UML Activity Diagrams, State-Machine Diagrams and Modelling

Lecture # 2

Department of Computer Science and Technology
University of Bedfordshire

Written by David Goodwin,
based on the book *Applying UML and Patterns (3rd ed.)*
by *C. Larman* (2005).

Modelling and Simulation, 2012

Outline

University of Bedfordshire

Activity Diagrams

What is an Activity Diagram?

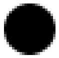
- An Activity Diagram is on of the Behaviour diargams.
- Activity modelling is the sequence and conditions for coordinating lower-level behaviours, rather than which classifiers own those behaviours.
- These are commonly called control flow and object flow models.
- The behaviours coordinated by these models can be initiated because other behaviours finish executing, because objects and data become available, or because events occur external to the flow.
- A UML Activity Diagram shows sequential and parallel activities in a process.
- Useful for modelling:
  - Business processes
  - Workflows
  - Data flows
  - Complex algorithms

Basic UML Activity Diagram notation

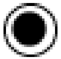
Action	It does something. There is automatic transition on its completion. A transition (arrows between actions) supports modelling of control flow.
Fork	One incoming transition, and multiple outgoing parallel transitions and/or object flows.
Join	Multiple incoming transitions and/or object flows; one outgoing transitions. The outgoing continuation does not happen until all the inputs arrive from all flows.
Object Node	An object produced or used by actions. This allows us to model data flows or object flows.

Initial and Final Nodes

- Initial Node:
  - An initial node is a control node at which flow starts when the activity is invoked.
  - An activity may have more than one initial node.

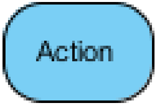


- Final Node:
  - An activity may have more than one activity final node; the first one reached stops all flows in the activity.



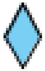
Action

- Action:
  - An action represents a single step within an activity that is not further decomposed within the activity.
  - An activity represents a behaviour that is composed of individual elements that are actions.
  - An action is simple from the point of view of the activity containing it, but may be complex in its effect and not be atomic.
  - An activity can be reused in many places, whereas an instance of an action is only used once at a particular point in an activity.
  - An action will not begin execution until all of its input conditions are satisfied.



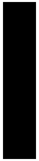
Merge and Decision Nodes

- Merge Node:
  - A merge node is a control node that brings together multiple alternate flows.
  - It is not used to synchronize concurrent flows but to accept one among several alternate flows.
  - A merge node has multiple incoming edges and a single outgoing edge.
- Decision Node:
  - A decision node accepts tokens on an incoming edge and presents them to multiple outgoing edges.
  - Which of the edges is actually traversed depends on the evaluation of the guards on the outgoing edges.



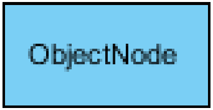
Join and Fork Nodes

- Join Node:
  - A join node is a control node that synchronizes multiple flows.
  - A join node has multiple incoming edges and one outgoing edge.
- Fork Node:
  - A fork node is a control node that splits a flow into multiple concurrent flows.
  - A fork node has one incoming edge and multiple outgoing edges.



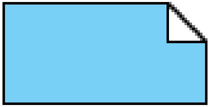
Object Node

- Object Node:
  - An object node is an activity node that indicates an instance of a particular classifier, possibly in a particular state, may be available at a particular point in the activity.
  - Object nodes can be used in a variety of ways, depending on where objects are flowing from and to.



Note

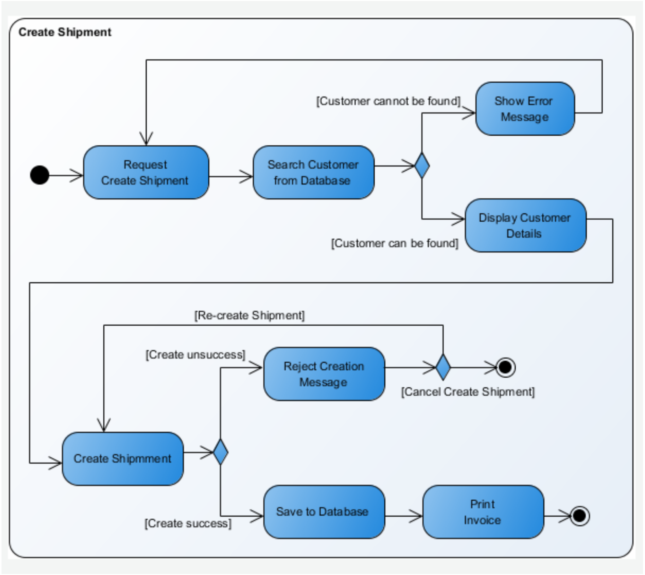
- Note:
  - A note (comment) gives the ability to attach various remarks to elements.
  - A comment carries no semantic force, but may contain information that is useful to a modeller.



