Software Engineering

Lesson #07 - Practice

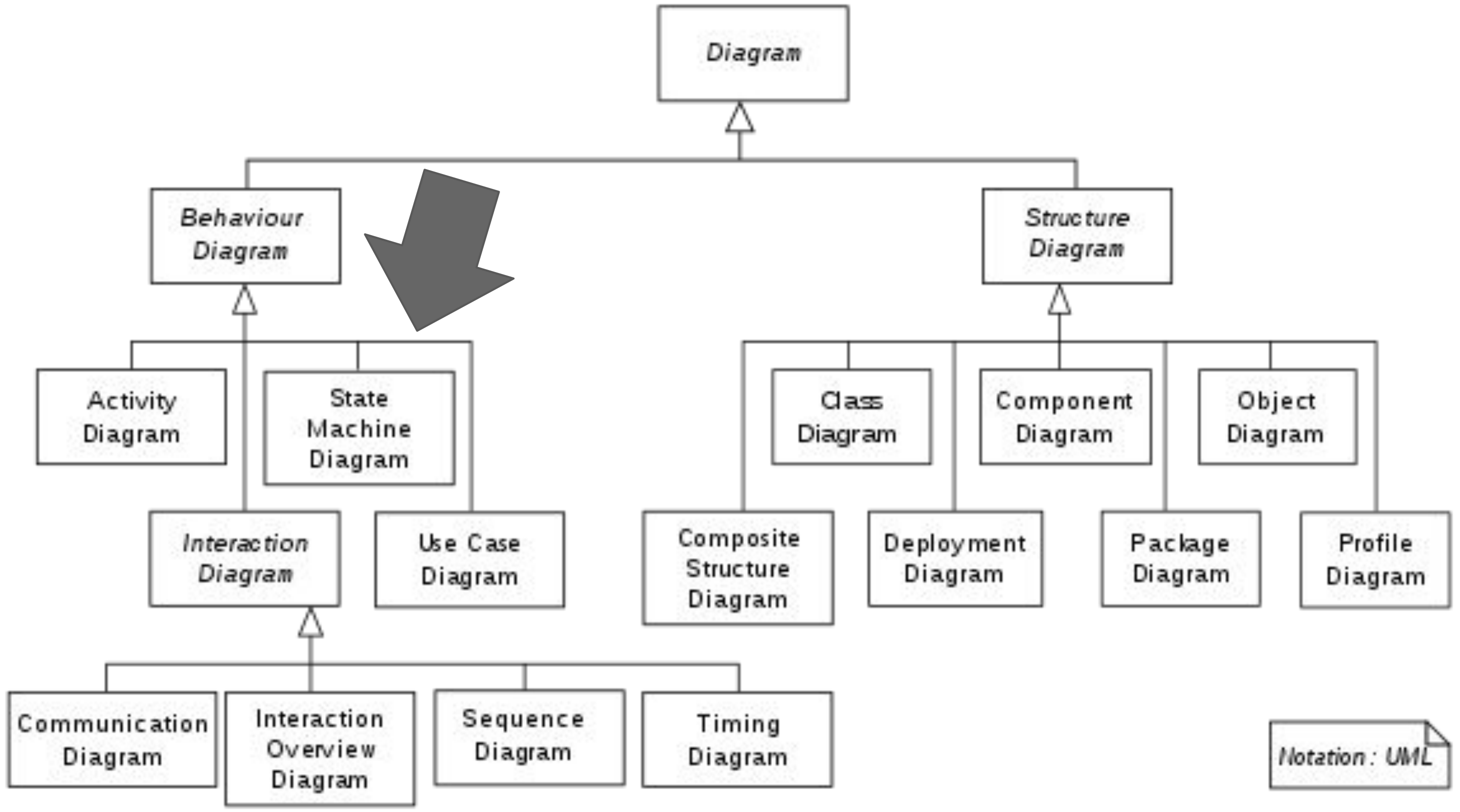
Lesson #07 - Practice

Agenda: Lesson #07 - Software Engineering - Practice

1	State Machine Diagrams
2	Activity Diagrams
3	Class Work
4	Q & A

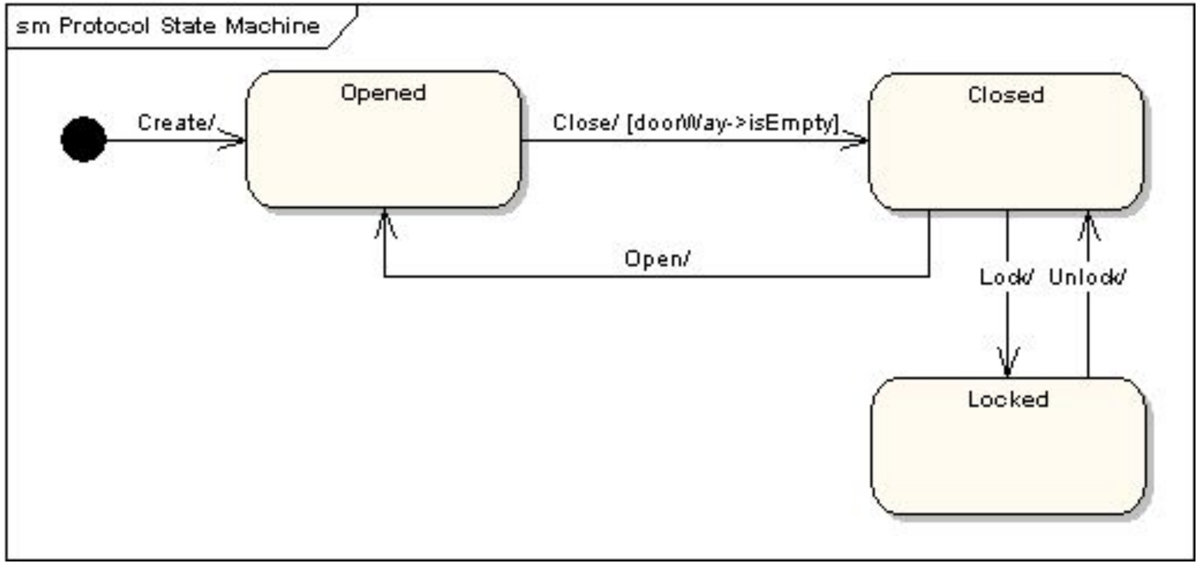
Agenda: Lesson #07 - Software Engineering - Practice

State Machine Diagrams



State Machine Diagrams

State machine diagrams are a familiar technique to describe the behavior of a system



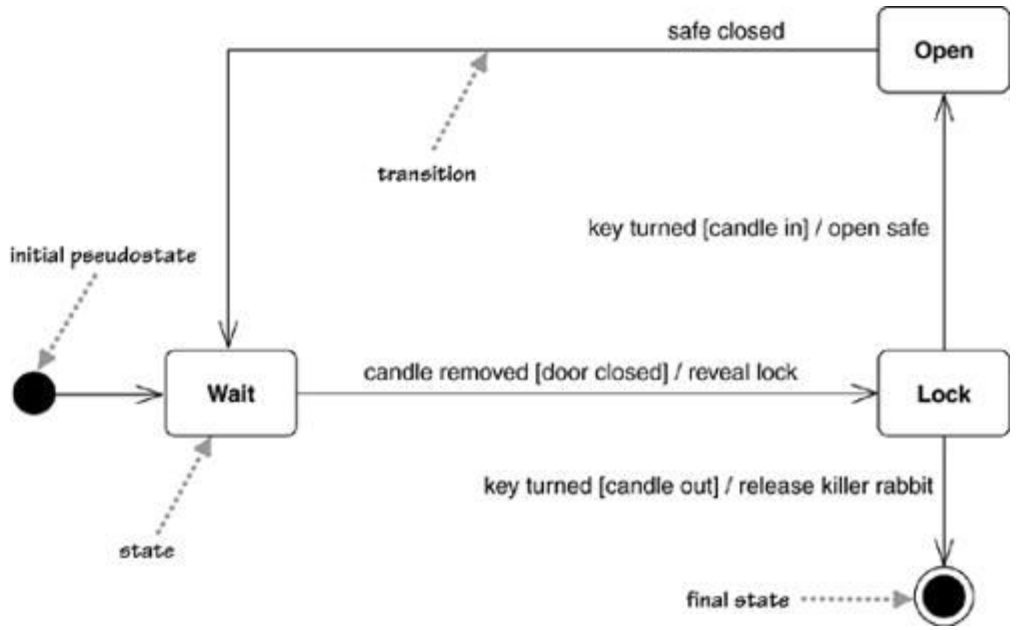
State Machine Diagrams

Various forms of state diagrams have been around since the 1960s and the earliest object-oriented techniques adopted them to show behavior

