Software modelling

- Models and Diagrams
- Diagram examples
  - Use-case diagrams
  - Class diagrams
  - Package diagrams
  - Sequence diagrams
  - Collaboration diagrams
  - Type diagrams
- Activity Diagrams

CE202 Software Engineering, Autumn term

Dr Cunjin Luo, School of Computer Science and Electronic Engineering, University of Essex

Introduction to software modelling

- Purpose of modelling
  - Managing complexity
    - Abstraction
  - Communication: between stakeholders (programmers, designers, architects, clients, …)
    - Documentation: for maintenance and evolution
- Kinds of models
  - Object models: classes, objects, messages, relations, …
  - Functional models: what the system ‘does’, e.g., flow charts, Statecharts
  - Physical models: which physical units the systems constitute
  - Task models: units of work, schedule, resources. E.g.: PERT, Gantt, …
- Notations
  - (OMT: Predecessor to UML, Object Modelling Techniques (Rumbaugh et al))
  - UML: Unified Modeling Language
  - Statecharts
  - Data flow diagrams
  - Type diagrams

What is a Model

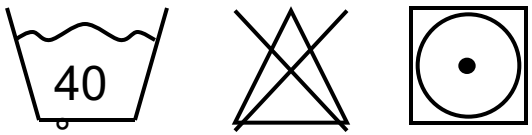
- Like a map, a model represents something else
- A useful model has the right level of detail and represents only what is important for the task in hand
- Many things can be modelled: bridges, traffic flow, buildings, economic policy

Why Use a Model?

- A model is quicker and easier to build than the real thing
- A model can be used in a simulation
- A model can evolve as we learn
- We can choose which details to include in a model
- A model can represent real or imaginary things from any domain

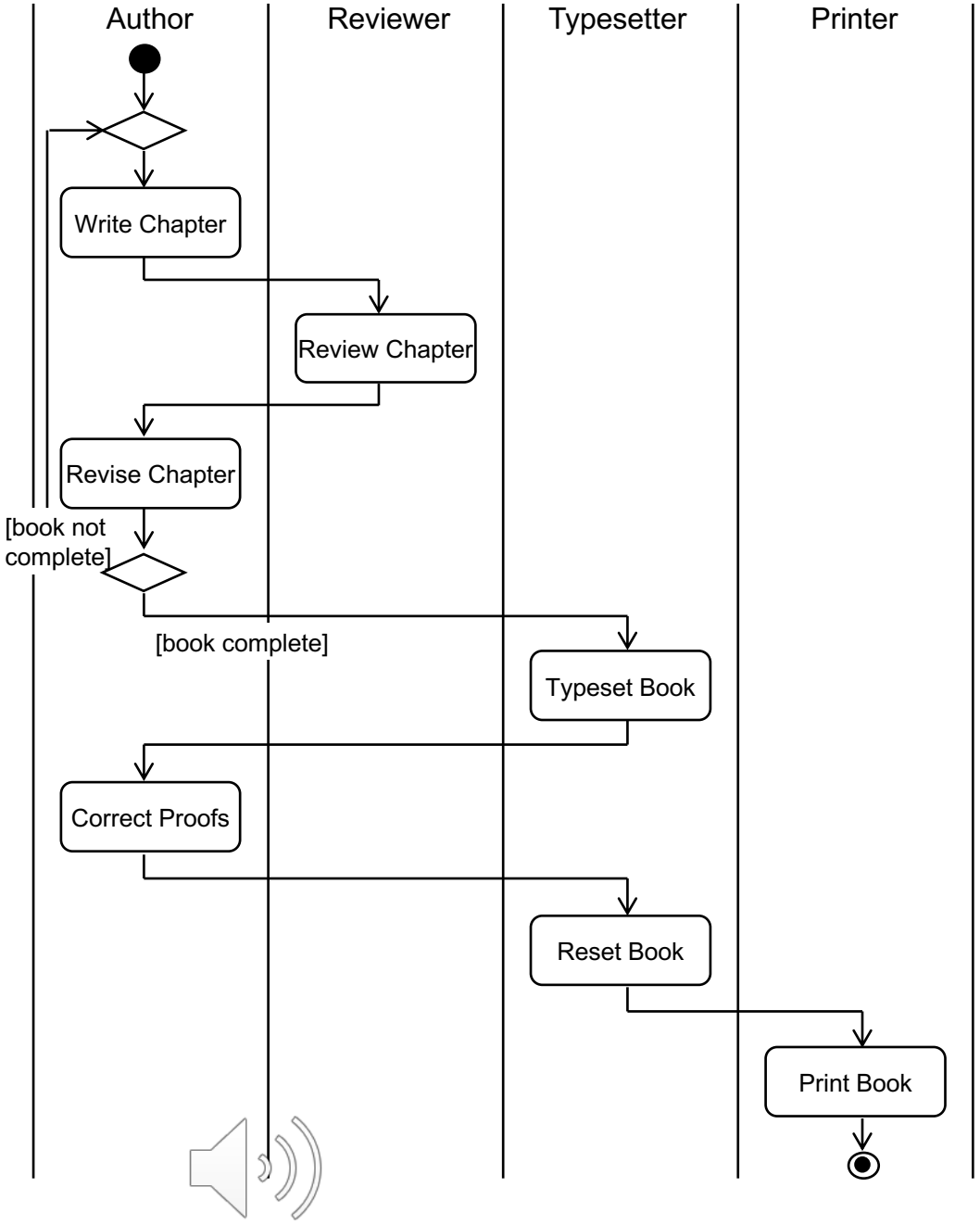
What is a Diagram?

- Abstract shapes are used to represent things or actions from the real world
- Diagrams follow rules or standards
- The standards make sure that different people will interpret the diagram in the same way

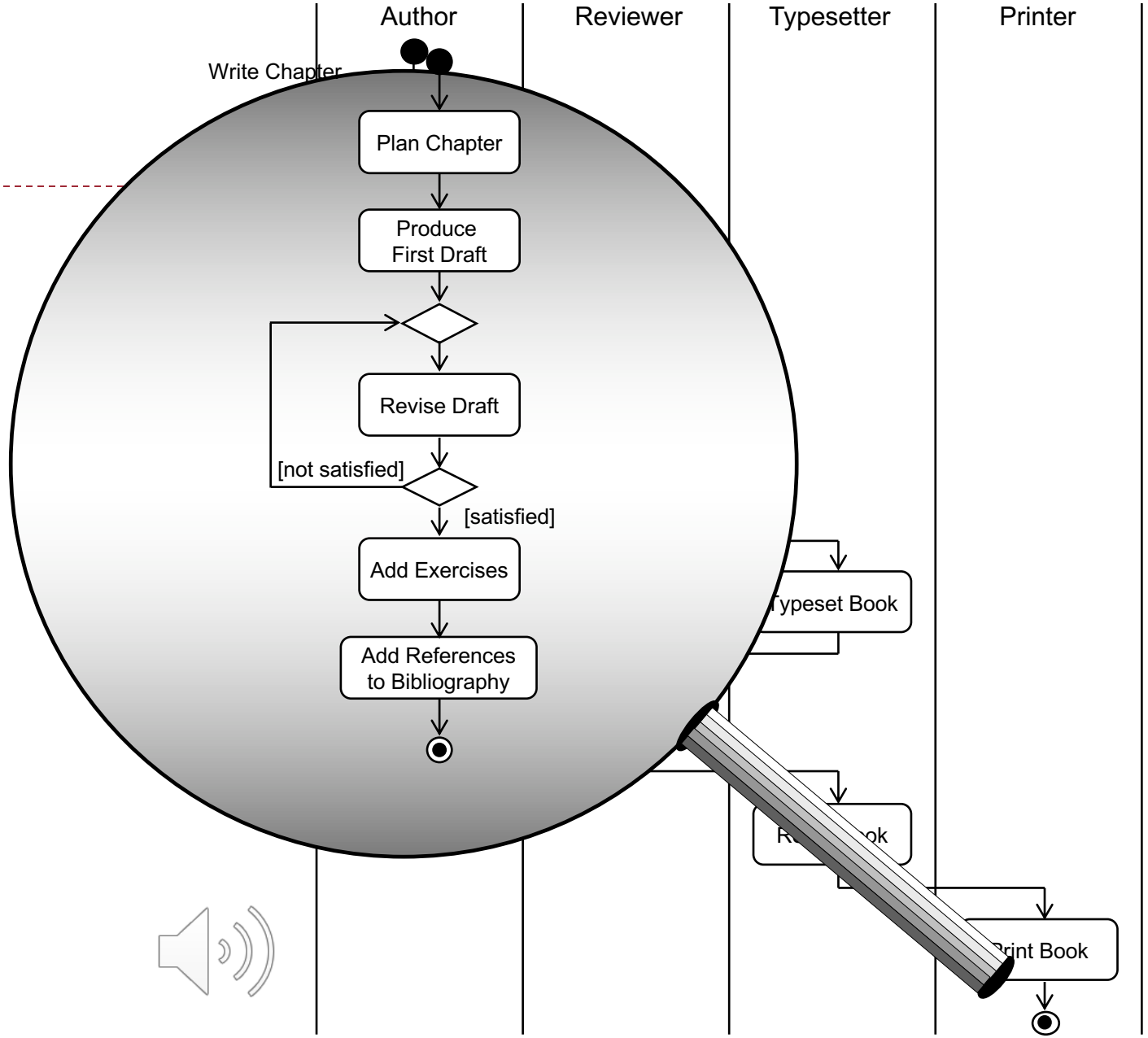


An Example of a Diagram

- An activity diagram of the tasks involved in producing a book.

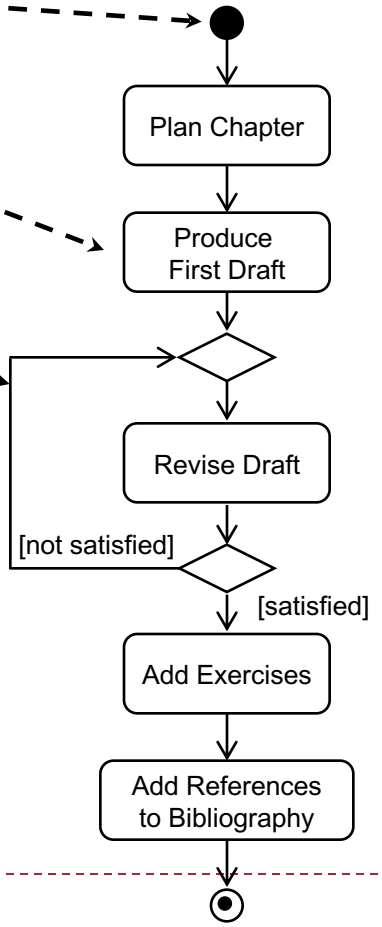


Hiding Detail



Diagrams in UML

- UML diagrams consist of:
  - icons
  - two-dimensional symbols
  - paths
  - Strings
- UML diagrams are defined in the UML specification.

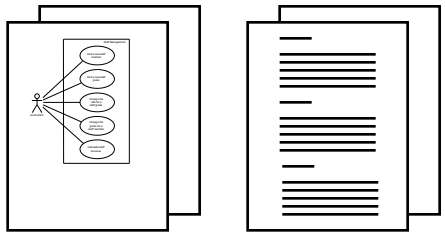


Developing Models

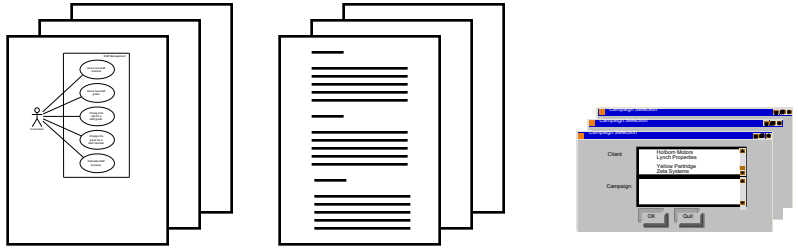
- During the life of a project using an iterative life cycle, models change along the dimensions of:
  - abstraction—they become more concrete
  - formality—they become more formally specified
  - level of detail—additional detail is added as understanding improves

Development of the Use Case Model

Iteration 1
Obvious use cases.
Simple use case descriptions.



