

Basic Behavioral Modeling

Lecture 4

Part I

Use Cases

Use Cases

- A **use case model** describes a system's functionality from the point of view of the different actors that interact with the system to receive services.
- A **use case diagram** shows the relationships among actors and use cases within a system. Use case diagrams address the static use case view of a system.
- An **actor** (actor class) is a predefined stereotype of type denoting an entity outside the system that interacts with use cases in the system.
- A **use case** is a class that defines a set of use case instances. It specifies a set of actions that a system performs that yields a result of value to an actor.

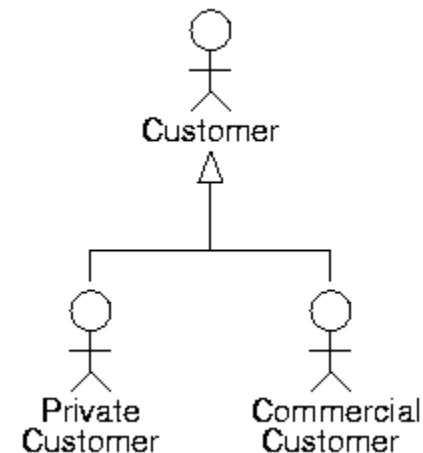
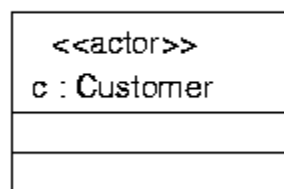
Actors

- A coherent set of roles that users of use cases play when they interact with use cases
- Typically represents the role that a human, hardware device or another system plays with the system
- Actors are not part of the system
- As an actor is a class, it can be generalized

collapsed view

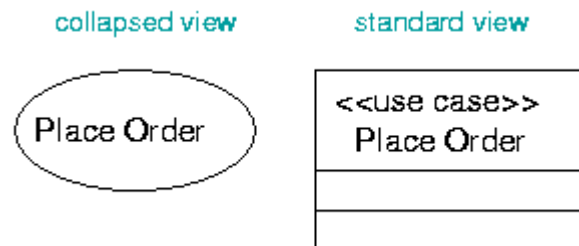


standard view



Use Cases

- A coherent unit of externally visible functionality provided by a system unit
- Used to define a piece of coherent behavior without revealing the internal details
- A use case describes what the system does, but does not specify how it does it.
- The behavior of a use case can be specified by describing on or more flow of events written in plain text in a note in the diagram



Example: ATM

- ValidateUser use case
 - main flow of events:
 - the use case starts when the system prompts the Customer for a PIN number. The Customer then enters a PIN via the keypad, and presses the enter button. The system then checks if the PIN is valid. If the PIN is valid, the system acknowledges the entry, thus ending the use case.
 - exceptional flows of events:
 - the Customer can cancel a transaction any time by pressing the Cancel button, thus restarting the use case
 - the Customer can clear a PIN any time before committing it and reenter a new PIN
 - If the Customer enters an invalid PIN, the use case restarts

Use Cases and Scenarios


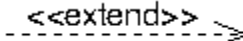

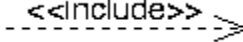
- Flows of events are first described in text
- As requirements are more understood, sequence diagrams are used to describe the main flow and exceptional flows of events
- Each sequence is called a scenario
- Each scenario is an instance of the use case
- Modestly complex systems might have a few dozen use cases and each of these can expand to several dozen scenarios

Use Cases and Collaborations

- Use cases capture an abstract view of the behavior of the system
- Ultimately, use cases have to be implemented as a society of interacting objects
- This society of elements is modeled as a collaboration diagram
- Finding the minimal set of well-structured collaborations that satisfy all the scenarios specified in all the use cases of the system is the focus of the system's architecture design

