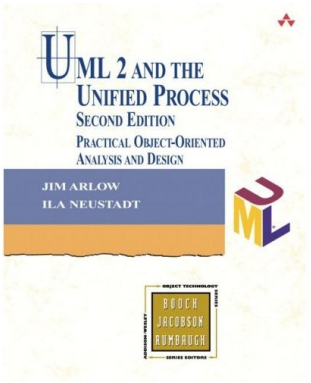
IMPORTANT NOTICE TO STUDENTS

These slides are **NOT** to be used as a replacement for student notes.
These **slides** are sometimes **vague and incomplete on purpose** to spark a class discussion

Introduction to Unified Modelling Language (UML)

(part 1- Building Blocks & Use Case Modelling)

CS 446/646 ECE452
May 4th, 2010



Material covered in this lecture is based on various chapters from ***UML 2 and the Unified Process- 2nd Edition Practical Object Oriented Analysis & Design***

WATERLOO
CHERITON SCHOOL OF COMPUTER SCIENCE

Outline

Introduction

Use case Modelling

Introduction

What is UML

- visual modelling language
- UML is a language not a methodology?
  - Q: why is this distinction important?

UML Model

- repository of all **things** and **relationships**

Building Blocks

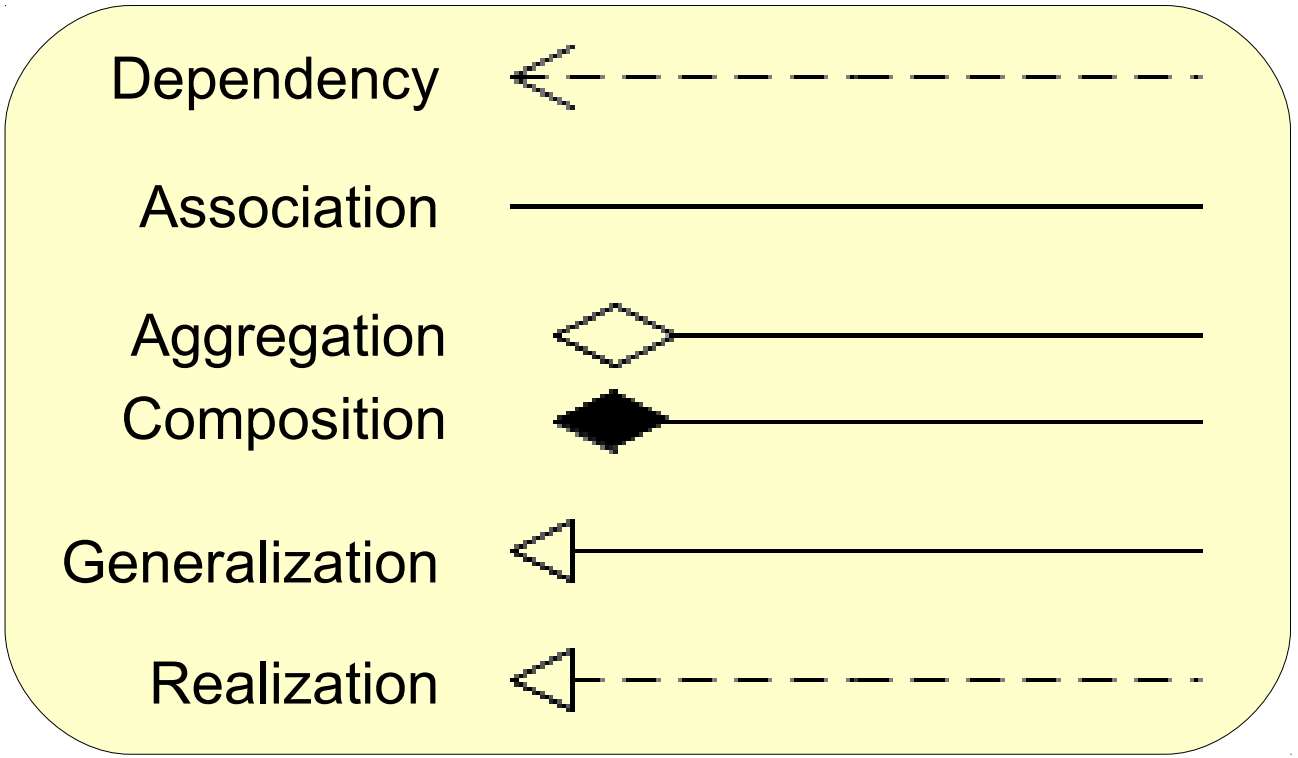
Things

- Things are the nouns of a UML model
- Three types
  - structural things:
    - *nouns (class, interface, use case etc.)*
  - behaviour things:
    - *verbs (interactions, activities etc.)*
  - grouping things:
    - *package*
  - annotational things:
    - *note*

Building Blocks

Relationships

- *representation* of how things relate to each other
- adds *semantics* to connections between entities

