MODULE -III

BASIC BEHAVIORAL MODELING

Interactions

- An interaction is a behavior that comprises a set of messages exchanged among a set of objects within a context to accomplish a purpose.
- A message is a specification of a communication between objects that conveys information with the expectation that activity will ensue.

Context

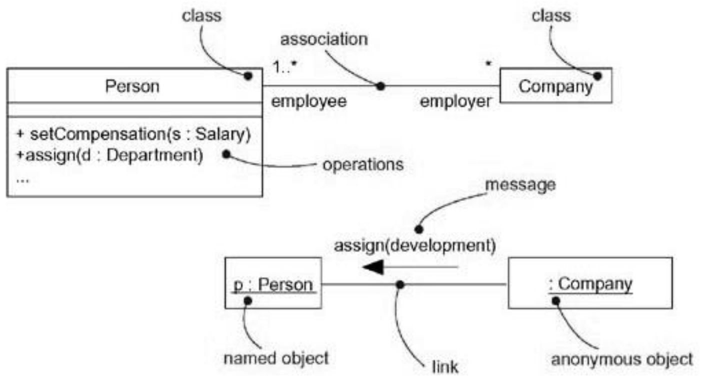
- We can use interactions to visualize, specify, construct, and document the semantics of a class
- We may find an interaction wherever objects are linked to one another.
- We'll find interactions in the collaboration of objects that exist in the context of your system or subsystem.
- We will also find interactions in the context of an operation.
- We might create interactions that show how the attributes of that class collaborate with one another
- Finally, you'll find interactions in the context of a class.

Objects and Roles

- The objects that participate in an interaction are either concrete things or prototypical things.
- As a concrete thing, an object represents something in the real world. For example, p, an instance of the class Person, might denote a particular human
- As a prototypical thing, p might represent any instance of Person.
- Although abstract classes and interfaces, by definition, may not have any direct instances, you may find instances of these things in an interaction
- Such instances do not represent direct instances of the abstract class or of the interface, but may represent, respectively, indirect (or prototypical) instances of any concrete children of the abstract class of some concrete class that realizes that interface.

Links

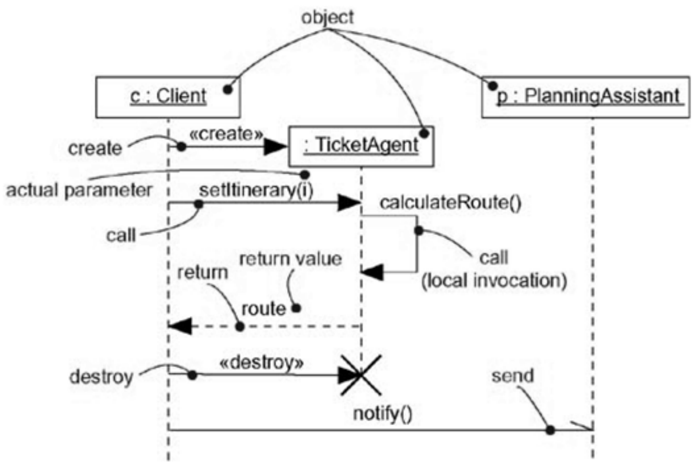
- A link is a semantic connection among objects. In general, a link is an instance of an association
- Wherever a class has an association to another class, there may be a link between the instances of the two classes. Wherever there is a link between two objects, one object can send a message to the other object
- A link specifies a path along which one object can dispatch a message to another (or the same) object.



Links and Associations

Messages

- A message is the specification of a communication among objects that conveys information with the expectation that activity will ensue.
- The receipt of a message instance may be considered an instance of an event.
- When you pass a message, the action that results is an executable statement that forms an abstraction of a computational procedure. An action may result in a change in state.
- In the UML, you can model several kinds of actions
- The UML provides a visual distinction among these kinds of messages, as follows

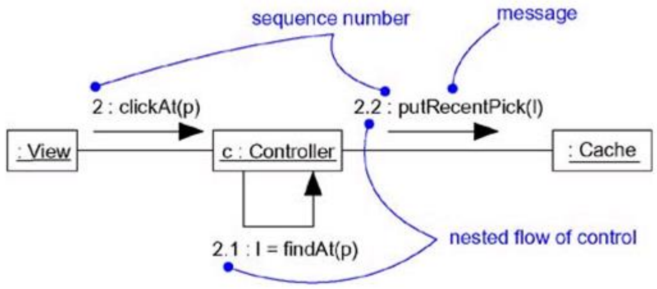


Messages

Messages

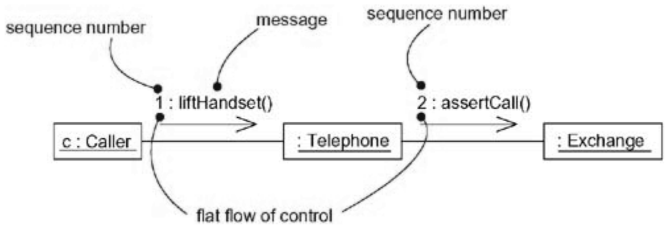
Sequencing

- When an object passes a message to another object the receiving object might in turn send a message to another object, which might send a message to yet a different object, and so on. This stream of messages forms a sequence
- Any sequence must have a beginning; the start of every sequence is rooted in some process or thread.