Buisness Process Modelling

- Example: Parcel shipping
  - The process of shipping a parcel is non-trivial; there are many parties involved (customer, driver,. . .) and many steps.
  - The process can be captured by a Use Case diagram, but activity diagrams are great example of “a picture being worth a thousand words”.
  - Object nodes are useful for illustrating what is moving around.

Activity Diagram Parcel shipping



Guideline for Activity Modelling

- The technique proves most valuable for very complex processes, usually involving many parties.
- On a first overview "level 0" diagram, keep all the actions at a very high level of abstraction, so that the diagram is short. Then expand details in sub-diagrams at the "level 1" level,. . . etc.
- Try to make the level of abstraction of action nodes roughly equal withn a diagram (Very different levels of abstraction might be a node labelled "Deliver Order" and a node labelled "Calculate Tax").

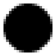
State-Machine Diagrams

What is a State-Machine Diagram?

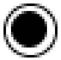
- State-Machine Diagrams specify state machines.
- As with Activity Diagrams, UML State-Machine Diagrams show a dynamic flow.
- The UML includes notation to illustrate events and states of things (transactions, Use Cases, people,...etc.).
    - An **event** is a significant or noteworthy occurence *e.g. a telephone receiver is taken off the hook*.
    - A **state** is the condition of an object at a moment in time *e.g. a telephone is in the state of being "idle" after the receiver is place on the hook and until it is taken off the hook*.
    - A **transition** is a relationship between two states that indicates when an event occurs *e.g. when the event "off hook" occurs, transition the telephone from "idle" to "active" state*.

Initial Pseudo State and Final State

- Initial Pseudo State:
  - An initial pseudostate represents a default vertex that is the source for a single transition to the default state.
  - There can be at most one initial vertex in a region.

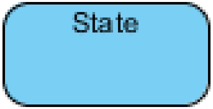


- Final State:
  - A special kind of state signifying that the enclosing region is completed, leading to the entire state machine being completed.



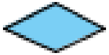
State

- State:
  - A state models a situation during which some (usually implicit) invariant condition holds.
  - The invariant may represent a static situation such as an object waiting for some external event to occur.
  - However, it can also model dynamic conditions such as the process of performing some behaviour (i.e., the model element under consideration enters the state when the behaviour commences and leaves it as soon as the behaviour is completed).



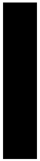
Choice

- Choice:
  - choice vertices result in the dynamic evaluation of the guards of the triggers of its outgoing transitions.
  - It allows splitting of transitions into multiple outgoing paths.
  - If more than one of the guards evaluates to true, an arbitrary one is selected. If none of the guards evaluates to true, then the model is considered ill-formed. (To avoid this define one outgoing transition with the predefined else guard for every choice vertex).



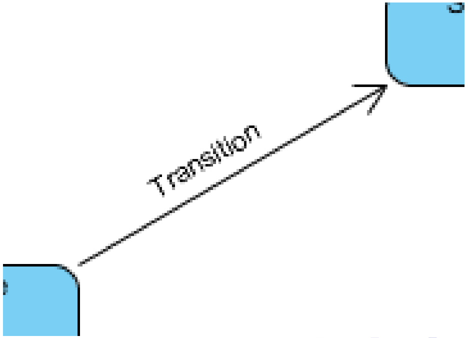
Join and Fork

- Join:
  - Join vertices serve to merge several transitions emanating from source vertices in different orthogonal regions.
  - The transitions entering a join vertex cannot have guards or triggers.
- Fork:
  - Fork vertices serve to split an incoming transition into two or more transitions terminating on orthogonal target vertices.
  - The segments outgoing from a fork vertex must not have guards or triggers.



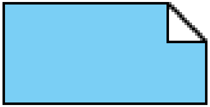
Transition

- Transition:
  - A transition is a directed relationship between a source vertex and a target vertex.
  - It may be part of a compound transition, which takes the state machine from one state configuration to another, representing the complete response of the state machine to an occurrence of an event of a particular type.