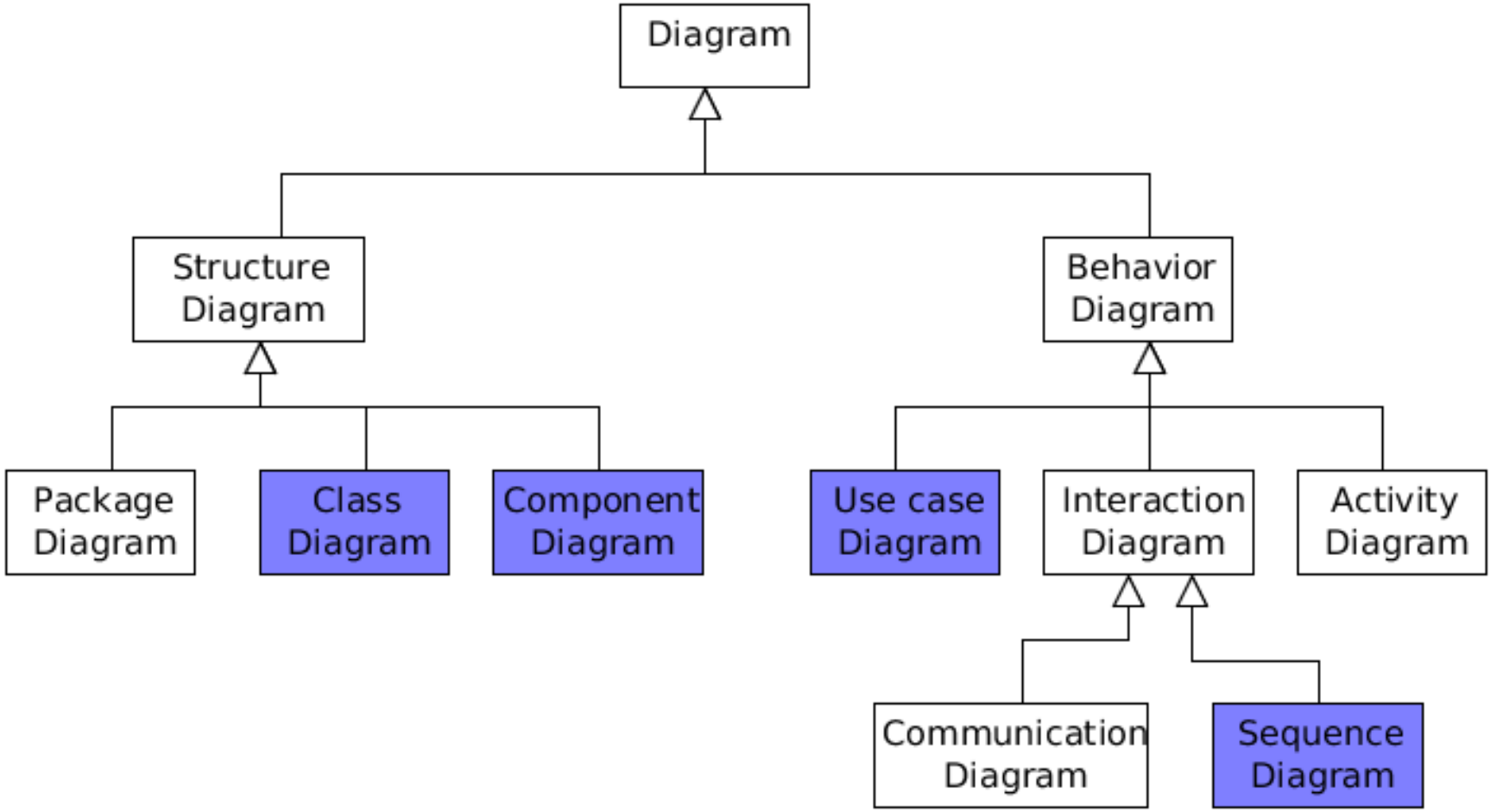


Building Blocks

Diagrams

- views of a UML model
- not a model itself
  - *things/relationships can be omitted from diagrams but still be part of the model*

Building Blocks



Building Blocks