In object-oriented approaches, you draw a state machine diagram for a single class to show the lifetime behavior of a single object

State Machine Diagrams

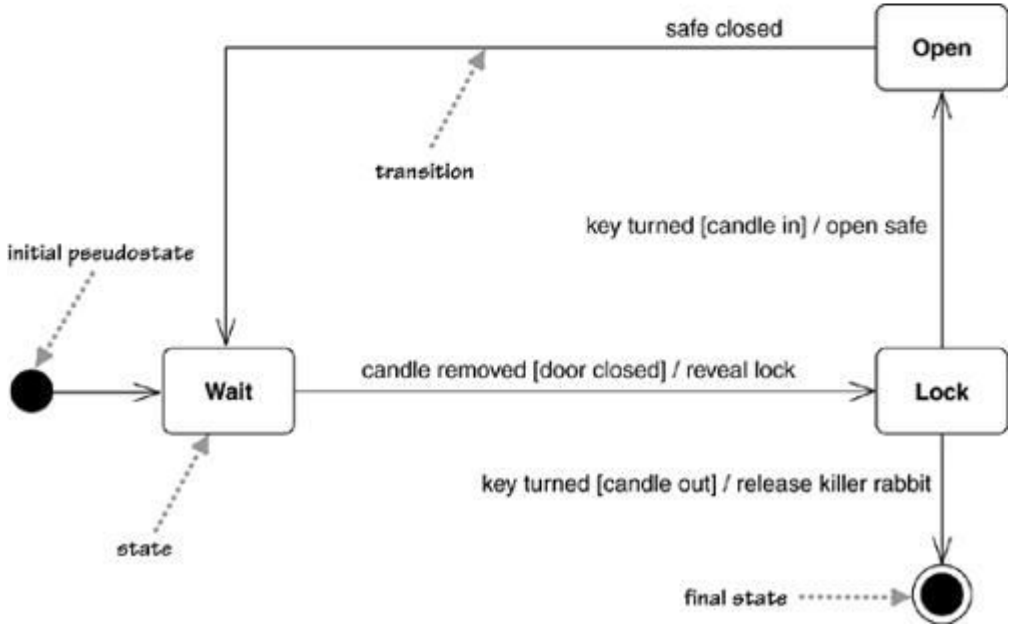
The diagram shows that the controller can be in three states: Wait, Lock, and Open


safe closed
Open
transition
key turned [candle in] / open safe
initial pseudostate
Wait
candle removed [door closed] / reveal lock
Lock
key turned [candle out] / release killer rabbit
state
final state

State Machine Diagrams

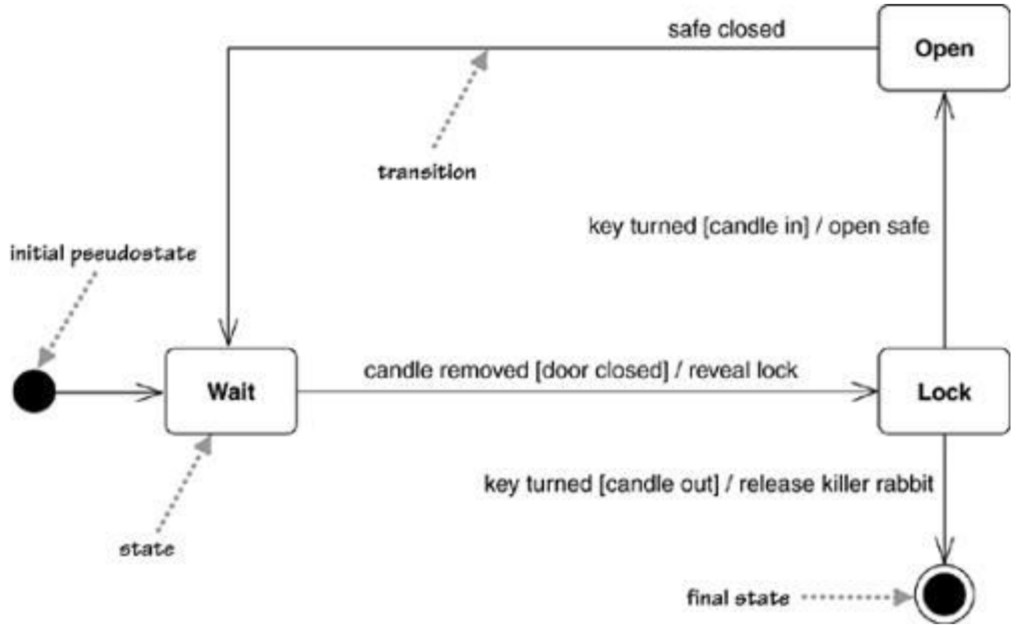
The diagram also gives the rules by which the controller changes from state to state

These rules are in the form of transitions: the lines that connect the states


safe closed
Open
transition
key turned [candle in] / open safe
initial pseudostate
Wait
candle removed [door closed] / reveal lock
Lock
key turned [candle out] / release killer rabbit
state
final state

State Machine Diagrams

The transition indicates a movement from one state to another


safe closed
Open
transition
key turned [candle in] / open safe
initial pseudostate
Wait
candle removed [door closed] / reveal lock
Lock
key turned [candle out] / release killer rabbit
state
final state

State Machine Diagrams

Internal Activities

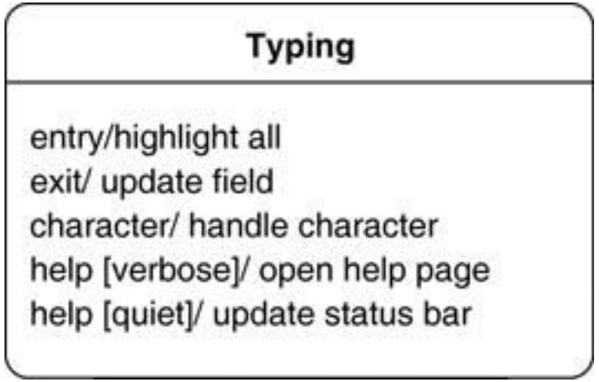
- States can react to events without transition, using internal activities: putting the event, guard, and activity inside the state box itself

State Machine Diagrams

Internal Activities

- Diagram shows two special activities: the entry and exit activities. The entry activity is executed whenever you enter a state; the exit activity, whenever you leave

- However, internal activities do not trigger the entry and exit activities; that is the difference between internal activities and self-transitions


Typing
entry/highlight all
exit/ update field
character/ handle character
help [verbose]/ open help page
help [quiet]/ update status bar

State Machine Diagrams

Activity States

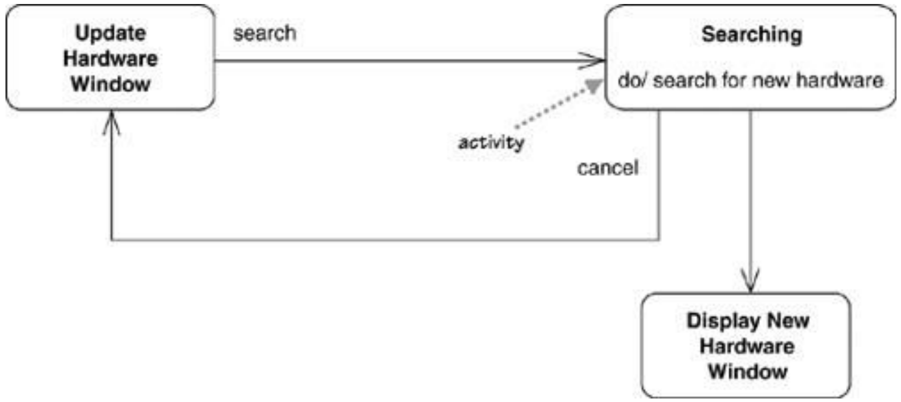
- In the states I've described so far, the object is quiet and waiting for the next event before it does something

- However, you can have states in which the object is doing some ongoing work

State Machine Diagrams

Activity States

The Searching state in diagram is such an activity state: The ongoing activity is marked with the do/; hence the term do-activity