Iteration 2
Additional use cases.
Simple use case descriptions.
Prototypes.



Iteration 3
Structured use cases.
Structured use case descriptions.
Prototypes.

Some example diagrams …..

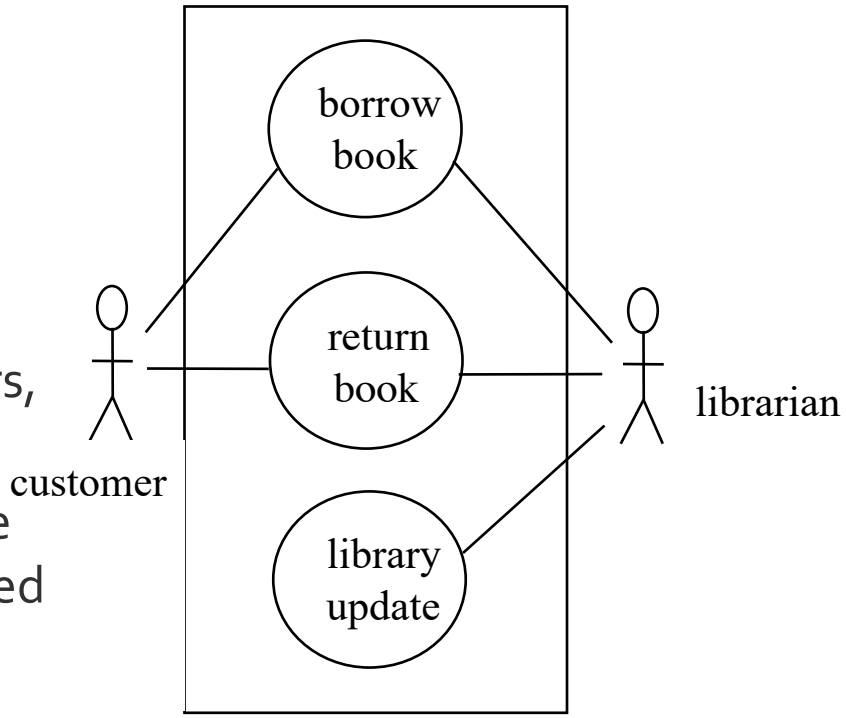
We will explore these in more detail in future lectures

Types of diagram: Static vs. dynamic models

- **Static** models describe structural aspects:
  - Class Diagrams
  - Package Diagrams
  - Type Diagrams
- **Dynamic** models describe behavioral aspects:
  - Use Case Diagrams
  - Sequence Diagrams
  - Collaboration Diagrams
  - Activity Diagrams

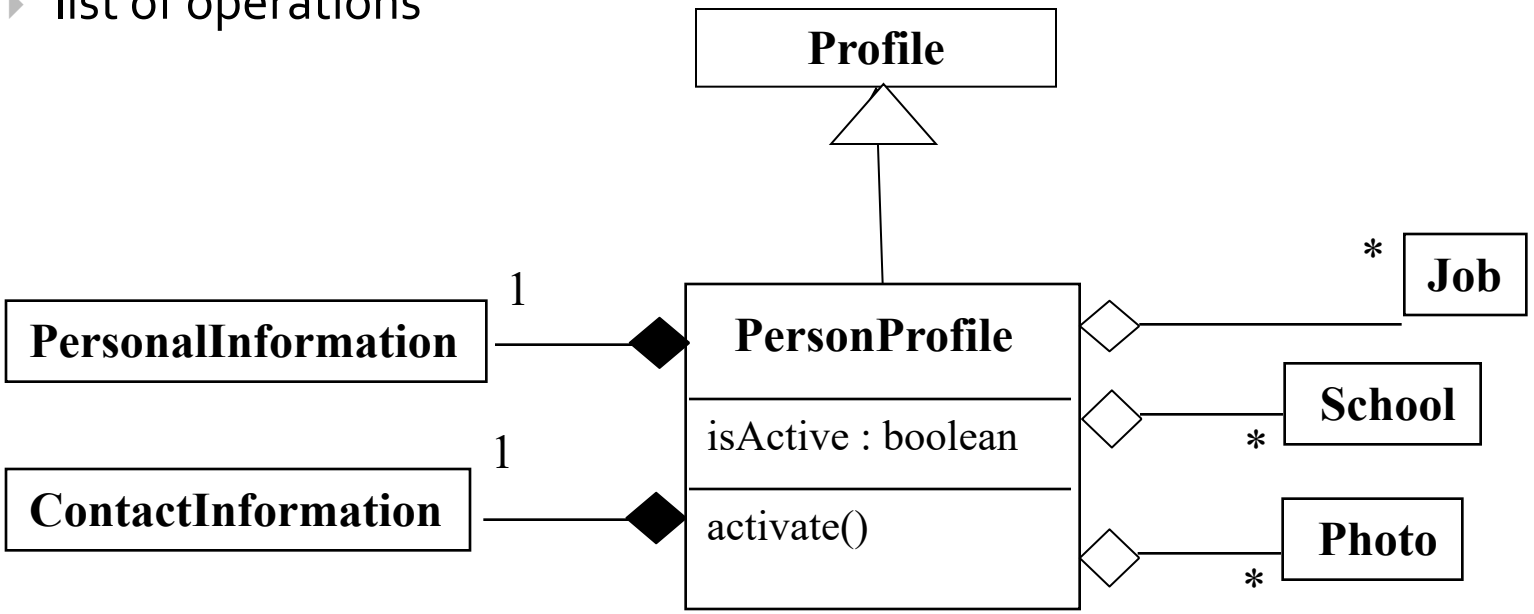
UML use-case diagrams

- Specify ONLY user views of essential system behaviour
- Do NOT specify the flow of processes in the implementation
- Components
  - A large box: system boundary
  - Stick figures outside the box: actors, both human and systems
  - Each oval inside the box: a use case that represents some major required functionality and its variant
  - A line between an actor and use case: the actor participates in the use case



UML class diagrams

- Classes, associations, interfaces
  - Class: solid rectangle with 3 compartments
    - class name and other general properties of the class
    - list of attributes
    - list of operations