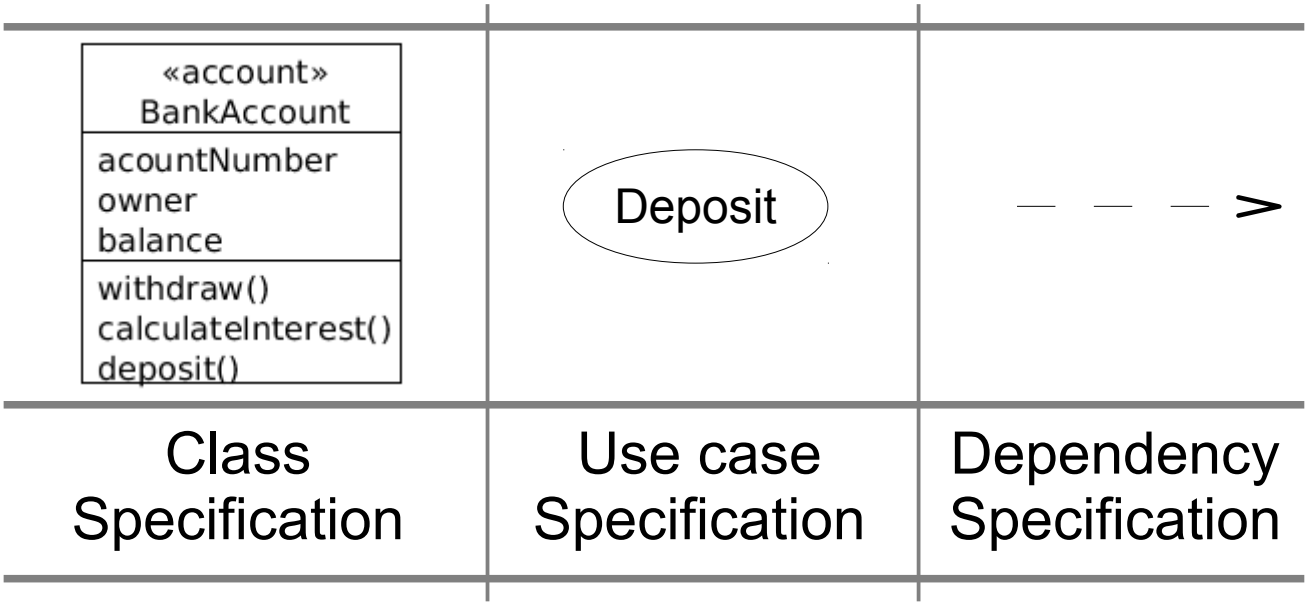
Types of UML Diagrams that

- represent static model
  - things
  - the structural relationship between things
- represent dynamic behaviour
  - how things *interact* to generate the required *functionality*