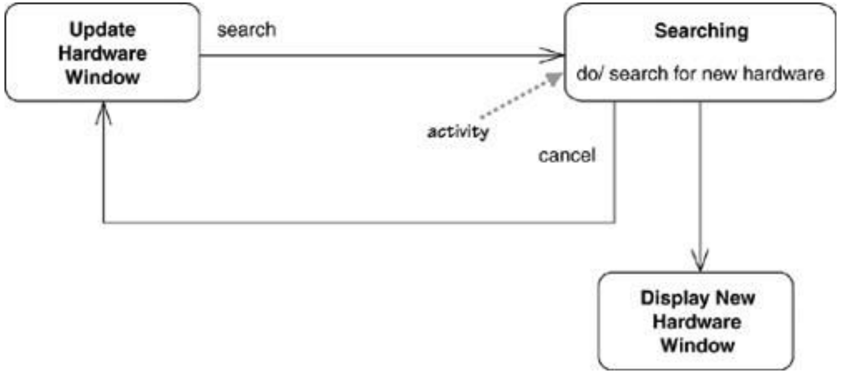


State Machine Diagrams

Activity States

Once the search is completed, any transitions without an activity, such as the one to display new hardware, are taken

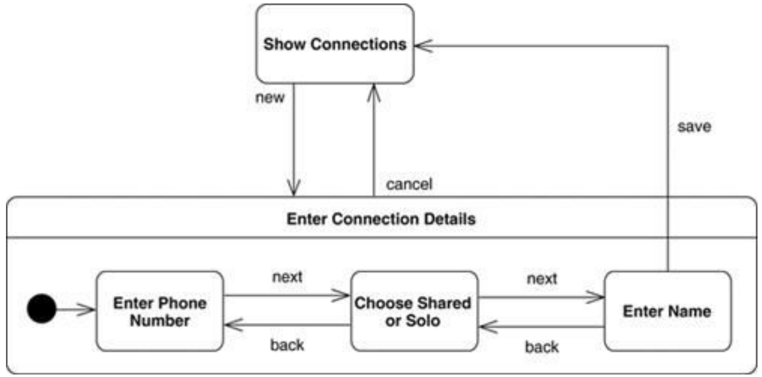
If the cancel event occurs during the activity, the do-activity is unceremoniously halted, and we go back to the Update Hardware Window state



State Machine Diagrams

Superstates

- Often, you'll find that several states share common transitions and internal activities
- In these cases, you can make them sub-states and move the shared behavior into a superstate, as in diagram

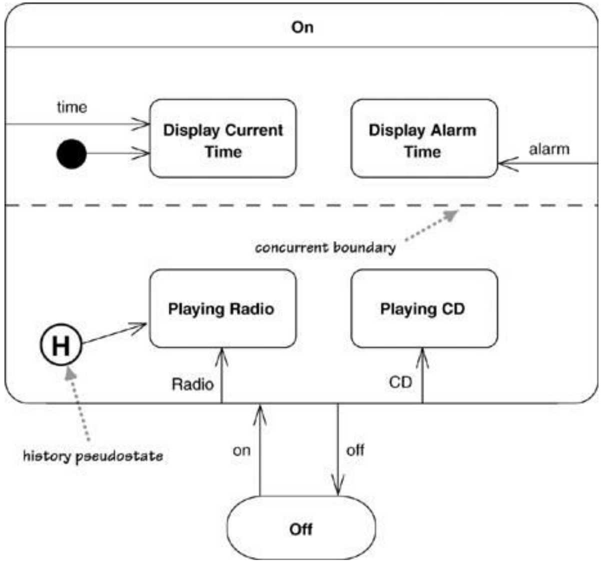


State Machine Diagrams

Concurrent States

- States can be broken into several orthogonal state diagrams that run concurrently

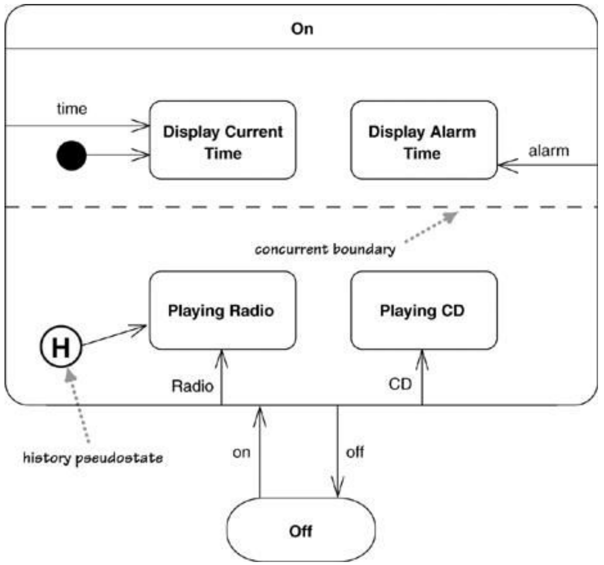
- Diagram shows a pathetically simple alarm clock that can play either CDs or the radio and show either the current time or the alarm time



State Machine Diagrams

Concurrent States

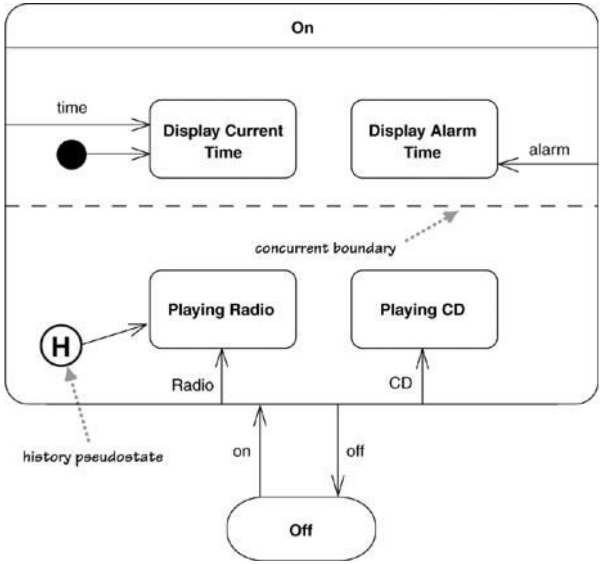
- Diagram also includes a history pseudostate
- This indicates that when the clock is switched on, the radio/CD choice goes back to the state the clock was in when it was turned off



State Machine Diagrams

Concurrent States

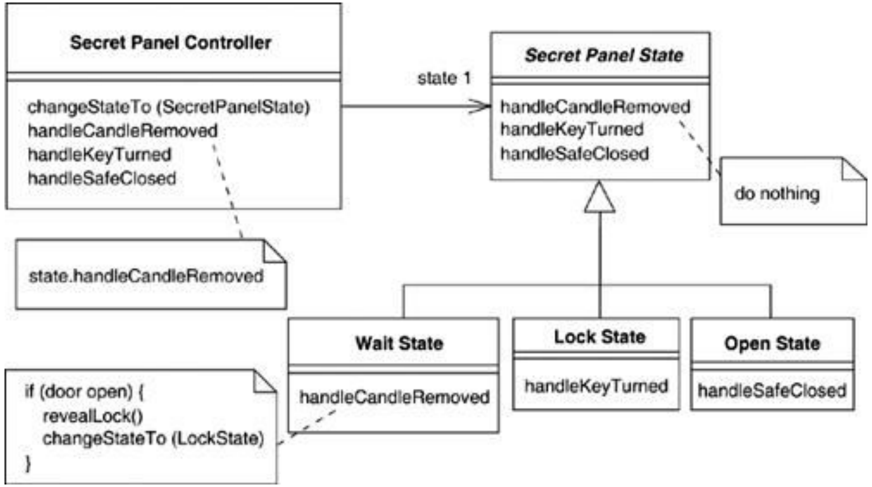
- The arrow from the history pseudostate indicates what state to be in on the first time when there is no history



State Machine Diagrams

Implementing State Diagrams

- A state diagram can be implemented in three main ways: nested switch, the State pattern, and state tables



When to Use State Diagrams

State diagrams are good at describing the behavior of an object across several use cases

State diagrams are not very good at describing behavior that involves a number of objects collaborating

