

Unit 3: Advanced State Diagrams

Syllabus

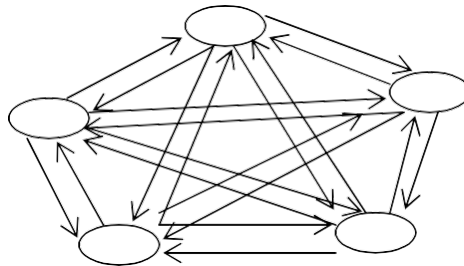
-----7hr

- **Nested state diagram**
- **Nested states**
- **Signal generalization**
- **Concurrency**
- **A sample state mode**
- **Relation of class and state models**
- **Relation of class and state models**
- **Use case models**
- **Sequence models**
- **Activity models**

Problem with flat state diagrams

- Flat unstructured state diagram are impractical for large problems, because – representing an object with n independent Boolean attribute requires 2^n states. By partitioning the state into n independent state diagram requires $2n$ states only.

● Eg:



Above figure requires n^2 transition to connect every state to other state. This can be reduced to as low as n by using sub diagrams structure.

Expanding states

- One way to organize a model is by having high level diagram with sub diagrams expanding certain state. This is like a macro substitution in programming language
- A submachine is a state diagram that may be invoked as part of another state diagram

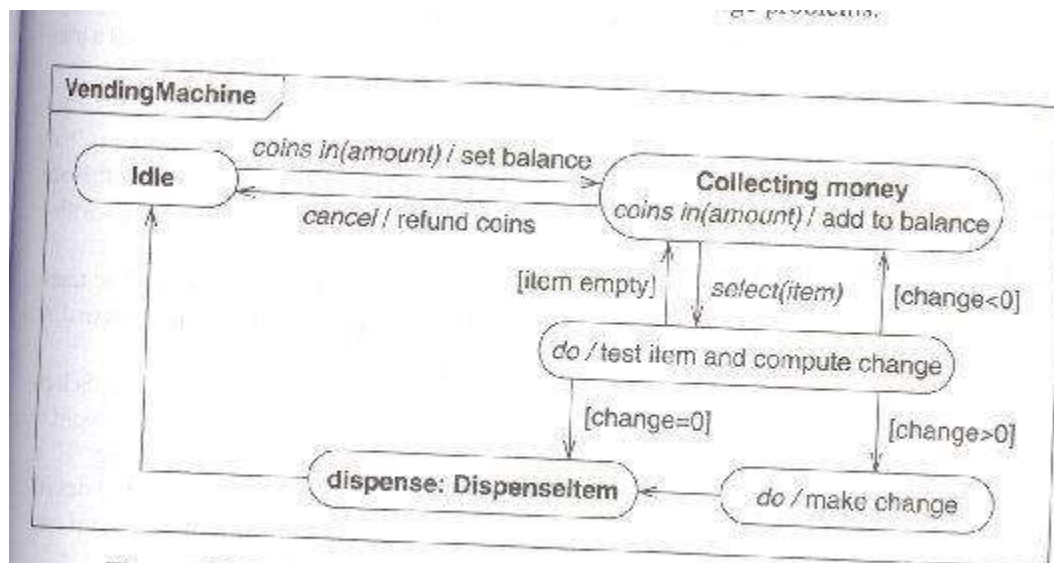


Figure 6.2 Vending machine state diagram. You can simplify state diagrams by using subdiagrams.

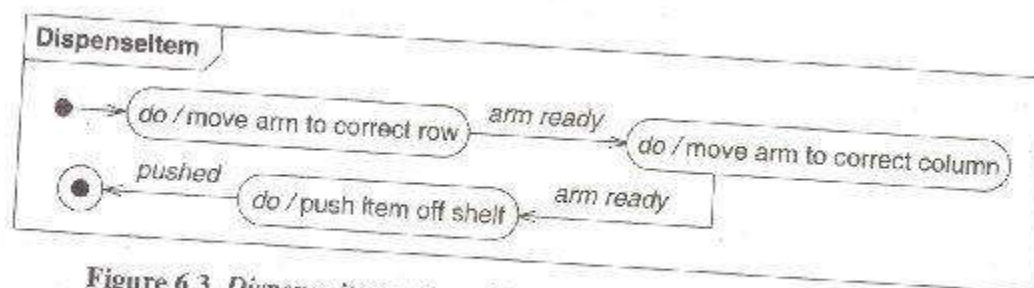


Figure 6.3 Dispense item submachine of vending machine. A lower-level state diagram can elaborate a state.

6.2 Nested States

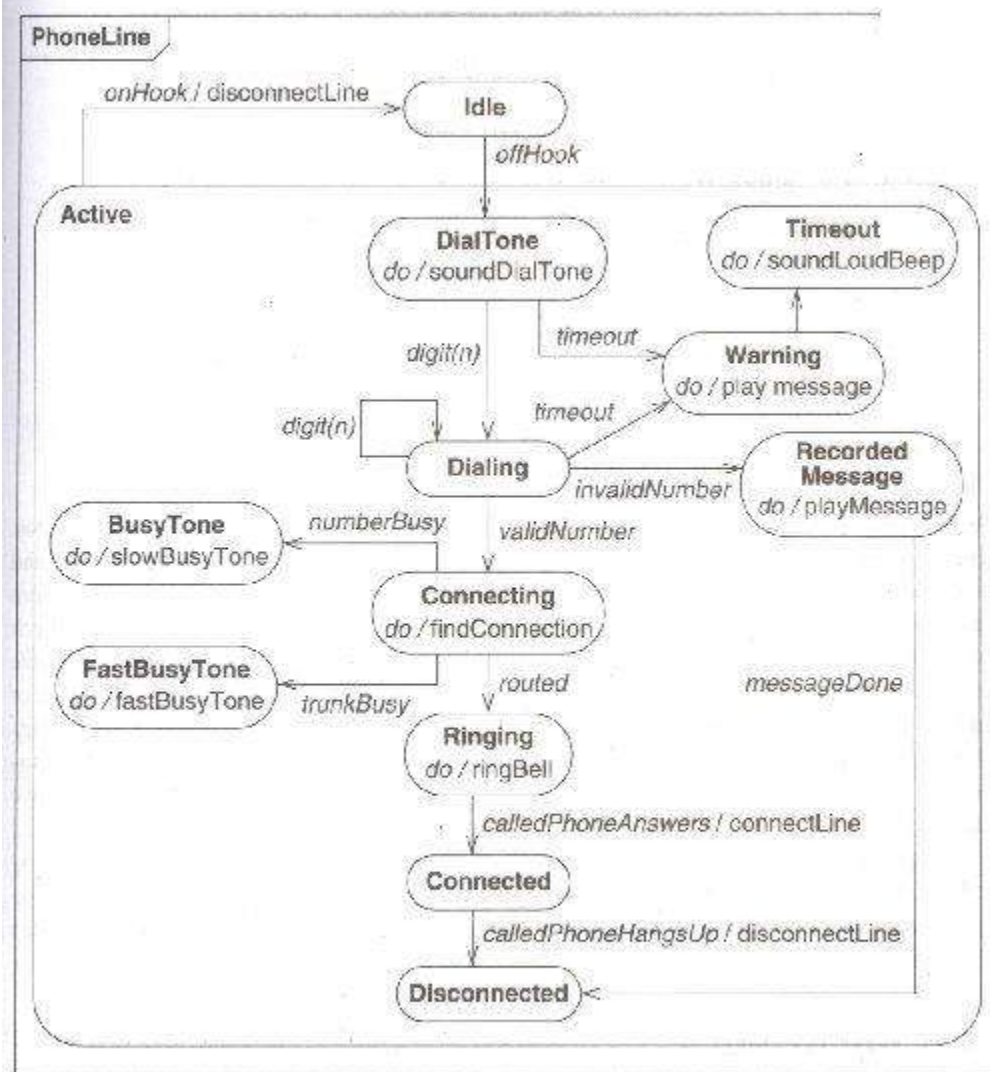


Figure 6.4 Nested states for a phone line. A nested state receives the outgoing transitions of its enclosing state.

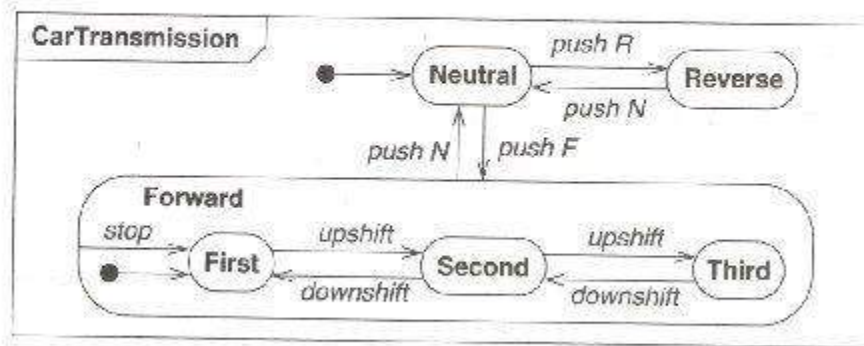


Figure 6.5 Nested states. You can nest states to an arbitrary depth.