Organizing Use Cases

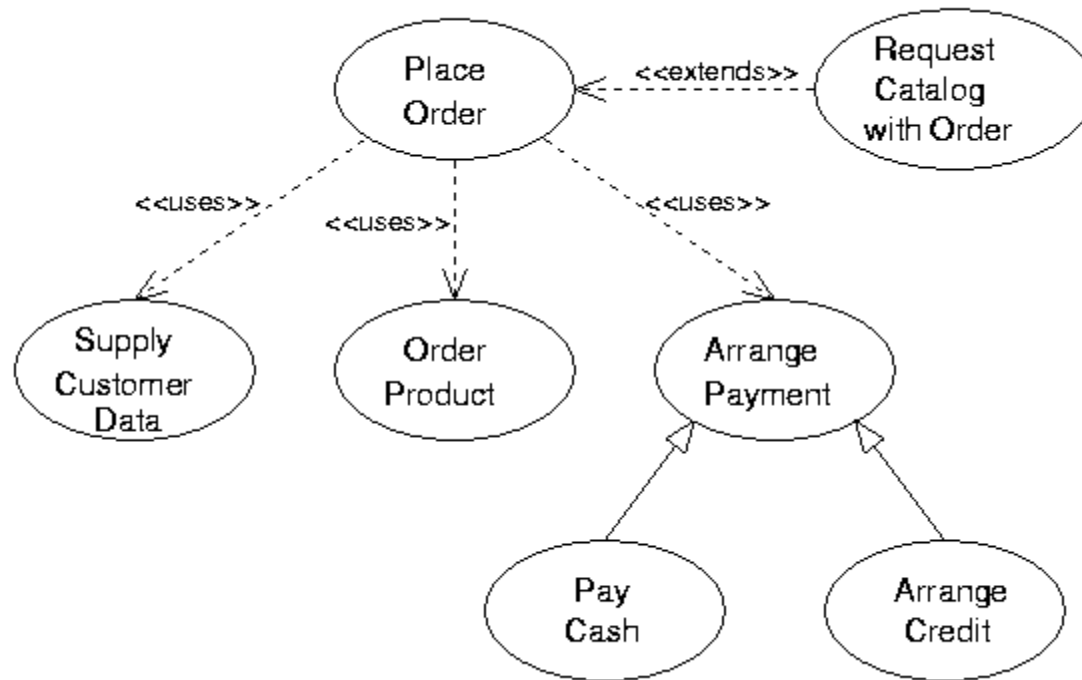
- Can be grouped in packages
- Can be organized by specifying generalization, include and extend relationships among them

association: communication path between actors and use cases	
extend: insertion of an additional behavior into a base use case. Can be used to add optional behavior	
generalization: specifies inheritance between two use cases	
include: factorization of use cases into simpler use cases	

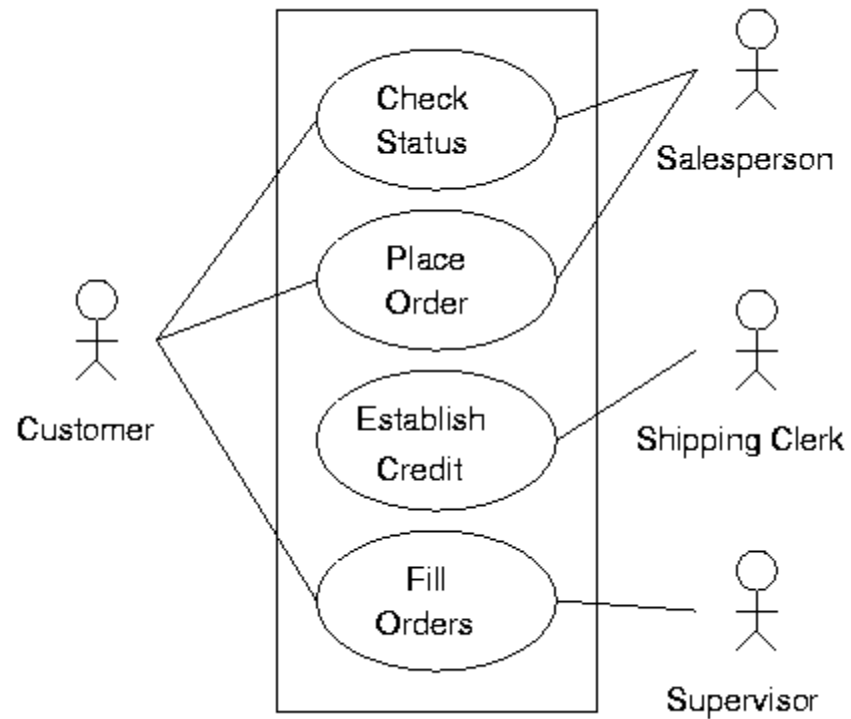
Use Case Diagrams

- Roles
 - Model the context of the system. Define what are the actors that are external to the system
 - Model the requirements of the system. Define what the system should do from an external point of view

Example



Example



Modeling the behavior of an element

- Use cases can be used at various levels of abstraction: system, subsystem, module, ... , class
- Can be used as a forum between the domain experts, the end users and the developers
- Gives a first impression of the element to developers by giving the intent of the element
- Useful for testing purposes

Modeling the behavior of an element

- Identify the actors that interact with the element.
- Organize actors using generalizations
- For each actor, consider its main scenario of interaction with the element
- Consider also the exceptional interaction scenarios
- Organize these behaviors as use cases, applying include and extend relationships to factor common behavior

