SOFT 437

Software Performance Analysis

UML Tutorial

What is UML?

- Unified Modeling Language (UML) is a standard language for specifying, visualizing, constructing, and documenting the artifacts for software systems, as well as for business modeling and other non-software systems.
- The UML is a very important part of developing object oriented software and the software development process
- The UML uses mostly graphical notations to express the design of software projects

UML Tools

- Standalone tools
  - Web-based tools (e.g., creatly.com)
  - Packages (e.g., ArgoUML)

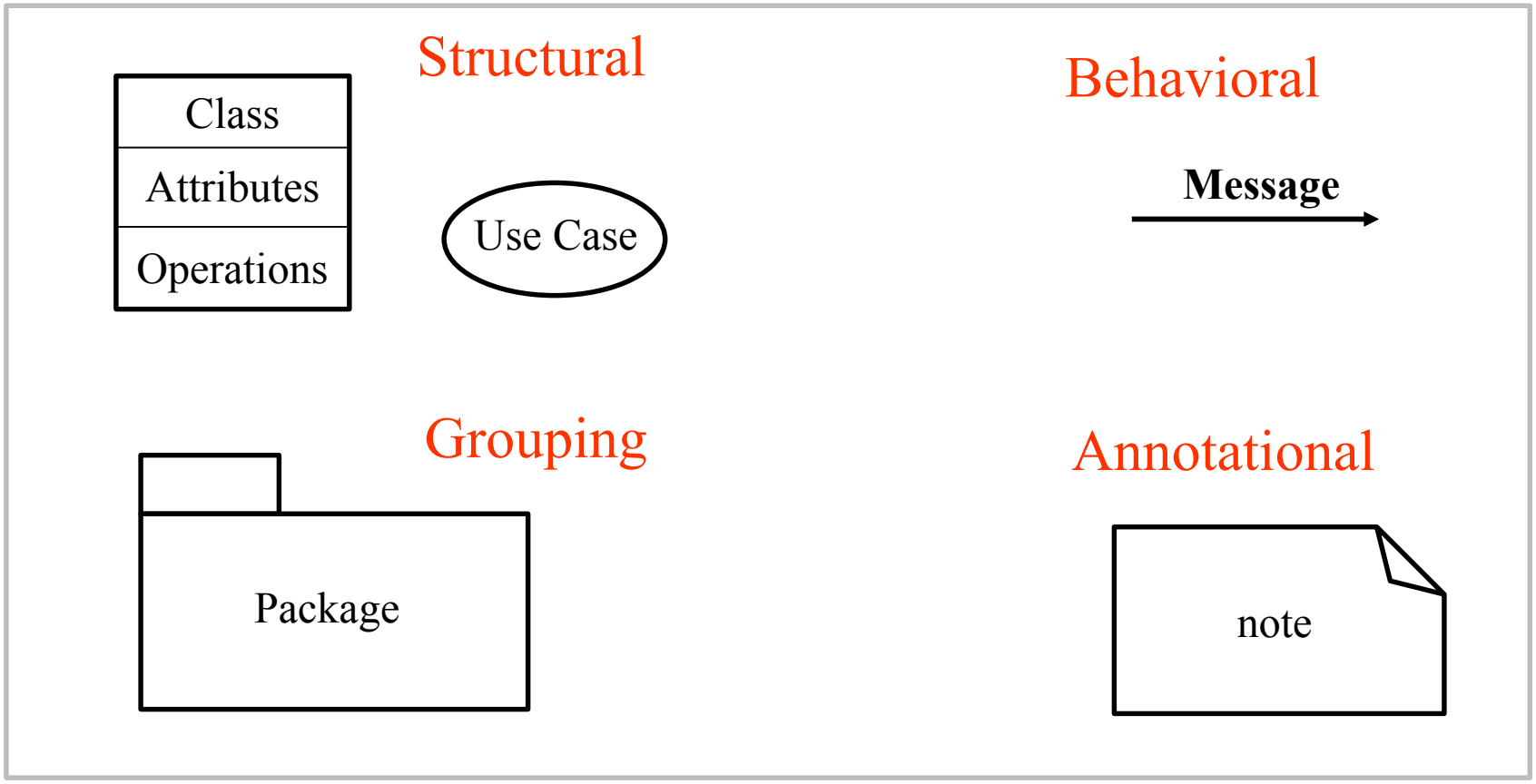
- Plugin tools
  - Integrate with IDEs (e.g., UML2-SDK for Eclipse)
- Tutorials
  - Many many tutorials online

UML Building Blocks

- Things
  - Structural Behavioral
  - Grouping Annotational
- Relationships
- Diagrams

UML Building Blocks

- Things
  - Structural Behavioral
  - Grouping Annotational

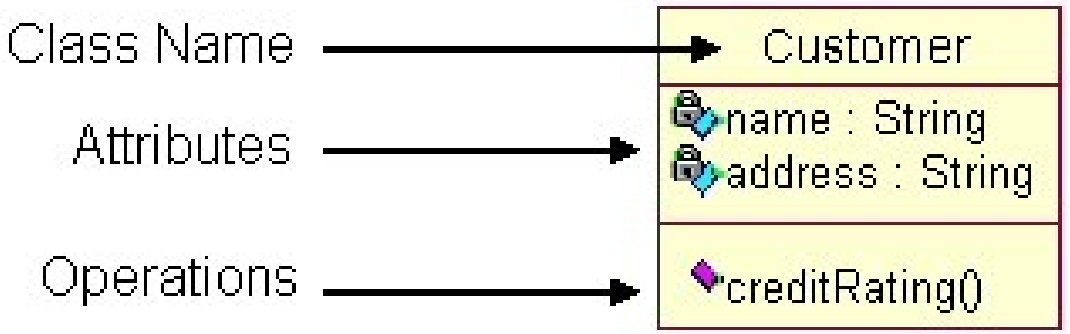


UML Diagrams

1. Class diagram
2. Object diagram
3. Use case diagram
4. Sequence diagram
5. Collaboration diagram
6. Activity diagram
7. Statechart diagram
8. Deployment diagram
9. Component diagram

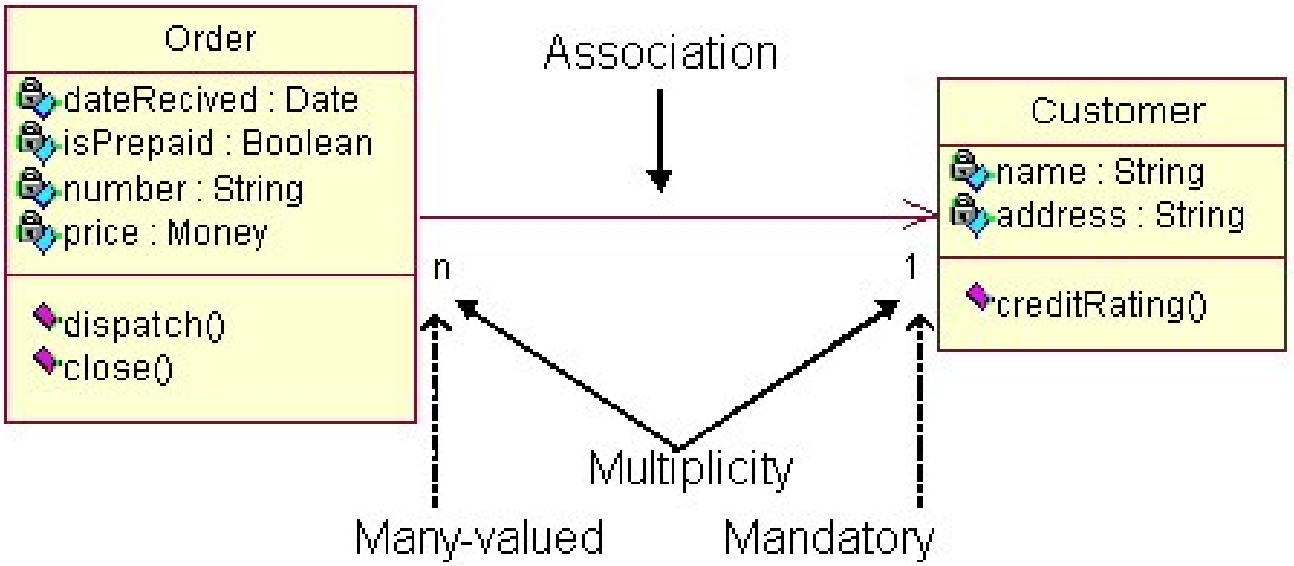
Class Diagrams

- Classes are composed of three components: a name, attributes, and operations.



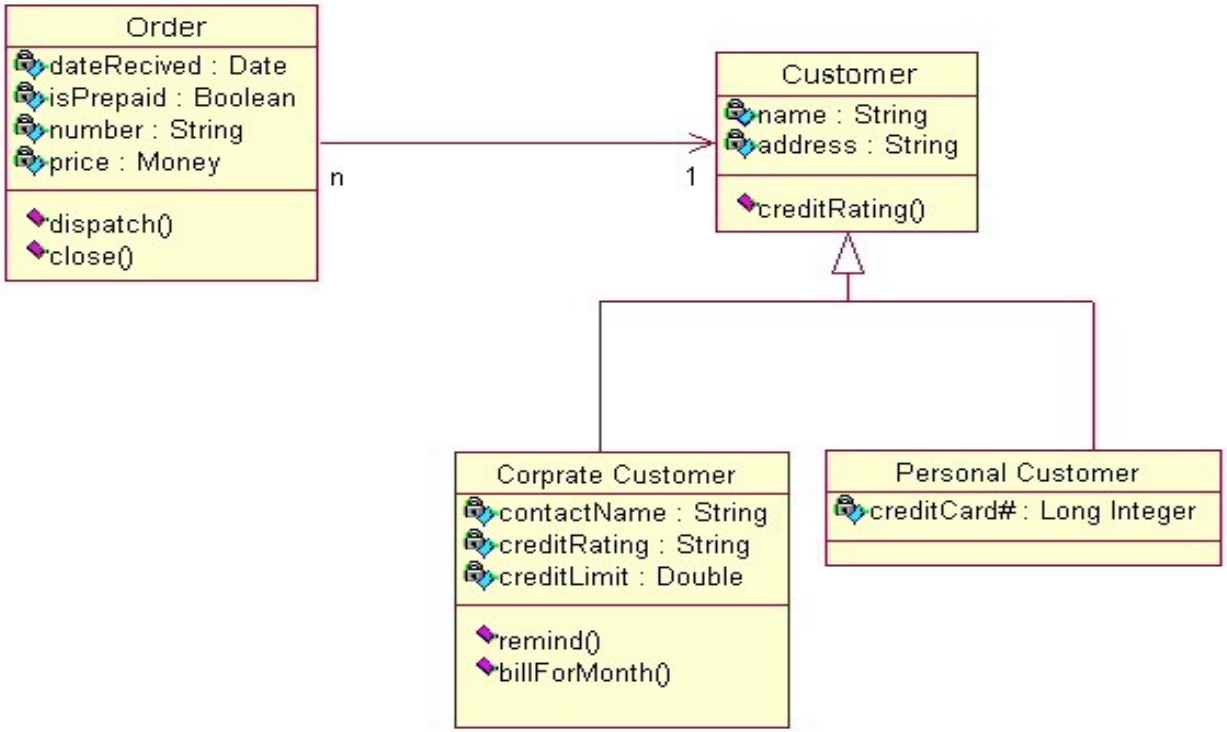
Class Diagrams (con't)