Procedural Sequence

- We can specify a flat flow of control, rendered using a stick arrowhead, to model the nonprocedural progression of control from step to step.



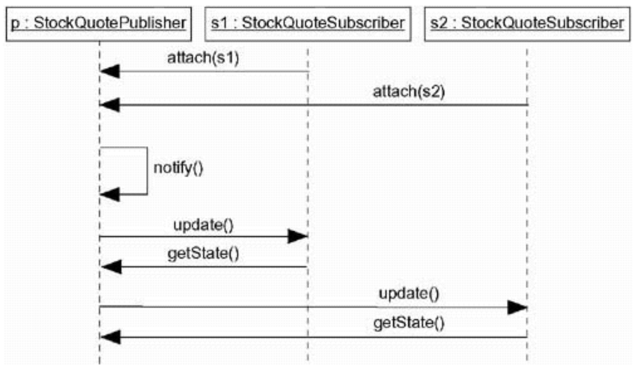
Flat Sequence

Common Modeling Techniques

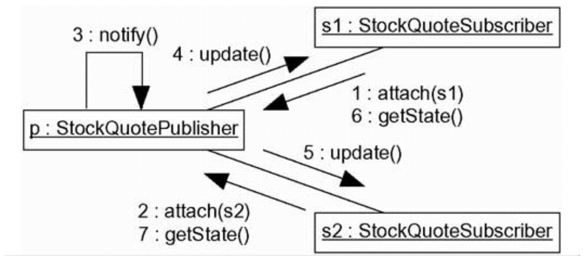
Modeling a Flow of Control

- To model a flow of control

  - Set the context for the interaction, whether it is the system as a whole, a class, or an individual operation.
  - Set the stage for the interaction by identifying which objects play a role; set their initial properties, including their attribute values, state, and role.
  - If your model emphasizes the structural organization of these objects, identify the links that connect them, relevant to the paths of communication that take place in this interaction. Specify the nature of the links using the UML's standard stereotypes and constraints, as necessary.
  - In time order, specify the messages that pass from object to object. As necessary, distinguish the different kinds of messages; include parameters and return values to convey the necessary detail of this interaction.
  - Also to convey the necessary detail of this interaction, adorn each object at every moment in time with its state and role.



Flow of Control by Time



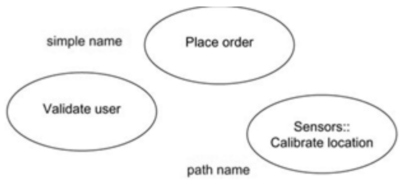
Flow of Control by Organization

Use Cases

- A use case is a description of a set of sequences of actions, including variants, that a system performs to yield an observable result of value to an actor.
- Graphically, a use case is rendered as an ellipse.

Names

- Every use case must have a name that distinguishes it from other use cases. A name is a textual string.
- That name alone is known as **a simple name**; **a path name** is the use case name prefixed by the name of the package in which that use case lives.
- A use case is typically drawn showing only its name

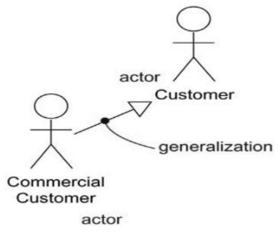


Simple and Path Names

Use Cases and Actors

- An actor represents a coherent set of roles that users of use cases play when interacting with these
  use cases.

- Typically, an actor represents a role that a human, a hardware device, or another system plays with a system.
- An instance of an actor, therefore, represents an individual interacting with the system in a specific way
- Actors may be connected to use cases only by association
- An association between an actor and a use case indicates that the actor and the use case communicate with one another, each one possibly sending and receiving messages.



Actors

Use Cases and Flow of Events

- A use case describes what a system does but it does not specify how it does it.
- You can specify the behavior of a use case by describing a flow of events in text clearly enough for an outsider to understand it easily
- When you write this flow of events, you should include how and when the use case starts and ends
- When the use case interacts with the actors and what objects are exchanged, and the basic flow and alternative flows of the behavior.

➢ **Main flow of events:**

The use case starts when the system prompts the Customer for a PIN number. The Customer can now enter a PIN number via the keypad. The Customer commits the entry by pressing the Enter button. The system then checks this PIN number to see if it is valid. If the PIN number is valid, the system acknowledges the entry, thus ending the use case.