Hints and Tips

- Every use case should represent a distinct and identifiable behavior of an element of the system
- Factor common behavior and apply include relationships
- Factor optional or variants of behavior and apply extend relationships
- Define a minimal set of scenarios
- Each diagram should serve a purpose and be restricted to a specific context, hiding details that are not part of this context

Part II

Interaction Diagrams

Interaction Diagrams

- Shows an interaction (a set of objects and their relationships) including exchanged message
- Two kinds of interaction diagrams:
 - **sequence diagrams**: emphasizes the time ordering of messages exchanged to provide a service or react to events. Used to model scenarios (normally a single scenario per diagram)
 - **collaboration diagrams**: emphasizes the structural organization of the objects that send or receive messages in . Used to model whole use cases (normally a collection of scenarios)

Sequence Diagrams

- Objects that participate in the interaction are placed along the X-axis, initiators to the left.
- Exchanged messages are placed along the Y-axis, in order of occurrence from top to bottom.
- Two features differentiate sequence diagrams from collaboration diagrams:
 - the **object lifeline**: represents the existence of an object over a period of time. Represented by a vertical dashed line.
 - The **focus of control**: represents the period of time during which an object is performing an action. Represented by a vertical thin rectangle over the object lifeline.

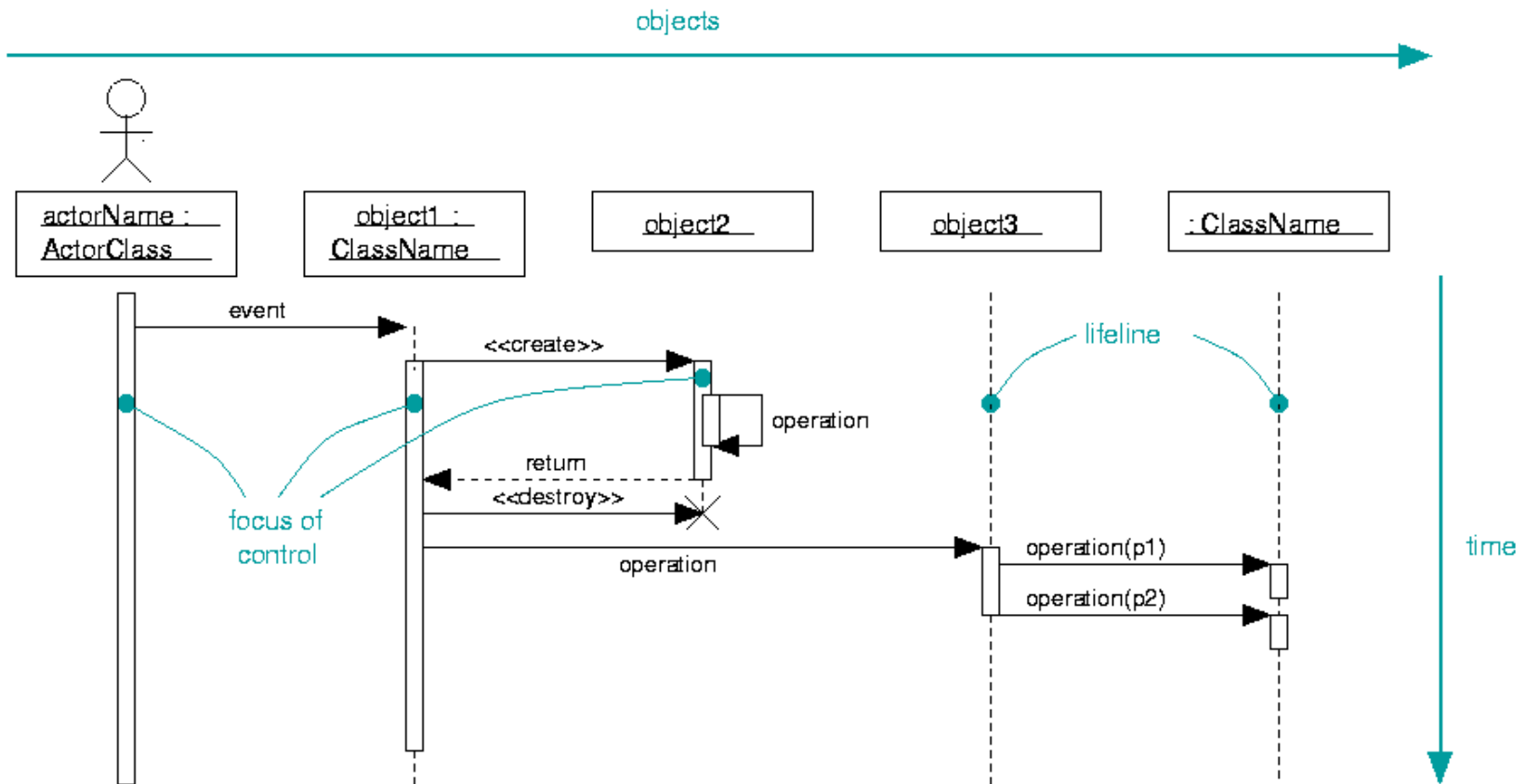
Sequence Diagrams