- Class diagrams also display relationships such as containment, inheritance, and association



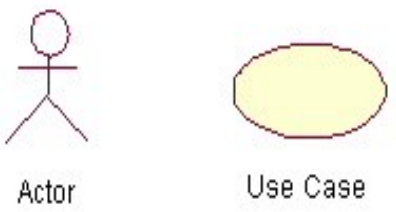
Class Diagrams (con't)

- Model class structure and contents

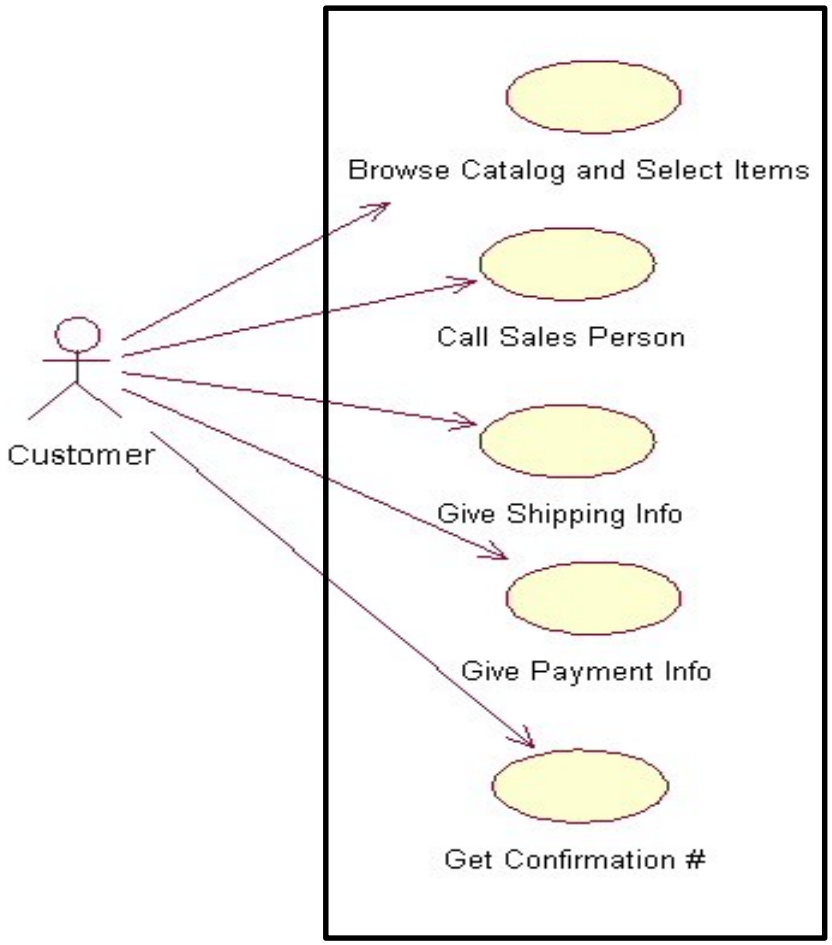


Use Cases

- Use cases describe the behavior of a system or a subsystem
  - A set of actions that a system performs and yields an observable result of value to an actor
  - An actor is an entity outside the system (e.g., a user or another system) and interacts directly with the system
- A use case does not reveal internal details of interactions between actors and the system



Example Use Cases



Use Case Diagrams

- A use case diagram shows
  - a set of use cases,
  - the actors that interact with use cases
  - the relationships
- An actor can be represented by stick figures or stereotyped icons
- A use case is represented by an ellipse that contains the name of the use case

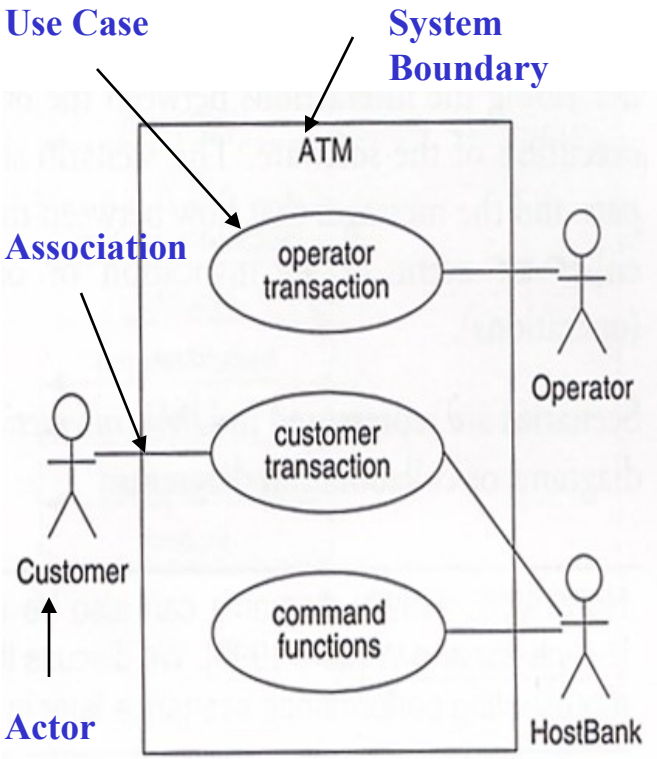


Figure 3-5: Use Case Diagram

Critical Use Cases

- Use cases are employed to
  - model the context of the system: the system boundary indicates which ***features*** are part of (inside) the system, which ***actors*** interact with system, and the ***meaning of interaction***
  - specify the requirements for the system (i.e., what the system should do from the point of view of actors)
- From a performance point of view, use case diagrams are used to identify ***the critical functions of the system that are most important to performance***
- The ***critical use cases*** are considered, including
  - are ***critical*** to the ***operation*** of the system
  - influence users' perception of ***responsiveness***
  - represent a ***risk*** that performance goals might not be met

Use Cases and Scenarios

- The **SPE** process focuses on use cases and the scenarios that describe them
- By examining the system's use cases, you can identify the functions of the system that are significant to performance
- A scenario is an **instance** of a use case
- Performance scenario are the scenarios that have the most impact on performance

Scenarios

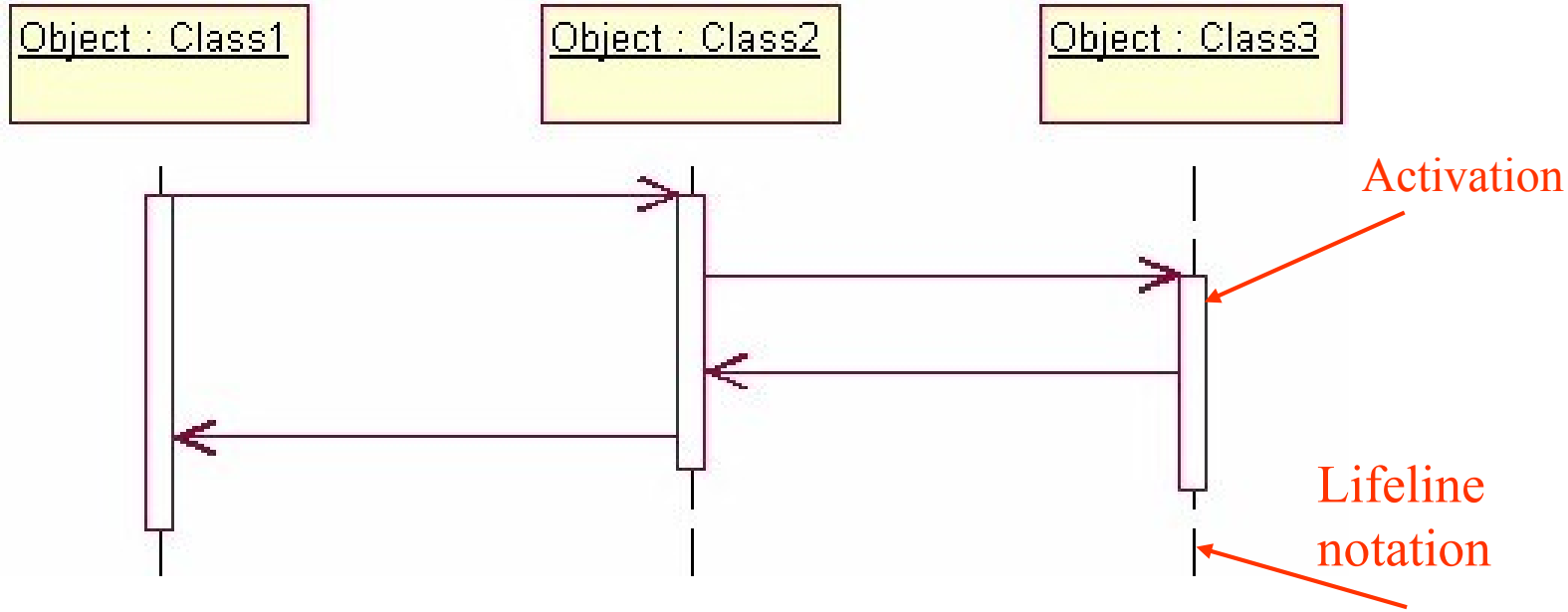
- A scenario is an instance of a use case
- It consists of a sequence of steps describing the interactions between the objects involved in a particular execution of the software
- The scenario shows
  - objects that participate
  - messages (e.g., event or method invocation) that flow between them

Scenarios (cont.)