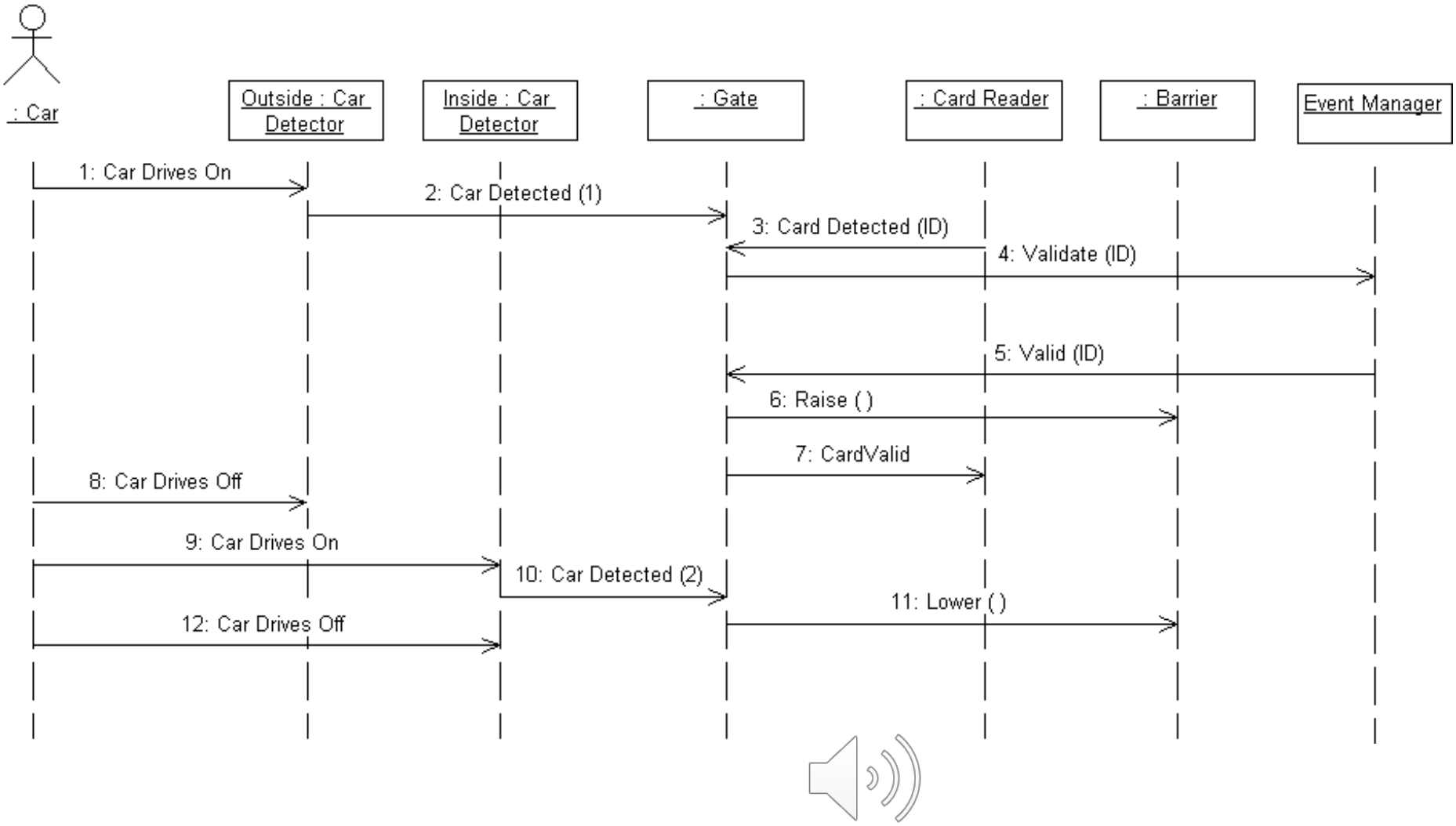


UML package diagram

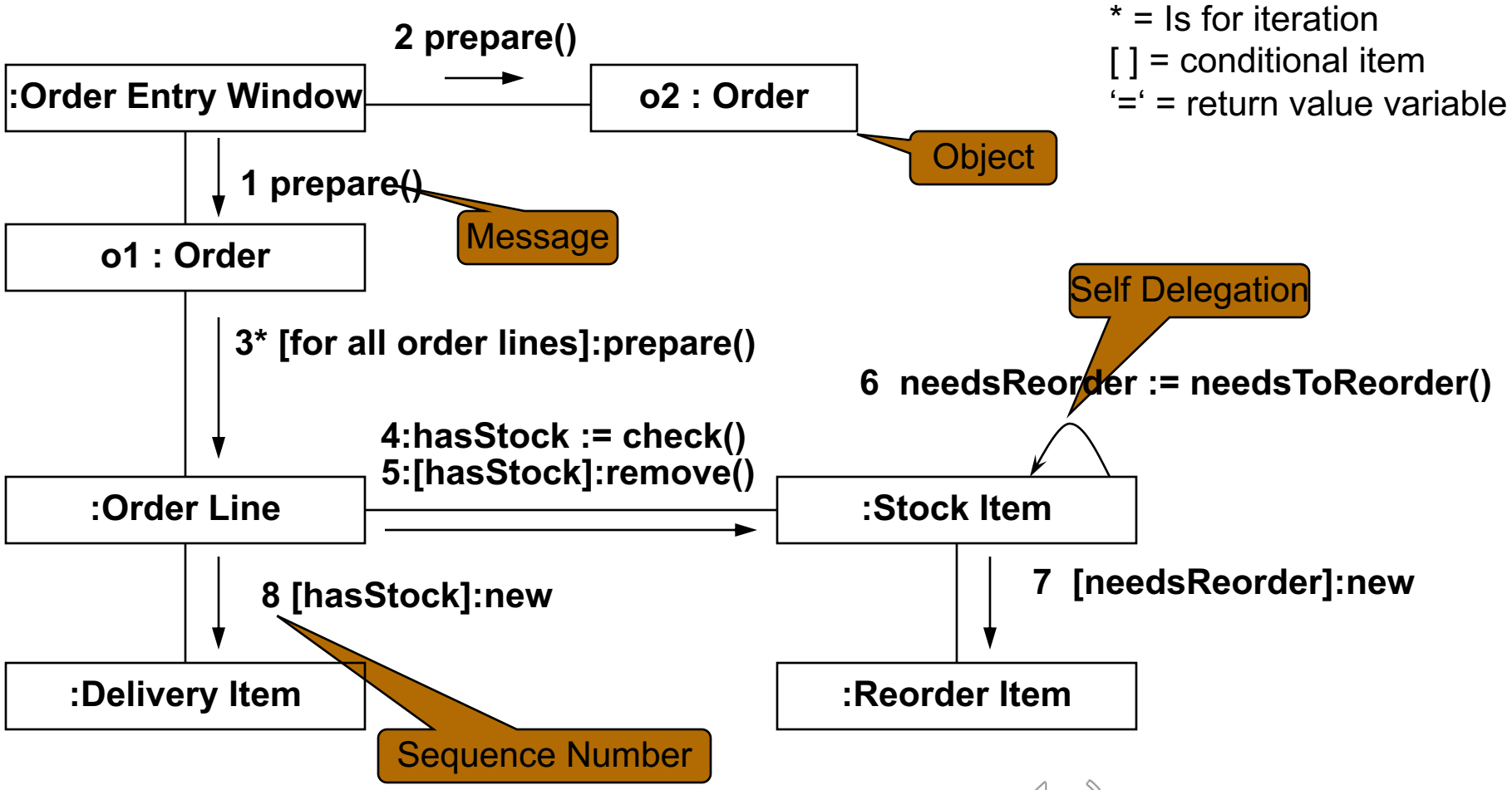
- UML package diagrams allow viewing a system as a small collection of packages each of which may be expanded to a larger set of classes

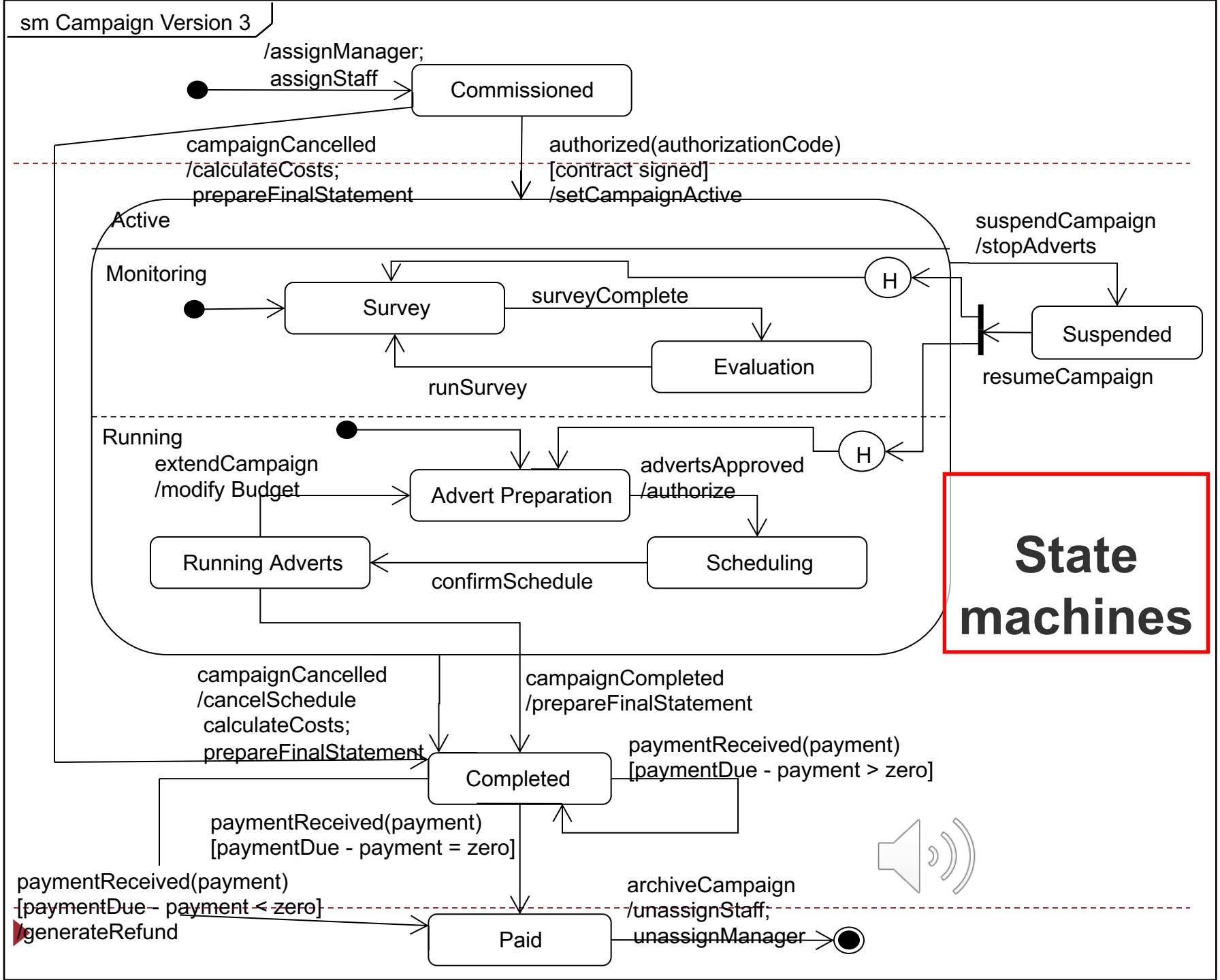
- See p120 & 244 Bennett

UML sequence diagrams



UML collaboration diagrams



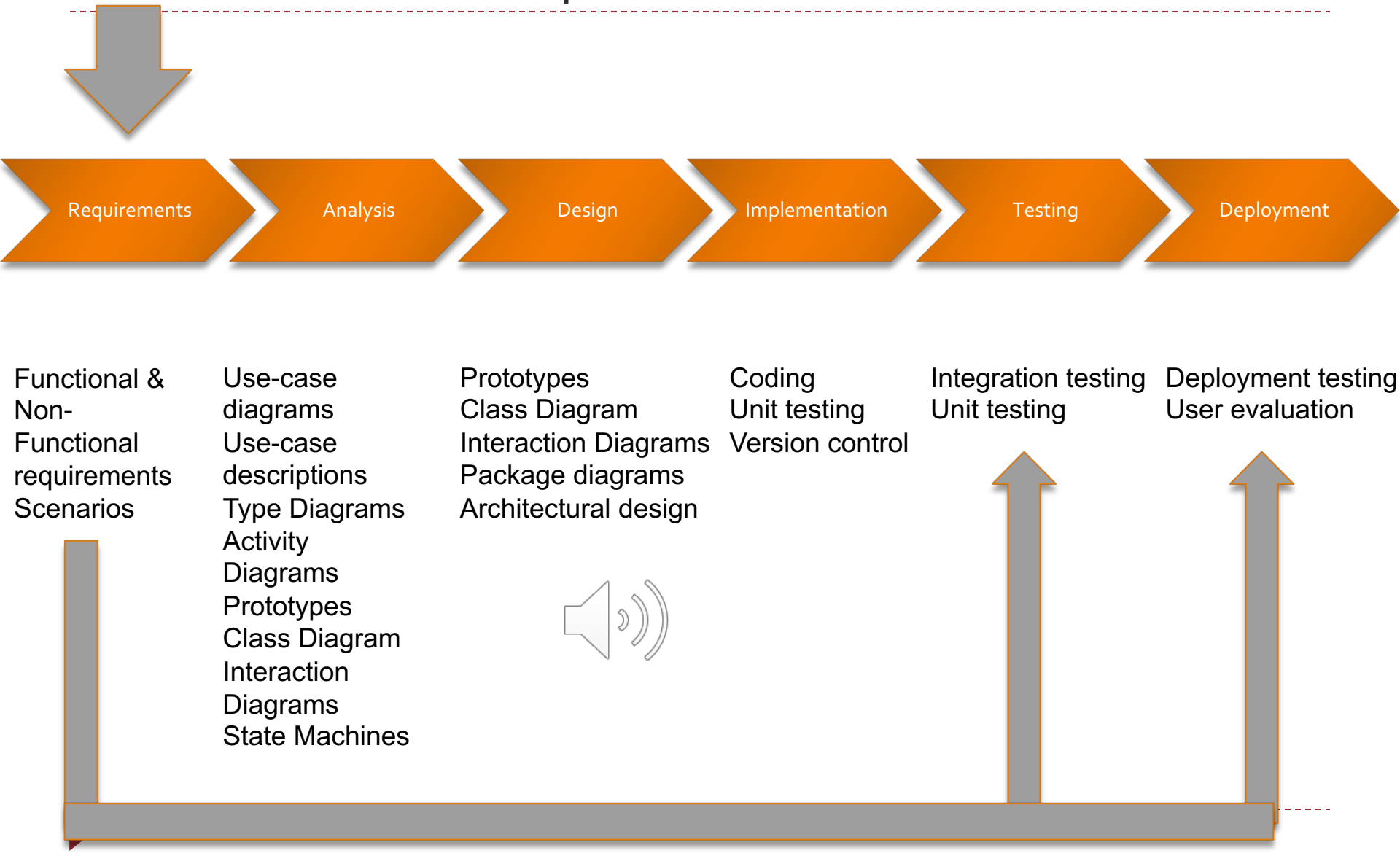

sm Campaign Version 3
/assignManager;
assignStaff
Commissioned
campaignCancelled
/calculateCosts;
prepareFinalStatement
authorized(authorizationCode)
[contract signed]
/setCampaignActive
Active
suspendCampaign
/stopAdverts
Monitoring
H
Survey
surveyComplete
Suspended
Evaluation
runSurvey
resumeCampaign
Running
H
extendCampaign
/modify Budget
Advert Preparation
advertsApproved
/authorize
Running Adverts
Scheduling
confirmSchedule
State machines
campaignCancelled
/cancelSchedule
calculateCosts;
prepareFinalStatement
campaignCompleted
/prepareFinalStatement
paymentReceived(payment)
[paymentDue - payment > zero]
Completed
paymentReceived(payment)
[paymentDue - payment = zero]
paymentReceived(payment)
[paymentDue - payment < zero]
/generateRefund
archiveCampaign
/unassignStaff;
unassignManager
Paid

UML pros and cons

- Pros
  - De-facto industry standard for software representation
  - A standard of communication
    - A common vocabulary
    - Accepted notation
    - A visual abstraction
- Cons
  - Language? No well-defined semantics
    → No means for proving/refuting consistency with implementation
    → No automated verification
  - Modelling? Low level of abstraction
    → Inadequate means for abstraction & navigation
  - Unified? Little integration between sub-notations