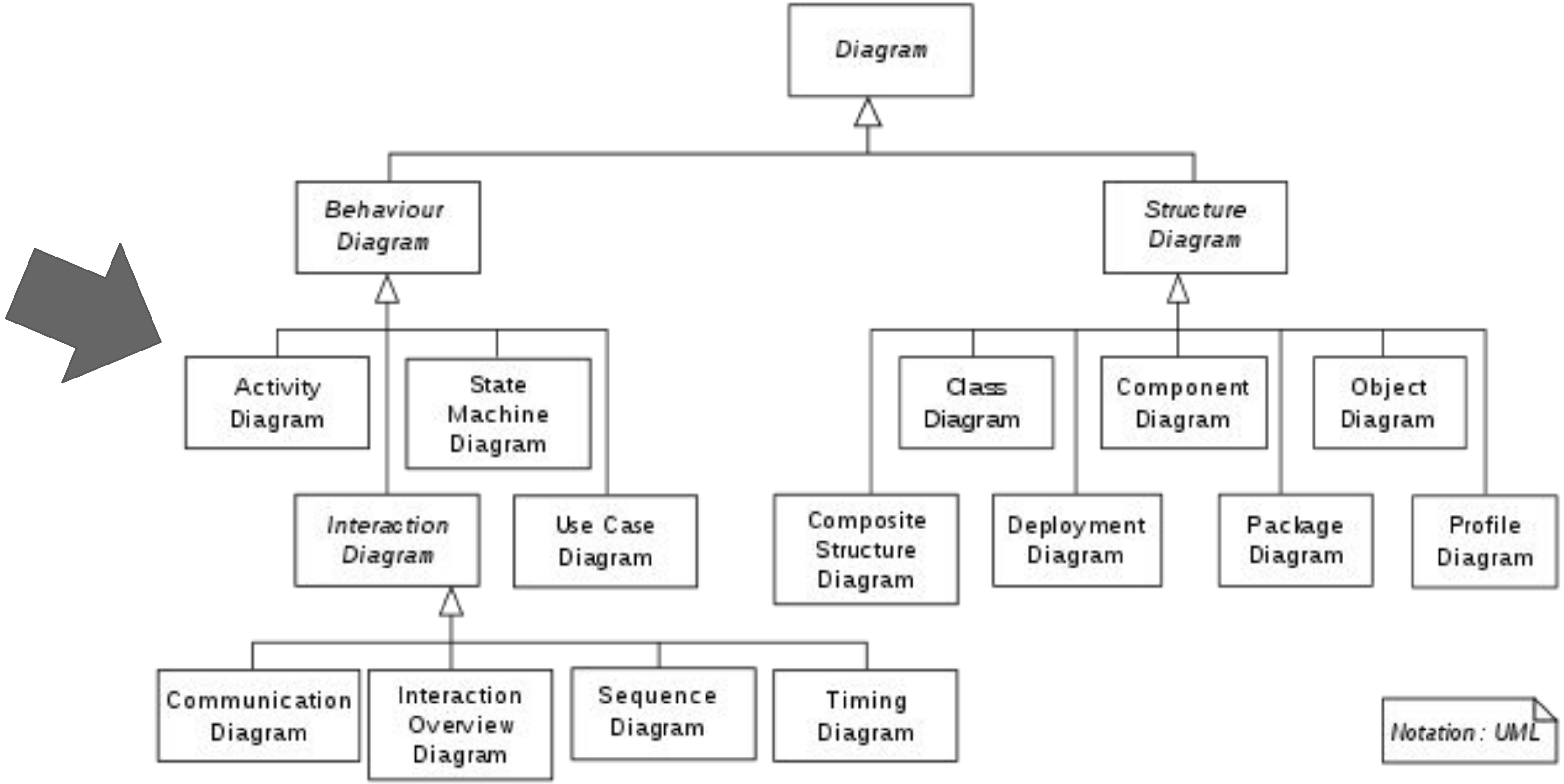
As such, it is useful to combine state diagrams with other techniques

When to Use State Diagrams

For instance, interaction diagrams are good at describing the behavior of several objects in a single use case, and activity diagrams are good at showing the general sequence of activities for several objects and use cases

Agenda: Lesson #07 - Software Engineering - Practice

Activity Diagrams



Activity Diagrams

Activity diagrams are a technique to describe procedural logic, business process, and workflow

In many ways, they play a role similar to flowcharts, but the principal difference between them and flowchart notation is that they support parallel behavior

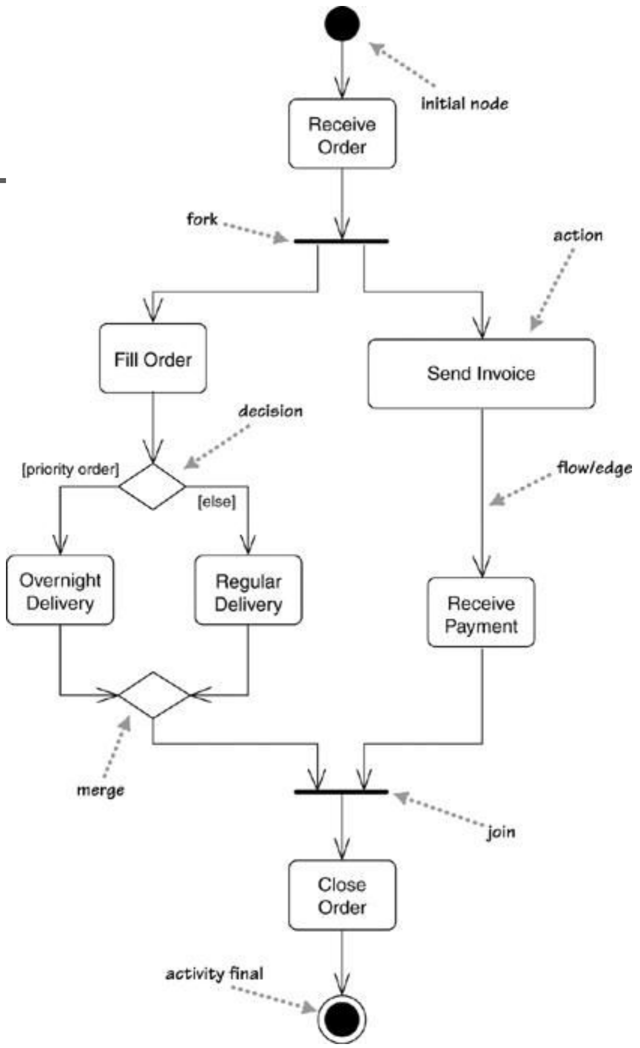
Activity Diagrams

Diagram shows a simple example of an activity diagram

We begin at the initial node action and then do the action Receive Order

Once that is done, we encounter a fork

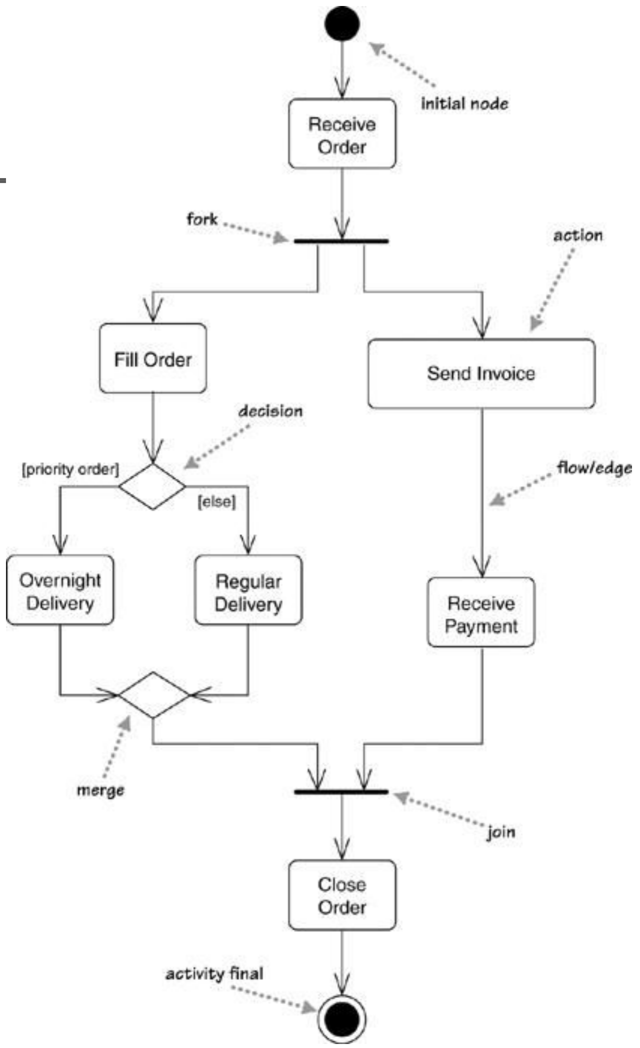
A fork has one incoming flow and several outgoing concurrent flows