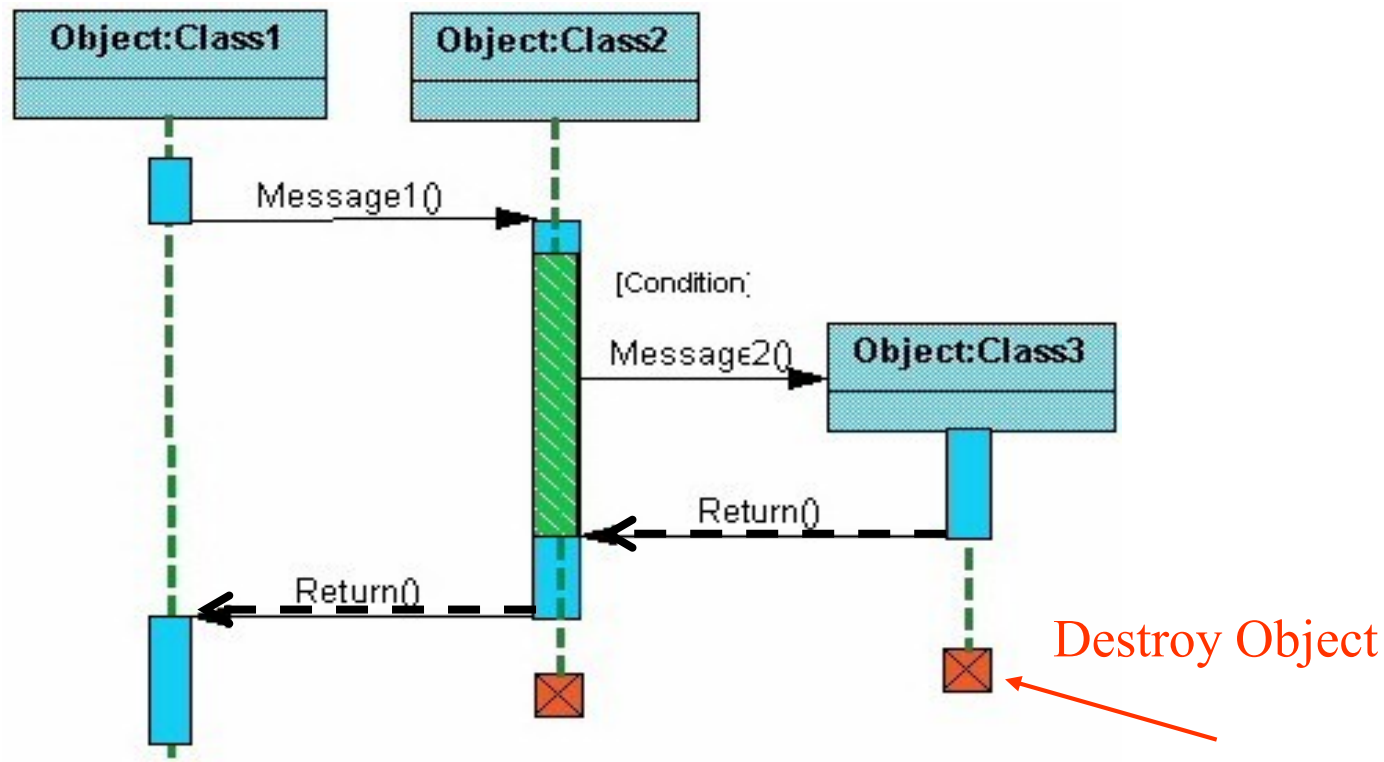
- Scenarios are represented by either **sequence diagrams** or **collaboration diagrams**
- Sequence diagrams emphasize the time-ordering of message
- Collaboration diagrams emphasize the structural organization of the collection of interacting objects
- Sequence diagrams are more natural to use for constructing performance models

Sequence Diagrams

- Sequence diagrams demonstrate the behaviour of objects in a use case by describing the objects and the messages they pass.



Sequence Diagrams (con't)



Creation and Destruction

- An activation indicates a period of time when the object is busy performing some action
- Object creation and destruction are indicated by the stereotyped messages

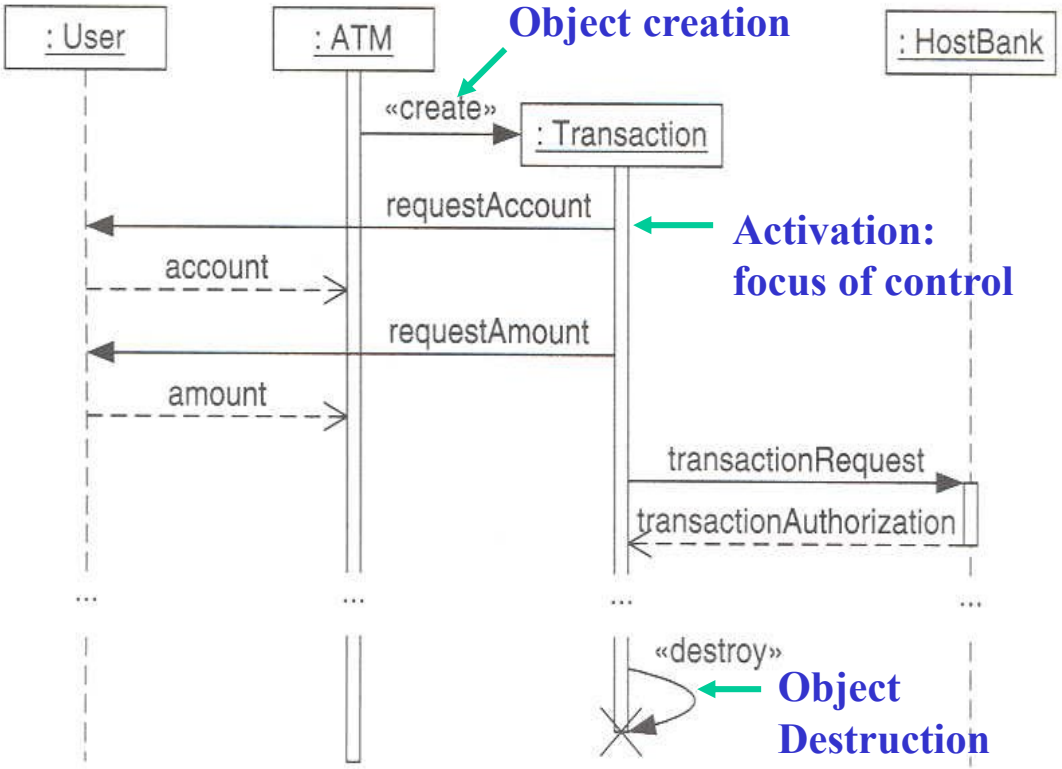


Figure 3-7: Creation and Destruction

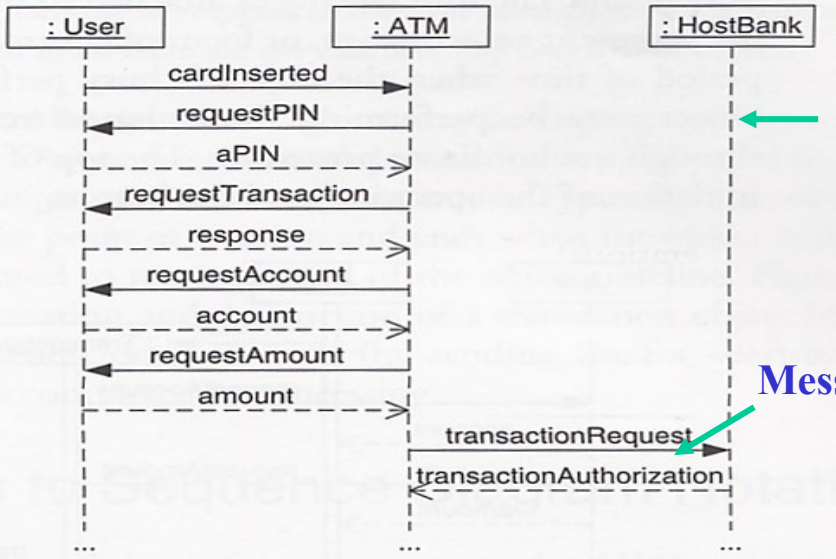


Figure 3-6: Sequence Diagram

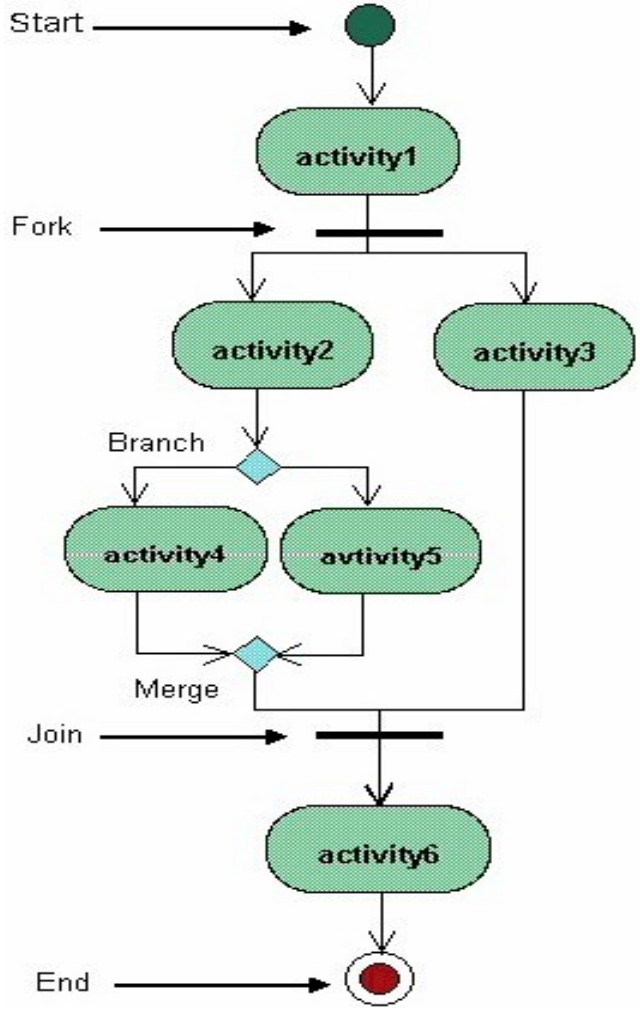
Table 3-1: Message Control Flow

→	A solid arrowhead indicates procedural flow of control. Any nested operations are completed before the outer-level sequence of actions continues. This models ordinary procedure calls as well as concurrent interactions where the sender waits for a nested sequence of behavior to complete before proceeding.
→	A stick arrowhead indicates nonprocedural, or flat, flow of control. Each arrow simply indicates the next step in the sequence.
→	A half-stick arrowhead shows asynchronous communication between two objects.
⇢	A dashed arrow with a stick arrowhead shows a return from a procedure call. The return arrow is sometimes omitted because a return is implicit at the end of an activation.