Reminder: Waterfall Lifecycle and Main Outputs

Problem Definition
Viewpoints/Scope



Analysis Flow in UML (& linkages between diagrams)

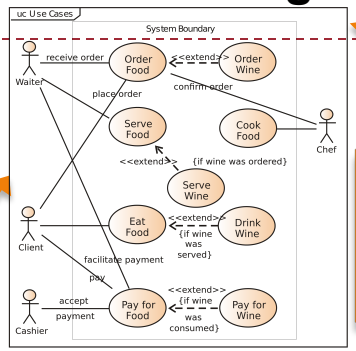
1

Requirements

Problem Definition
Viewpoints/Scope
Functional & Non-Functional requirements
Scenarios

2

Use-case diagrams



Actors

A sequence diagram for each use-case description

3

Use-case descriptions

Names in bubbles match with use-case description names

Name	The Use Case name. Typ… …me is of the format <action> + <object>.
ID	An identifier that is unique to … …se Case.
Description	A brief sentence that states wh… the user wants to be able to do and what benefit he will derive.
Actors	The type of user who interacts with the system to accomplish the task. Actors are identified by role name.
Organizational Benefits	The value the organization expects to receive from having the functionality described. Ideally this is a link directly to a Business Objective.
Frequency of Use	How often the Use Case is executed.
Triggers	Concrete actions made by the user within the system to start the Use Case.
Preconditions	Any states that the system must be in or conditions that must be met before the Use Case is started.
Postconditions	Any states that the system must be in or conditions that must be met after the Use Case is completed successfully. These will be met if the Main Course or any Alternate Courses are followed. Some Exceptions may result in failure to meet the Postconditions.
Main Course	The most common path of interactions between the user and the system. 1. Step 1 2. Step 2
Alternate Courses	Alternate paths through the system. AC1: <condition for the alternate to be called> 1. Step 1 2. Step 2 AC2: <condition for the alternate to be called> 1. Step 1
Exceptions	Exception handling by the system. EX1: <condition for the exception to be called> 1. Step 1 2. Step 2 EX2 <condition for the exception to be called> 1. Step 1

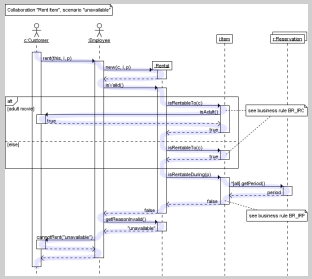
4

List of Candidate Classes:
Campaign
Order
Customer

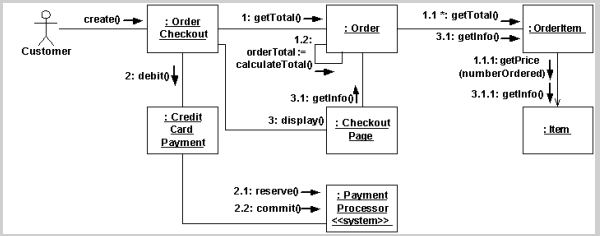
Class names

5

Sequence diagrams



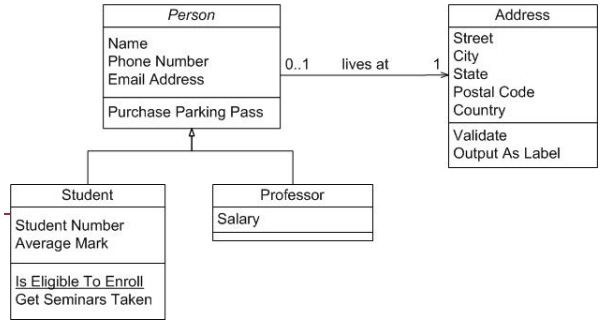
Collaboration diagrams



Class Names
Messages/Associations

6

Class diagram



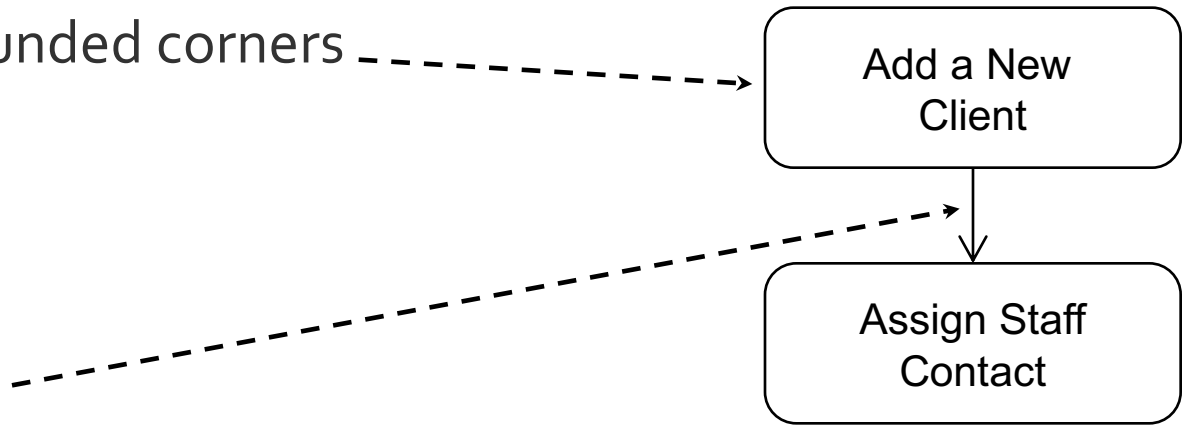
UML Activity Diagrams

Our first example: Activity Diagrams

- Purpose
  - to model a task (for example in business modelling)
  - to describe a function of a system represented by a use case
  - to describe the logic of an operation

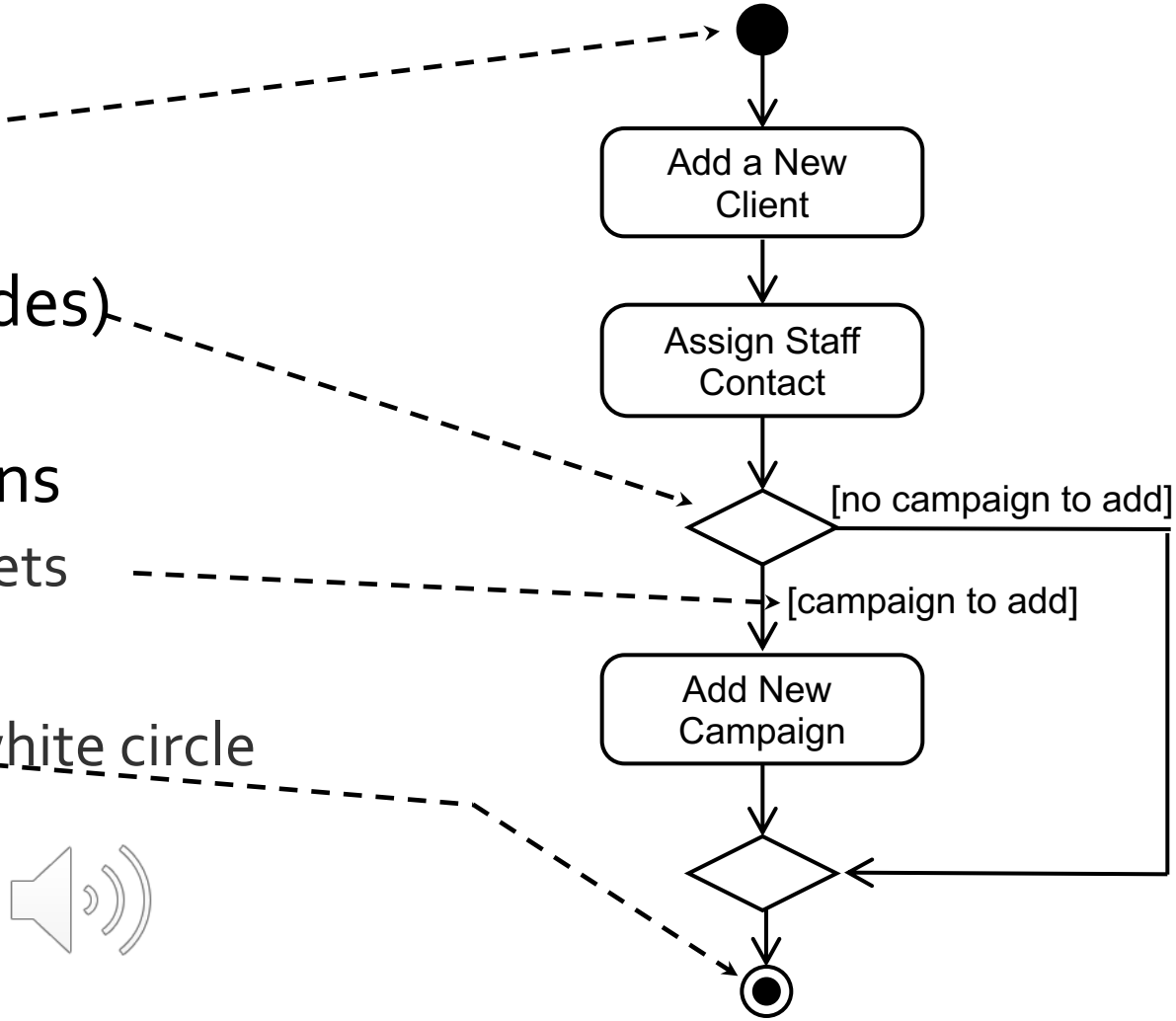
Notation of Activity Diagrams

- Actions
  - rectangle with rounded corners
  - meaningful name
- Control flows
  - arrows with open arrowheads



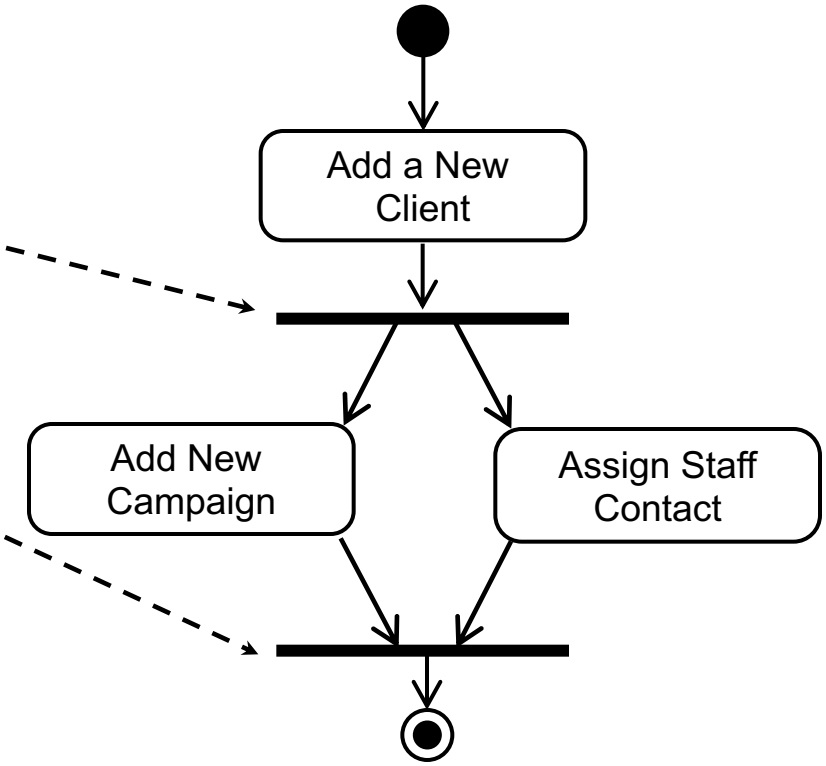
Notation of Activity Diagrams

- Initial node
  - black circle
- Decision nodes (and merge nodes)
  - diamond
- Guard conditions
  - in square brackets
- Final node
  - black circle in white circle



