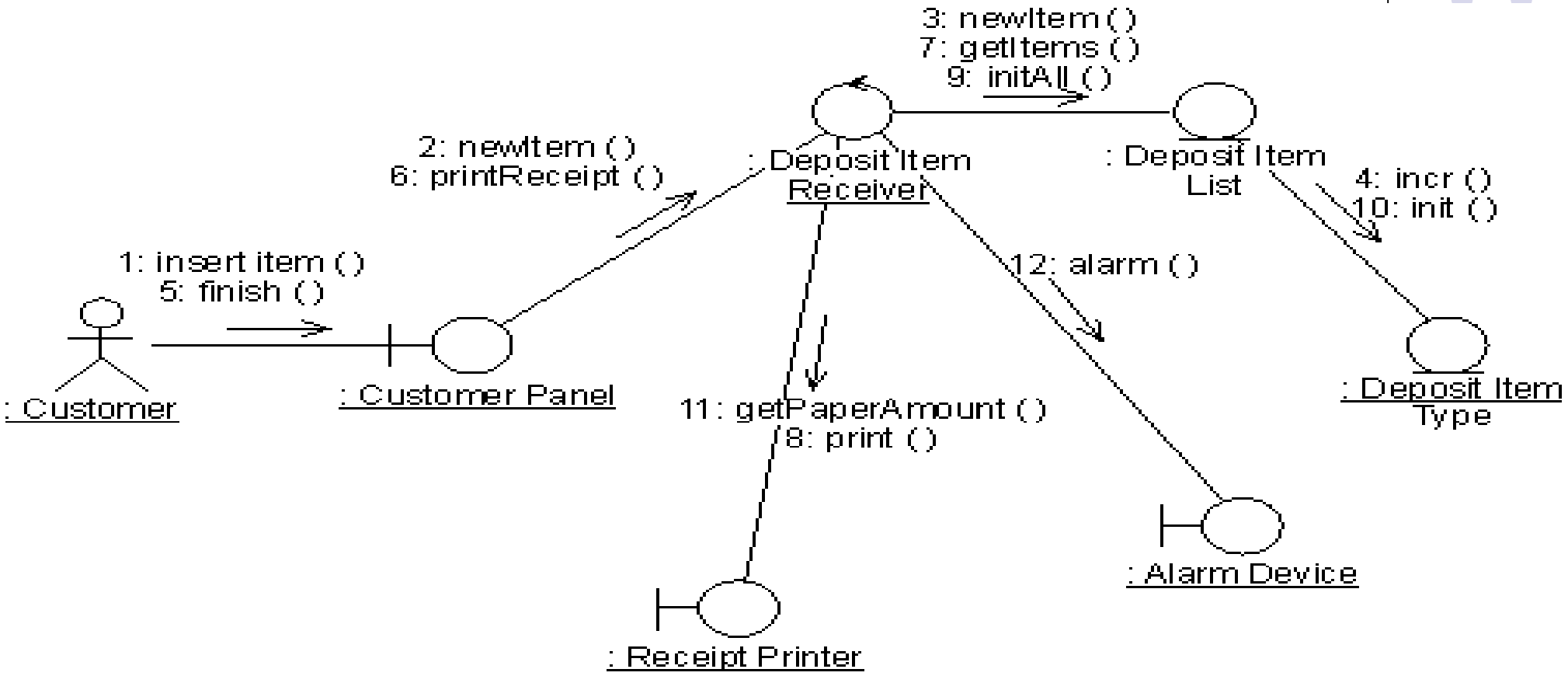


- A link is a relationship among objects across which messages can be sent (shown as a solid line between two objects).

- An object interacts with, or navigates to, other objects through its links to them.

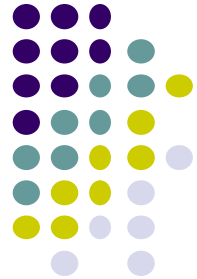
- A link can be an instance of an association, or it can be **anonymous**, meaning that its association is unspecified.