- Can represent the dynamic creation and destroying of objects
- The lifeline of an object starts only when the object is created and ends when the object is destroyed.
- Stereotypes <<create>> and <<destroy>> can be used to represent these lifecycle milestones.
- A return message can be sent when a focus of control ends
- Recursion and internal function calls can be represented with nested focus of controls
- A sequence diagram can begin or end at any point in a sequence. It is reasonable to break-up parts of a large flow in separate diagrams.

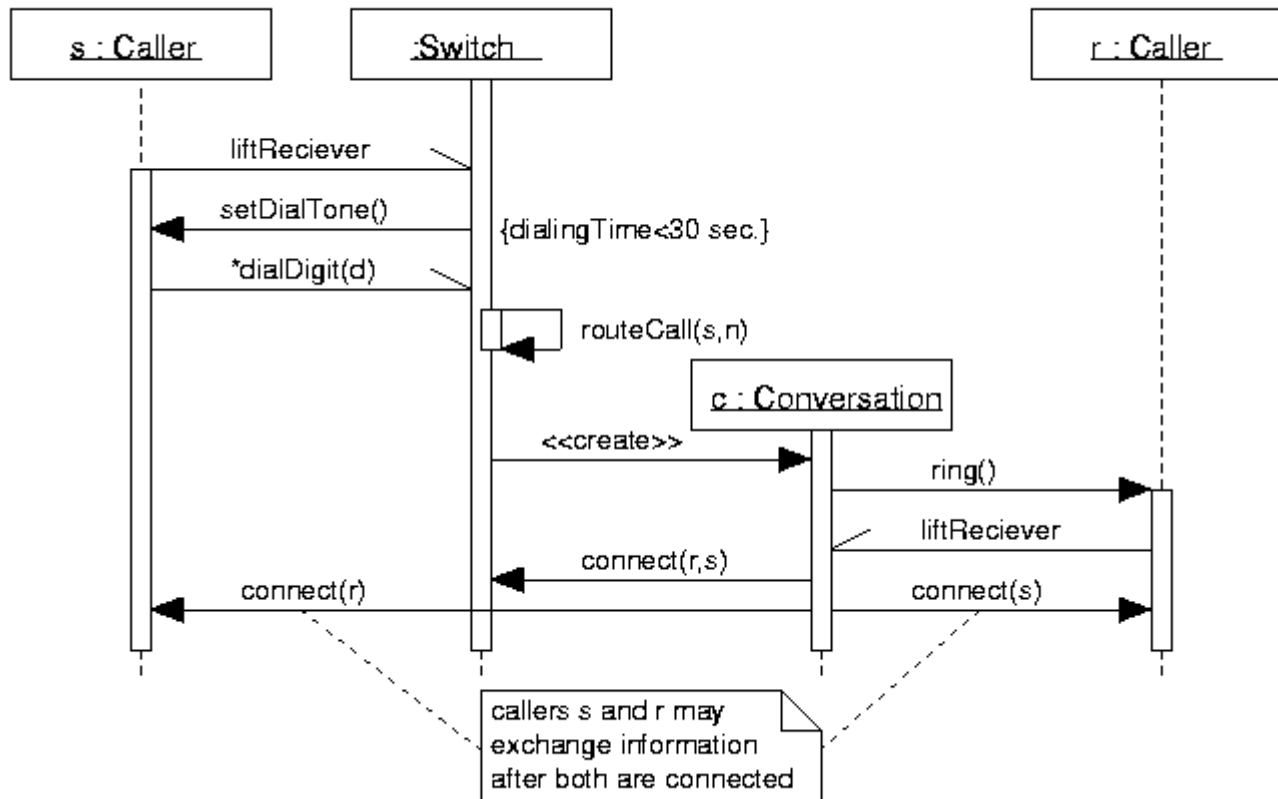
Sequence Diagram Construction

- Set the context for the interaction, e.g. system, subsystem, operation, class, scenario
- Identify the objects that play a role in the interaction
- Set the lifeline for each object
- Starting with the initiating message, lay out the sequence of messages from top to bottom
- Adorn each object's lifeline with focuses of control if needed
- If several sequence diagrams are needed to model different scenarios, group them in a package, with meaningful names to differentiate each diagram in the package

Example of Sequence Diagram



Example of Sequence Diagram



Collaboration Diagrams

- A collection of vertices representing objects, connected with arcs to signify interaction representing message passing
- Two features that differentiate collaboration diagrams from sequence diagrams:
 - **path stereotypes**: indicates how an object is linked to another, e.g. <<local>>, <<global>>, <<parameter>>. Appended on the far end of a link.
 - **sequence number**: indicates the time ordering of message sending. Prefixes message names.