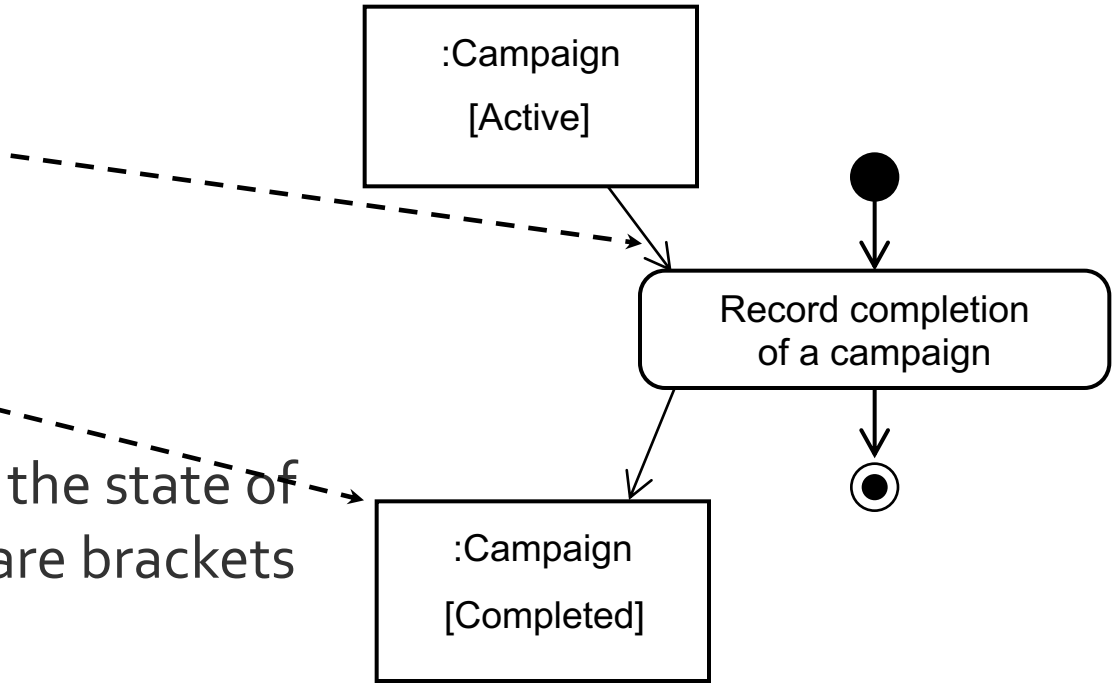
Notation of Activity Diagrams

- Fork nodes and join nodes
  - thick bar
- Actions carried out in parallel



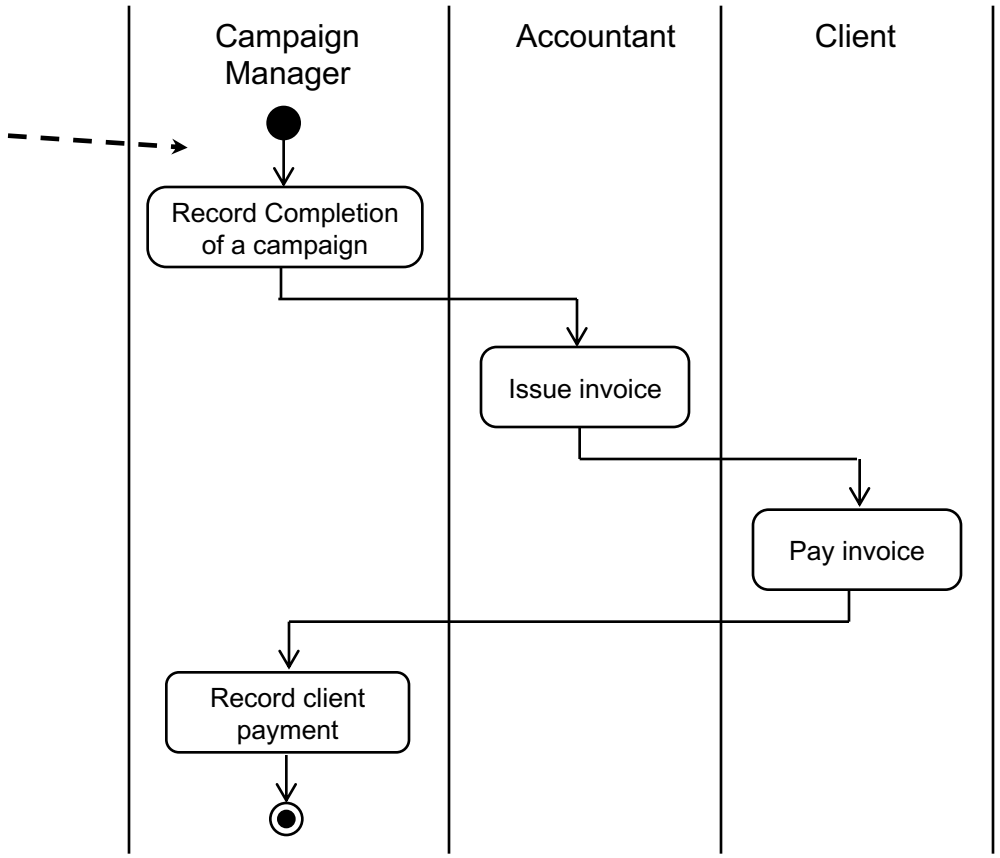
Notation of Activity Diagrams

- Object flows
  - open arrow
- Objects
  - rectangle
  - optionally shows the state of the object in square brackets



Notation of Activity Diagrams

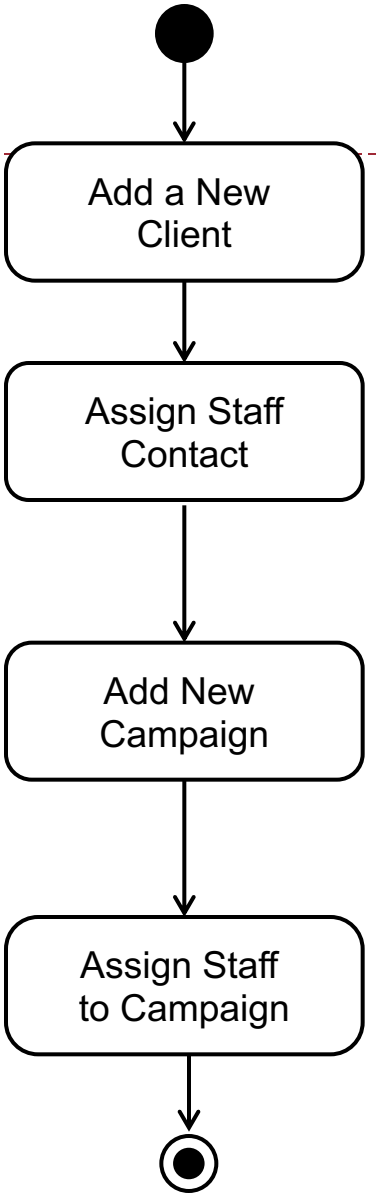
- Activity Partitions (Swimlanes)
  - vertical columns
  - labelled with the person, organisation, department or system responsible for the activities in that column



Drawing Activity Diagrams

- Identify actions
  - Eg. What happens when a new client is added in the system?
    - Add a New Client
    - Assign Staff Contact
    - Add New Campaign
    - Assign Staff to Campaign
- Organise the actions in order with flows

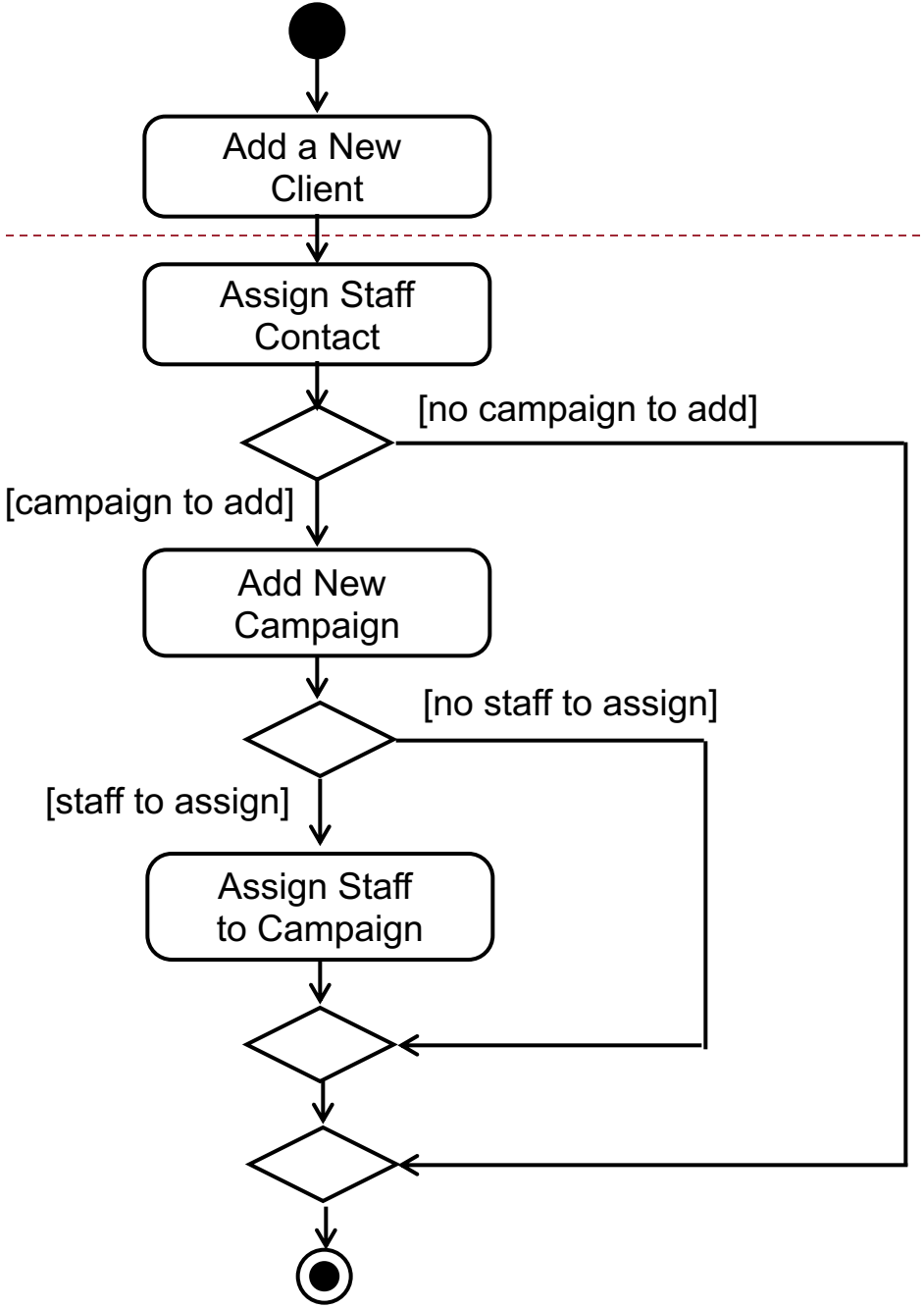
Drawing Activity Diagrams



Drawing Activity Diagrams

- Identify any alternative flows and the conditions on them
  - sometimes there is a new campaign to add for a new client, sometimes not
  - sometimes they will want to assign staff to the campaign, sometimes not
- Add decision and merge nodes, flows and guard conditions to the diagram