Example of Communication Diagram



Events' flow of the use case *Receive Deposit Item* in the *Recycling-Machine System*

Contents of Communication Diagrams – Messages 1/2

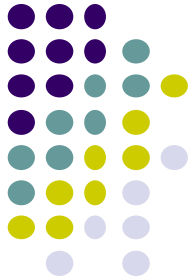


- A message is a communication between objects that conveys information with the expectation that activity will ensue

- In collaboration diagrams, a message is shown as a labeled arrow placed near a link. This means that the link is used to transport, or otherwise implement the delivery of the message to the target object

- The arrow points along the link in the direction of the target object (the one that receives the message).

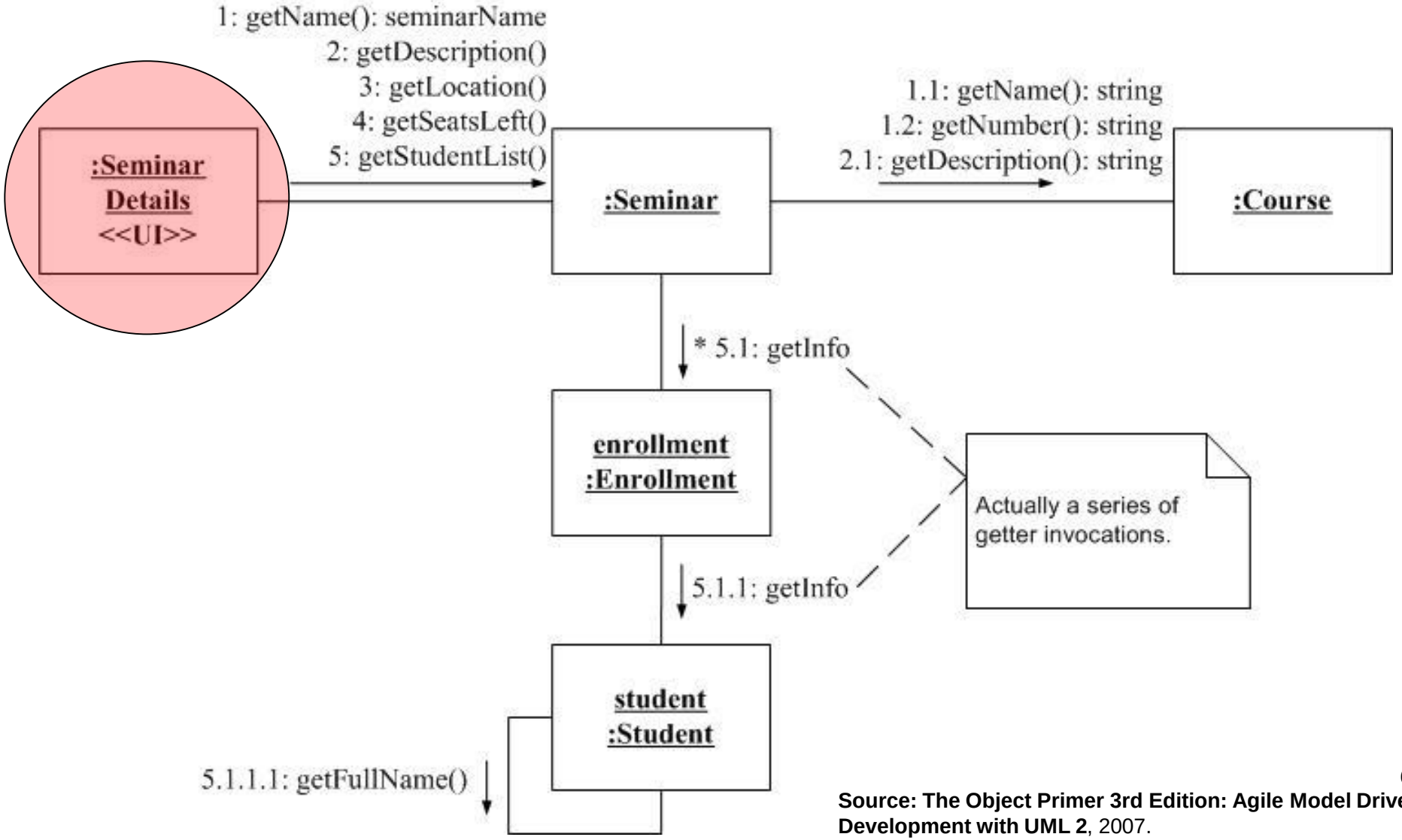
Contents of Communication Diagrams – Messages 2/2