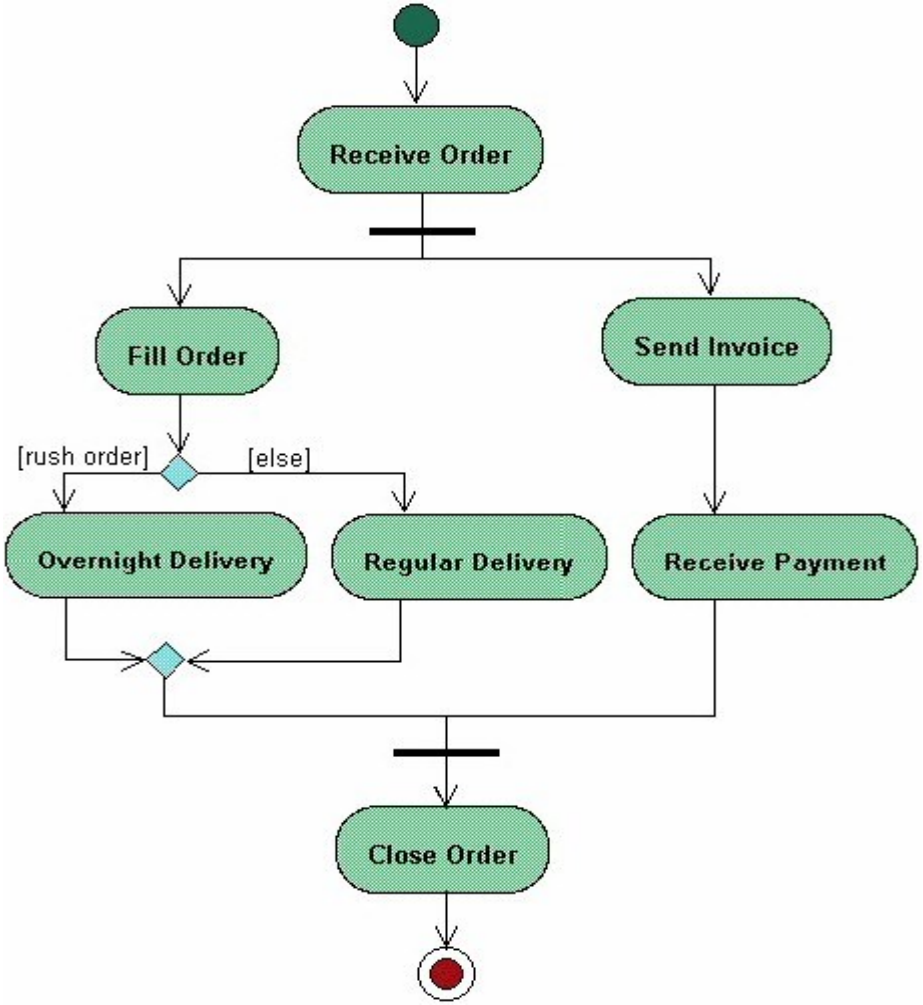
Basic Sequence Diagrams & Message Control Flow

Activity Diagrams

- Show the flow of activities through the system



Activity Diagrams (con't)



Quick Quiz

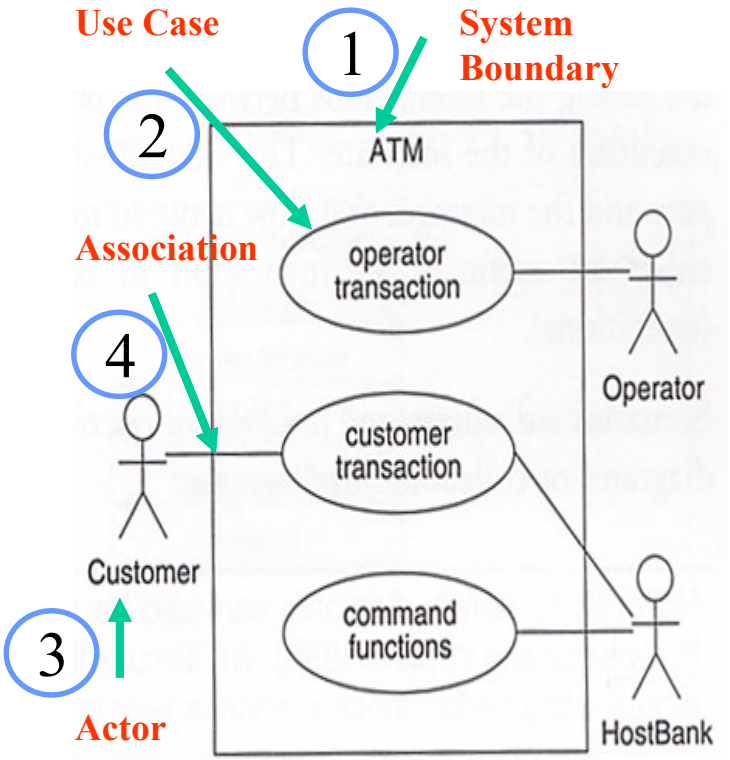
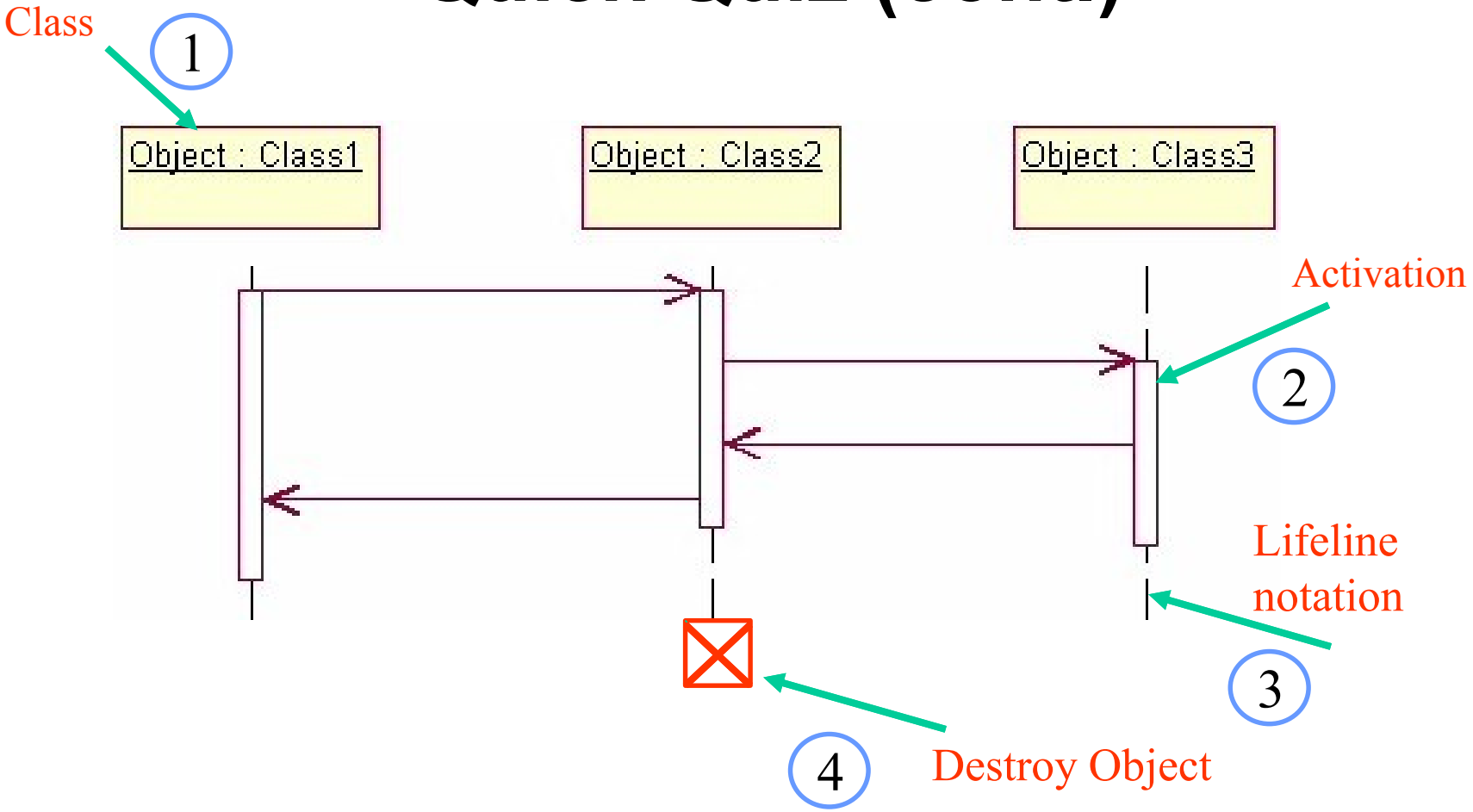


Figure 3-5: Use Case Diagram

Quick Quiz (cont.)



SOFT 437

Software Performance Analysis

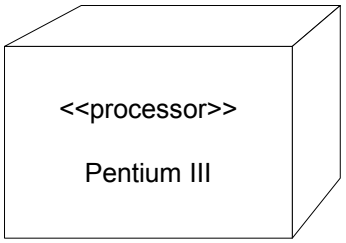
Chapter 3: SPE and the UML

Extending the UML

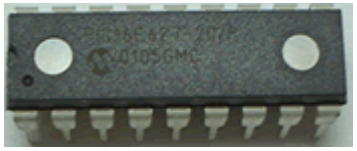
- UML provides built-in extension mechanisms that allow you to tailor the notation for particular purposes.
- These mechanisms are
  - stereotypes
  - tagged values
  - constraints

Stereotypes

- A stereotype allows you to create new model elements
  - derived from existing UML elements
  - specific to a problem domain
- The stereotype is represented as
  - a string enclosed in guillemets (<< >>), or
  - a graphic elements, such as icon



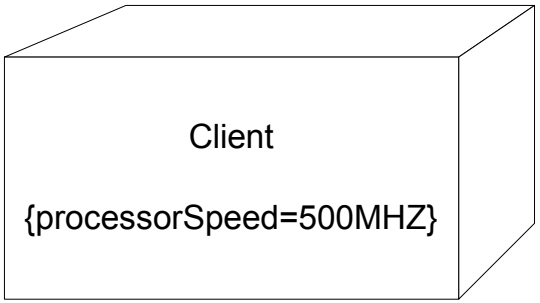
stereotypes



:Processor

Tagged Values

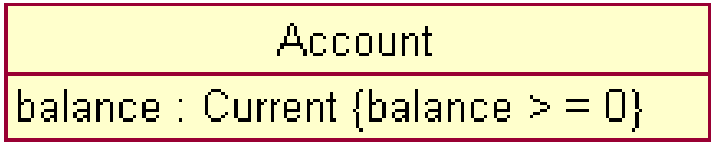
- A tagged value allows you to include new properties for model elements
- A tagged value is a pair of strings -- a tag and a value
  - {name of a property = value of the property}



Tagged Values

Constraints

- A constraint is a condition or restriction that defines additional model semantics
- A constraint may be attached to an individual model element or a collection of elements
- A constraint is written as a string enclosed in braces ({})



Constraints

Stereotypes, Tagged Values, Constraints

- We use stereotypes and tagged values to capture information about the software execution environment
  - e.g., processor type, processor speed, network speed
- We use constraints to specify performance objectives
  - e.g., response time or throughput

Extensions to Sequence Diagram Notation

- Sequence diagram notation is extended to represent
  - hierarchical structure (instance decomposition and references)
  - looping
  - alternation
  - concurrency