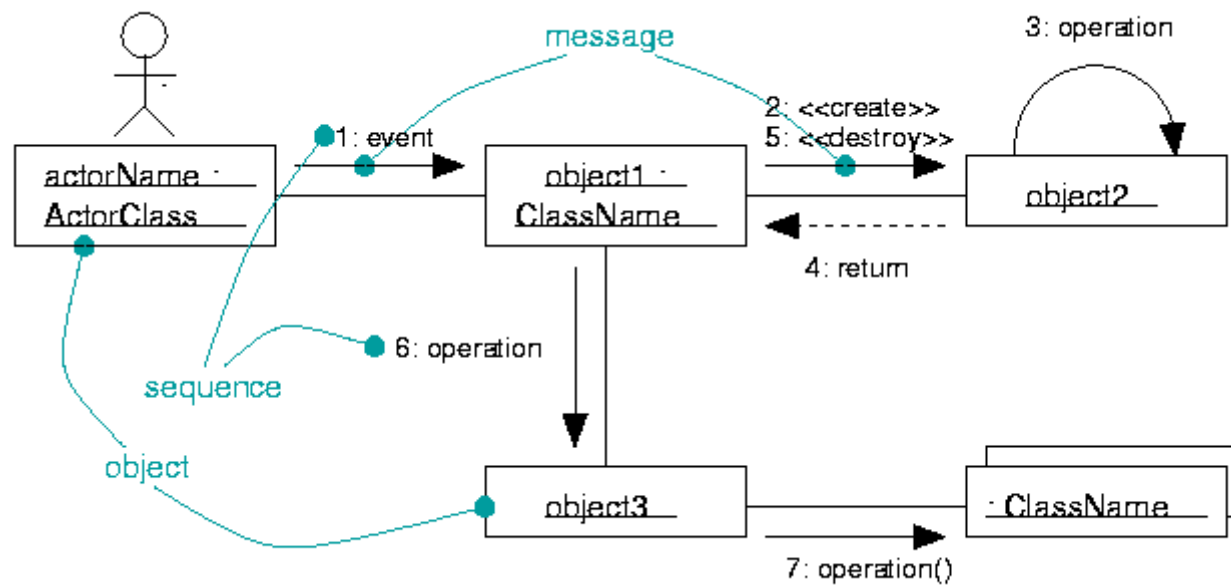
Collaboration Diagrams

- Links can be adorned with the names and direction of the messages exchanged
- Complex branching such as iteration and alternatives can be represented using iteration expressions (e.g. `*[1 := 1..n]`) and condition clauses (e.g. `[x<0]`) respectively.
- As for sequence diagrams, stereotypes `<<create>>` and `<<destroy>>` can be used to represent the dynamic creation and deletion of objects
- Normally used to directly model use cases (reverse engineering) or as a tool to integrate several sequence diagrams (forward engineering).

Constr. Collaboration Diagrams

- Set the context for the interaction, e.g. system, subsystem, operation, class, scenario or use case
- Identify the objects that play a role in the interaction and lay them as vertices in a graph
- Specify the links between these objects as edges between the objects
- Starting with the initiating message, attach message names to each edge, assigning them incremental sequence numbers.
- If several sequence diagrams are needed to model different aspects of a use case, group them in a package, with meaningful names to differentiate each diagram in the package

Collaboration diagram example



Part III

Activity Diagrams

Activity Diagrams

- Shows the flow of control from activity to activity
- **Activity**: an ongoing non-automatic execution within a state machine
- Activities ultimately result in some action
- **Action**: a sequence of atomic computations that result in a change in the state of the system or the return of a value
 - call another operation, send signal, create or destroy objects, evaluate expressions/statements
- A collection of vertices and arcs

Example

- Insert figure 19-1

Action and Activity States

- An *activity diagram* decomposes an activity in terms of its component activities and actions
- Atomic (lowest level) sub-activities are represented by *action states*
 - cannot be interrupted, considered to take an insignificant amount of time
- Non-atomic (abstract) activities are represented by *activity states*
 - can be decomposed (as an activity diagram), can be interrupted