Signal generalization

You can organize signals into generalization hierarchy with inheritance of signal attributes

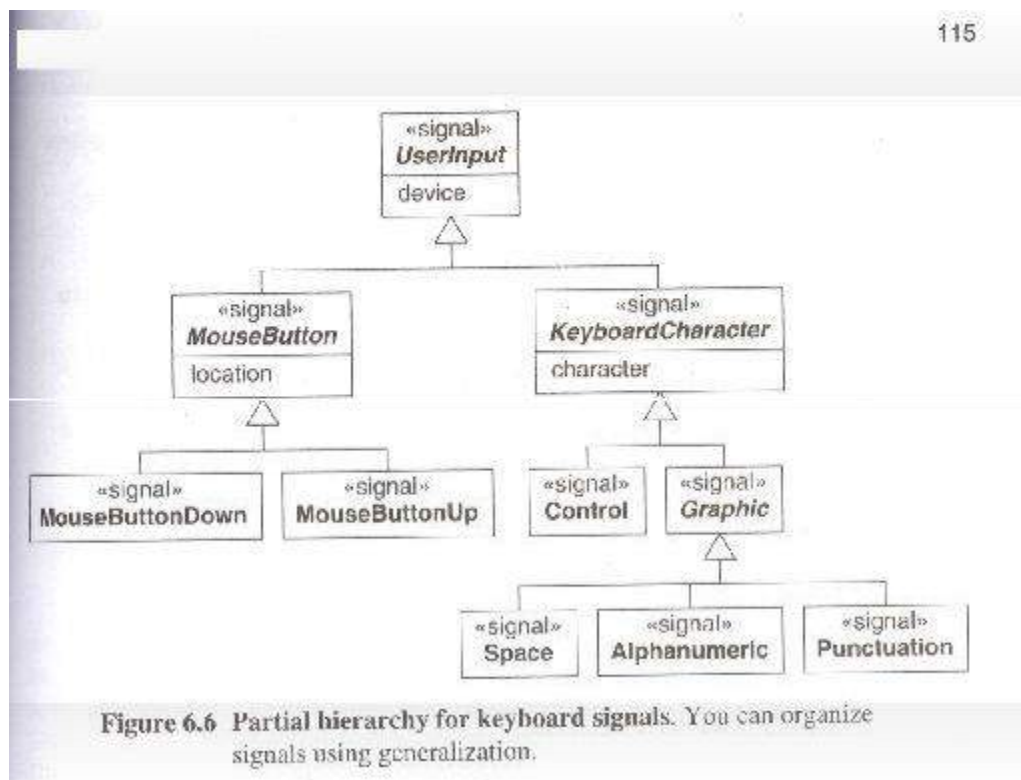
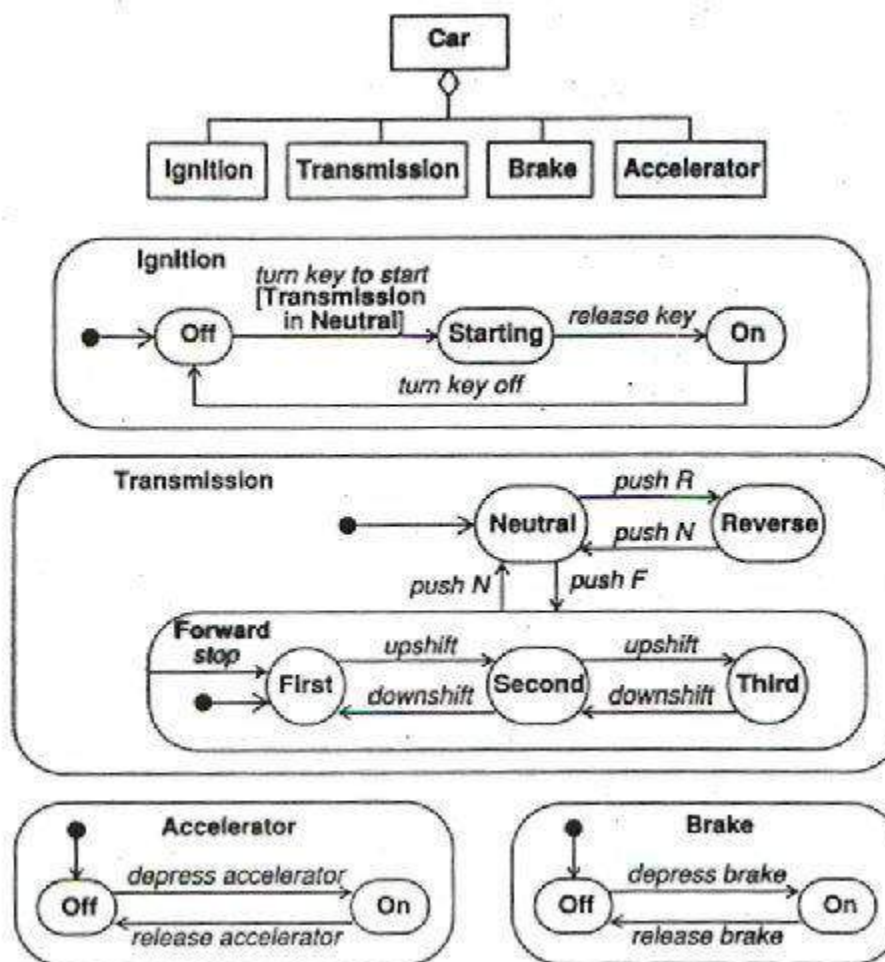


Figure 6.6 Partial hierarchy for keyboard signals. You can organize signals using generalization.

- Ultimately, we can view every actual signal as a leaf on a generalization tree of signals

- In a state diagram, a received signal triggers transitions that are defined for any ancestor signal type.
- For eg: typing an 'a' would trigger a transition on a signal alphanumeric as well as key board character.
- **Concurrency 1:**
- The state model implicitly supports concurrency among objects.
- In general, objects are autonomous entities that can act and change state independent of one another. However objects need not be completely independent and may be subject to shared constraints that cause some correspondence among their state changes.

1 Aggregation concurrency



2 concurrency within an object

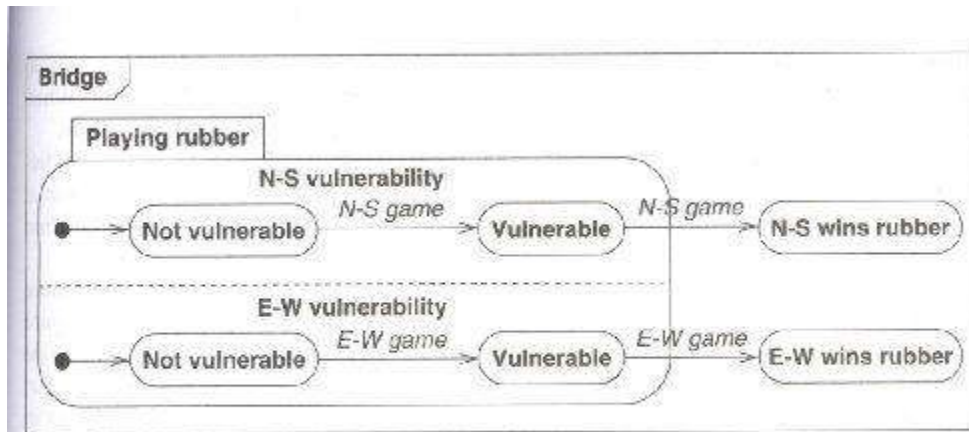


Figure 6.8 Bridge game with concurrent states. You can partition some objects into subsets of attributes or links, each of which has its own subdiagram.

3 synchronization of concurrent activities

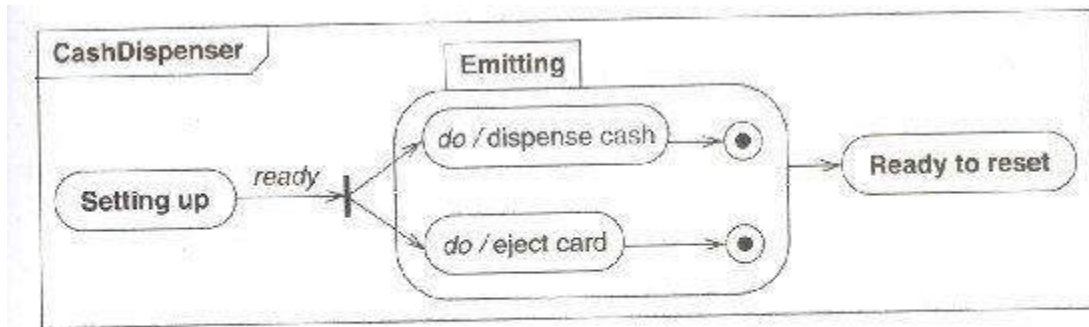


Figure 6.9 Synchronization of control. Control can split into concurrent activities that subsequently merge.

Interaction Models

- The class model describes the objects in a system and their relationship.
- The state model describes the life cycles of the objects.
- The interaction model describes how the objects interact.

The **interaction model** starts with **use cases** that are then elaborated with **sequence** and **activity diagrams**

- **Use case:** focuses on functionality of a system- i.e, what a system does for users
- **Sequence diagrams:** shows the object that interact and the time sequence of their interactions
- **Activity diagrams:** elaborates important processing steps

Use Case models**Actors**

- A direct external user of a system
- Not part of the system
- For example
 - Traveler, agent, and airline for a travel agency system. Can be
 - a person, devices and other system
 - An actor has a single well-defined purpose

Use Cases

- A use case is a coherent piece of functionality that a system can provide by interacting with actors.
- For example:
 - A *customer* actor can *buy a beverage* from a vending machine.
 - A *repair technician* can *perform scheduled maintenance* on a vending machine.
- Each use case involves one or more actors as well as the system itself.

A Vending Machine

- **Buy a beverage.** The vending machine delivers a beverage after a customer selects and pays for it.
- **Perform scheduled maintenance.** A repair technician performs the periodic service on the vending machine necessary to keep it in good working condition.
- **Make repairs.** A repair technician performs the unexpected service on the vending machine necessary to repair a problem in its operation.
- **Load items.** A stock clerk adds items into the vending machine to replenish its stock of beverages.

Figure 7.1 Use case summaries for a vending machine. A use case is a coherent piece of functionality that a system can provide by interacting with actors.

Object Oriented Modeling and Design with UML, Second Edition by Michael Blaha and James Pumbaugh, ISBN 0-13-1-35823-4, © 2005 Pearson Education, Inc., Upper Saddle River, NJ. All rights reserved.

- A use case involves a sequence of messages among the system and its actors.
- Error conditions are also part of a use case.
- A use case brings together all of the behavior relevant to a slice of system functionality.

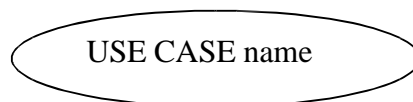
Use Case Description (see text book fug 7.2)

- Use Case Name
- Summary
- Actors
- Preconditions
- Description
- Exception
- Postcondition

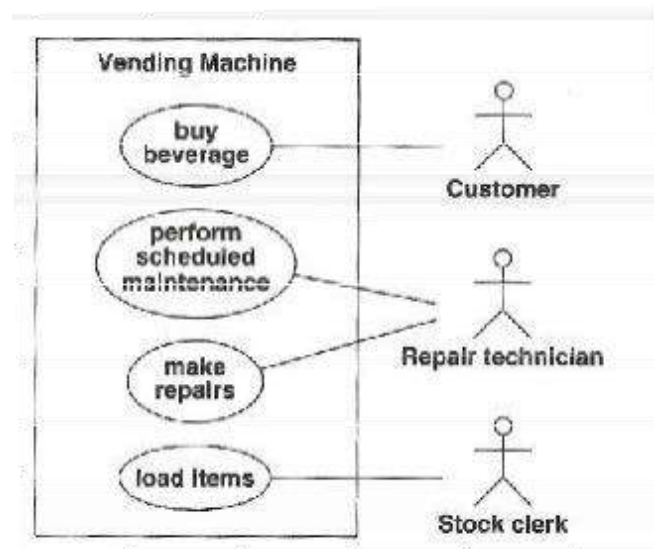
● Actor



● Use Case



A Vending Machine



Guidelines for Use Case

- First determine the system boundary
- Ensure that actors are focused
- Each use case must provide value to users
- Relate use cases and actors
- Remember that use cases are informal
- Use cases can be structured

Use Case Relationships

- Include Relationship
 - Incorporate one use case within the behavior sequence of another use case.
- Extend Relationship
 - Add incremental behavior to a use case.
- Generalization
 - Show specific variations on a general use case.