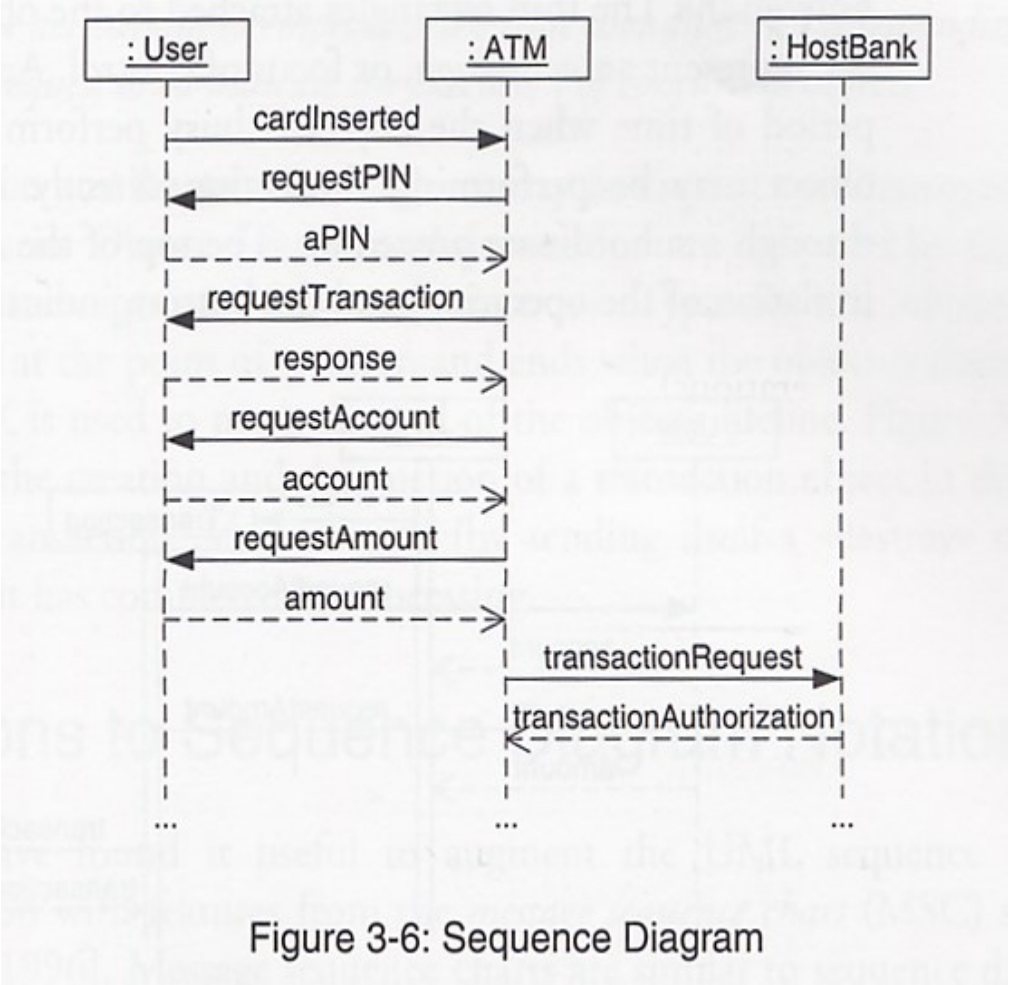


Figure 3-6: Sequence Diagram

Instance Decomposition

- Uses instance decomposition to indicate the refinement of sequence diagrams
- Makes it possible to attach another sequence diagram to an object lifeline
- Allows expansion of a high-level sequence diagram to show lower-level interactions

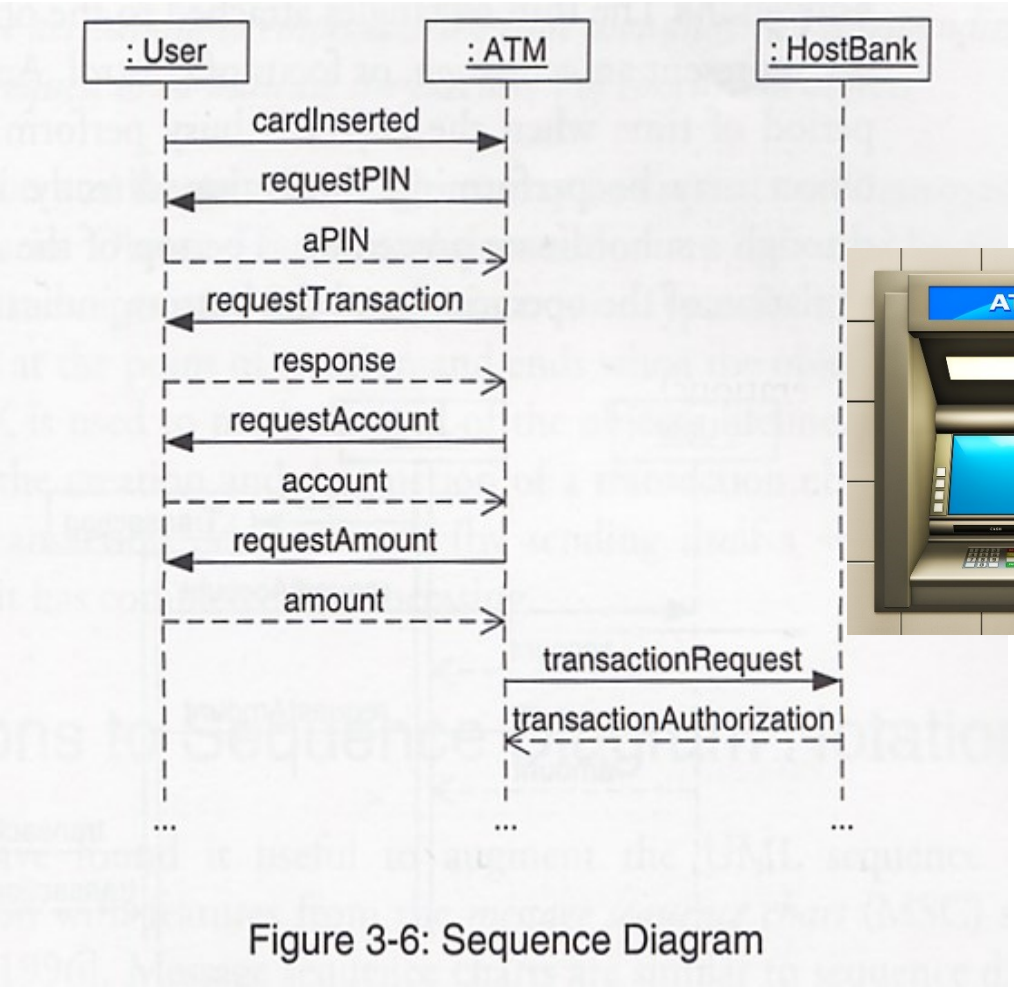


Figure 3-6: Sequence Diagram

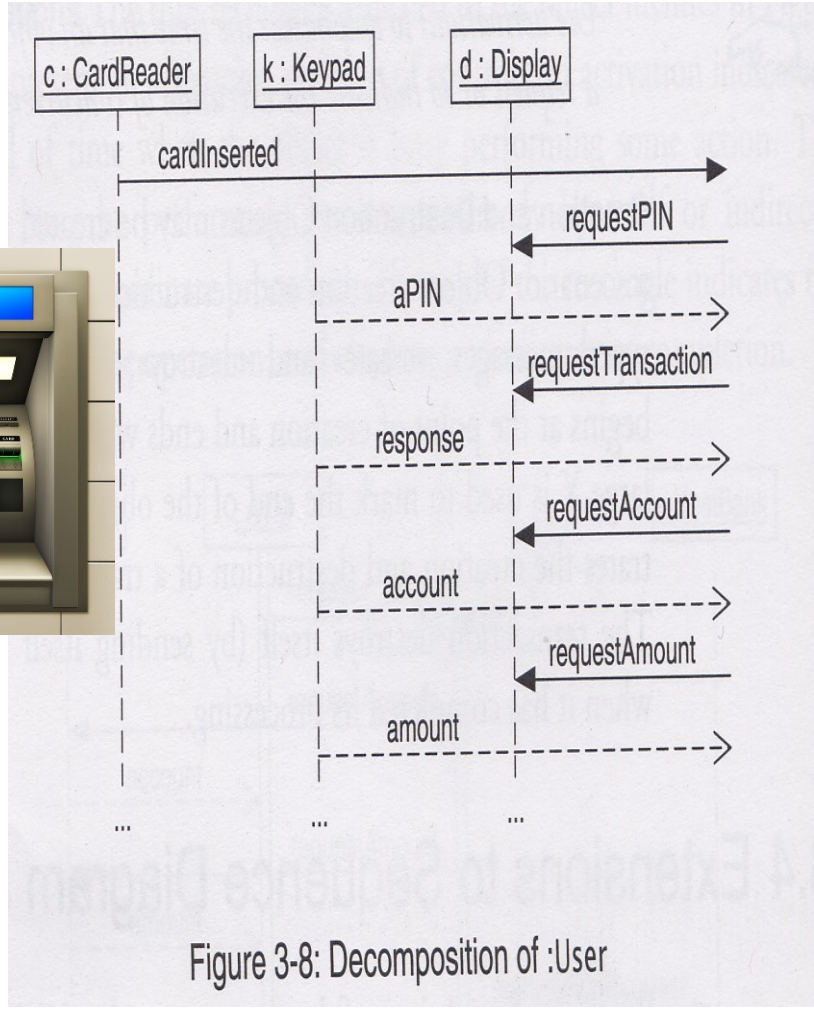


Figure 3-8: Decomposition of :User

ATTENTION!

For the decomposition to be meaningful, ***the order of messages on the decomposed instance must be preserved***

Benefits of Instance Decomposition

- Elaborate the sequence diagram as we learn more about the system, without having to re-draw the diagram each time
- Ensure the consistency with the scenario as it was originally described