- The arrow is labeled with the name of the message, and its parameters. The arrow may also be labeled with a sequence number to show the sequence of the message in the overall interaction.
- A message can be unassigned, meaning that its name is a temporary string that describes the overall meaning of the message. You can later assign the message by specifying the operation of the message's destination object.
- Message notation:

[sequenceNumber:] methodName(parameters) [: returnValue]

Sample communication diagram

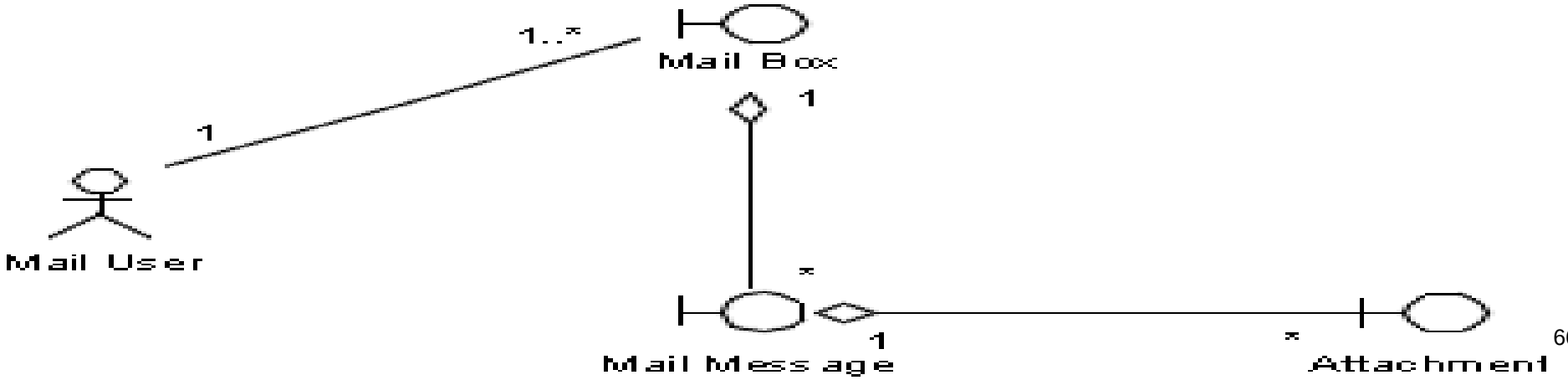


Source: The Object Primer 3rd Edition: Agile Model Driven Development with UML 2, 2007.

Boundary Objects in Interaction Diagr.