Activity Diagrams

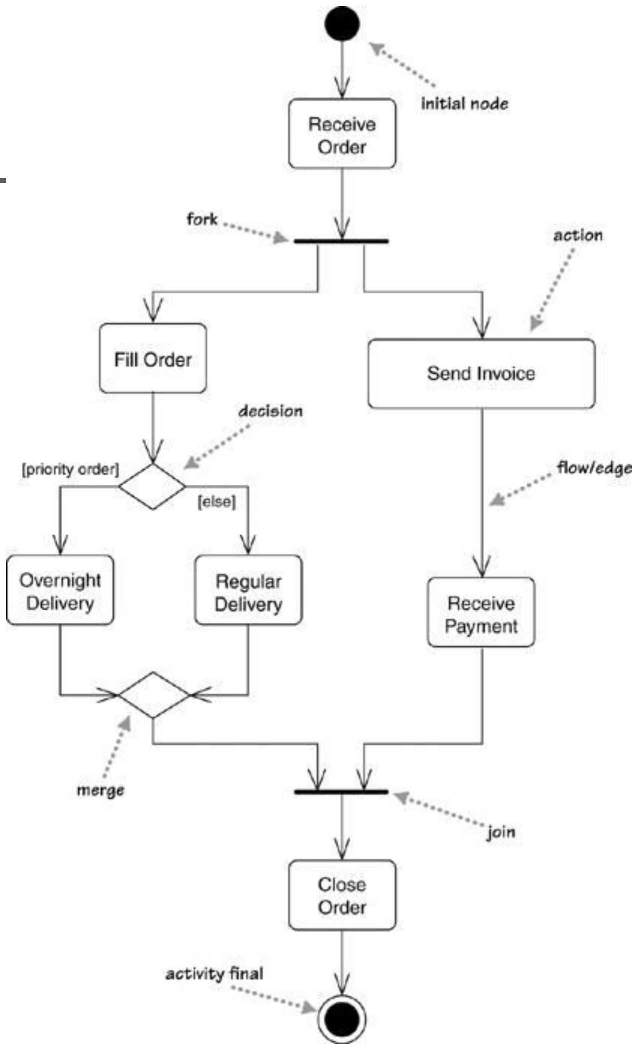
Diagram says that Fill Order, Send Invoice, and the subsequent actions occur in parallel

Essentially, this means that the sequence between them is irrelevant



Activity Diagrams

I could fill the order, send the invoice, deliver, and then receive payment; or, I could send the invoice, receive the payment, fill the order, and then deliver: You get the picture



Activity Diagrams

The activity diagram allows whoever is doing the process to choose the order in which to do things

In other words, the diagram merely states the essential sequencing rules I have to follow

This is important for business modeling because processes often occur in parallel

Activity Diagrams

It's also useful for concurrent algorithms, in which independent threads can do things in parallel

When you have parallelism, you'll need to synchronize

We don't close the order until it is delivered and paid for

Activity Diagrams

We show this with the join before the Close Order action

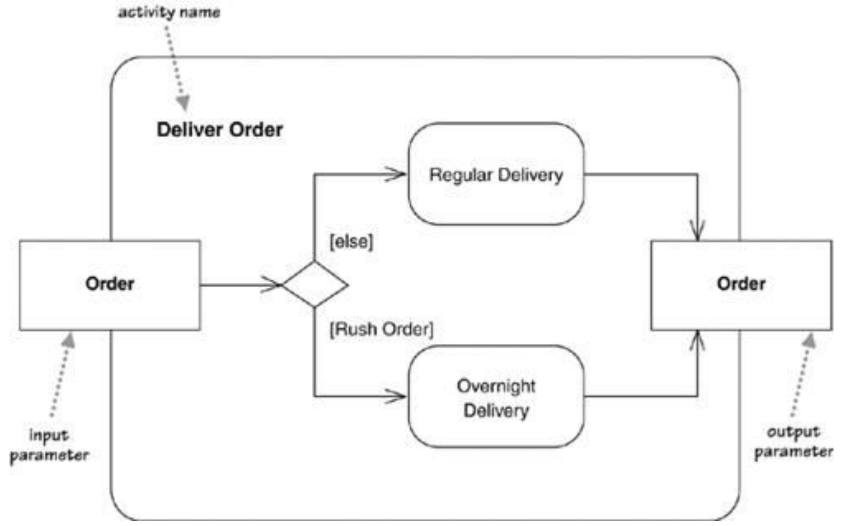
With a join, the outgoing flow is taken only when all the incoming flows reach the join

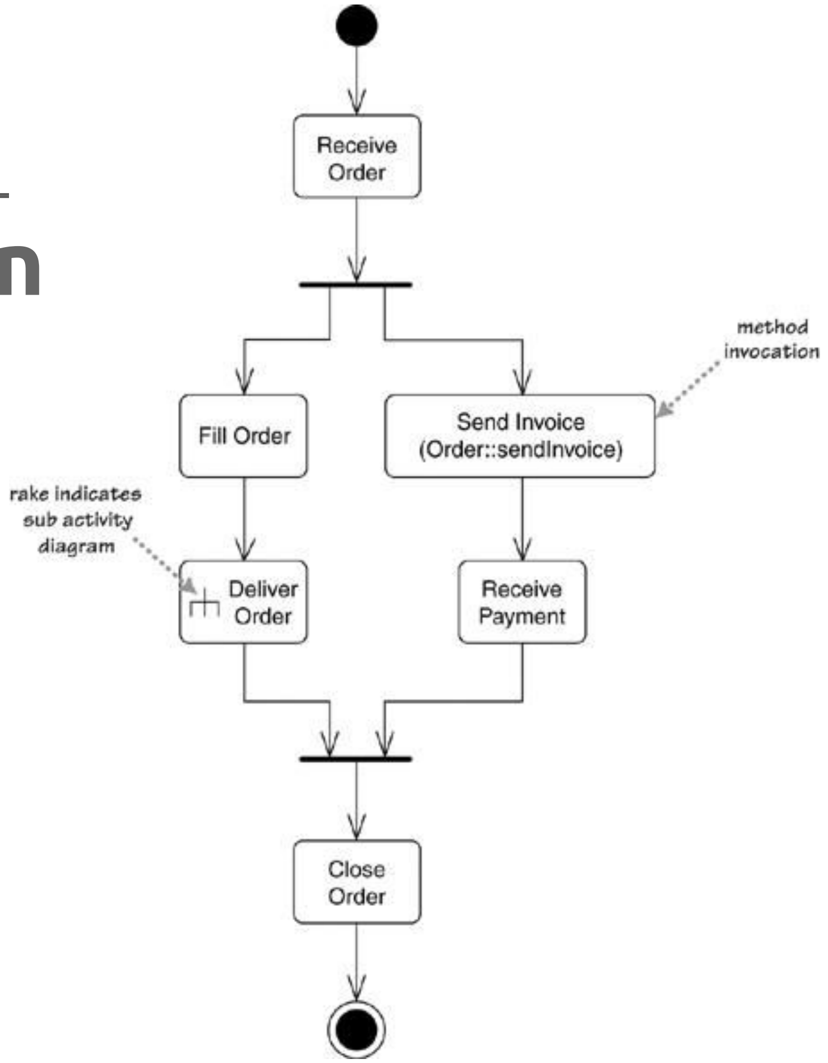
So you can close the order only when you have both received the payment and delivered

Activity Diagrams

Decomposing an Action

Actions can be decomposed into subactivities





When to Use Activity Diagrams

The great strength of activity diagrams lies in the fact that they support and encourage parallel behavior

This makes them a great tool for workflow and process modeling, and indeed much of the push in UML 2 has come from people involved in workflow

When to Use Activity Diagrams

You can also use an activity diagram as a UML-compliant flowchart. Although this allows you to do flowcharts in a way that sticks with the UML, it's hardly very exciting

In principle, you can take advantages of the forks and joins to describe parallel algorithms for concurrent programs

When to Use Activity Diagrams

The main strength of doing this may come with people using UML as a programming language

In this case, activity diagrams represent an important technique to represent behavioral logic