Building Blocks

UML Model

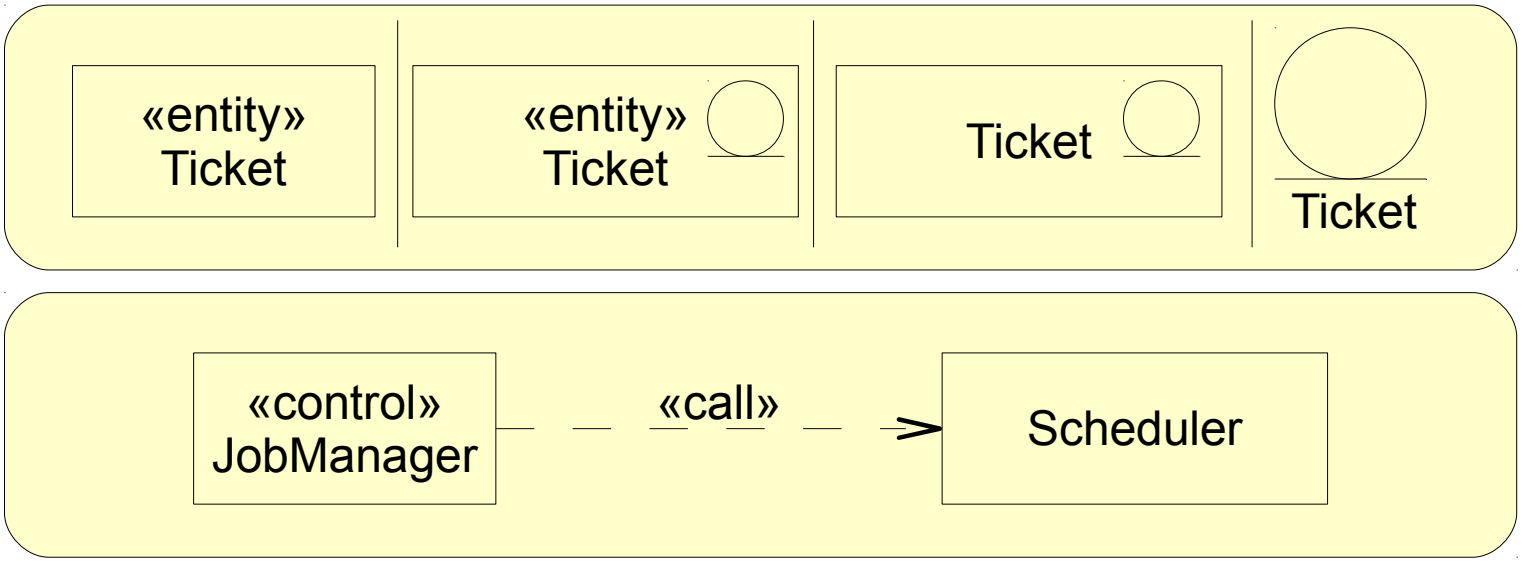
- graphical for visualization
- specifications
  - provide meaning to the visual components



Building Blocks

Extensibility Mechanisms

- stereotypes
  - allow defining new modelling elements based on existing
  - must define the semantics of the new elements
    - or else the new model is just a picture
    - usually attach a note to the new element



Use Case Modelling

Use Case Modelling

Definition

- *formalism to capture **system requirements***

Purpose

- to capture discrete unit of interactions between
  - **system** components & actors
  - **not** between **actors**

- Q: is use case diagram a behavioural or structural diagram?