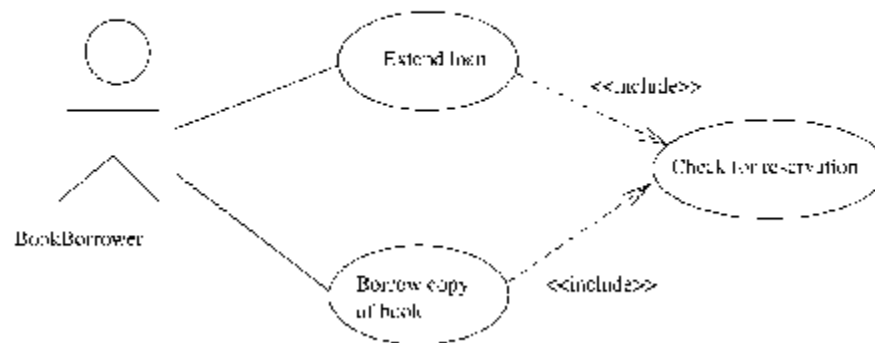
Use case Relationships



Examples:

<<include>> for common behavior

(1)



(2)

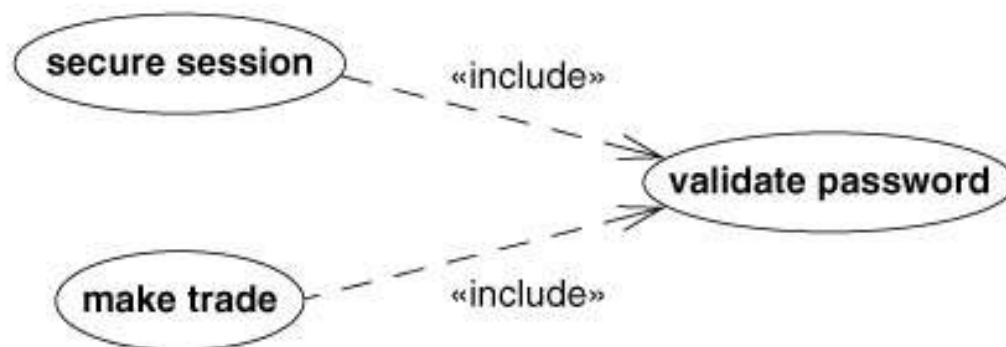
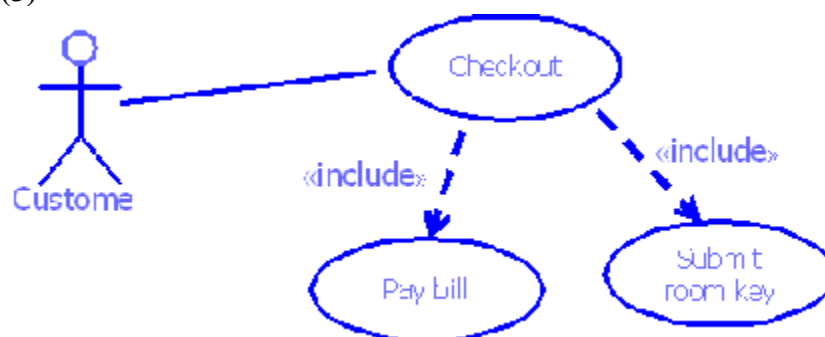


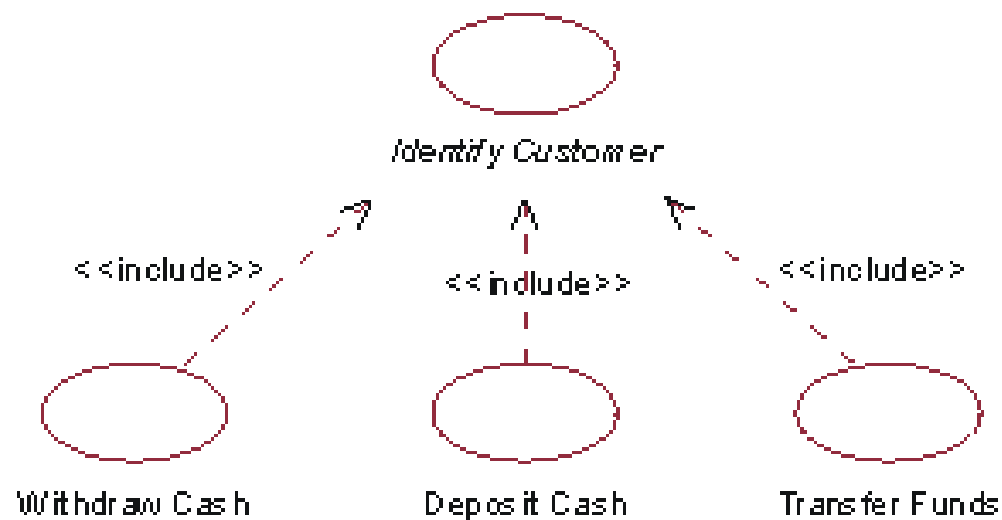
Figure 8.1 Use case inclusion. The *include* relationship lets a base use case incorporate behavior from another use case.

Object-Oriented Modeling and Design with UML, Second Edition by Michael Blaha and James Rumbaugh. ISBN 0-13-1-015920-4. © 2005 Pearson Education, Inc., Upper Saddle River, NJ. All rights reserved.

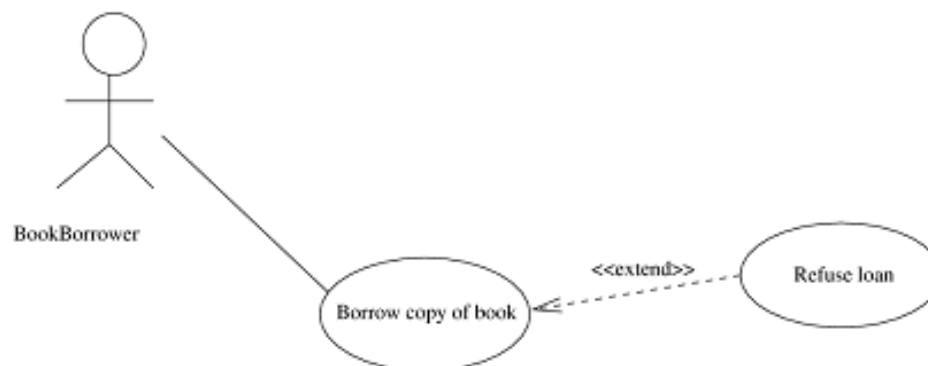
(3)



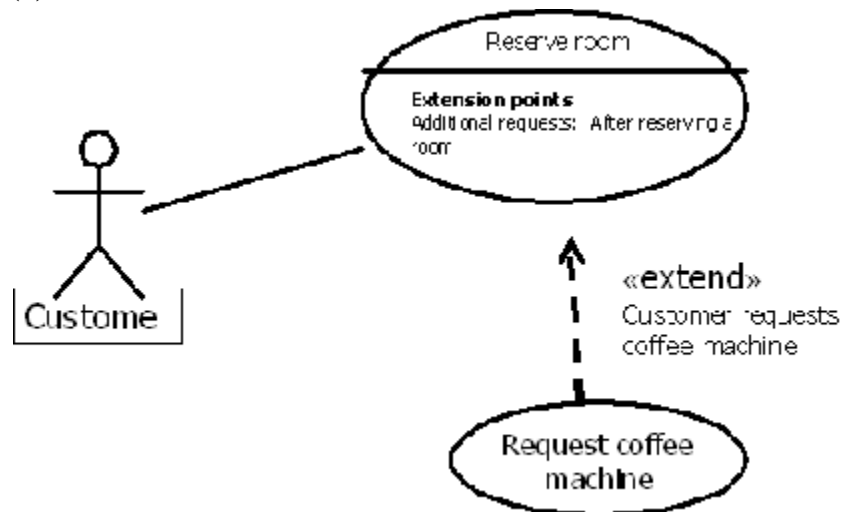
(4)

**Extend Relationship examples:**`<<extend>>` for special cases:

(1)



(2)



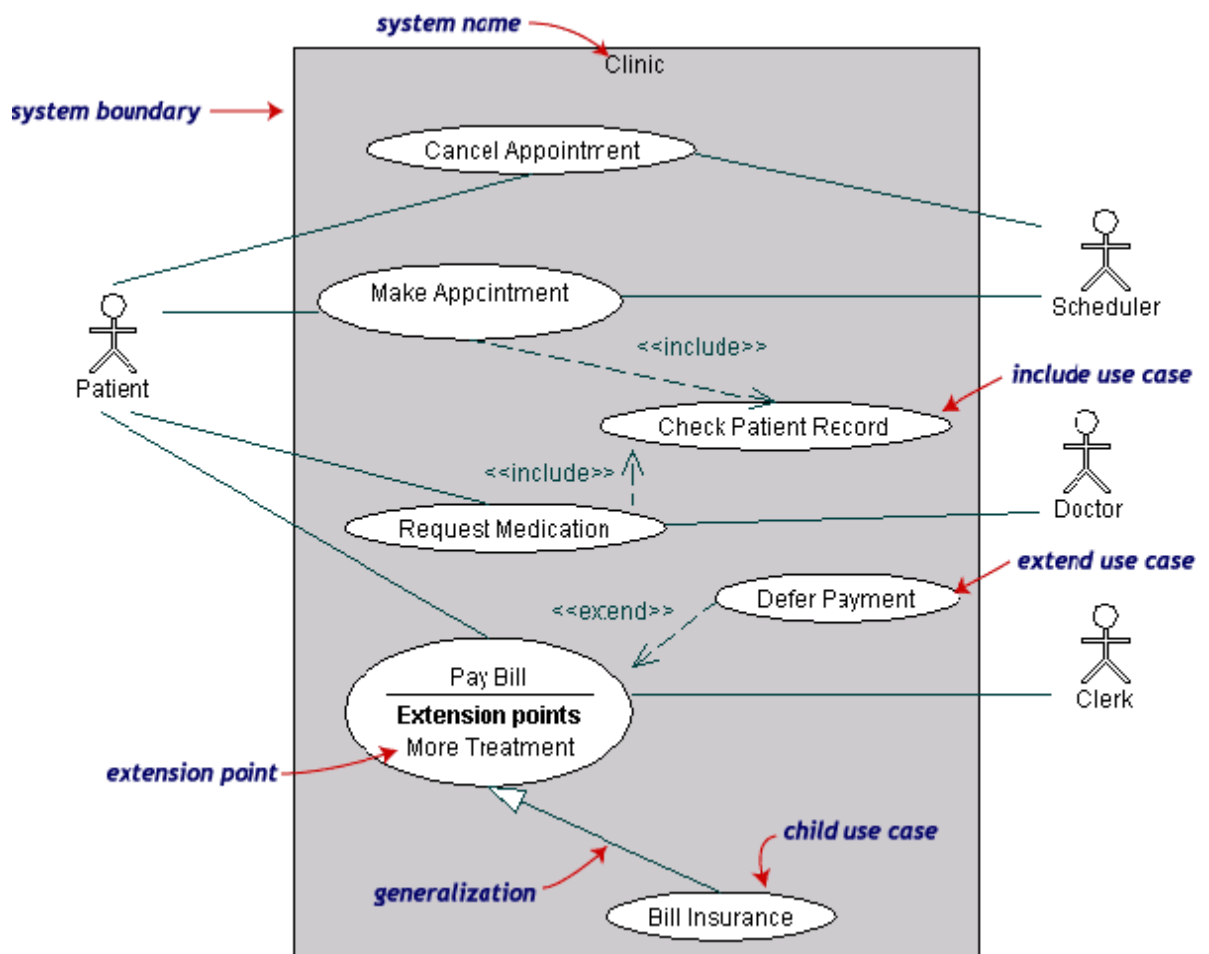
(3)



Figure 8.2 Use case extension. The *extend* relationship is like an *include* relationship looked at from the opposite direction. The extension adds itself to the base.