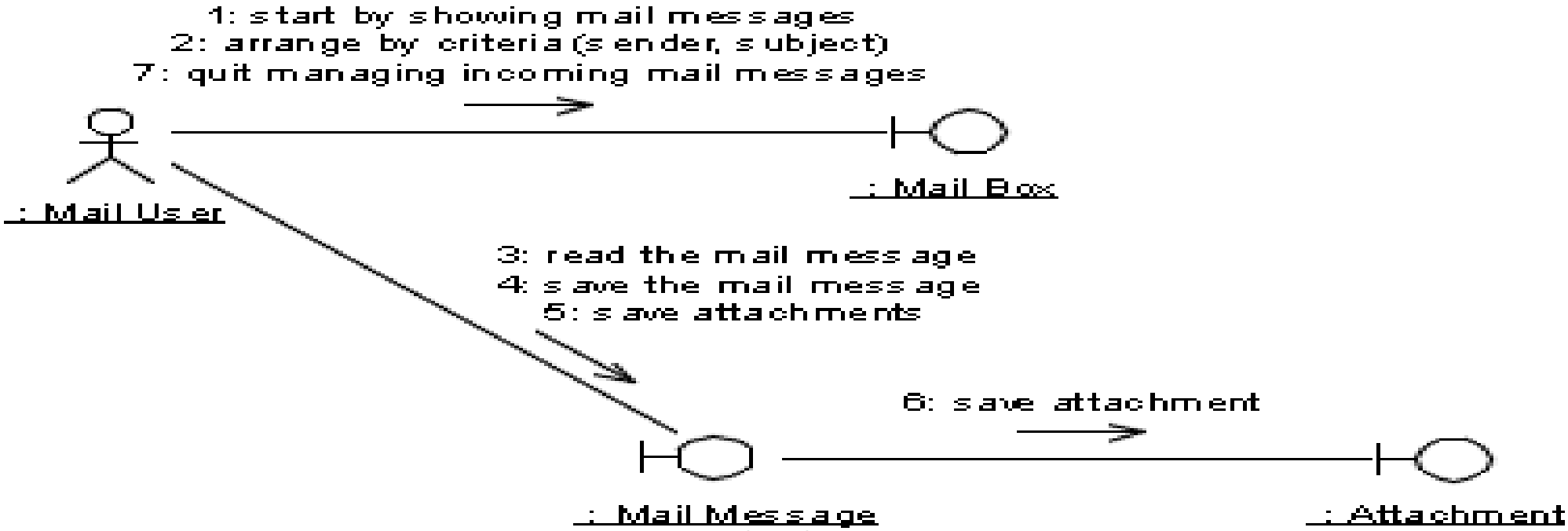
To illustrate the boundary objects participating in the use-case storyboard (a logical and conceptual description of how a use case is provided by the user interface), and their interactions with the user, we use communication or sequence diagrams. This is useful for use cases with complex sequences or flows of events.

Example: Class diagram including the Mail User actor and the boundary classes Mail Box, Mail Message, and Attachment, realizing ***Manage Incoming Mail Messages use case***.



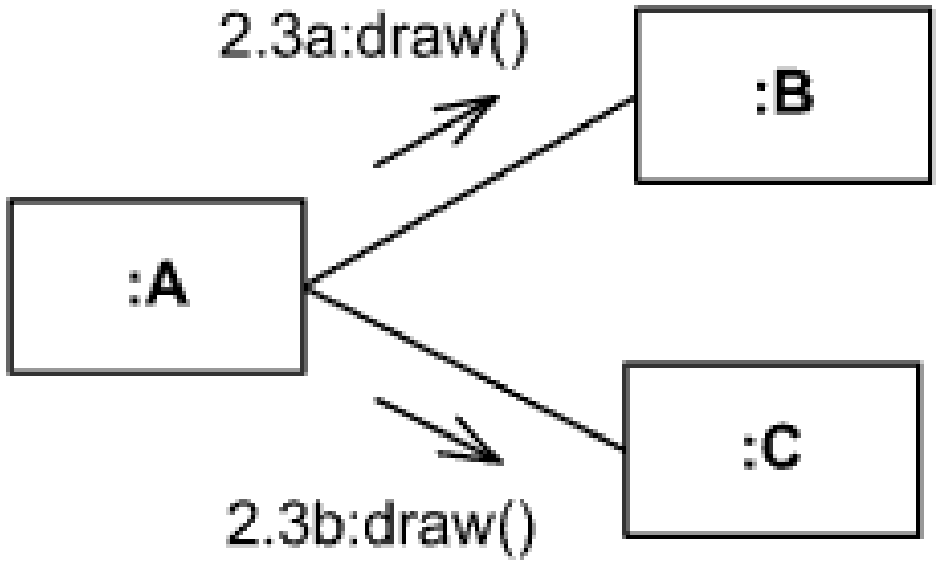
Boundary Object Collaboration/Communication Diagrams - Example

CD including the Mail User actor and boundary objects of Mail Box, Mail Message, and Attachment, participating in a use-case storyboard realizing the ***Manage Incoming Mail Messages*** use case:

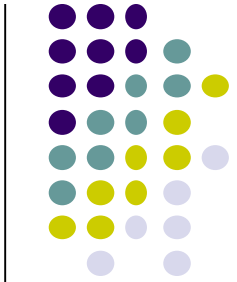


Concurrency

- *Instance of A sends draw() messages concurrently to instance of B and to instance of C*
- Messages 2.3a and 2.3b are concurrent within activation 2.3



Recurrence (Repetition)



- The **recurrence** defines **conditional** or **iterative** execution of zero or more messages that are executed depending on the specified condition.