Note

- Note:
  - A note (comment) gives the ability to attach various remarks to elements.
  - A comment carries no semantic force, but may contain information that is useful to a modeller.



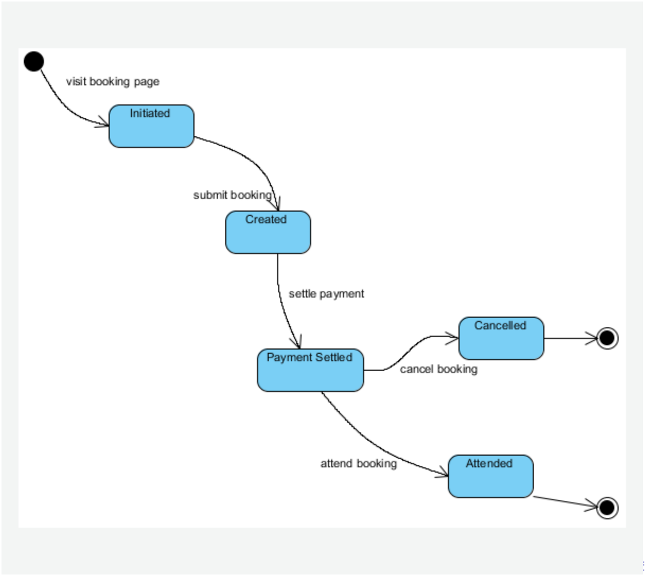
State-Independent and State Dependent Objects

- If an object always responds the same way to an event, then it is considered **state-independent** with respect to that event.
- If for all events of interest an object always reacts the same way, it is a **state-independent object**.
- By contrast **state-dependent objects** react differently to events depending on their state.

State-Independent and State Dependent Objects

- Business information systems have few state-dependent classes, so it is not helpful to apply a state machine modelling.
- Process control, device control, protocol handlers, and telecommunication domains often have many state-dependent objects; state machine modelling would be useful in these cases.
    - e.g. a telephone is state-dependent; the phone's reaction to pushing a particular button depends on the current mode of the phone (off hook, engaged,. . . etc).

Example: Booking a psychiatrist appointment

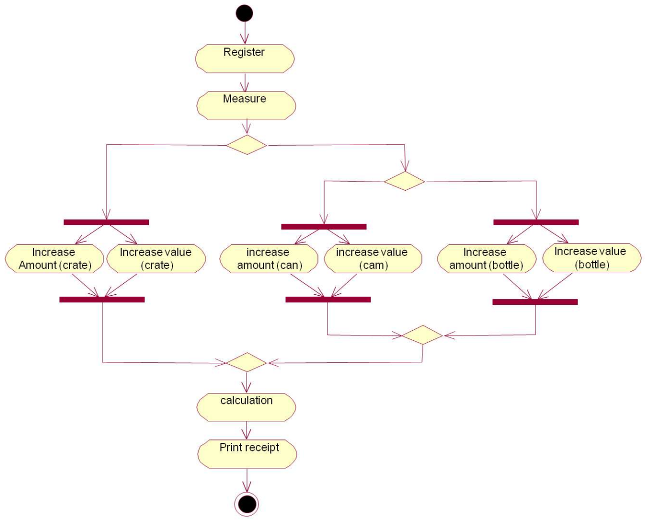


Further Examples

Recycling machine - deposit items

- When a customer returns his/her first deposit item, the machine will **register** the customer.
- The machine will **measure** the returned item. The measurements are used to determine what kind of can, bottle or crate has been returned. If acceptable, the total number of items and the total value of this type returned by the customer **increments**.
- When the customer presses the receipt button, the printer **prints** the date. The total number of items he returned and the lump sum is **calculated**. The followings are printed out: customer number, number returned, deposit value, total of this type and lump sum.