Object-Oriented Modeling and Design with UML, Second Edition by Michael Blaha and James Rumbaugh. ISBN 0-13-1-015920-4. © 2005 Pearson Education, Inc., Upper Saddle River, NJ. All rights reserved.

Medical Clinic: «include» and «extend»

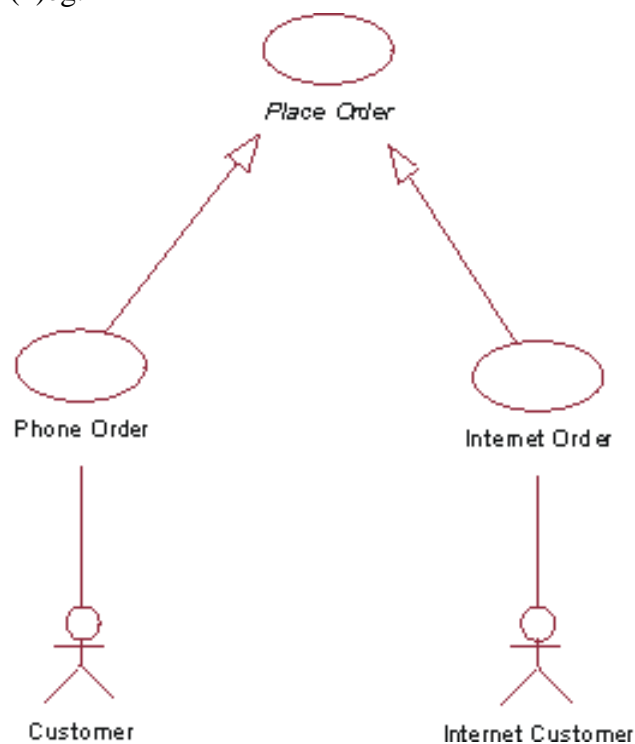


Generalization



Figure 8.3 Use case generalization. A parent use case has common behavior and child use cases add variations, analogous to generalization among classes.

(2)eg:



Use Case Relationships

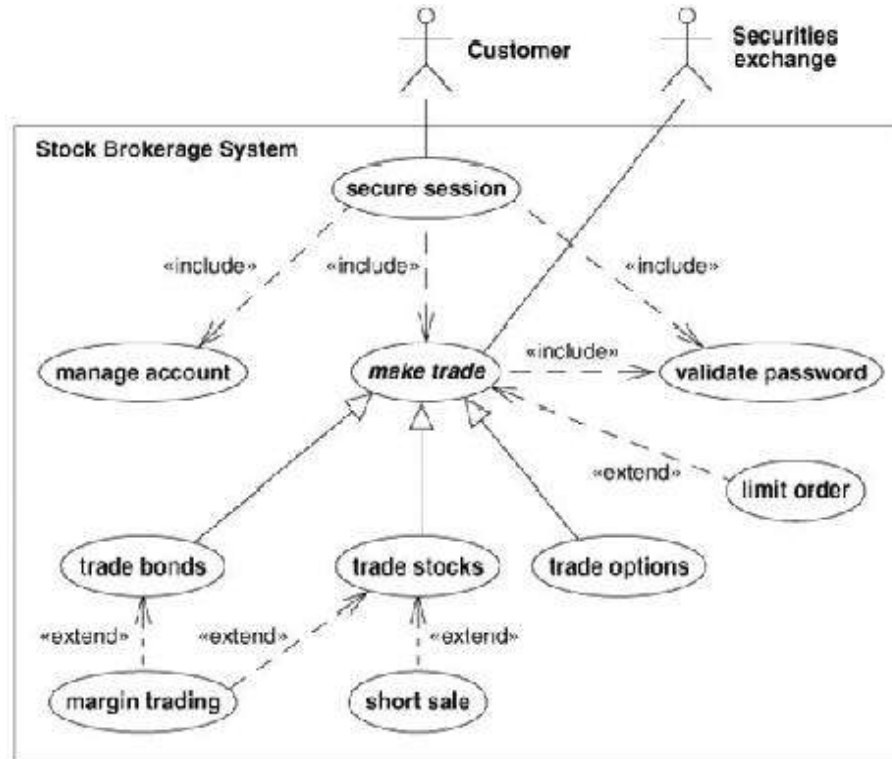


Figure 5.4 Use case relationships. A single use case diagram may combine several kinds of relationships.


Sequence Models

- The sequence model elaborates the themes of use cases.
- Two kinds of sequence models
 - Scenarios
 - Sequence diagrams

Scenarios

- A scenario is a sequence of events that occurs during one particular execution of a system.
- For example:
 - *John Doe logs in* transmits a message from John Doe to the broker system.

Scenario for a stock broker



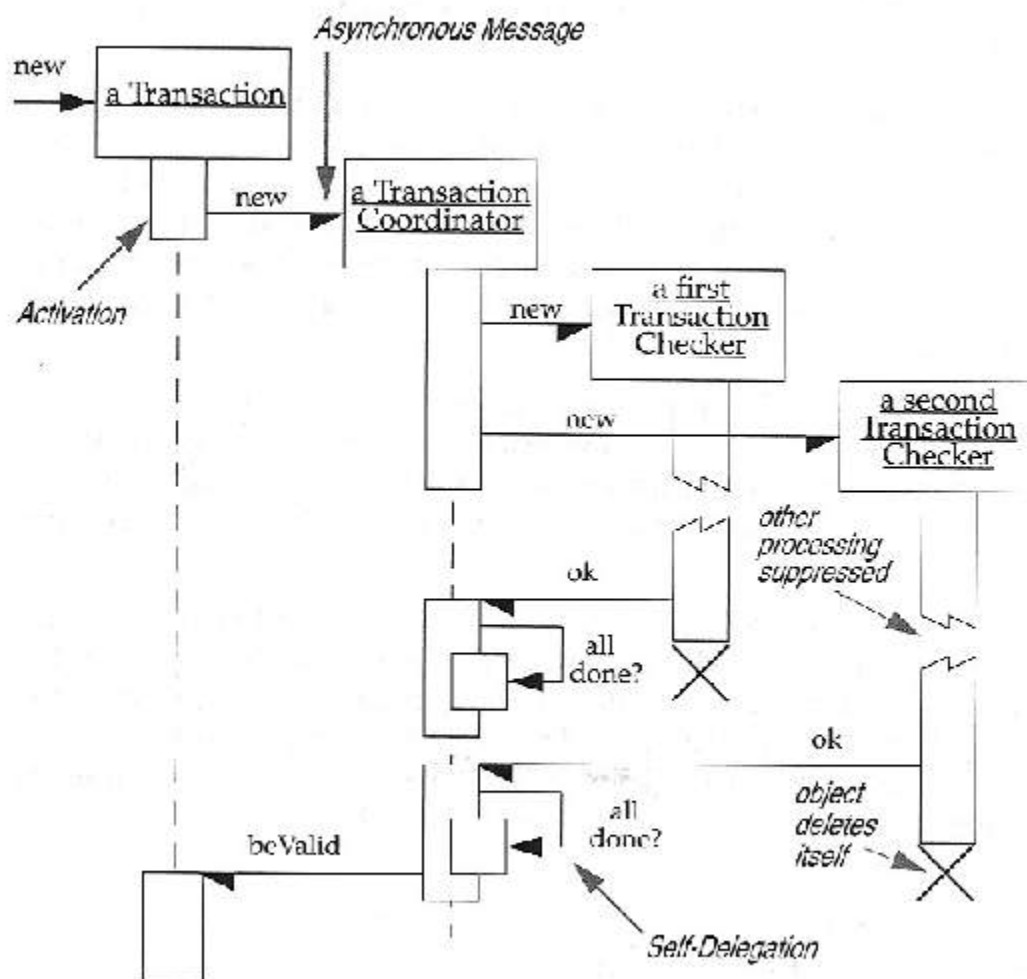
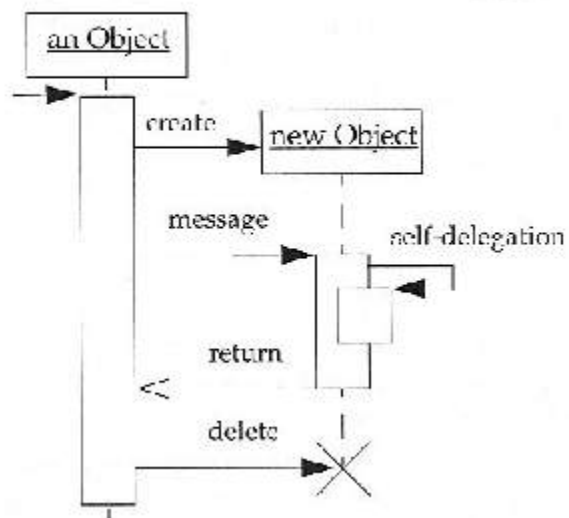
John Doe logs in.
System establishes secure communications.
System displays portfolio information.
John Doe enters a buy order for 100 shares of GE at the market price.
System verifies sufficient funds for purchase.
System displays confirmation screen with estimated cost.
John Doe confirms purchase.
System places order on securities exchange.
System displays transaction tracking number.
John Doe logs out.
System establishes insecure communication.
System displays good-bye screen.
Securities exchange reports results of trade.