Use instance decomposition to elaborate high-level objects as the design evolves

Loop and Alternation

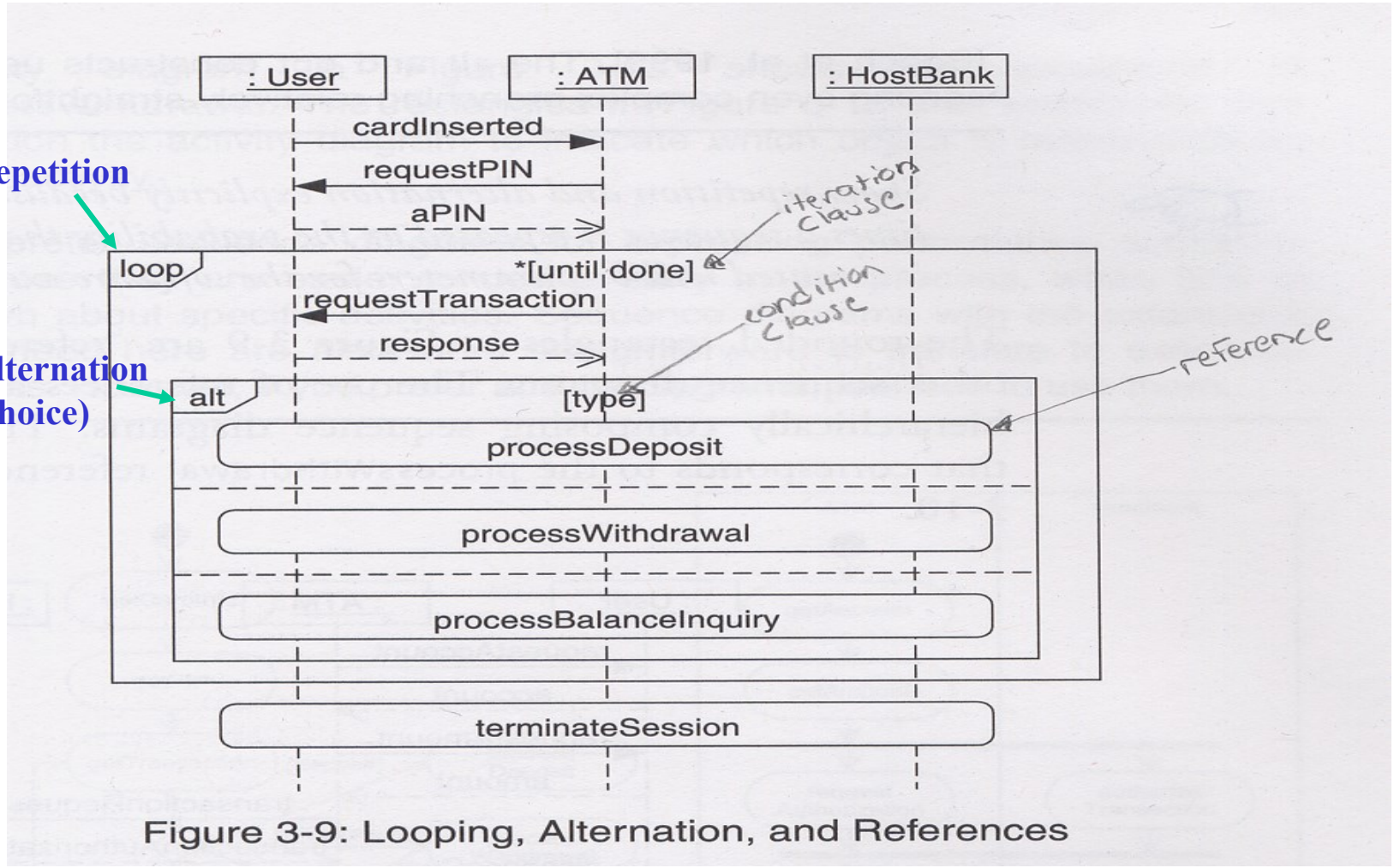


Figure 3-9: Looping, Alternation, and References

References

- ***References*** allows for referring to other sequence diagrams
- Use references to reduce the complexity of sequence diagrams.

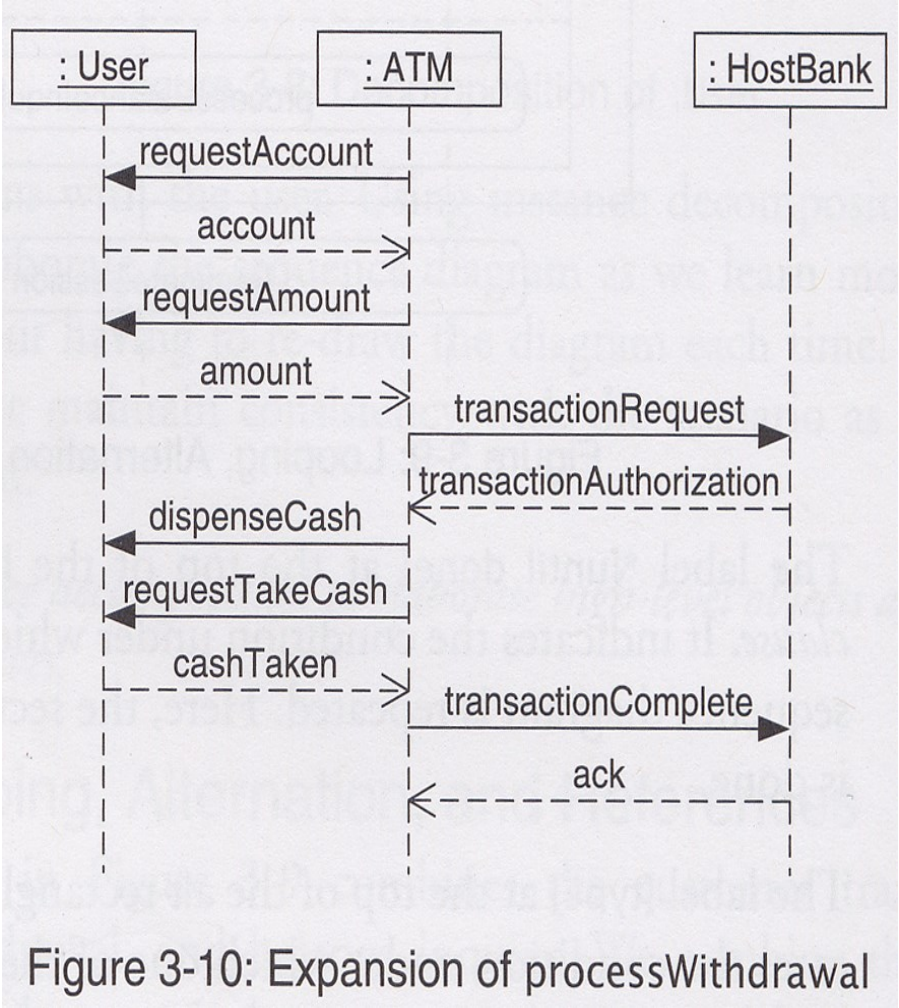
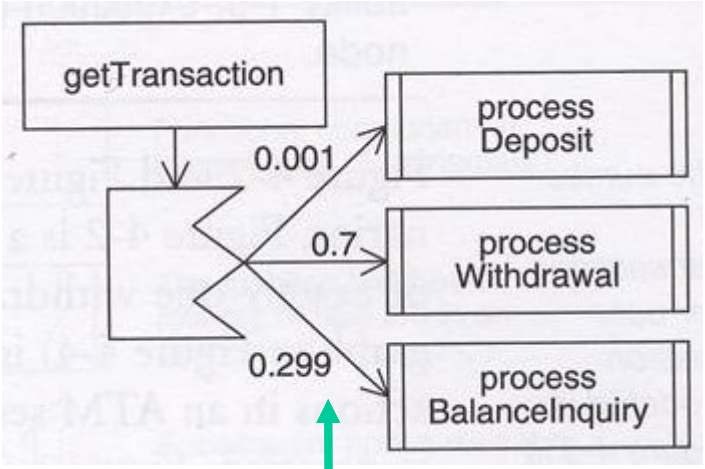


Figure 3-10: Expansion of processWithdrawal

Loop and Alternation (con't)

- ***Loop*** can be used when a sequence is repeated while, loop, etc.
- ***Alternation*** can be used when several possible transitions will be executed If-then, switch-case, etc.
- A probability of execution can be attached to a given sequence