Transitions

- When the action or activity of a state completes, control passes immediately to the next action or activity state
- These dependencies, called *state transitions*, are represented by a directed line showing the direction of flow of control
- Execution starts at the start state and continues indefinitely or until a stop state is reached
- Entry and exit actions can be specified

Branching

- Sequential transitions are common, but most activities include conditional branching dependent to the system state
- Branching can be used in activity diagrams, which is represented by a diamond
- A branch has only one entry point and possibly several exit points
- Alternate exit transitions are *guarded*. *Guards* should be defined to cover all possibilities and avoid overlapping
- Iteration can be defined using a branch

Forking and Joining

- Concurrent flows can exist in many situations
- The activation (fork) and synchronization (join) of parallel flows of control can be represented by synchronization bars, represented by a thick horizontal or vertical line
- To each fork with x outgoing transitions corresponds a join with x incoming transitions

Swimlanes

- Activity states can be partitioned into groups, each group representing the entity responsible for those activities
- Such partitions are called swimlanes
- Swimlane names do not need to have a deep meaning, but normally represent modules, subsystems or objects
- Activities belong to exactly one swimlane, but transitions may cross lanes

Example

- Insert figure19-7

Object Flow

- Some activities produce results (objects) that can be used by or direct the execution other activities
- These can be included in activity diagrams and linked by dependencies to the activity/action states

Example

- Insert figure 19-8

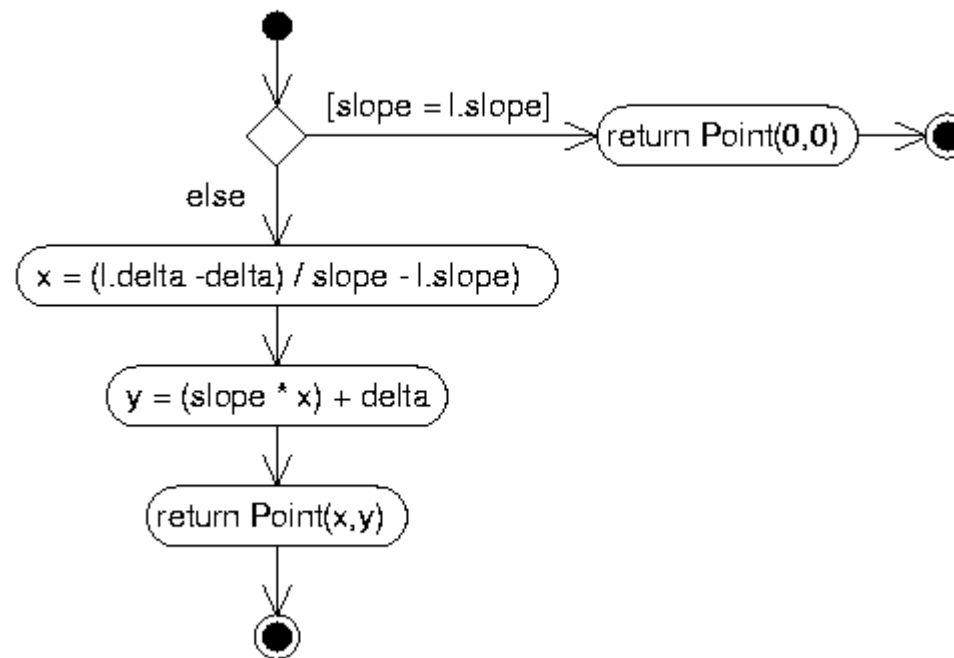
Uses: Modeling Workflows

- The business process (collaboration of human and automated systems) can be represented using activity diagrams
 - Establish the focus of the workflow
 - Find the business entities that interact in this workflow and elect them as swimlanes
 - Identify the pre and post conditions to the workflow
 - Beginning from the start state, identify the activities and actions pertaining to this workflow
 - Collapse out of focus complex action states into activity states
 - Identify the transitions that connect these activities consider branching and fork and join for complex/concurrent activities
 - Important or transition-related objects, along with a part of their state can be rendered in the diagram to ease comprehension

Uses: Modeling Operations

- Activity diagrams are used to convey the behavior of elements in terms of a sequence of activities
- The most common element to which an activity diagram is attached is an operation
- Same procedure as for workflows, except that forks and joins are rarely used in this context
- Used only for complex operations: sometimes an algorithm is sufficient