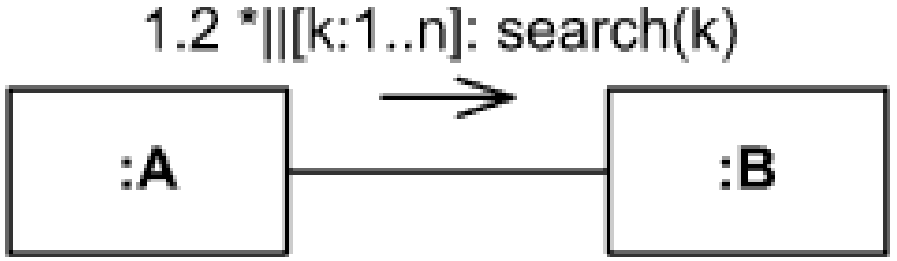
<recurrence> ::= <branch> | <loop>
<branch> ::= '[' <guard> ']'
<loop> ::= '*' ['||'] ['[' <iteration-clause> ']']

Guard condition

- *Instance of class A will send draw() message to instance of C if x > y*

Sequence vs Concurrency

- *Instance of class A will send search() message to instance of B n times,* ***one by one***
- *Instance of class A will send n* ***concurrent*** *search() messages to instance of B*

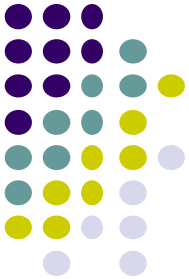


Comparing sequence and communication diagrams

(from Learning UML 2.0, by K. Hamilton, R. Miles)

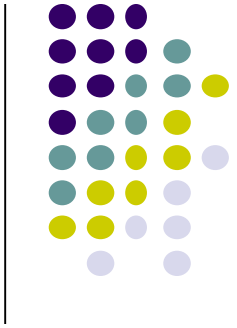
Feature	Sequence diagrams	Communication diagrams
Shows participants effectively	Participants are mostly arranged along the top of page, unless the drop-box participant creation notation is used. It is easy to gather the participants involved in a particular interaction.	Participants as well as links are the focus, so they are shown clearly as rectangles.
Showing the links between participants	Links are implied. If a message is passed from one participant to another, then it is implied that a link must exist between those participants.	Explicitly shows the links between participants. In fact, this is the primary purpose of these types of diagram.
Showing message signatures	Message signatures can be fully described.	Message signatures can be fully described.
Supports parallel messages	With the introduction of sequence fragments, sequence diagrams are much better.	Shown using the number-letter notation on message sequences.
Supports asynchronous messages (fire and forget)	Achieved using the asynchronous arrow.	Communication diagrams have no concept of the asynchronous message since its focus is not on message ordering.
Easy to read message ordering	This is a sequence diagram's forté. Sequence diagrams clearly show message ordering using the vertical placement of messages down the diagram's page.	Shown using the number-point-nested notation.
Easy to create and maintain the diagram	Creating a sequence diagram is fairly simple. However, maintaining sequence diagrams can be a nightmare unless a helpful UML tool is being used.	Communication diagrams are simple enough to create; however, maintenance, especially if message numbering needs to be changed, still ideally needs the support of a helpful UML tool.

Homework



- **To convert a sequence diagram into a communication diagram and vice versa on Visual Paradigm for UML**