Figure 7.4 Scenario for a session with an online stock broker. A scenario is a sequence of events that occurs during one particular execution of a system.

Sequence Diagram

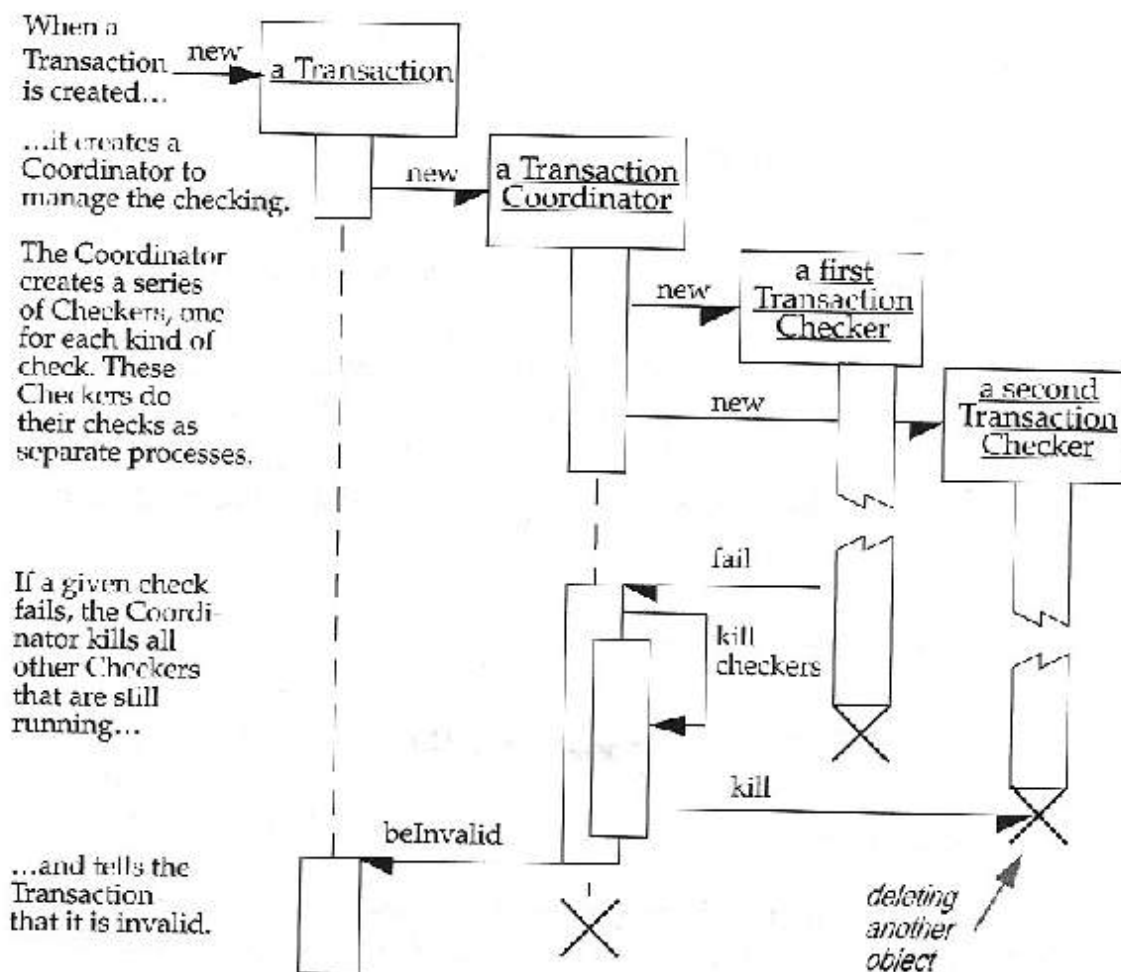
- A sequence diagram shows the participants in an interaction and the sequence of messages among them.
- A sequence diagram shows the interaction of a system with its actors to perform all or part of a use case.
- Each use case requires one or more sequence diagrams to describe its behavior.

Sequence Diagram



Concurrent Processes

- Activations - show when a method is active – either executing or waiting for a subroutine to return
- Asynchronous Message – (half arrow) a message which does not block the caller, allowing the caller to carry on with its own processing; asynchronous messages can:
 - Create a new thread
 - Create a new object
 - Communicate with a thread that is already running
- Deletion – an object deletes itself
- Synchronous Message – (full arrow) a message that blocks the caller



Sequence Diagram For a Session

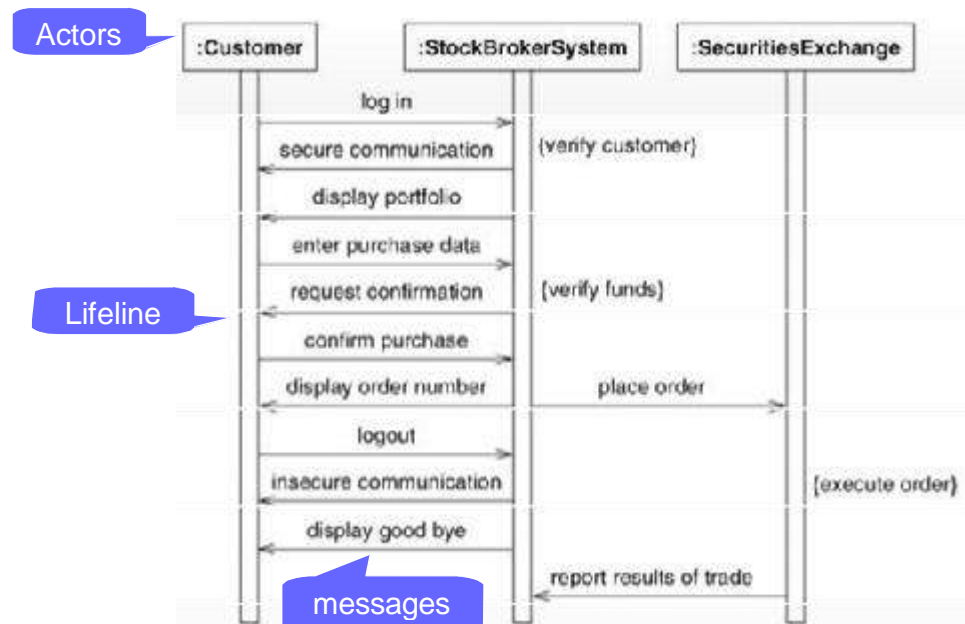


Figure 7.5 Sequence diagram for a session with an online stock broker. A sequence diagram shows the participants in an interaction and the sequence of messages among them.

A stock purchase

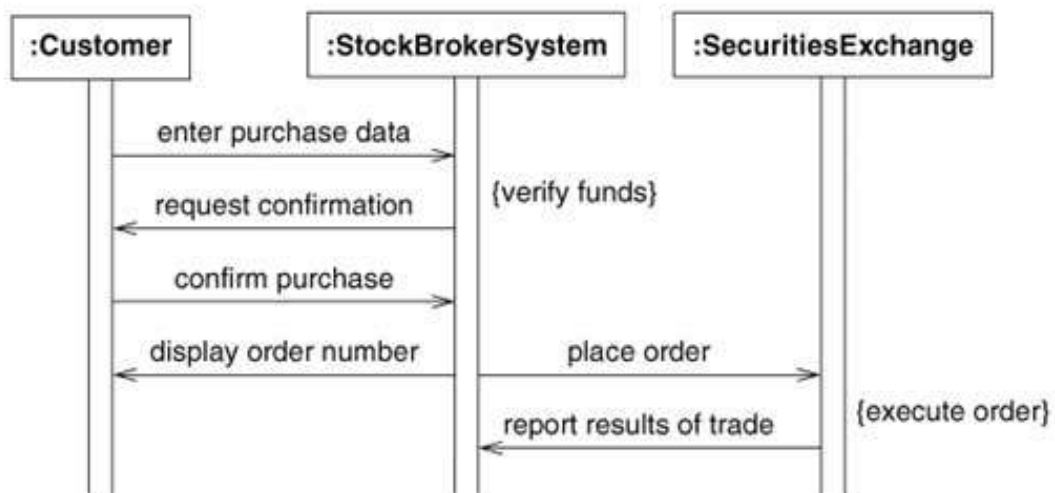


Figure 7.6 Sequence diagram for a stock purchase. Sequence diagrams can show large-scale interactions as well as smaller, constituent tasks.

A stock quote

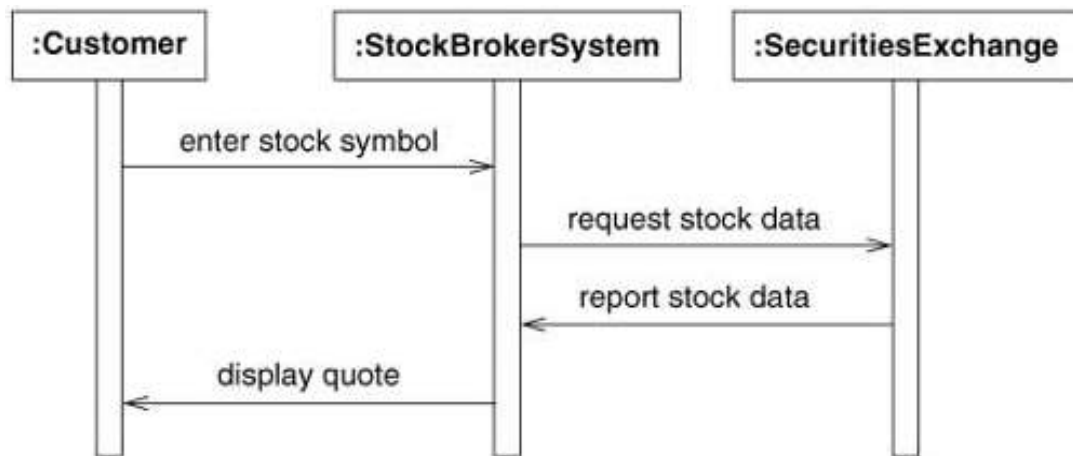


Figure 7.7 Sequence diagram for a stock quote.