Activity Diagrams

Partitions

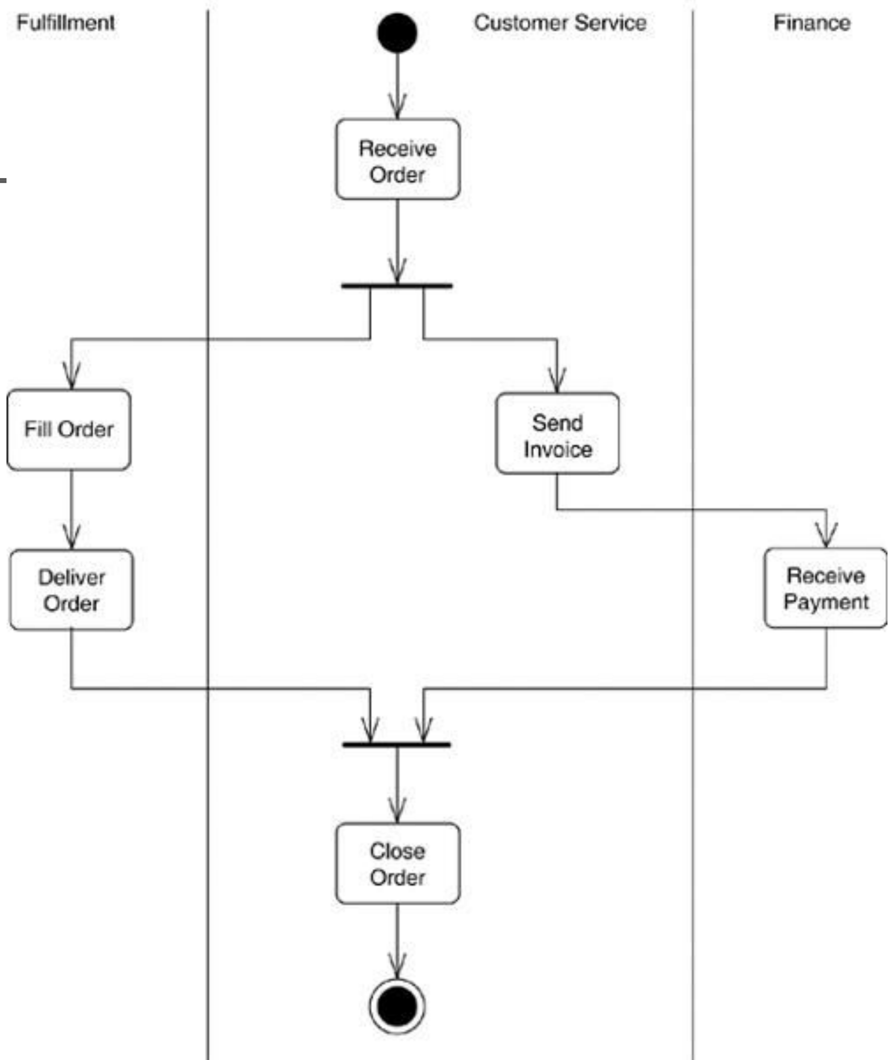
- Activity diagrams tell you what happens, but they do not tell you who does what
- In programming, this means that the diagram does not convey which class is responsible for each action
- In business process modeling, this does not convey which part of an organization carries out which action

Activity Diagrams

Activity Diagrams

Partitions

- Swim lanes



Activity Diagrams

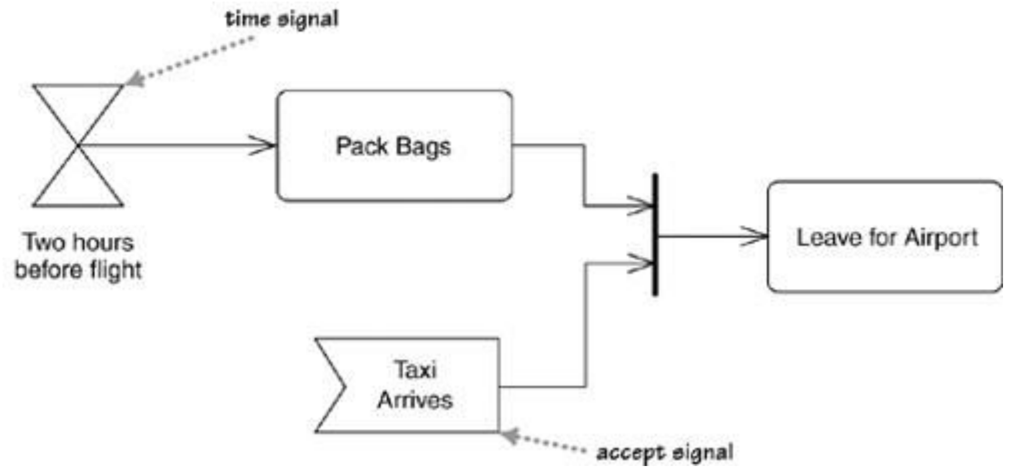
Signals

- A time signal occurs because of the passage of time
- Such signals might indicate the end of a month in a financial period or each microsecond in a real-time controller

Activity Diagrams

Signals

- Diagram shows an activity that listens for two signals



Activity Diagrams

Signals

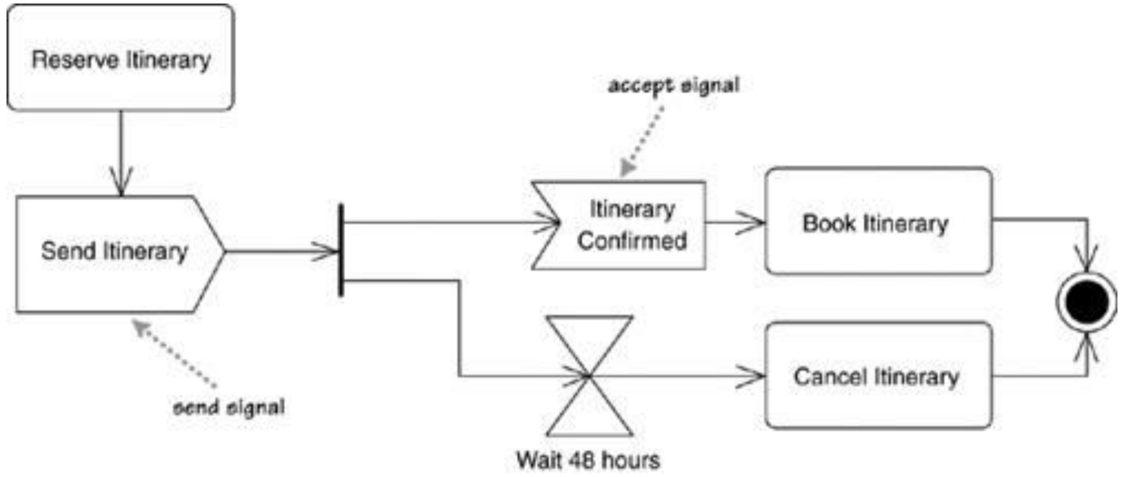
A signal indicates that the activity receives an event from an outside process

This indicates that the activity constantly listens for those signals, and the diagram defines how the activity reacts

Activity Diagrams

Signals

- Sending and receiving signals



Activity Diagrams

Tokens

- If you're sufficiently brave to venture into the demonic depths of the UML specification, you'll find that the activity section of the specification talks a lot about tokens and their production and consumption

- The initial node creates a token, which then passes to the next action, which executes and then passes the token to the next