Drawing Activity Diagrams



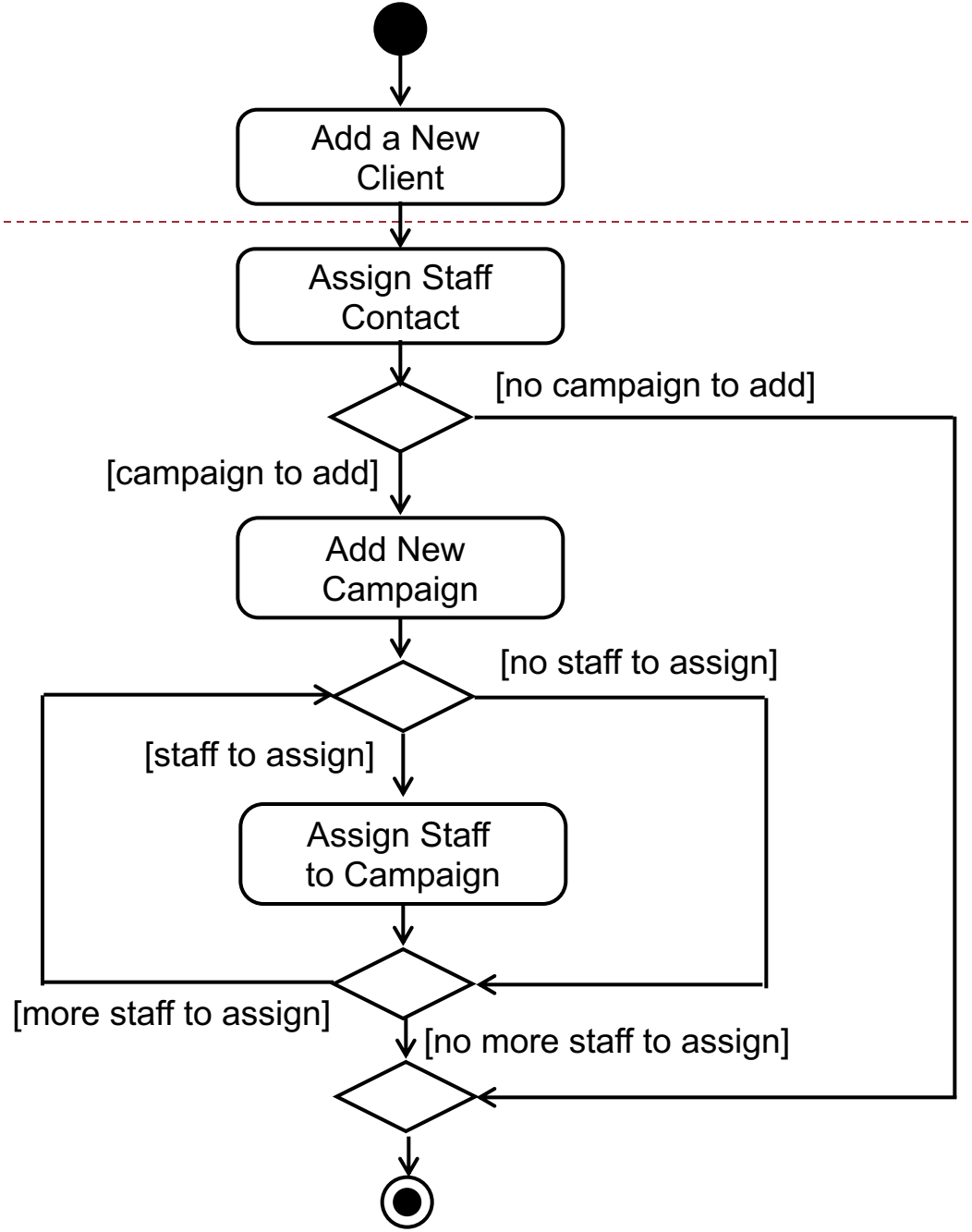
Drawing Activity Diagrams

- Identify any actions that are carried out in parallel
  - there are none in this example
- Add fork and join nodes and flows to the diagram

Drawing Activity Diagrams

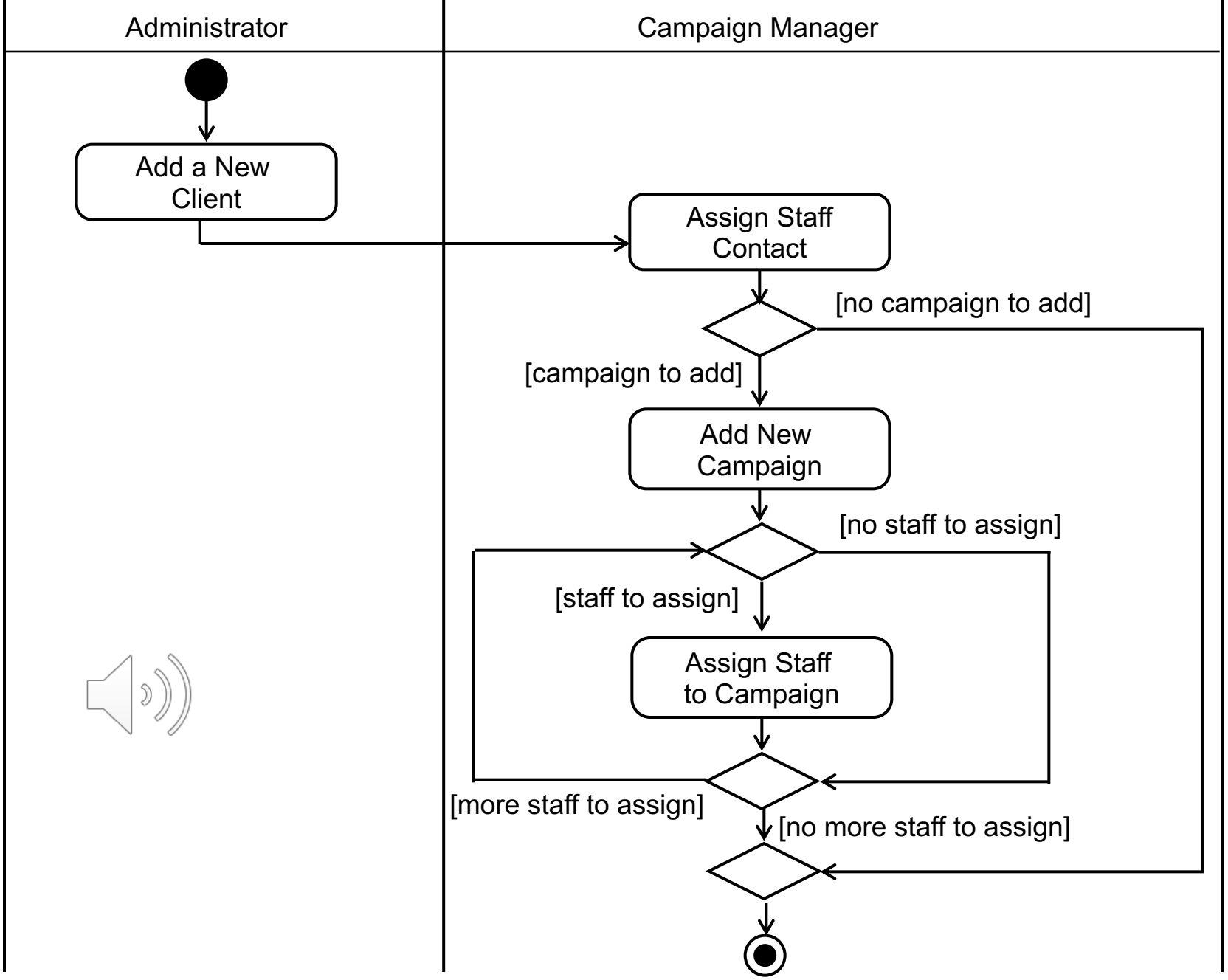
- Identify any processes that are repeated
  - they will want to assign staff to the campaign until there are no more staff to add
- Add decision and merge nodes, flows and guard conditions to the diagram

Drawing Activity Diagrams



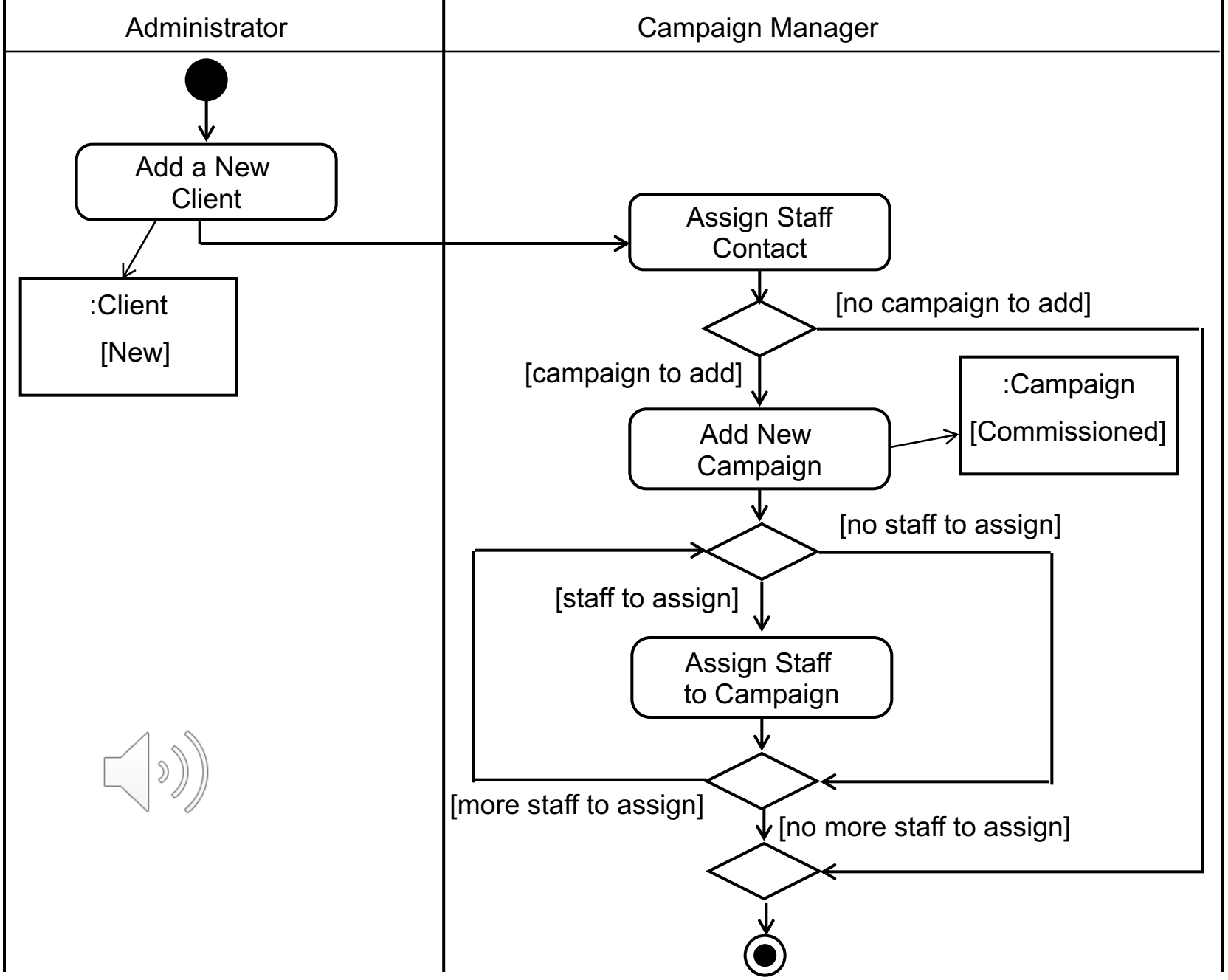
Drawing Activity Diagrams

- Are all the activities carried out by the same person, organisation or department?
- If not, then add swimlanes to show the responsibilities
- Name the swimlanes
- Show each activity in the appropriate swimlane