A exception case

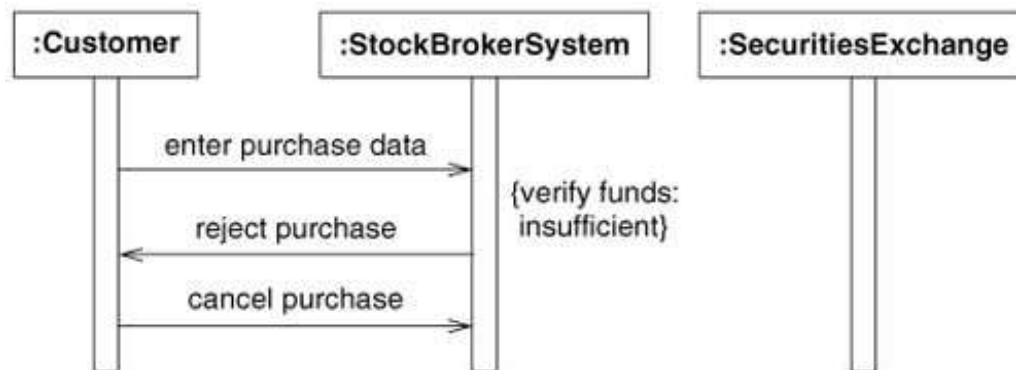


Figure 7.8 Sequence diagram for a stock purchase that fails.

Guidelines

- Prepare at least one scenario per use case
- Abstract the scenarios into sequence diagrams
- Divide complex interactions
- Prepare a sequence diagram for each error condition

Procedural Sequence Models

- Sequence Diagrams with Passive Objects
- A passive object is not activated until it has been called.

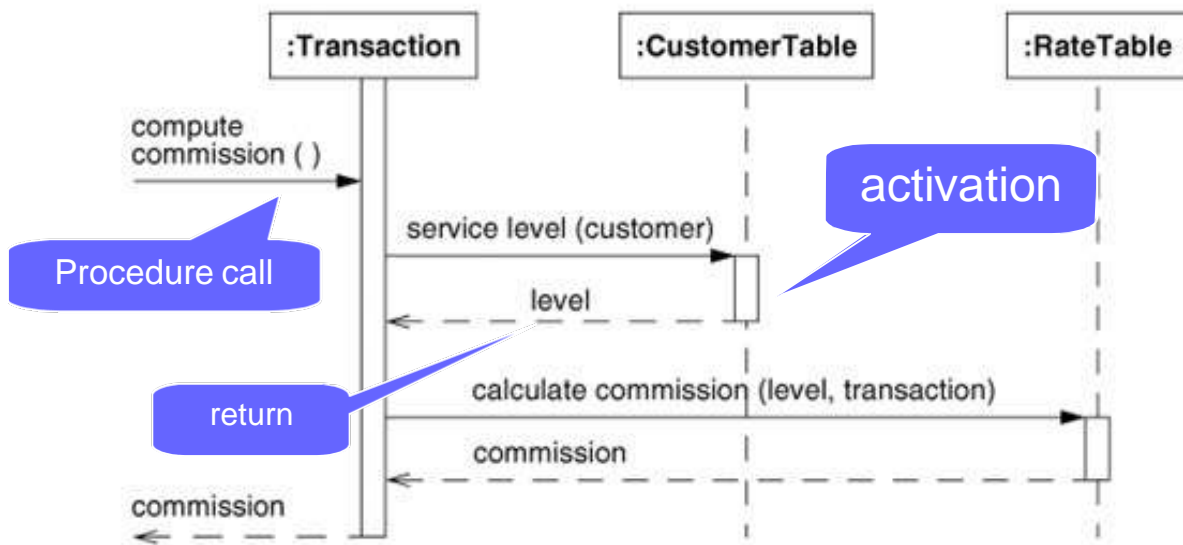


Figure 8.5 Sequence diagram with passive objects. Sequence diagrams can show the implementation of operations.

Sequence Diagrams with Transient Objects

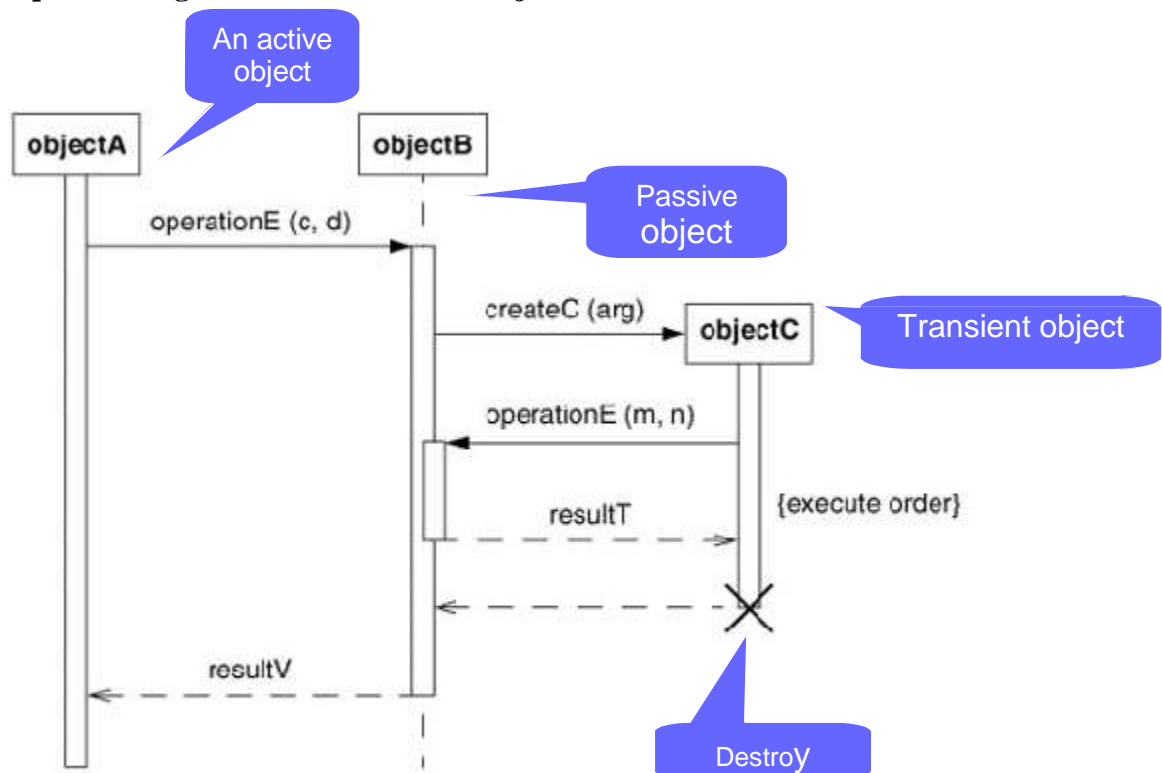


Figure 8.6 Sequence diagram with a transient object. Many applications have a mix of active and passive objects. They create and destroy objects.

Activity Models

● An activity diagram shows the sequence of steps that make up a complex process, such as an algorithm or workflow.

- Activity diagrams are most useful during the early stages of designing algorithms and workflows.

- Activity diagram is like a traditional flowchart in that it shows the flow of control from step to step

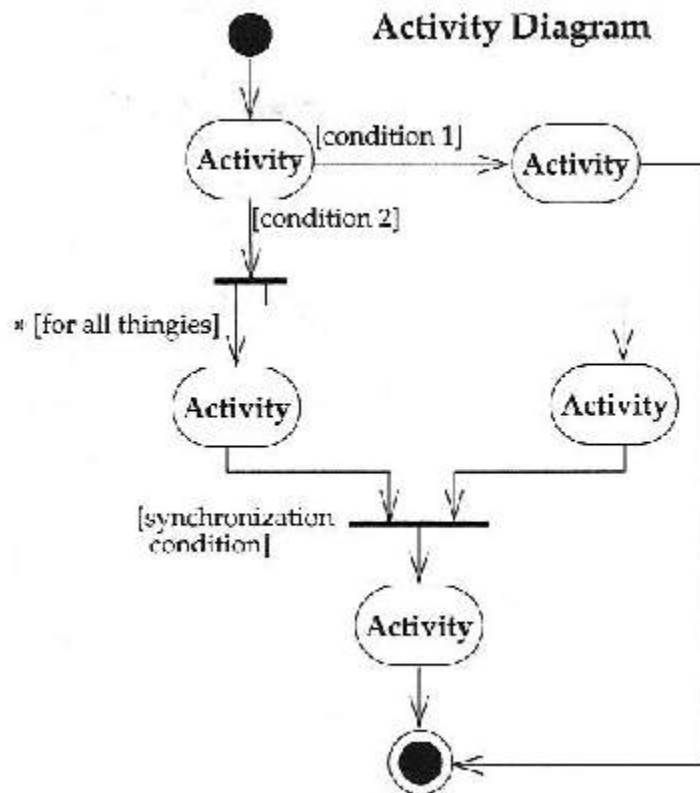
Activity diagram Notation

- Start at the top black circle
- If condition 1 is TRUE, go right; if condition 2 is TRUE, go down
- At first bar (a synchronization bar), break apart to follow 2 parallel paths
- At second bar, come together to proceed only when both parallel activities are done

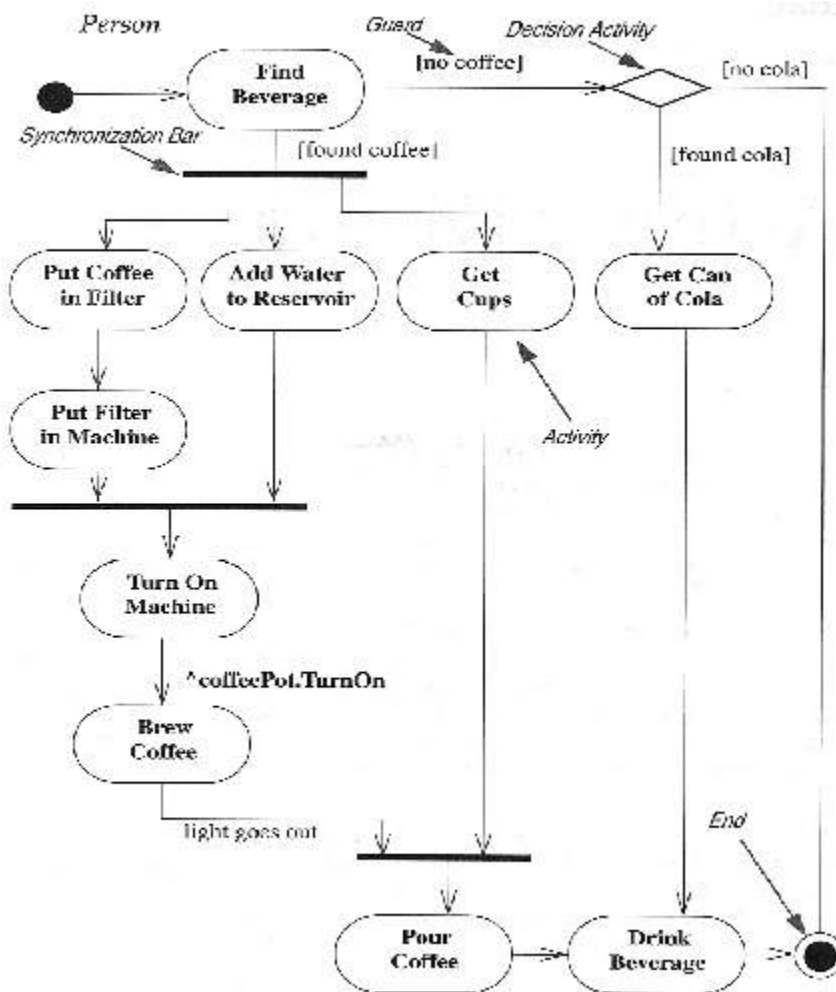
- Activity – an oval
- Trigger – path exiting an activity
- Guard – each trigger has a guard, a logical expression that evaluates to “true” or “false”