➢ **Exceptional flow of events:**

The Customer can cancel a transaction at any time by pressing the Cancel button, thus restarting the use case. No changes are made to the Customer's account.

➢ **Exceptional flow of events:**

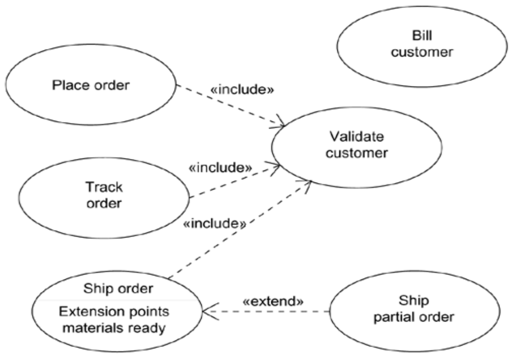
The Customer can clear a PIN number anytime before committing it and reenter a new PIN number.

Common Modeling Techniques

Modeling the Behavior of an Element

- The most common thing for which you'll apply use cases is to model the behavior of an element, whether it is the system as a whole, a subsystem, or a class.
- To model the behavior of an element

  - Identify the actors that interact with the element. Candidate actors include groups that require certain behavior to perform their tasks or that are needed directly or indirectly to perform the element's functions.
  - Organize actors by identifying general and more specialized roles.
  - For each actor, consider the primary ways in which that actor interacts with the element. Consider also interactions that change the state of the element or its environment or that involve a response to some event.
  - Consider also the exceptional ways in which each actor interacts with the element.
  - Organize these behaviors as use cases, applying include and extend relationships to factor common behavior and distinguish exceptional behavior.



Modeling the Behavior of an Element

Use Case Diagram

- A use case diagram is a diagram that shows a set of use cases and actors and their relationships.

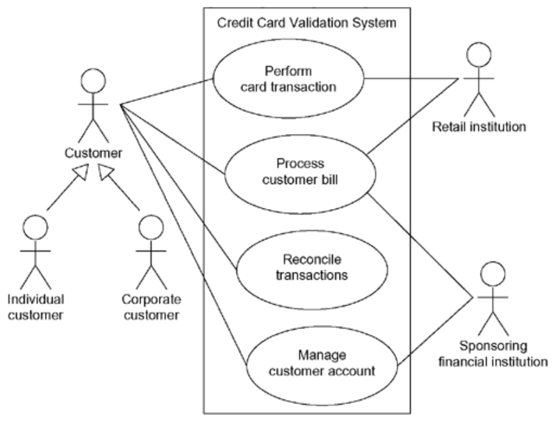
Contents

- Use case diagrams commonly contain
  - Use cases
  - Actors
  - Dependency, generalization, and association relationships
- Like all other diagrams, use case diagrams may contain notes and constraints.
- Use case diagrams may also contain packages
- Occasionally, you'll want to place instances of use cases in your diagrams, as well, especially when you want to visualize a specific executing system.

Common Modeling Techniques

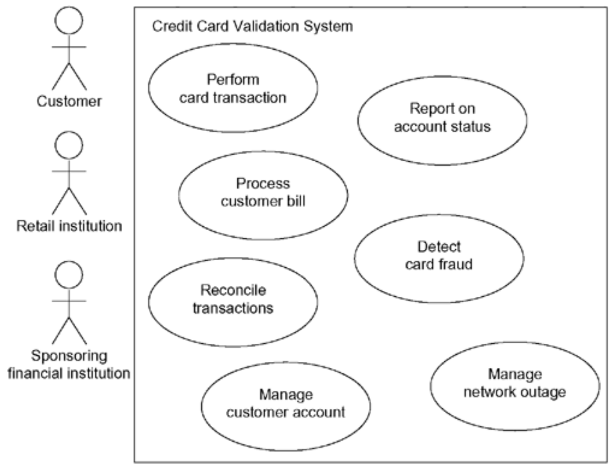
Modeling the context of a system involves drawing a line around the whole system and asserting which actors lie outside the system and interact with it.Here, you'll apply use case diagrams to specify the actors and the meaning of their roles.

Modeling the requirements of a system involves specifying what that system should do (from a point of view of outside the system), independent of how that system should do it. Here, you'll apply use case diagrams to specify the desired behavior of the system.



Modeling the Context of a System

○



Modeling the Requirements of a System

Forward and Reverse Engineering

- **Forward engineering** is the process of transforming a model into code through a mapping to an implementation language.
- A use case diagram can be forward engineered to form tests for the element to which it applies.
- Each use case in a use case diagram specifies a flow of events and these flows specify how the element is expected to behave

- **Reverse engineering** is the process of transforming code into a model through a mapping from a specific implementation language.
- The UML's use case diagrams simply give you a standard and expressive language in which to state what you discover.