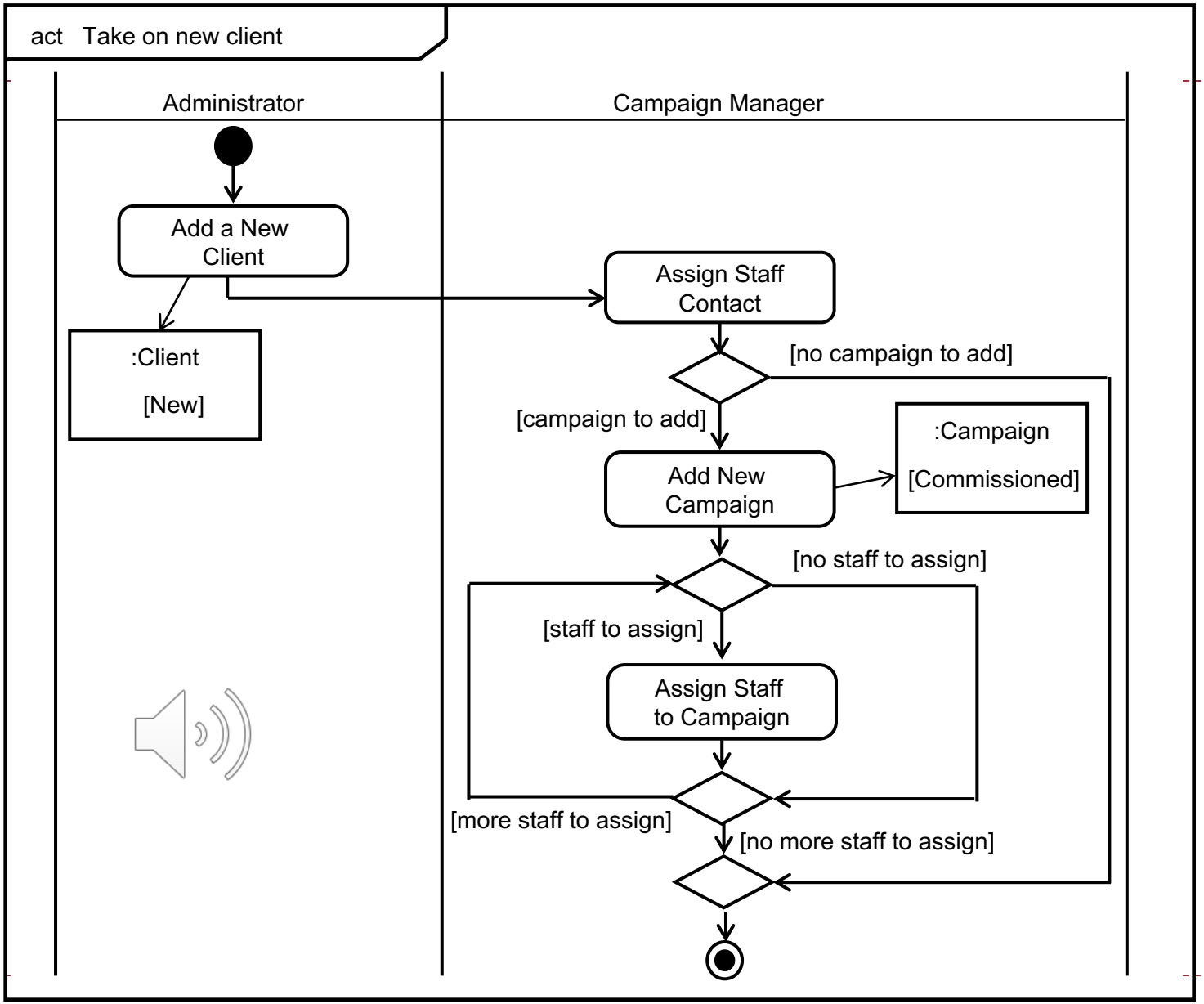

Administrator
Campaign Manager
Add a New Client
Assign Staff Contact
[no campaign to add]
[campaign to add]
Add New Campaign
[no staff to assign]
[staff to assign]
Assign Staff to Campaign
[more staff to assign]
[no more staff to assign]

Drawing Activity Diagrams

- Are there any object flows and objects to show?
  - these can be documents that are created or updated in a business activity diagram
  - these can be object instances that change state in an operation or a use case
- Add the object flows and objects

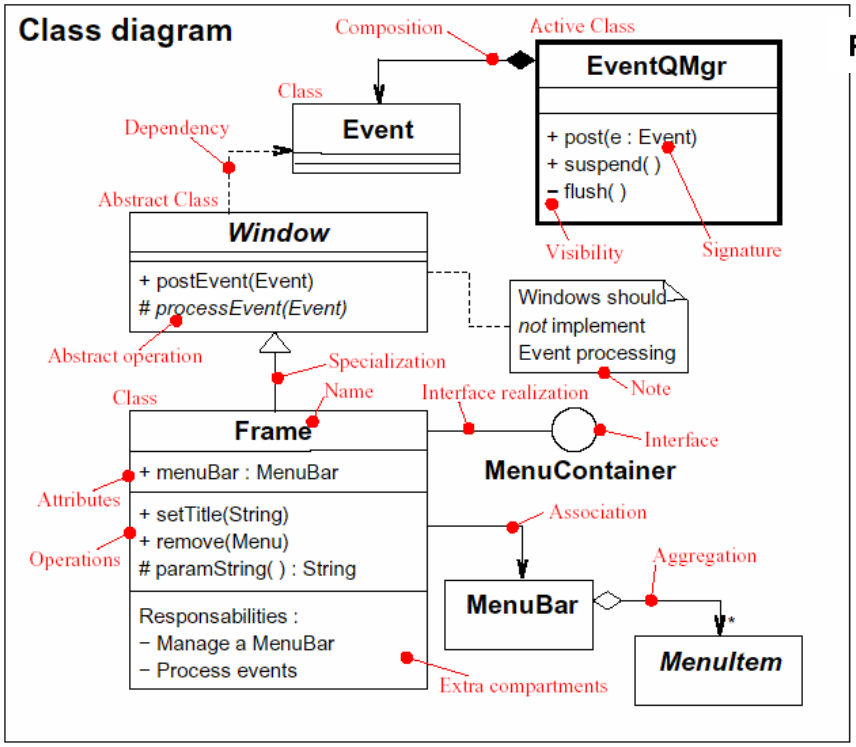

Administrator
Campaign Manager
Add a New Client
:Client
[New]
Assign Staff Contact
[no campaign to add]
[campaign to add]
Add New Campaign
:Campaign
[Commissioned]
[no staff to assign]
[staff to assign]
Assign Staff to Campaign
[more staff to assign]
[no more staff to assign]

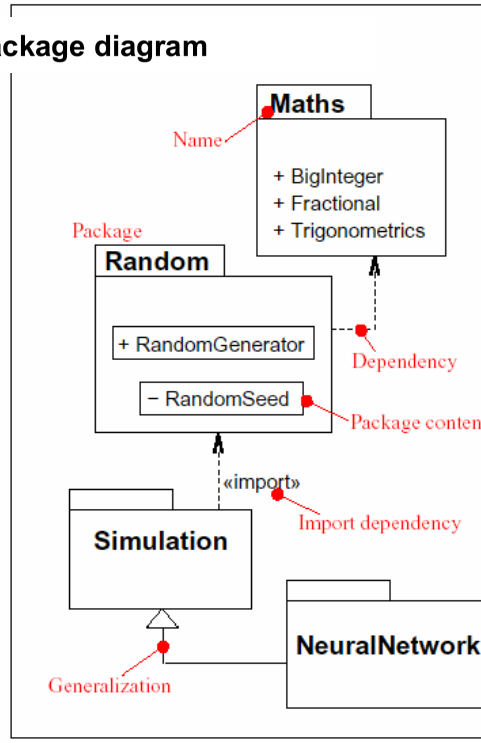
When printed or copied a frame is used

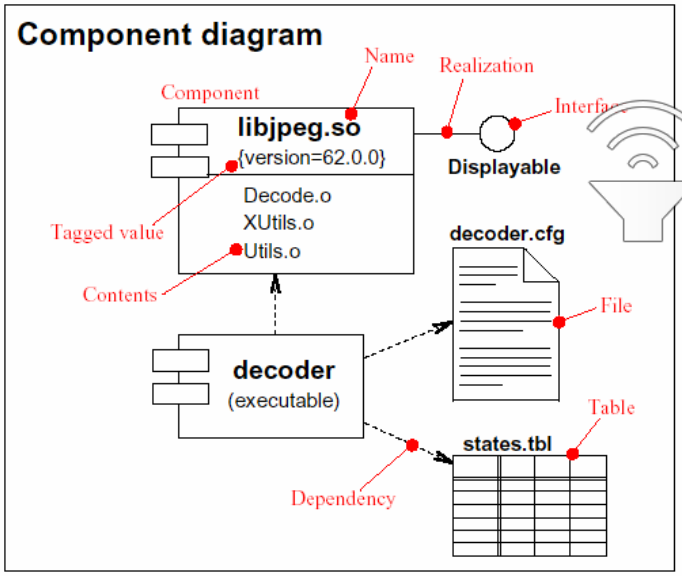


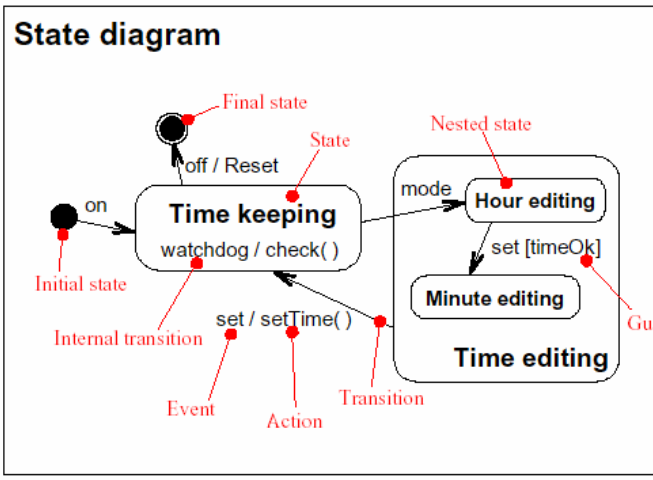
In this lecture we have looked at

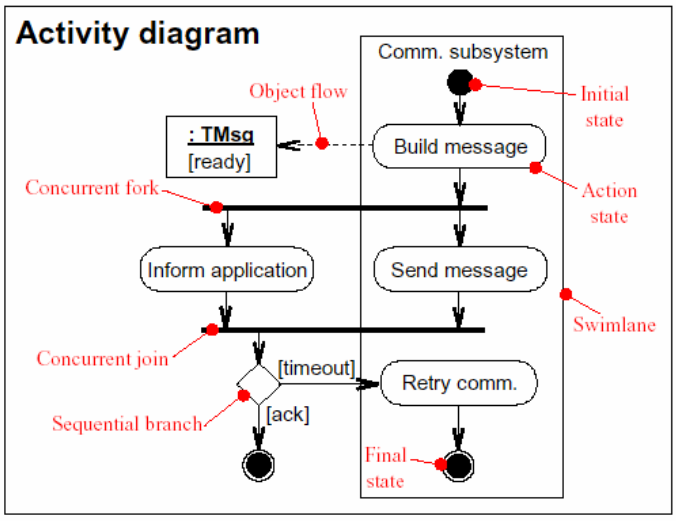
- Models and Diagrams in general
- Some diagram examples
  - Use-case diagrams
  - Class diagrams
  - Package diagrams
  - Sequence diagrams
  - Collaboration diagrams
  - Type diagrams
- Activity Diagrams in detail
- We will revisit these diagrams again during the rest of the module!