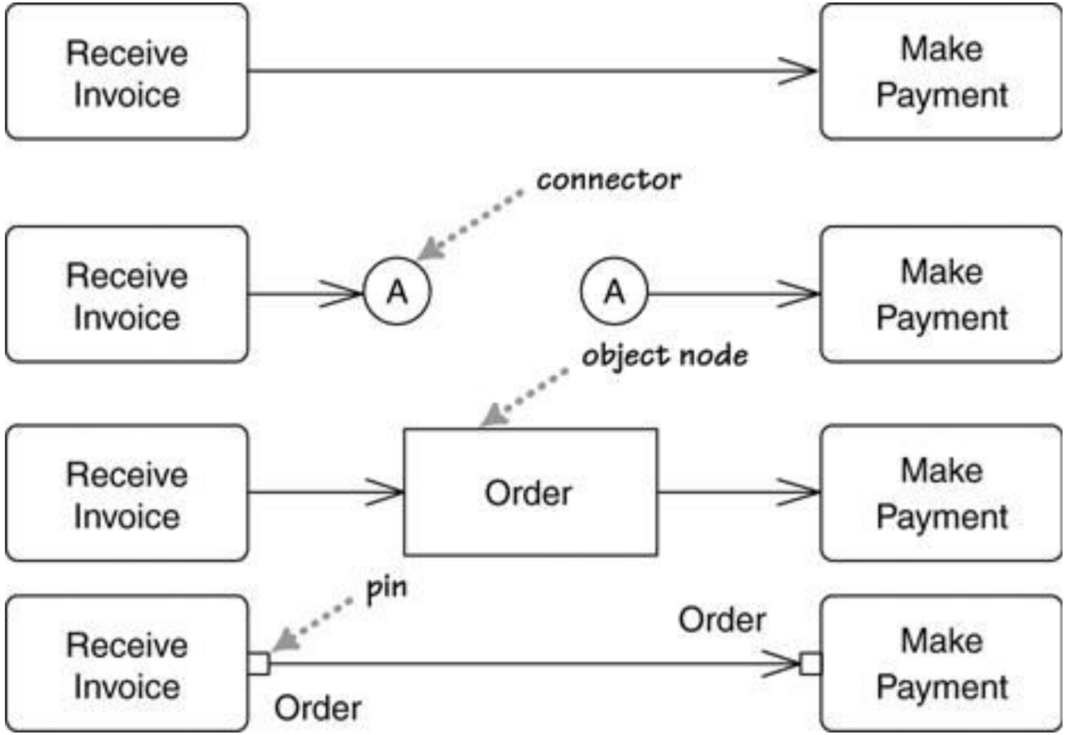
Activity Diagrams

Flows and Edges

- UML 2 uses the terms flow and edge synonymously to describe the connections between two actions
- The simplest kind of edge is the simple arrow between two actions
- You can give an edge a name if you like, but most of the time, a simple arrow will suffice

Activity Diagrams

Flows and Edges



Activity Diagrams

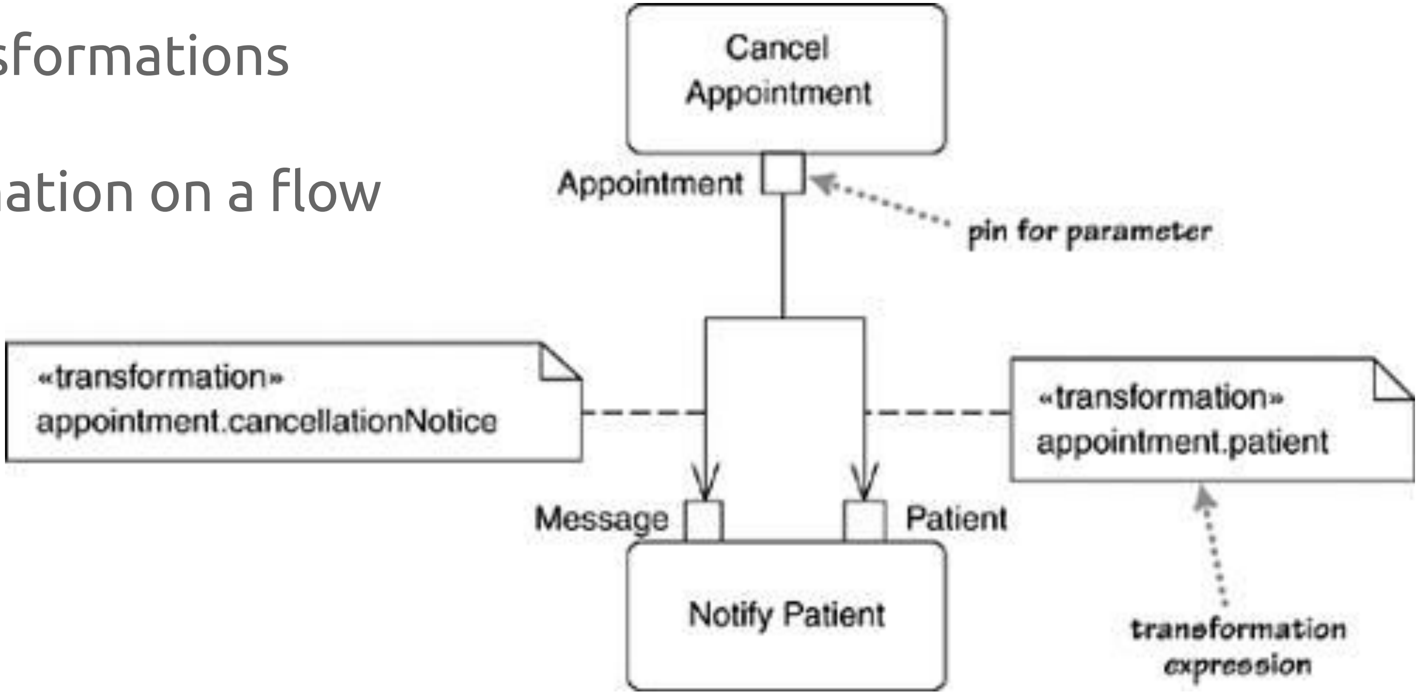
Pins and Transformations

- Actions can have parameters, just as methods do
- You don't need to show information about parameters on the activity diagram, but if you wish, you can show them with pins
- If you're decomposing an action, pins correspond to the parameter boxes on the decomposed diagram

Activity Diagrams

Pins and Transformations

- Transformation on a flow


Cancel Appointment
Appointment
pin for parameter
«transformation» appointment.cancellationNotice
«transformation» appointment.patient
Message
Patient
Notify Patient
transformation expression

Activity Diagrams

Expansion Regions

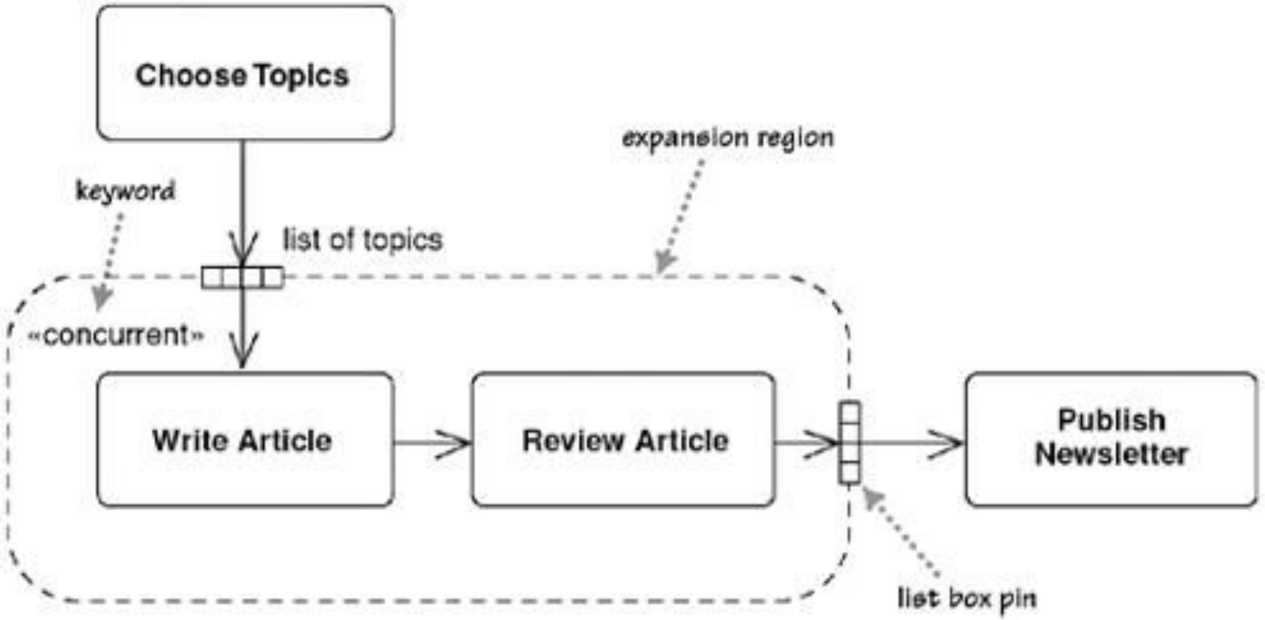
With activity diagrams, you often run into situations in which one action's output triggers multiple invocations of another action

There are several ways to show this, but the best way is to use an expansion region

An expansion region marks an activity diagram area where actions occur once for each item in a collection