The University Course Registration (UCR) Case Study



Object Name

History 101-Section2

Object Name & Class

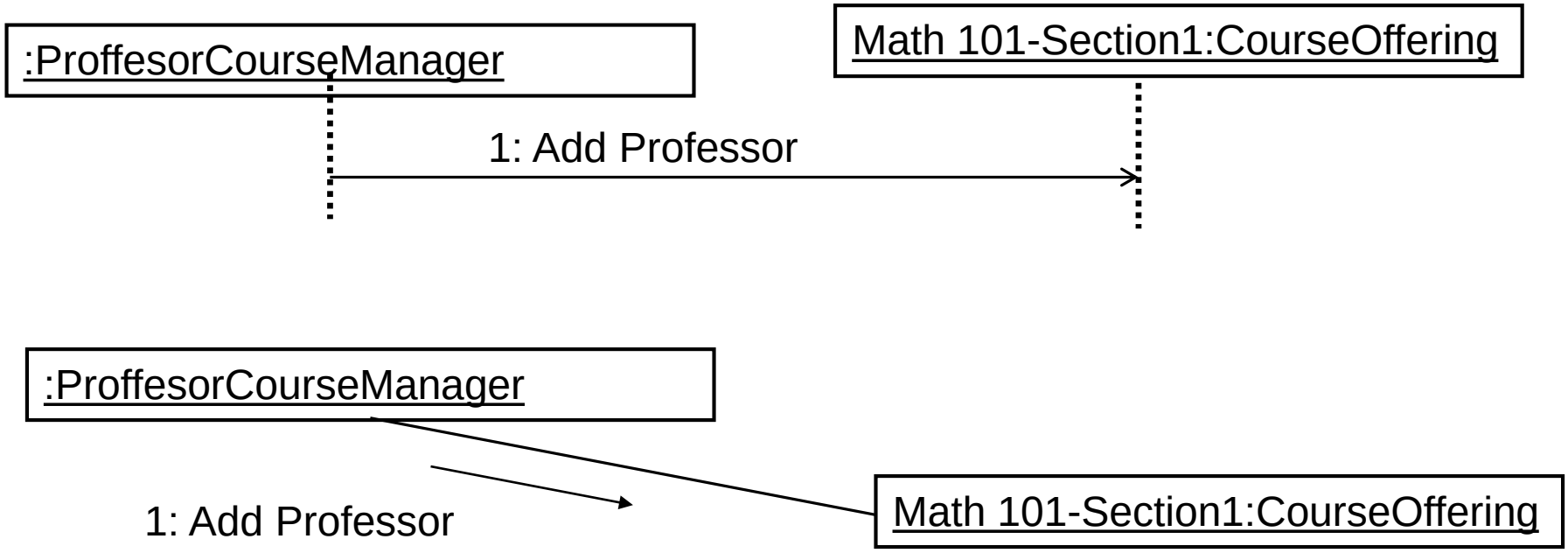
History 101-
Section7:CourseOffering

Class Name

:CourseOffering

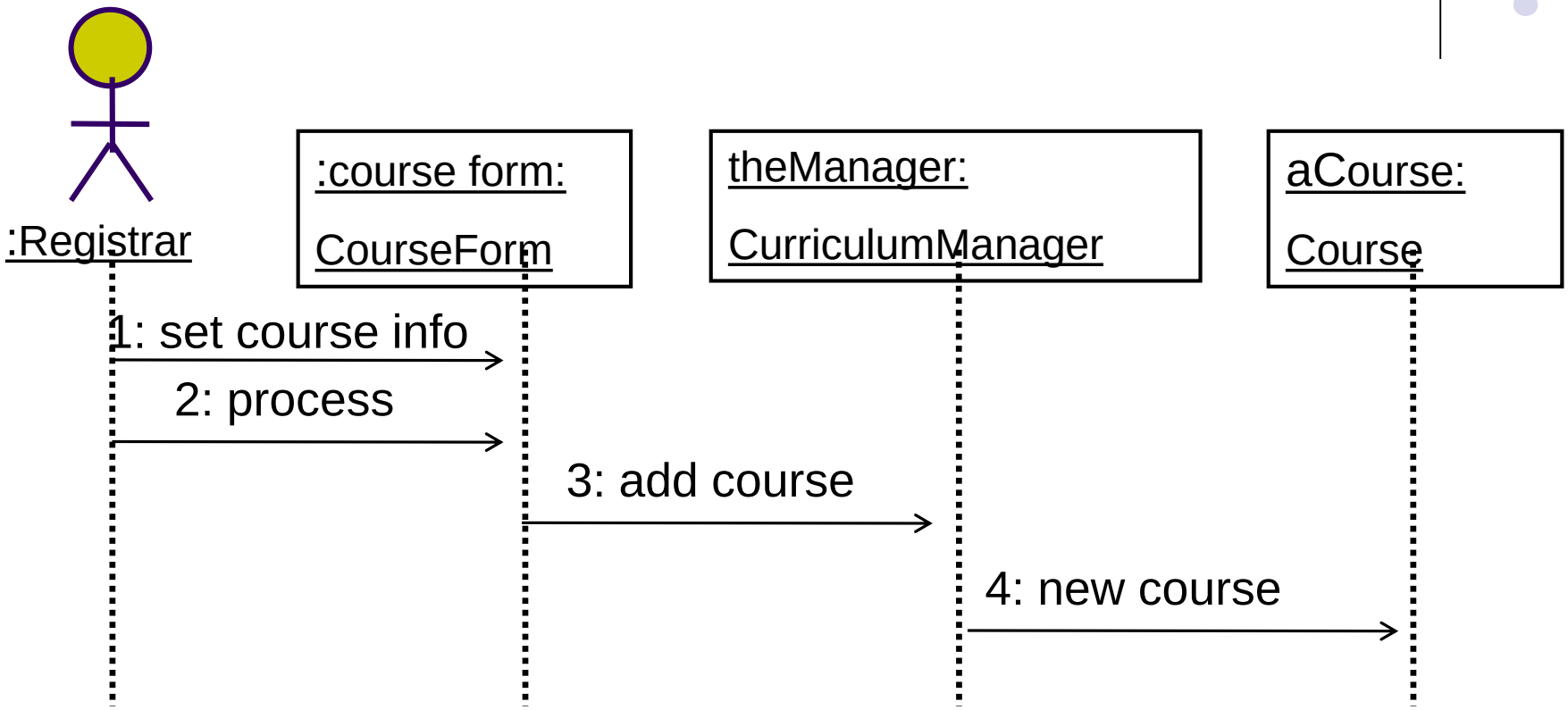