Example



Hints and Tips

- As for all other diagrams, an activity diagram:
 - has a focus
 - conveys information consistent with its level of abstraction
 - should minimize line crossings
- Activity states can be exploded into activity diagrams to convey more detailed information

Part IV

State Machines and State Charts

State Machines

- A **state machine** specifies the sequence of states an object goes through during its lifetime *in response to events*, together with its response to those events
- A **state** is a typical situation during the lifetime of an object
- An **event** is an occurrence of a stimulus that can trigger a state transition
- A state is rendered as a rectangle with rounded corners and a transition as a solid directed line

Statechart Diagrams

- A statechart diagram is a state machine emphasizing the flow of control from state to state for a *single object*
- States in a statechart can be either simple states or composite states
- Statechart diagrams are typically used to model event-driven (reactive) objects
- Operations and workflows are better rendered using activity diagrams, which are better suited for modeling the flow of activities over time

States

- A state is a *situation* during the lifetime of an object during which it satisfies some conditions, performs some activity, or waits for some event
- An object remains in a state for a finite amount of time
- A state has several parts:
 - name
 - entry/exit actions
 - substates (possibly)
- A state machine has one starting state and zero or more final states. The initial state does not have a trigger event

Transitions

- A transition is a relationship between two states indicating that an object in the *source state* will perform certain *actions* and enter the target state when a specified *event* occurs and specified *conditions* are satisfied
 - **source state**: state affected by the transition
 - **trigger event**: the event whose reception by the object in the source state makes the transition eligible to fire
 - **guard**: boolean condition that is evaluated when the trigger event is received. The transition is fired only if the guard evaluates to true
 - **action**: operation that is performed when the transition is fired
 - **target state**: state that is active after the completion of the transition

Modeling the Lifetime of Objects

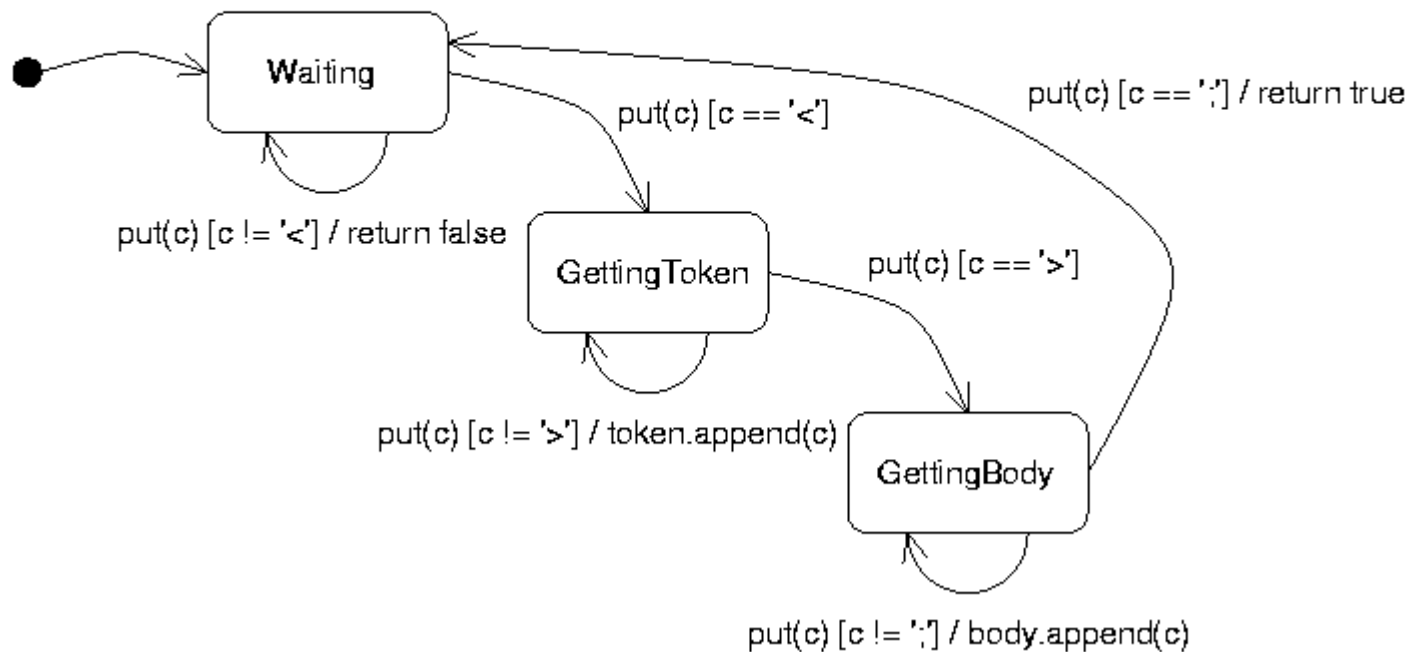
- Most common purpose for state machines
 - Set the context for the state machine, e.g. a class, use case, etc
 - Collect neighboring classes and consider those as candidate targets for actions and guard conditions
 - Establish the initial and final states of the object
 - Decide on the events to which this object may respond
 - starting from the initial state, lay out the states along with the appropriate transitions.
 - Add events and finally actions
 - Check that events and actions are supported by the neighboring classes
 - Check for unexpected sequences