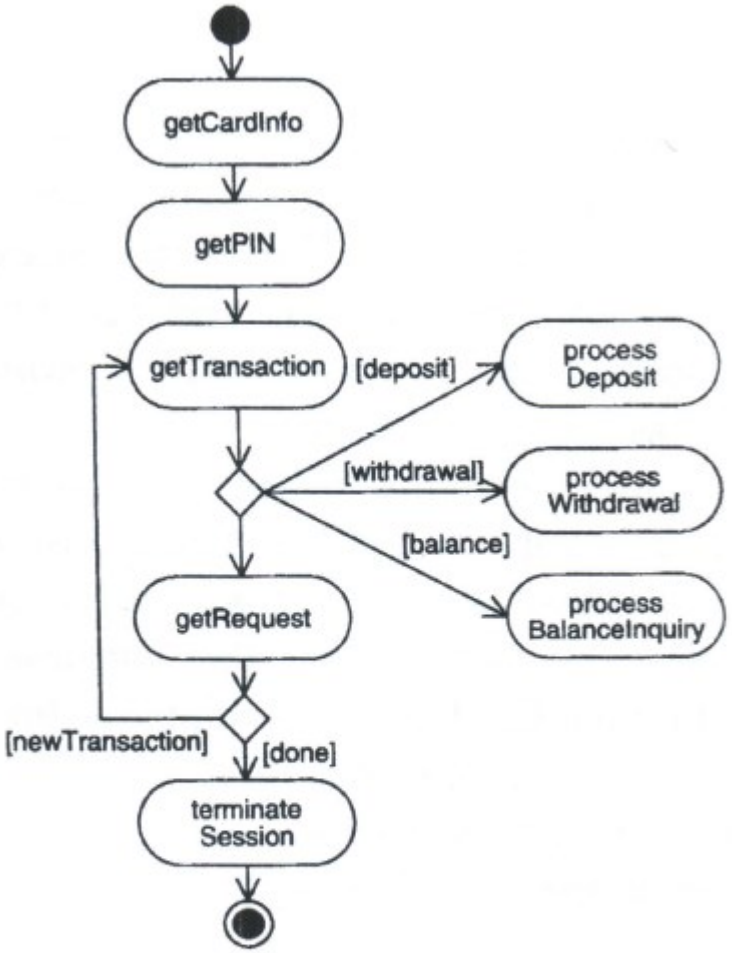


Figure 3-11: Activity Diagram for ATM

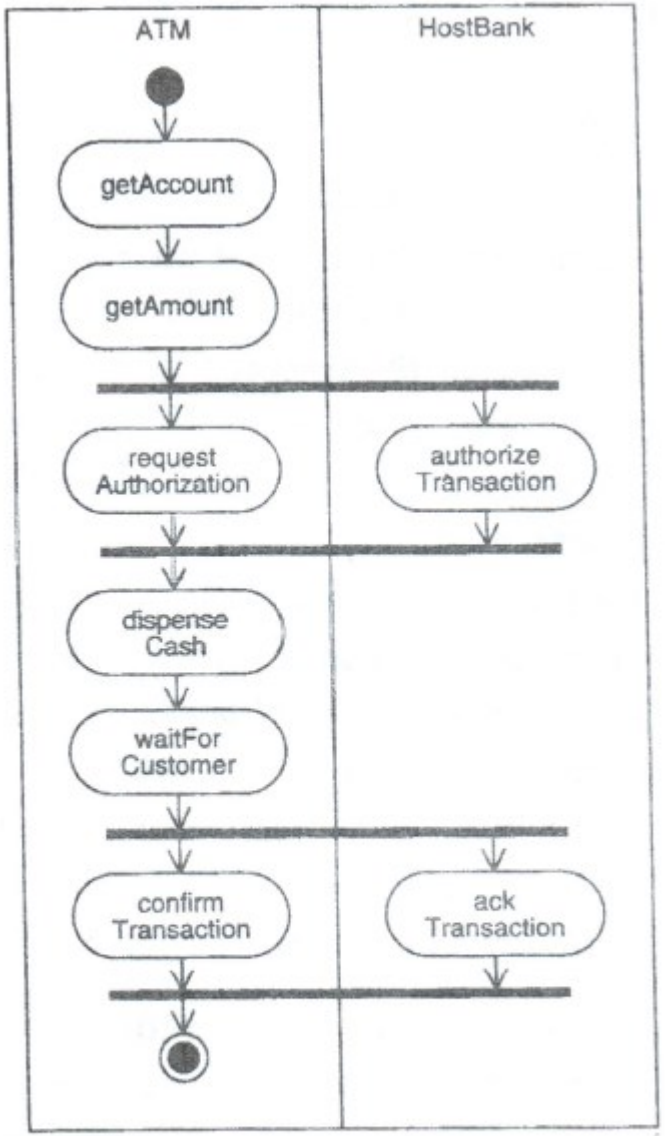
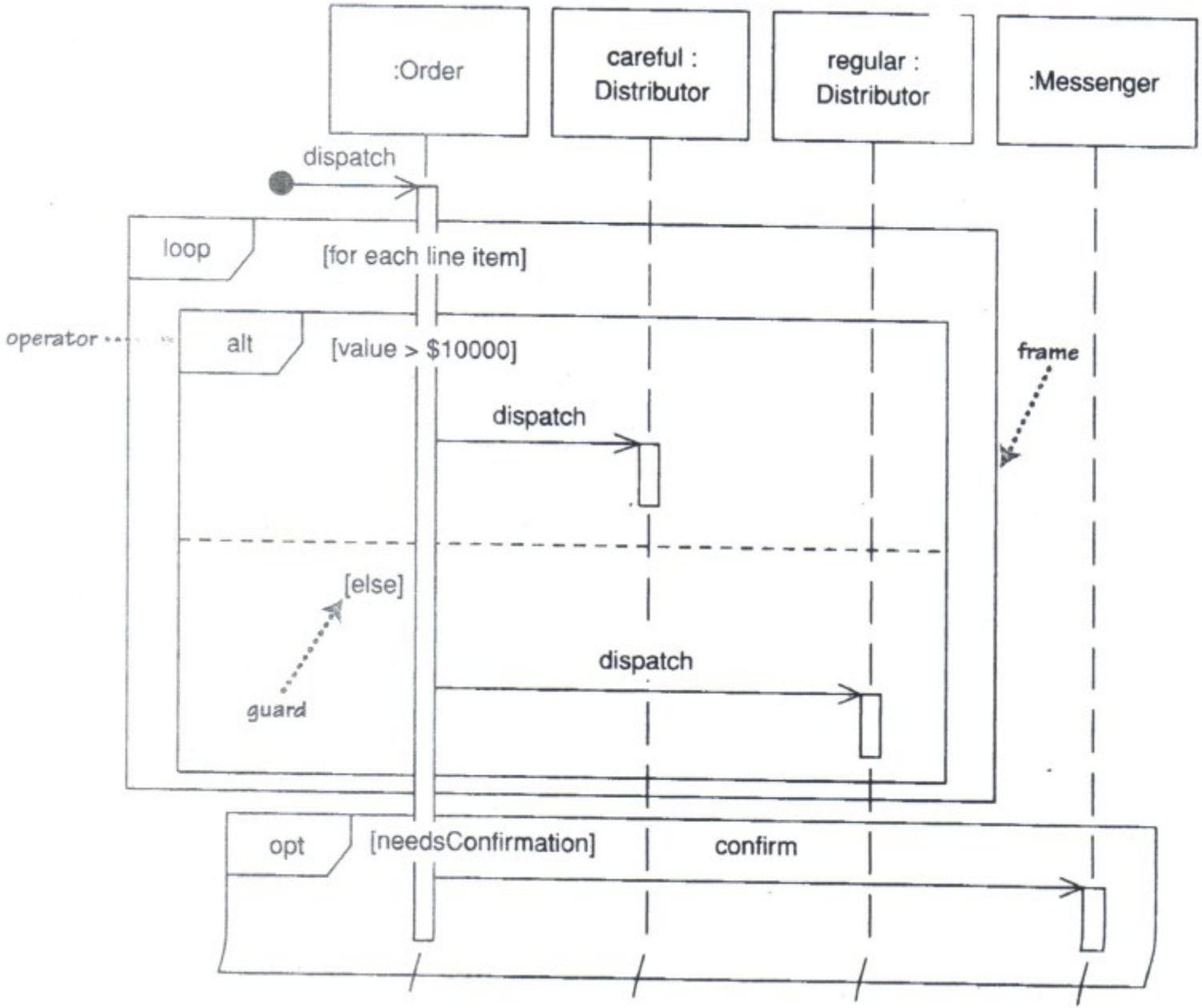


Figure 3-12: Expansion of processWithdrawal

Example

- Once an order is made, a dispatch message is sent.
- The following algorithm describes the dispatch steps:

```
procedure dispatch
    foreach (order.lineitem)
        if (product.value > $10K)
                careful.dispatch
        else
            regular.dispatch
        endif
    endfor
    if (needsConfirmation) meesenger.confirm
end procedure
```


:Order
careful : Distributor
regular : Distributor
:Messenger
dispatch
loop
[for each line item]
operator
alt
[value > $10000]
frame
dispatch
[else]
guard
dispatch
opt
[needsConfirmation]
confirm

Specifying Time

- The UML allows you to specify timing requirements through the use of
  - timing marks
  - time expressions
  - timing constraints

Timing Marks

- Denote the time at which a message or an event occurs
- For example:
  - message.sendTime() -- The time that the message is sent
  - message.recieveTime() -- The time that the message is received
  - where message is the name of the message

Time Expressions

- Evaluate to an absolute or relative value of time
- Express an elapsed time or the occurrence of some particular time
- For example,
  - after(500msec) -- time elapsed after a particular state is entered
  - when(t=08:00) -- the occurrence of an event at the time 08:00

Time Constraints

- Express a constraint based on the absolute or relative value of time
- For example
  - {a.recieveTime() - b.sendTime() < 10 msec}

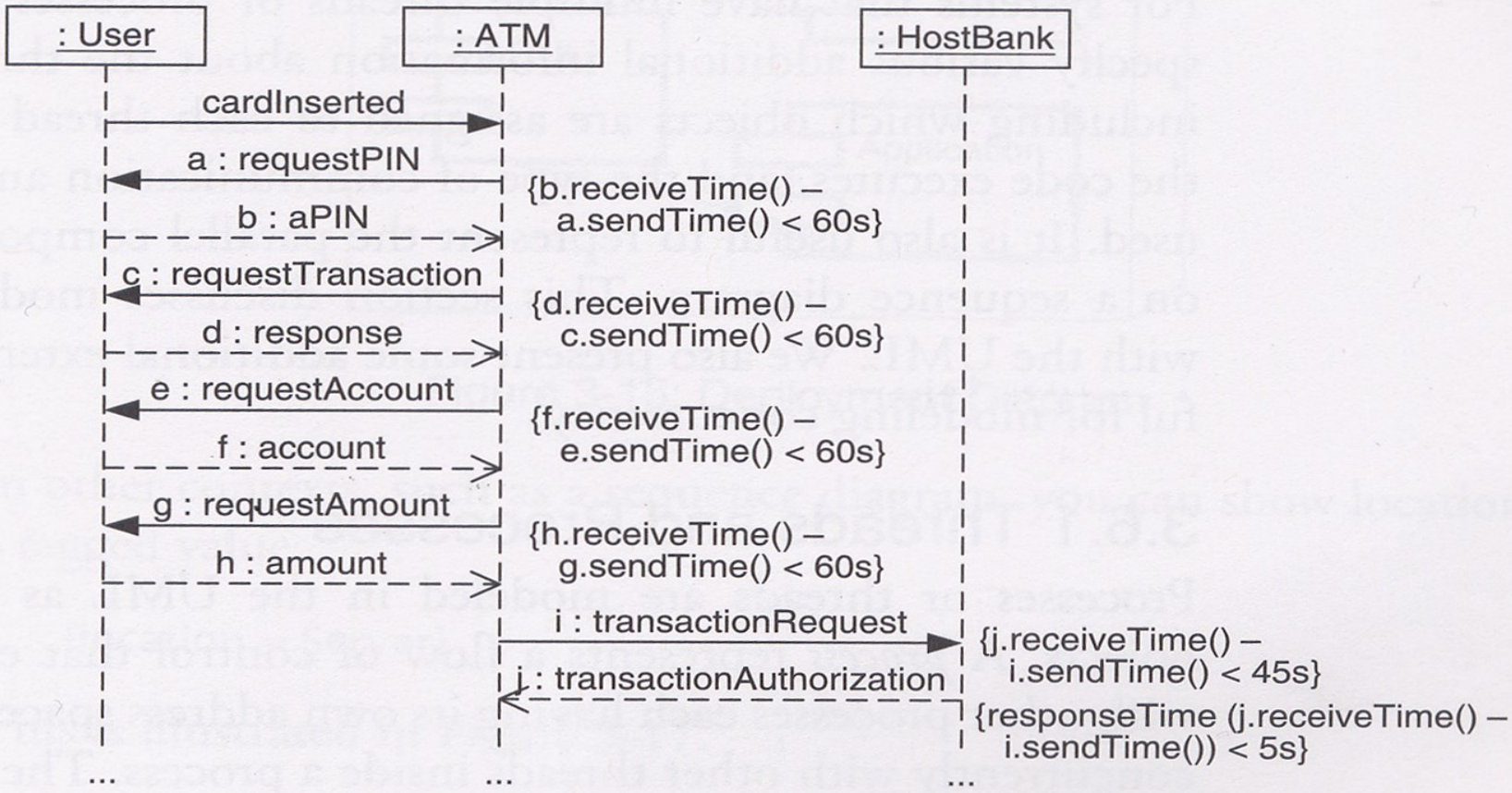


Figure 3-13: Time in Sequence Diagrams

Time Constraints (con't)

- Timeout conditions are not particularly useful from a performance perspective
- When specifying performance, we are more interested in response time
  - responseTime(j.receiveTime() – i.sendTime())
  - {responseTime(j.receiveTime() – i.sendTime()) < 5s} – a time constraint example

Use time expressions that are meaningful from a performance perspective, such as responseTime(), to specify performance objectives

Concurrency

- Modeling concurrency is important in the later stages of SPE for evaluating contention effects
- Concurrency issues are expressed by UML notations
  - Threads and Processes
  - Coregions
  - Parallel Composition
  - Synchronization

Threads and Processes

- A process represents a flow of control that executes in parallel with other processes
  - Each process has its own address space
  - represented by a standard stereotype ***<<process>>***
- A thread executes concurrently with other threads inside a process
  - all threads belonging to a process all share the same address space
  - represented by a standard stereotype ***<<thread>>***

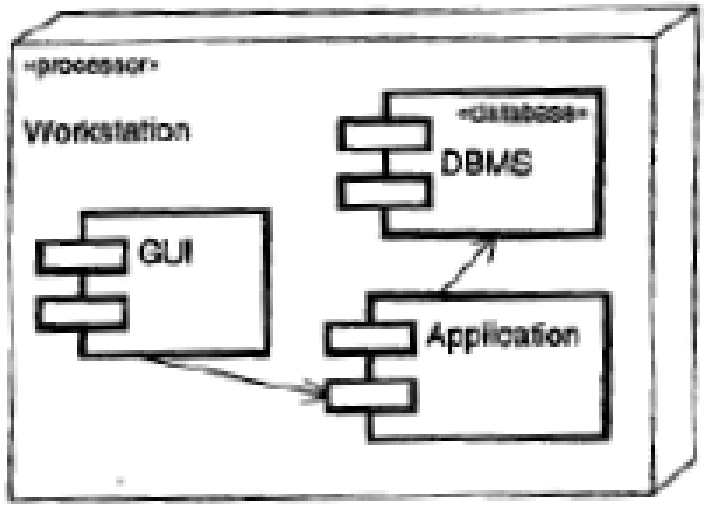


Figure 3-15: Deployment Diagram

Coregions

- A sequence diagram are strictly ordered in time
- ***Coregions*** allow an exception to total ordering whereby messages within a coregion are ***unordered***
- Coregions allow you to show the interleaving of messages that occur in parallel processing