- Synchronization Bar – can break a trigger into multiple triggers operating in parallel or can join multiple triggers into one when all are complete

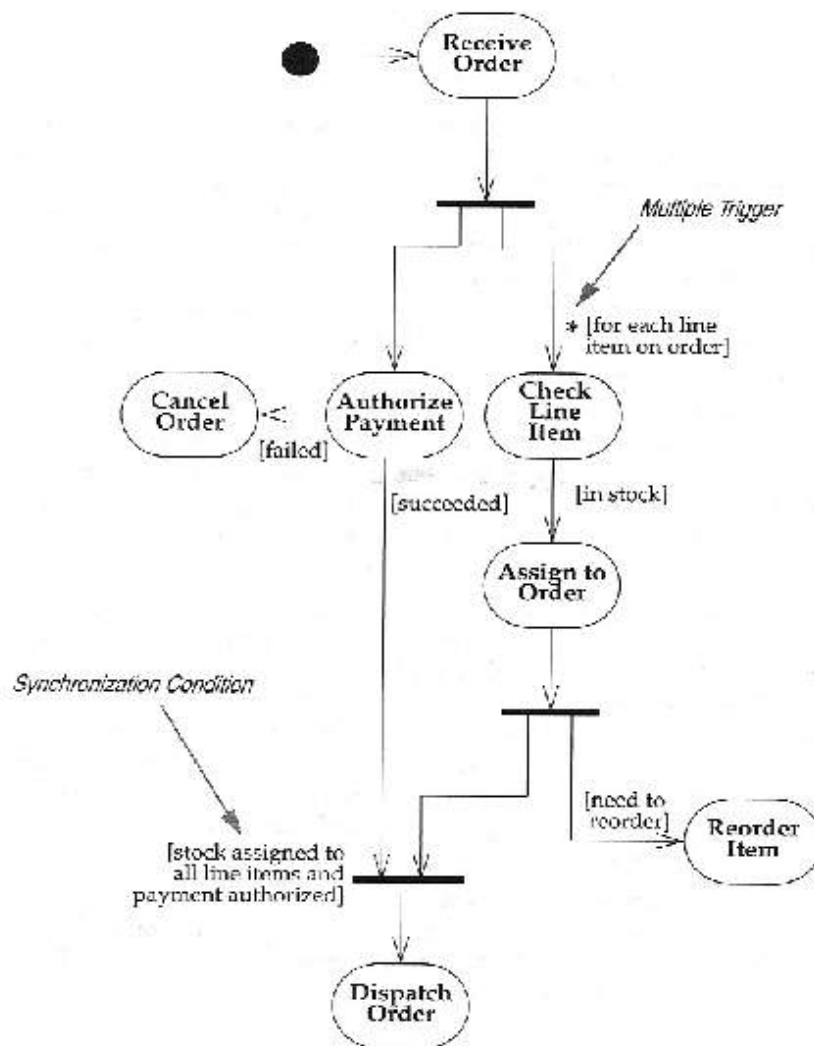
- Decision Diamond – used to describe nested decisions (the first decision is indicated by an activity with multiple triggers coming out of it)

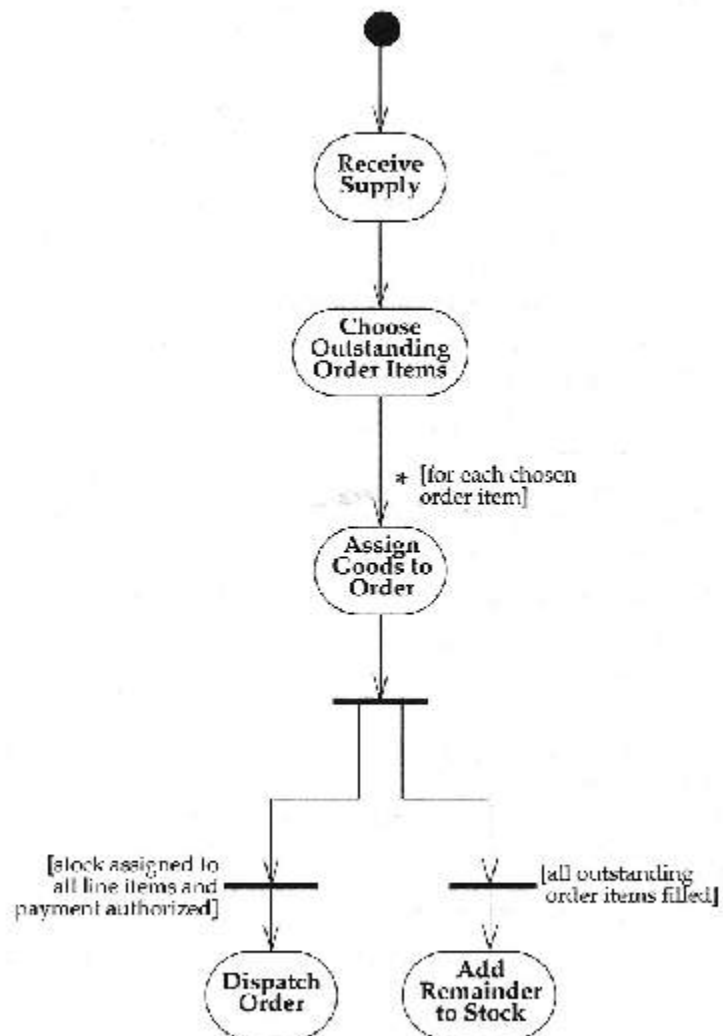


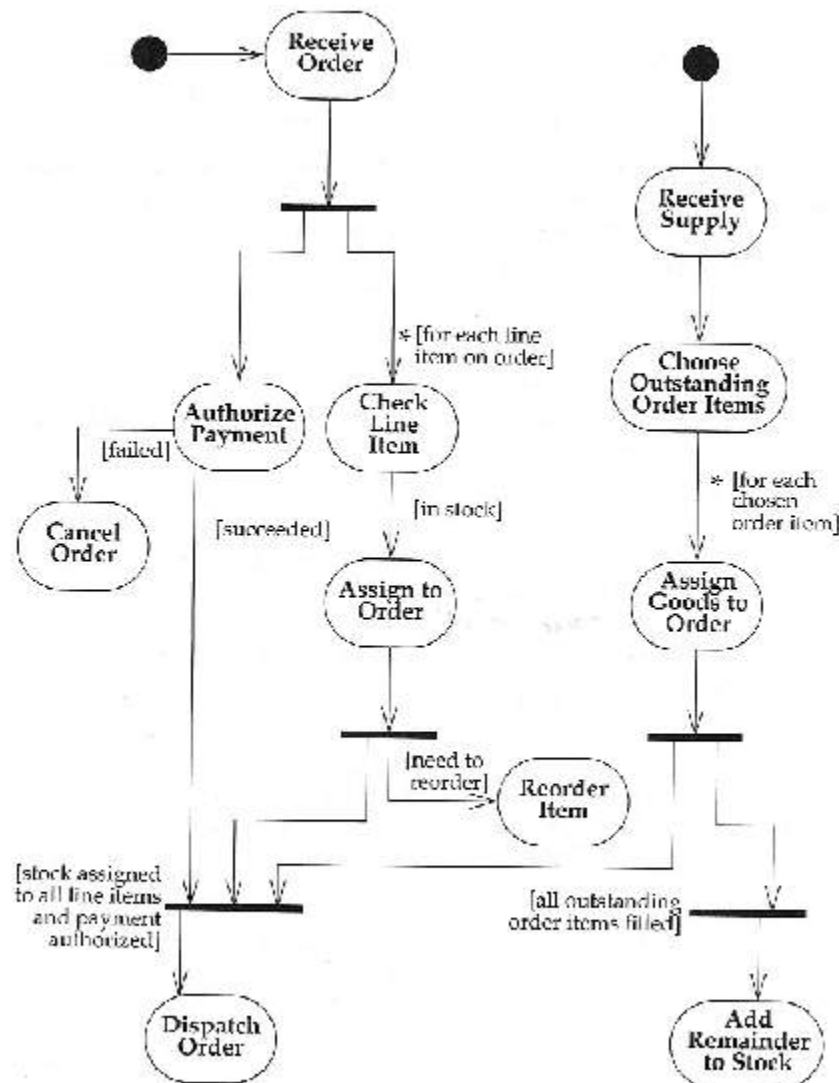
Eg:



Eg: activity diagram for Use Case: Receiving an Order



Activity diagram for Use Case: Receiving a Supply

Activity diagram for Use Case: Receiving an Order and Receiving a Supply

Activity diagram for stock trade processing

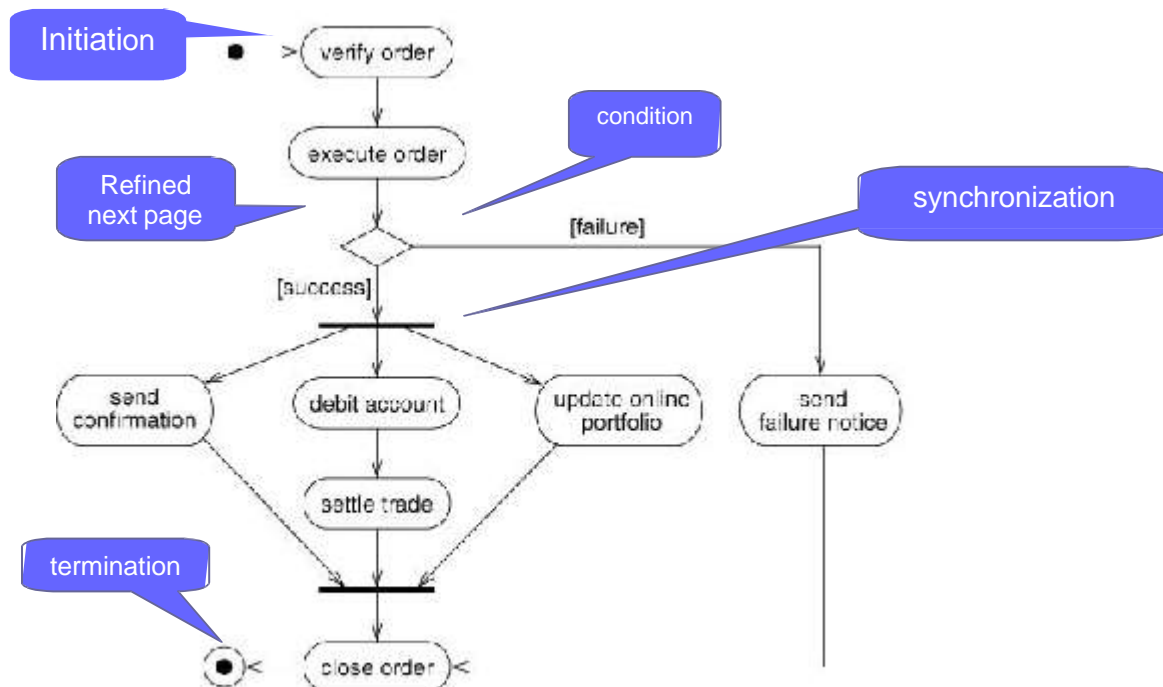


Figure 7.9 Activity diagram for stock trade processing. An activity diagram shows the sequence of steps that make up a complex process.