Modeling Reactive Objects

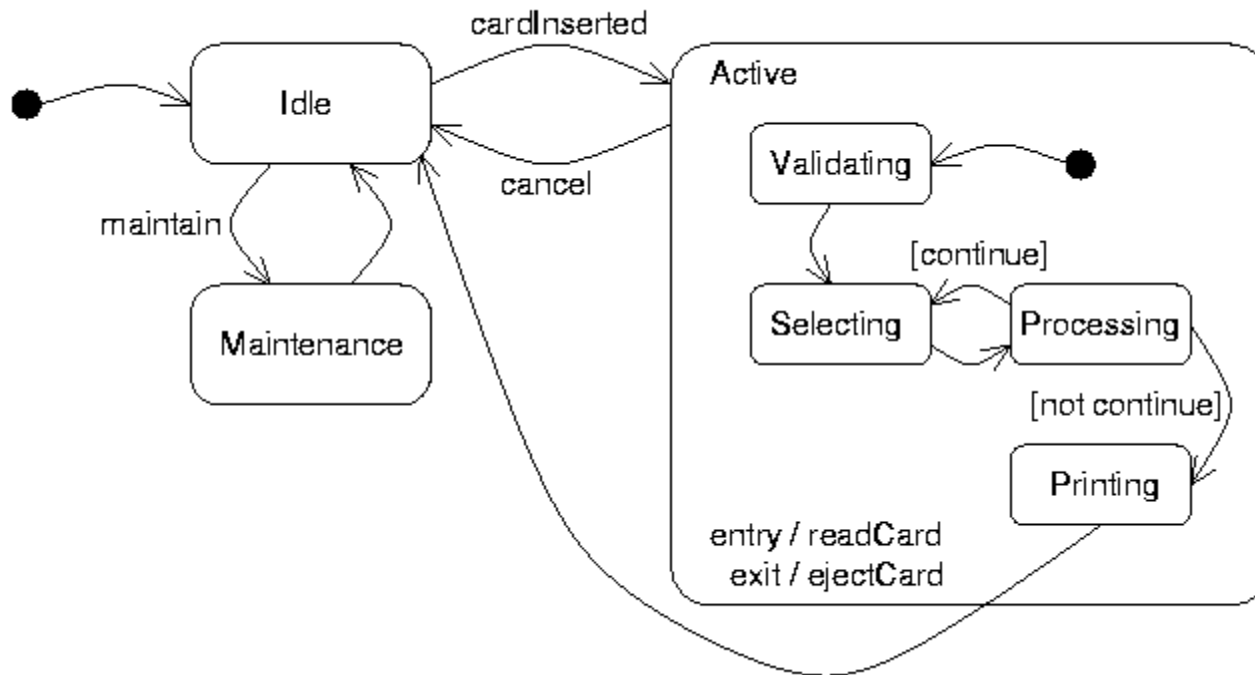
- A reactive system is a collection of objects reacting to events.
- Modeling a the behavior of a reactive object is about specifying three things:
 - the stable states in which that object may live
 - the events that trigger a transition between states
 - the actions that occur on each state change

Example

message ::= '<' string '>' string ';'



Example



Hints and Tips

- As for all other diagrams, a statechart diagram:
 - has a focus
 - conveys information consistent with its level of abstraction
 - should minimize line crossings
- Is not nested too deeply
- Should not make too much references to its context

Hints and Tips

- Sequence diagrams and collaboration diagrams provide a different view on the same information
- For complex systems, no single interaction diagram can convey all the information about the system's dynamic aspects
- Many interaction diagrams must be used to model the system as a whole, its subsystems, modules, classes, use cases and collaborations
- Each diagram must focus on its assigned goal

Hints and Tips

- The more diagrams we have, the more abstraction levels we need for comprehensibility
- Use sequence diagrams to represent the time ordering of messages in a restricted context e.g. scenarios
- Use collaboration diagrams to represent the structural organization of interacting object
- Use complex branching sparingly. It can be modeled best using activity diagrams.