Use Case Modelling

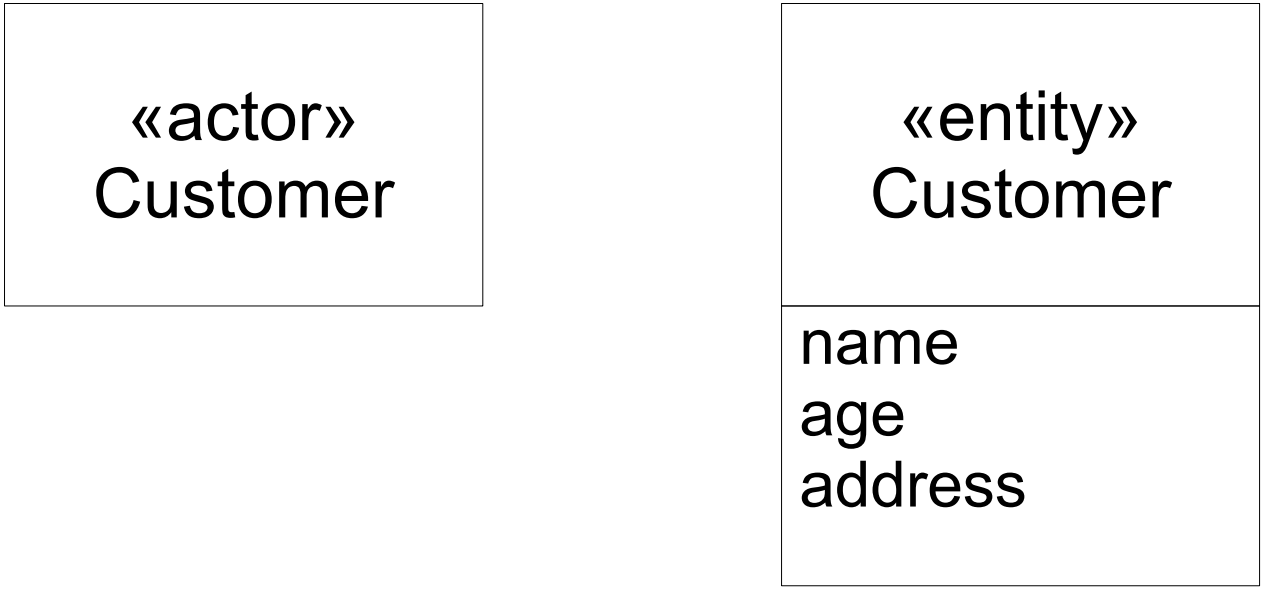
Components

- system/boundary
  - defines internal (system) & external parts
- actors
  - a **role** that an **external entity** adopts when interacting with the systems
- flows
  - main & alternative flows

Closer Look at Actors

Actors are External

- actors are **always external** to a system



  - *your system may contain internal representation of the external actors*

Closer Look at Actors

Identifying Actors

- consider who or what **uses/interacts** with your system

Use Roles not Individuals

- assuming a set of customers {Jim, Mike, Helen}
  - Q: who/what should be the actor here?