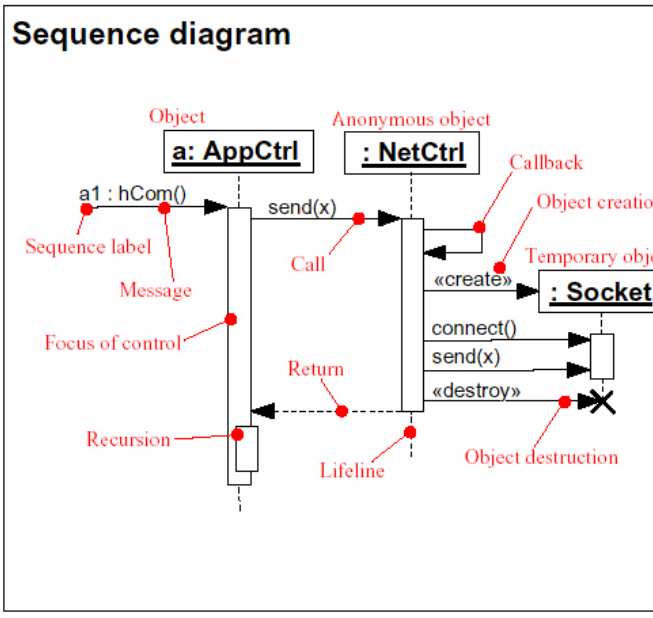

Class diagram
Composition
Active Class
EventQMgr
+ post(e : Event)
+ suspend( )
– flush( )
Class
Event
Dependency
Abstract Class
Window
+ postEvent(Event)
# processEvent(Event)
Visibility
Signature
Windows should not implement Event processing
Abstract operation
Specialization
Note
Class
Name
Interface realization
Frame
+ menuBar : MenuBar
+ setTitle(String)
+ remove(Menu)
# paramString( ) : String
Responsabilities :
– Manage a MenuBar
– Process events
Interface
MenuContainer
Attributes
Operations
Association
Aggregation
MenuBar
MenuItem
Extra compartments


Package diagram
Maths
Name
+ BigInteger
+ Fractional
+ Trigonometrics
Package
Random
+ RandomGenerator
– RandomSeed
Dependency
Package conten
«import»
Import dependency
Simulation
NeuralNetwork
Generalization

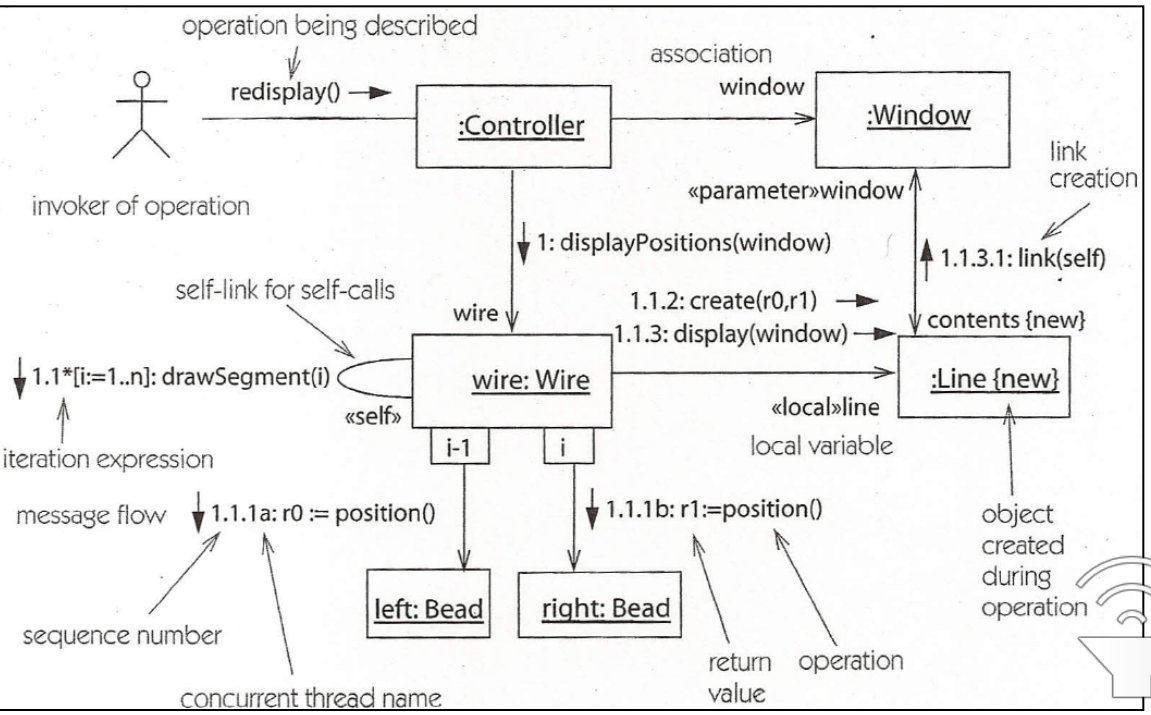

Component diagram
Name
Realization
Component
libjpeg.so
{version=62.0.0}
Displayable
Decode.o
XUtils.o
Utils.o
Tagged value
Contents
decoder.cfg
File
decoder
(executable)
Table
states.tbl
Dependency

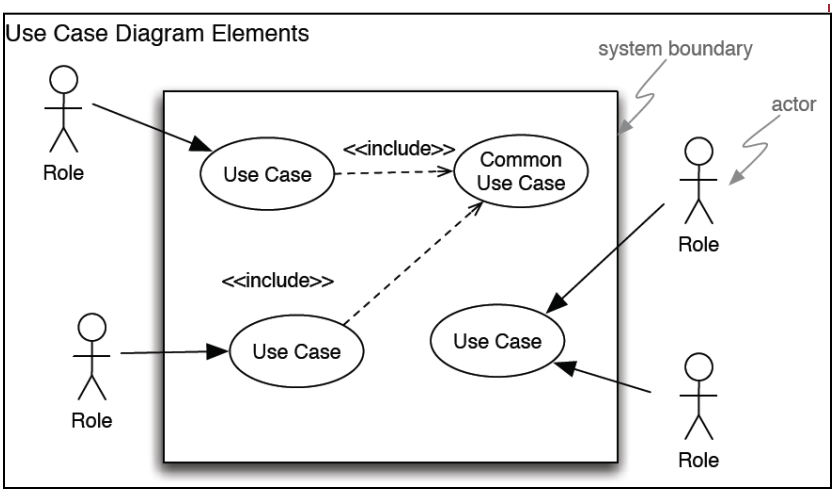

State diagram
Final state
State
Nested state
off / Reset
on
Time keeping
watchdog / check( )
mode
Hour editing
set [timeOk]
Minute editing
Time editing
Initial state
Internal transition
set / setTime( )
Event
Action
Transition


Activity diagram
Comm. subsystem
Object flow
Initial state
: TMsg
[ready]
Build message
Concurrent fork
Action state
Inform application
Send message
Swimlane
Concurrent join
[timeout]
Retry comm.
Sequential branch
[ack]
Final state

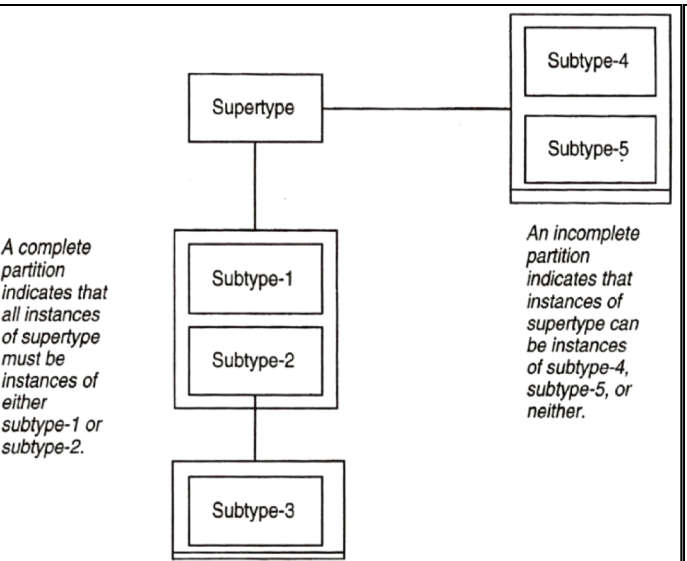

Sequence diagram
Object
Anonymous object
a: AppCtrl
: NetCtrl
Callback
a1 : hCom()
send(x)
Object creatio
Sequence label
Message
Call
Temporary obj
«create»
: Socket
Focus of control
connect()
send(x)
Return
«destroy»
Recursion
Lifeline
Object destruction

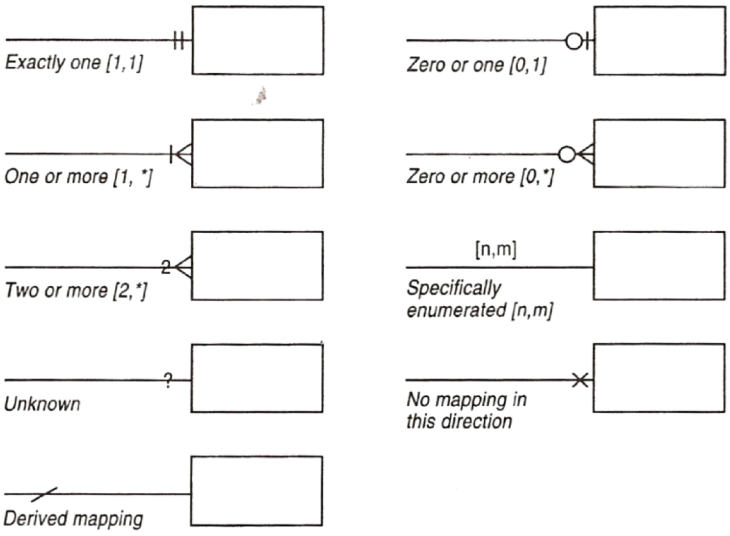
Collaboration diagram





Type diagrams





Class exercise (15 mins)

- Draw an activity diagram for the following use-case:
  - "Withdraw money from a bank account through an ATM"
- Add swimlanes to your diagram to show the main actors
- You could use Microsoft Visio to create the Diagram.

Some Hints for the class exercises

- You have three involved classes of the activity, they are Customer, ATM and Bank.
- Work out the activities for each class.

Hints for the class exercises

- You have three involved classes of the activity, they are Customer, ATM and Bank.
- The customer's activities are "insert card", "enter pin", "enter amount", "take money from slot" and "take card".
- The ATM Machine's activities are "show balance", and "Eject card".
- The Bank's activities are "authorize/verify PIN", "check account balance" and "debit account"

Further reading

- Package diagrams – pages 120 and 244 Bennett
- Chapter 7 Bennett – Class diagrams
- Chapter 9 Bennett – Object interaction
- Booch, Rumbaugh and Jacobson (1999)
- Bennett, Skelton and Lunn (2005)