A Finer Activity for *execute order*

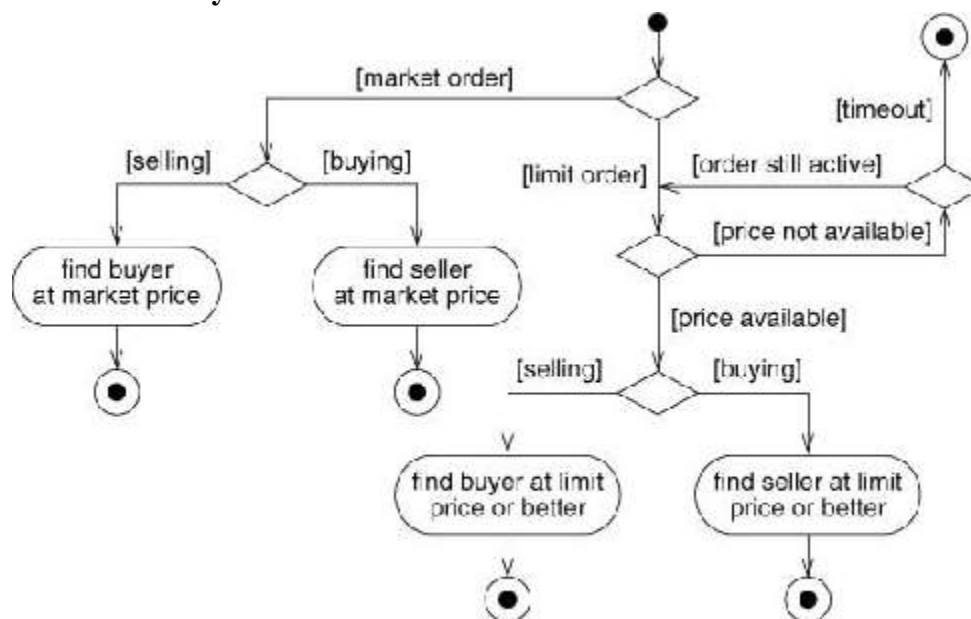


Figure 7.10 Activity diagram for *execute order*. An activity may be decomposed into finer activities.

Guidelines

- Don't misuse activity diagrams
- Do not be used as an excuse to develop software via flowcharts.

- Level diagrams
- Be careful with branches and conditions
- Be careful with concurrent activities
- Consider executable activity diagrams

Special constructs for activity diagrams

- Sending and receiving signals
- Swim lanes
- Object flows

Sending and Receiving Signals

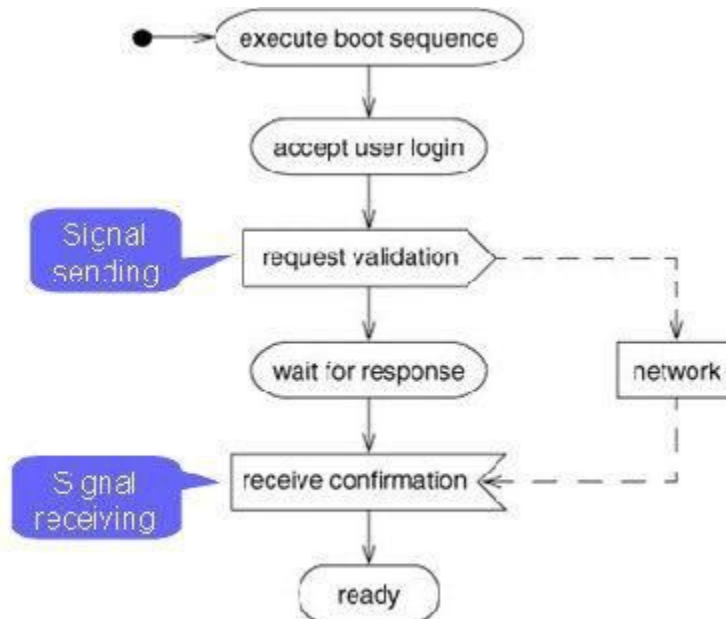


Figure 8.7 Activity diagram with signals. Activity diagrams can show fine control via sending and receiving events.

Swimlanes

- To know which human organization is responsible for an activity.

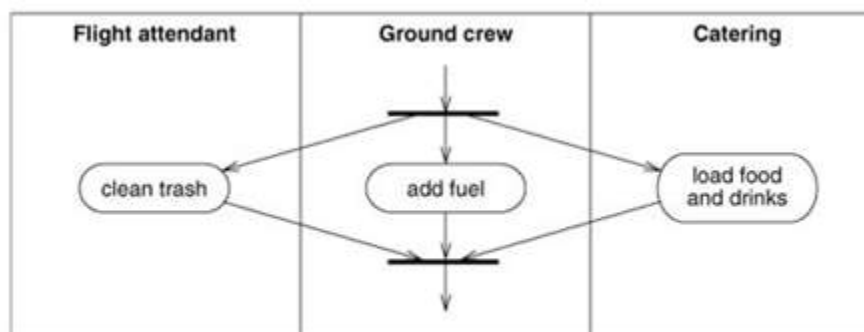
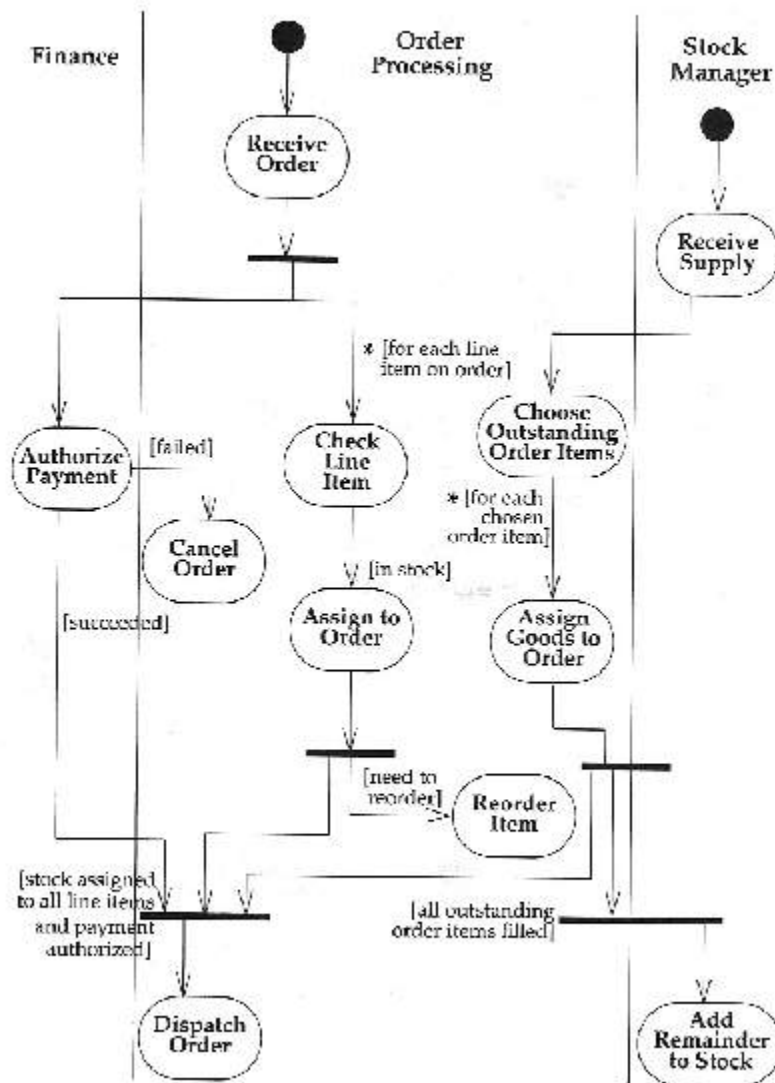


Figure 8.8 Activity diagram with swimlanes. Swimlanes can show organizational responsibility for activities.

Swimlanes - Activity Diagrams that show activities by class

- Arrange activity diagrams into vertical zones separated by lines
- Each zone represents the responsibilities of a particular class (in this example, a particular department)



Object Flows

- Show both the control and the progression of an object from state to state as activities act on it.

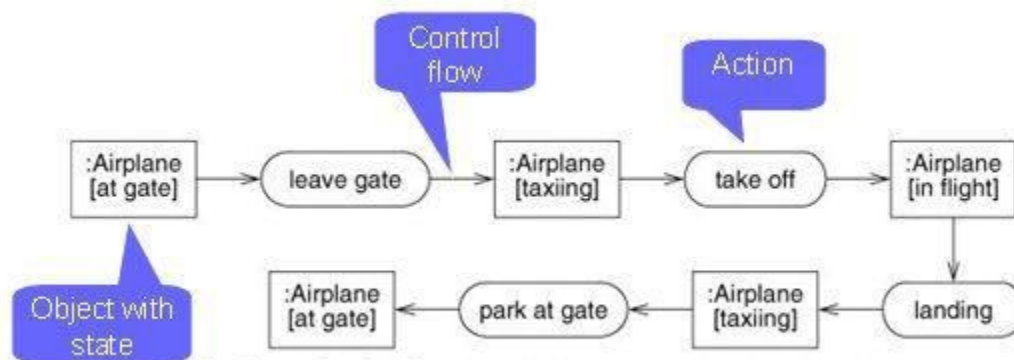


Figure 8.9 Activity diagram with object flows. An activity diagram can show the objects that are inputs or outputs of activities.