Other things as actors

- Q: what about time as an actor?
  - e.g. scheduled tasks

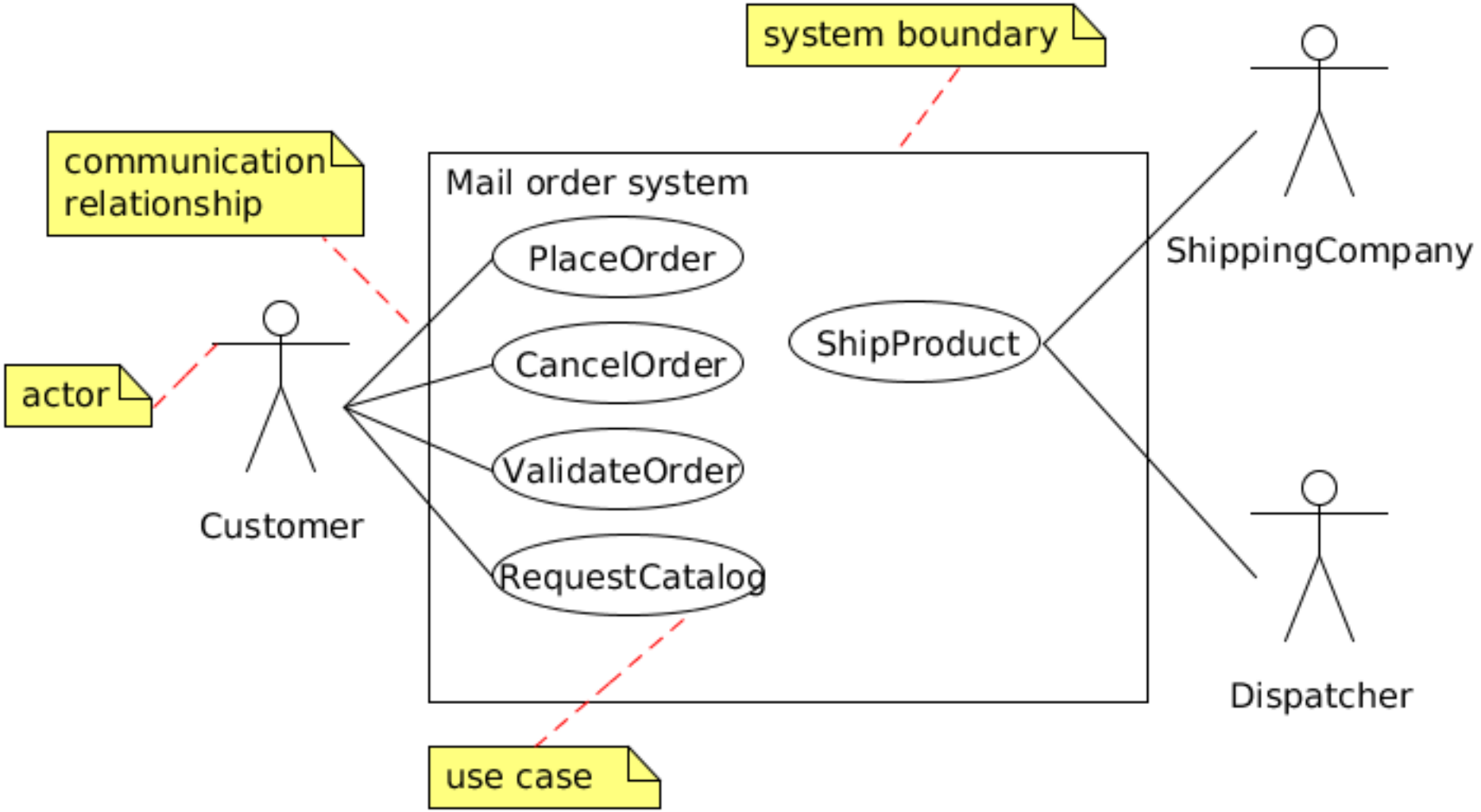
What are Use case

Definition

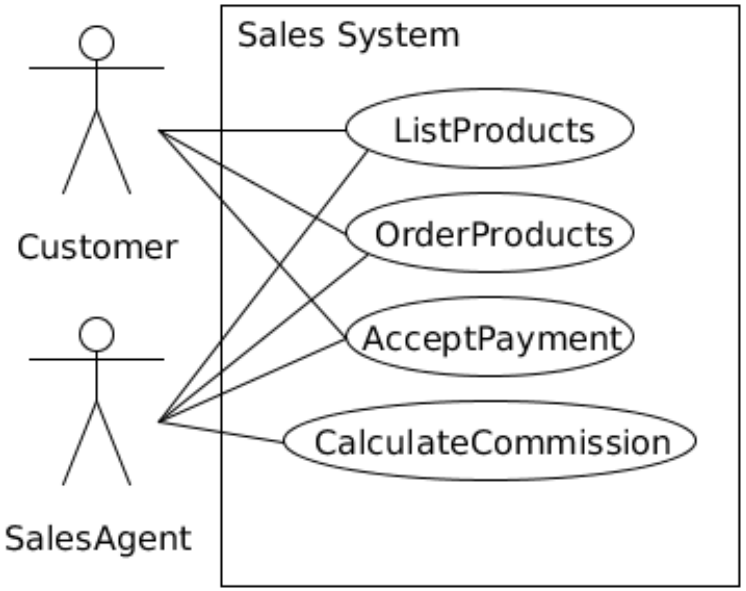
- *"A **specification** of sequence of **actions**, including variant sequences and error sequences, that a system, subsystem or class can perform by **interacting** with the with **outside actors**"* - *UML Reference Manual, 2nd Edition-2004*