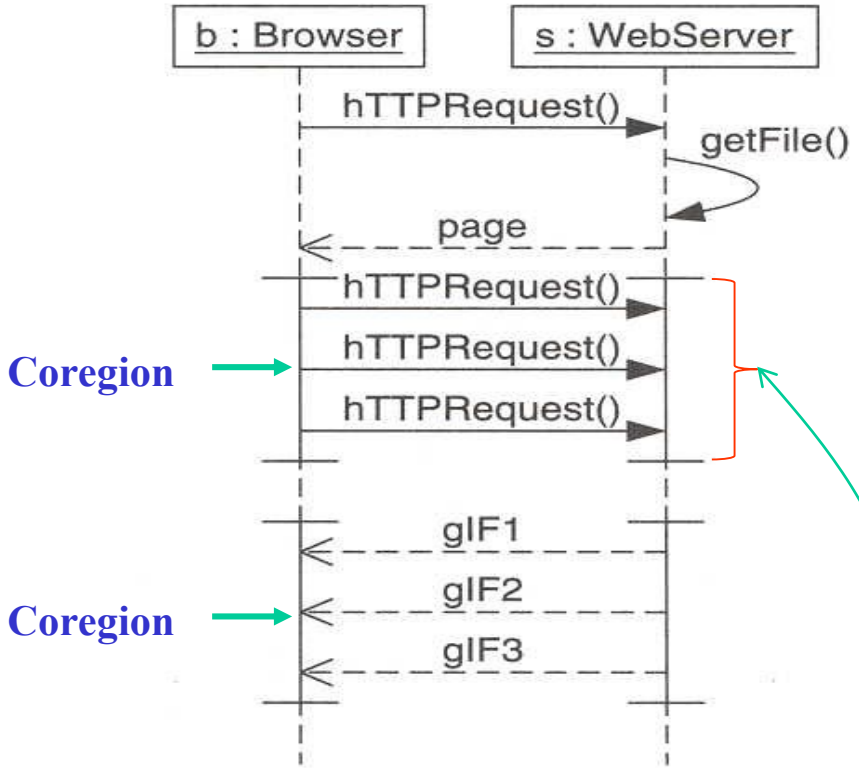


Figure 3-16: Coregions

{represented by a solid line with horizontal dividers}

Parallel Composition

- Indicates sections of the sequence diagram that are executed in parallel
- Shows the interleaving of messages that occur in parallel processing
- Allows more flexible representation of parallel processing

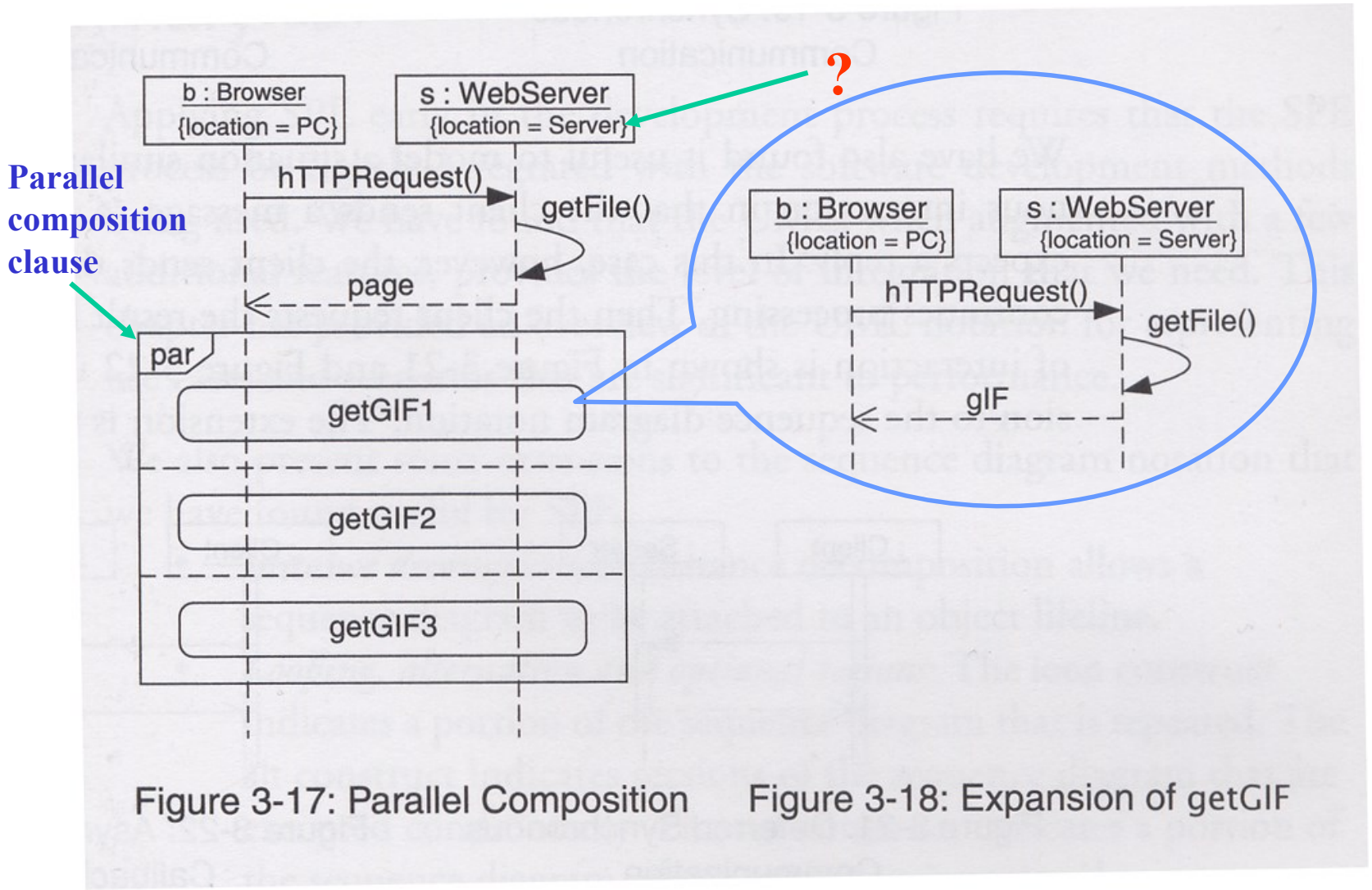


Figure 3-17: Parallel Composition

Figure 3-18: Expansion of getGIF

Synchronization

- UML provides different types of arrowheads to represent communications among objects

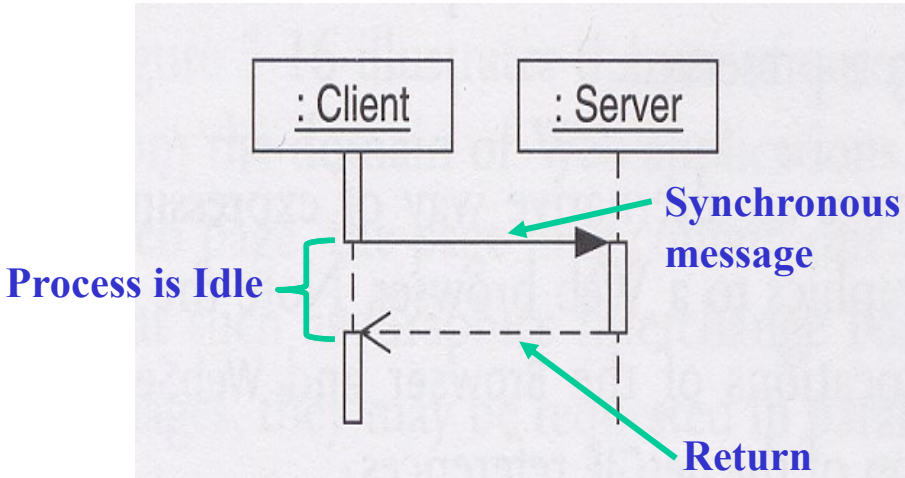


Figure 3-19: Synchronous Communication

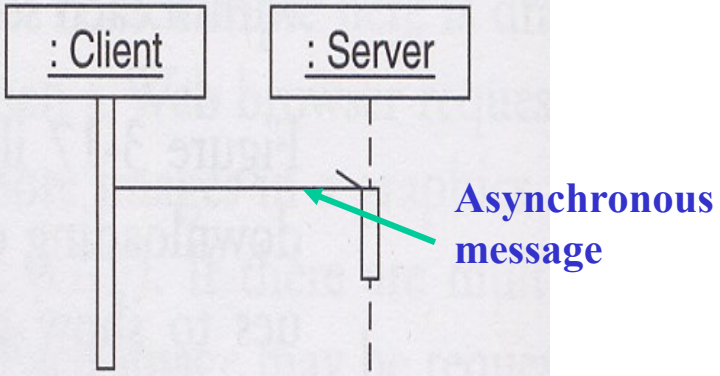


Figure 3-20: Asynchronous Communication

Synchronization (con't)

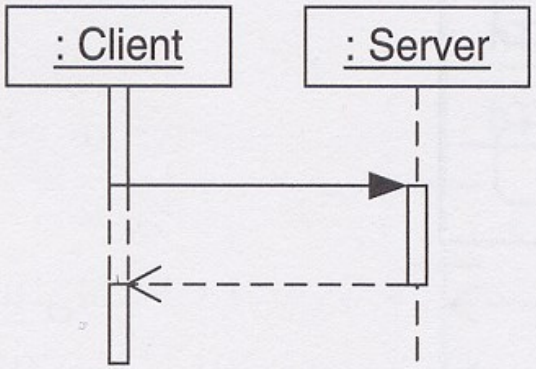


Figure 3-21: Deferred Synchronous Communication

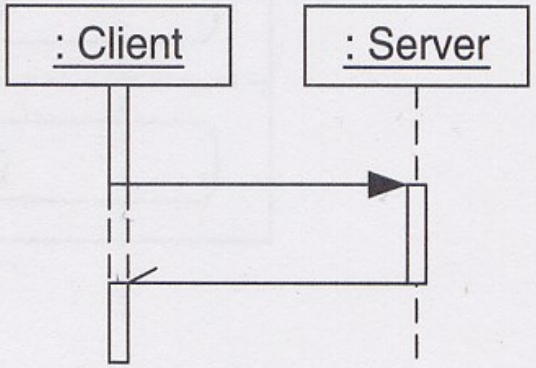


Figure 3-22: Asynchronous Callback

Contention Effects

- Modeling concurrency is important in the later stages of SPE for evaluating contention effects
- Early stages of the development process focus on software model without contention
- Concurrency and synchronization properties of the proposed software are considered later when your knowledge of the software system increases

References

- Lecture notes for CS399 by Bob Dugan at stonehill university