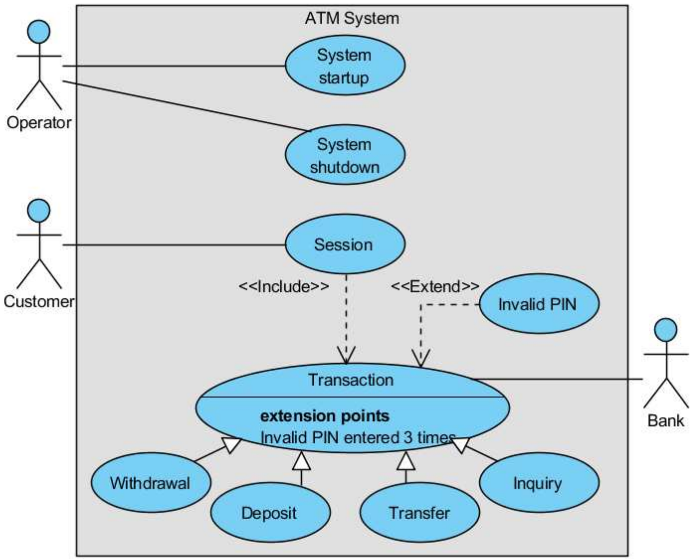
Recycling Activity Diagrams



Recycling machine - changing values. . .

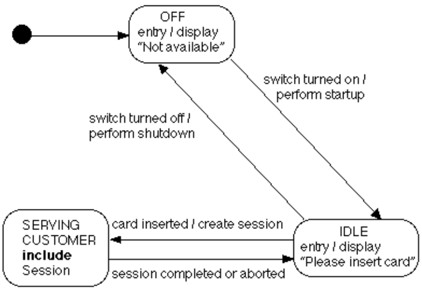
- When an operator changes items' values, the machine should first **display a menu** which contains three options: crate, can and bottle.
- When the operator chooses one option, the machine should **display** the current value of the corresponding item and a range in which the value can vary.
- If the operator **enters a value** that is within the range, the machine will ask the operator to **confirm** the change and then update the value of the item stored in the machine.

ATM Use Case


ATM System
Operator
System startup
System shutdown
Customer
Session
<<Include>>
<<Extend>>
Invalid PIN
Bank
Transaction
extension points
Invalid PIN entered 3 times
Withdrawal
Deposit
Transfer
Inquiry

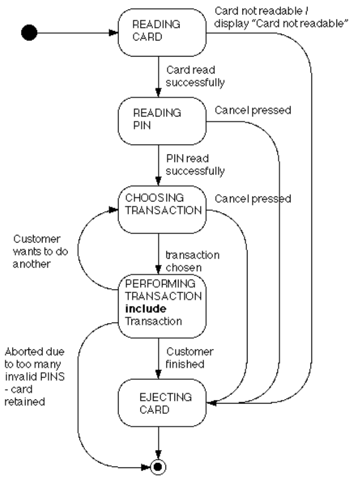
ATM State-machine diagrams

State-Chart for Overall ATM (includes System Startup and System Shutdown Use Cases)


OFF
entry / display "Not available"
switch turned on / perform startup
switch turned off / perform shutdown
IDLE
entry / display "Please insert card"
card inserted / create session
session completed or aborted
SERVING CUSTOMER
include Session

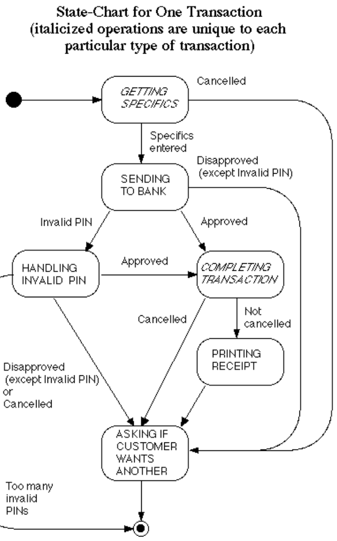
ATM State-machine diagrams

State-Chart for One Session



READING CARD
Card not readable / display "Card not readable"
Card read successfully
READING PIN
Cancel pressed
PIN read successfully
CHOOSING TRANSACTION
Cancel pressed
Customer wants to do another
transaction chosen
PERFORMING TRANSACTION include Transaction
Aborted due to too many invalid PINS - card retained
Customer finished
EJECTING CARD

ATM State-machine diagrams


State-Chart for One Transaction
(italicized operations are unique to each particular type of transaction)
GETTING SPECIFICS
Cancelled
Specifics entered
SENDING TO BANK
Disapproved (except Invalid PIN)
Invalid PIN
Approved
HANDLING INVALID PIN
Approved
COMPLETING TRANSACTION
Cancelled
Not cancelled
PRINTING RECEIPT
Disapproved (except Invalid PIN) or Cancelled
ASKING IF CUSTOMER WANTS ANOTHER
Too many invalid PINs