Naming Objects in a Sequence Diagram

UCR - 2



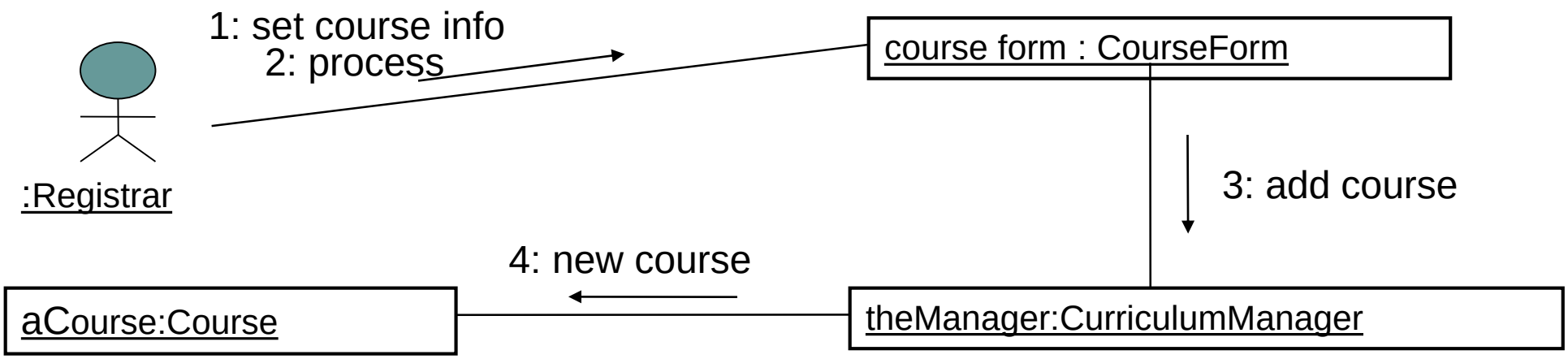
Sequence Diagram (over) versus Collaboration Diagram (bellow)

UCR - 3



Sequence Diagram with Objects Assigned to Classes

UCR - 4



Collaboration Diagram