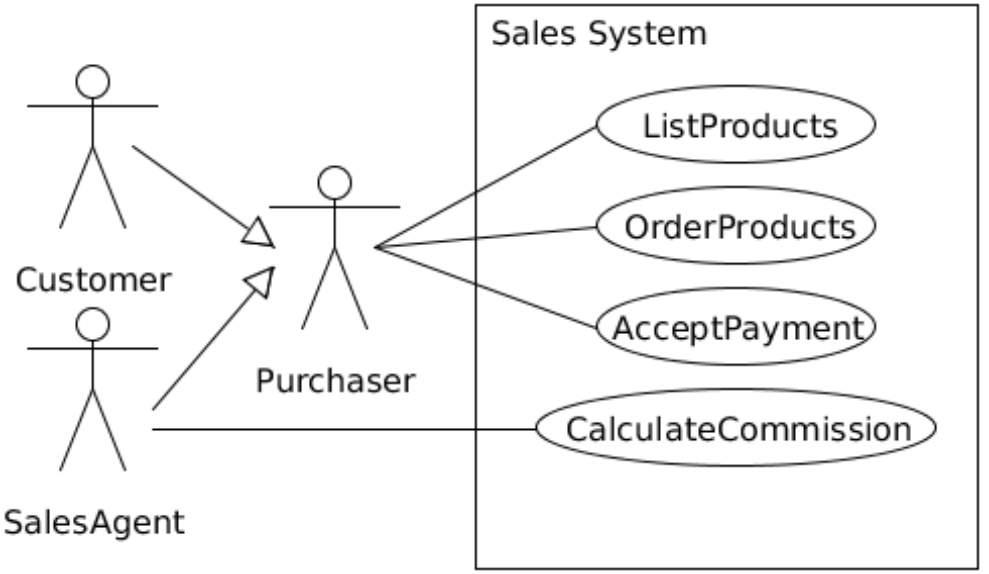
- A use case defines system behaviour during interactions with the actors

An Example

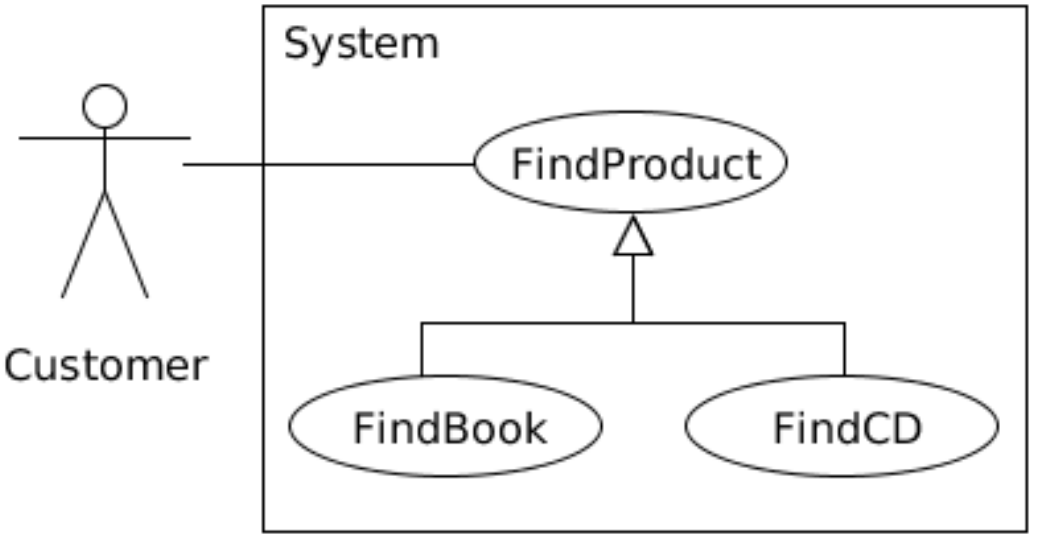

system boundary
communication relationship
Mail order system
PlaceOrder
CancelOrder
ValidateOrder
RequestCatalog
ShipProduct
ShippingCompany
actor
Customer
Dispatcher
use case

Actor Generalization





Use Case Generalization



Child use case may

- inherit features from the parent use case
- add new features
- override inherited features