Activity Diagrams

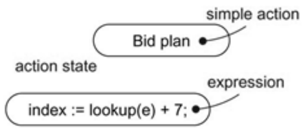
- An activity diagram shows the flow from activity to activity. An is an ongoing nonatomic execution within a state machine.
- Activities ultimately result in some action, which is made up of executable atomic computations that result in a change in state of the system or the return of a value.
- Actions encompass calling another operation, sending a signal, creating or destroying an object, or some pure computation, such as evaluating an expression.
- Graphically, an activity diagram is a collection of vertices and arcs.

Contents

- Activity diagrams commonly contain
  - Activity states and action states
  - Transitions
  - Objects
- Like all other diagrams, activity diagrams may contain notes and constraints.

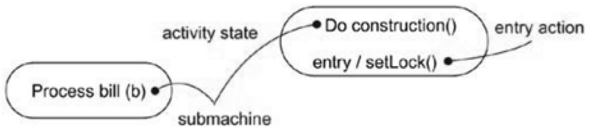
Action States and Activity States

- Executable, atomic computations are called **action states** because they are states of the system, each representing the execution of an action.
- We represent an action state using a lozenge shape (a symbol with horizontal top and bottom and convex sides). Inside that shape, you may write any expression.
- Action states can't be decomposed. Furthermore, action states are atomic, meaning that events may occur, but the work of the action state is not interrupted.
- Finally, the work of an action state is generally considered to take insignificant execution time.



Action States

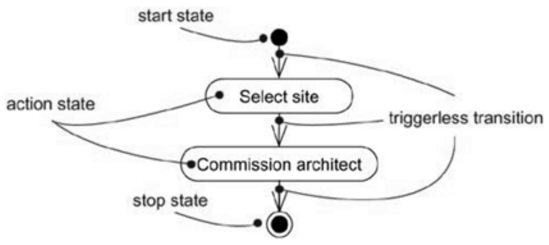
- **activity states** can be further decomposed, their activity being represented by other activity diagrams



Activity States

Transitions

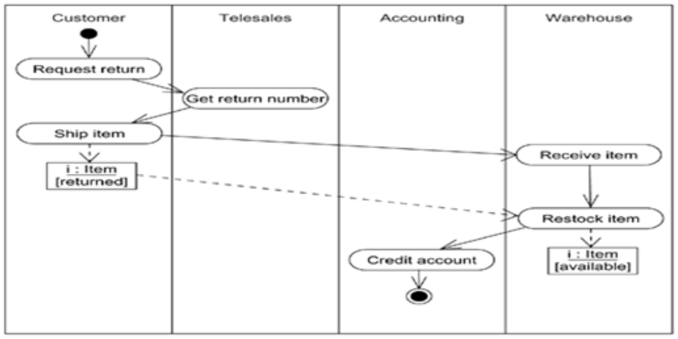
- When the action or activity of a state completes, flow of control passes immediately to the next action or activity state.



Common Modeling Techniques

Modeling a Workflow

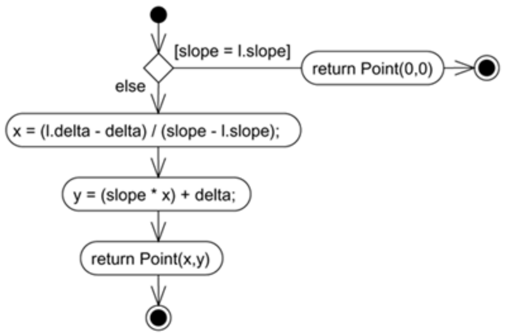
- No software-intensive system exists in isolation; there's always some context in which a system lives, and that context always encompasses actors that interact with the system.
- Especially for mission critical, enterprise software, you'll find automated systems working in the context of higher-level business processes.
- These business processes are kinds of workflows because they represent the flow of work and objects through the business.



Modeling a Workflow

Modeling an Operation

- An activity diagram can be attached to any modeling element for the purpose of visualizing, specifying, constructing, and documenting that element's behavior.
- You can attach activity diagrams to classes, interfaces, components, nodes, use cases, and collaborations.
- The most common element to which you'll attach an activity diagram is an operation.
- An activity diagram is simply a flowchart of an operation's actions.
- An activity diagram's primary advantage is that all the elements in the diagram are semantically tied to a rich underlying model.



Modeling an Operation

Forward and Reverse Engineering

- **Forward engineering** (the creation of code from a model) is possible for activity diagrams, especially if the context of the diagram is an operation.
- **Reverse engineering** (the creation of a model from code) is also possible for activity diagrams, especially if the context of the code is the body of an operation.