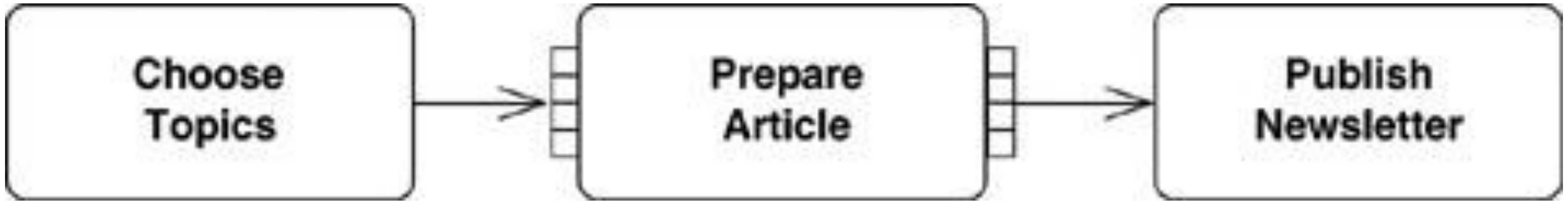
Activity Diagrams

Expansion Regions



Activity Diagrams

Expansion Regions



Activity Diagrams

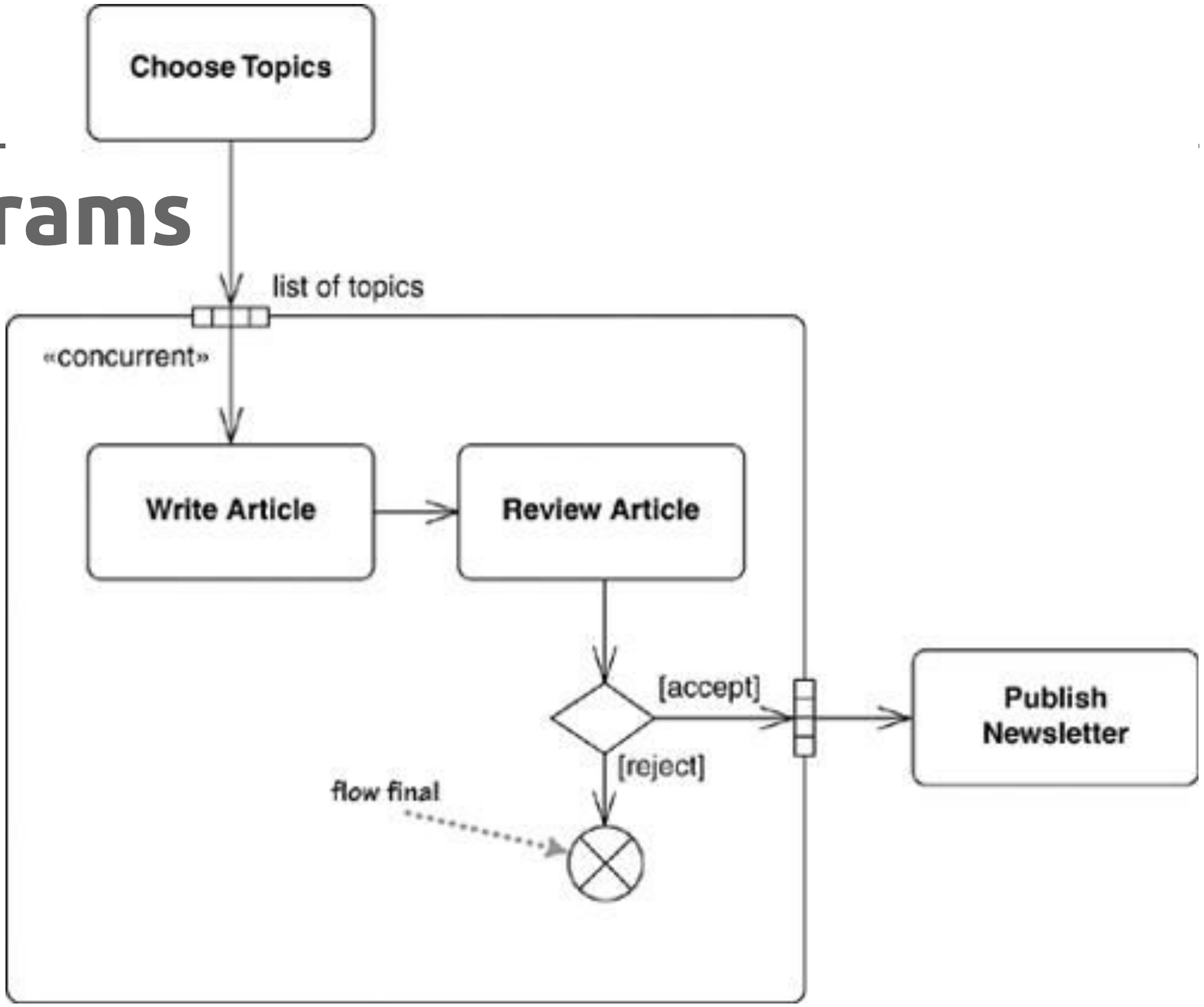
Flow Final

- Once you get multiple tokens, as in an expansion region, you often get flows that stop even when the activity as a whole doesn't end

- A flow final indicates the end of one particular flow, without terminating the whole activity

Activity Diagrams

Flow Final



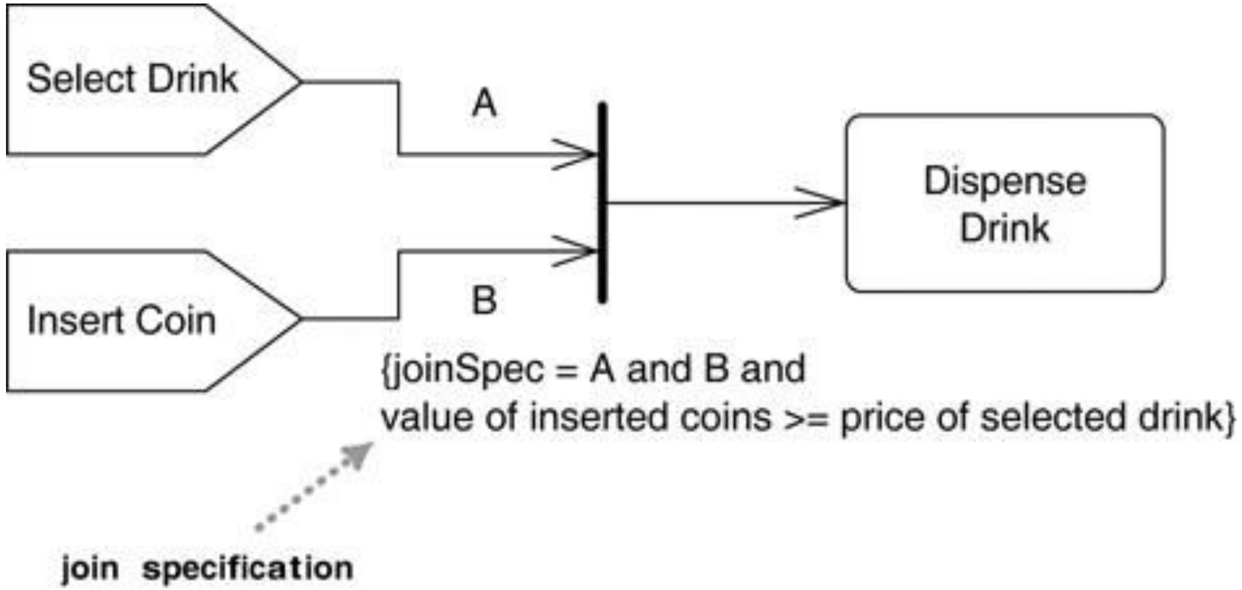
Activity Diagrams

Join Specifications

- A join specification is a Boolean expression attached to a join
- Each time a token arrives at the join, the join specification is evaluated and if true, an output token is emitted

Activity Diagrams

Join Specifications



Agenda: Lesson #07 - Software Engineering - Practice

Class work

Software Engineering, 10. Global Edition

At the end of class, please, send your work to z.aldamuratov@kbtu.kz, indicating Software Engineering Practice #07 & your surname and name

Class work

Software Engineering, 10. Global Edition

Tool: StarUML

You have been asked to develop a system that will help with planning large-scale events and parties such as weddings, graduation celebrations, and birthday parties. Using an activity diagram, model the process context for such a system that shows the activities involved in planning a party (booking a venue, organizing invitations, etc.) and the system elements that might be used at each stage.

Agenda: Lesson #07 - Software Engineering - Practice

1	State Machine Diagrams
2	Activity Diagrams
3	Class Work
4	**Q & A**

Agenda: Lesson #07 - Software Engineering - Practice

Q & A