Use Case Specification

id & name	**ID1: use case: PaySalesTax**
description	Pay sales tax to to the tax authority
actors	Primary actors: Secondary actors
pre conditions	Pre-Conditions: 1. it is the end of business quarter
main flow	Main Flow: 1. determine the amount of tax owed 2. prepare payment & other related documentation 3. send an electronic payment
post conditions	Post-Conditions: 1. tax authority receives the amount
alternate flows	Alternate flows: InvalidPaymentAmount Cancel

one main flow

0 or more alt flows may not merge with the main flow

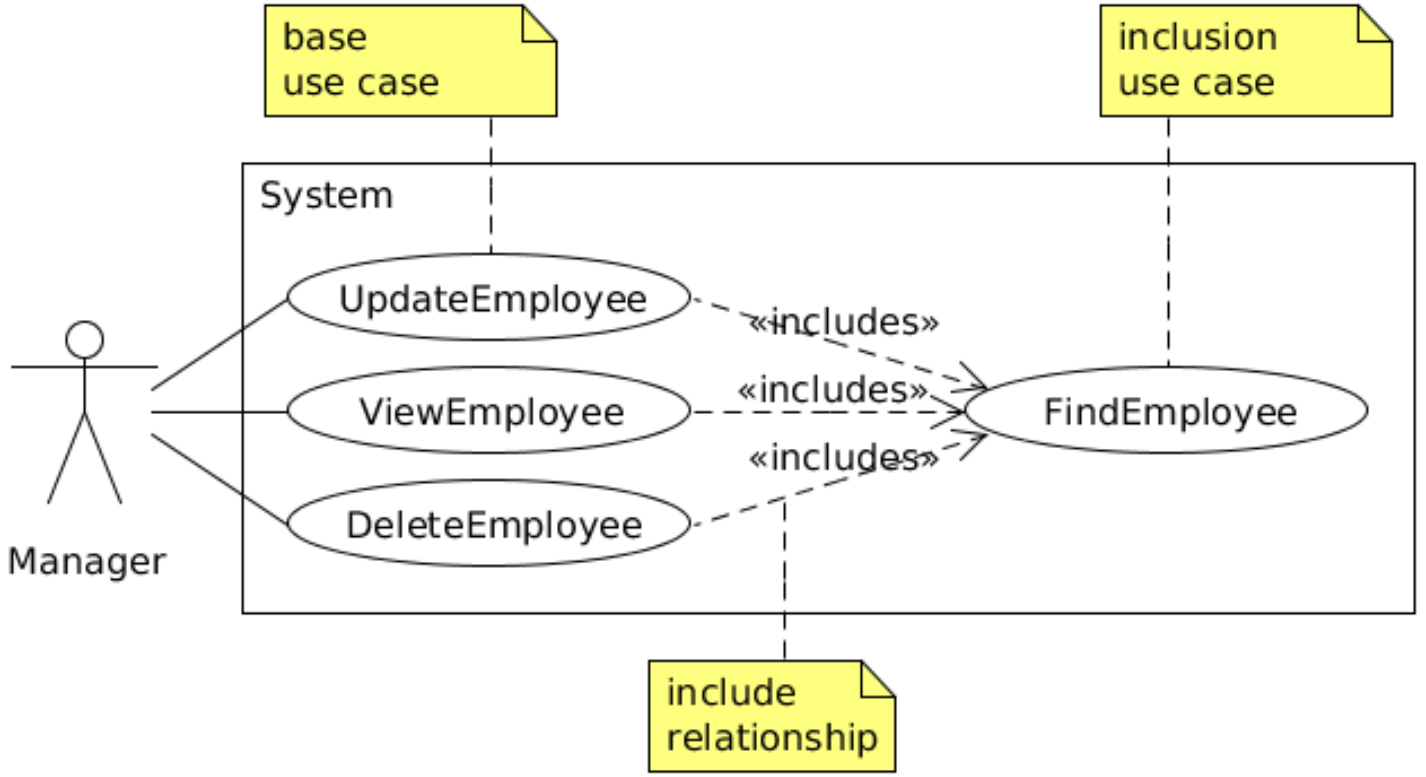
Use Case Generalization-Revisited

Use case feature	Inherit	Add	Override
Relationship	Y	Y	N
Extension point	Y	Y	N
Precondition	Y	Y	Y
Postcondition	Y	Y	Y
Step in main flow	Y	Y	Y
Alternative flow	Y	Y	Y

Use Case «include»

Purpose

- collect common use case steps into a single use case to be reused by other relevant (base) use cases



Use Case «extend»

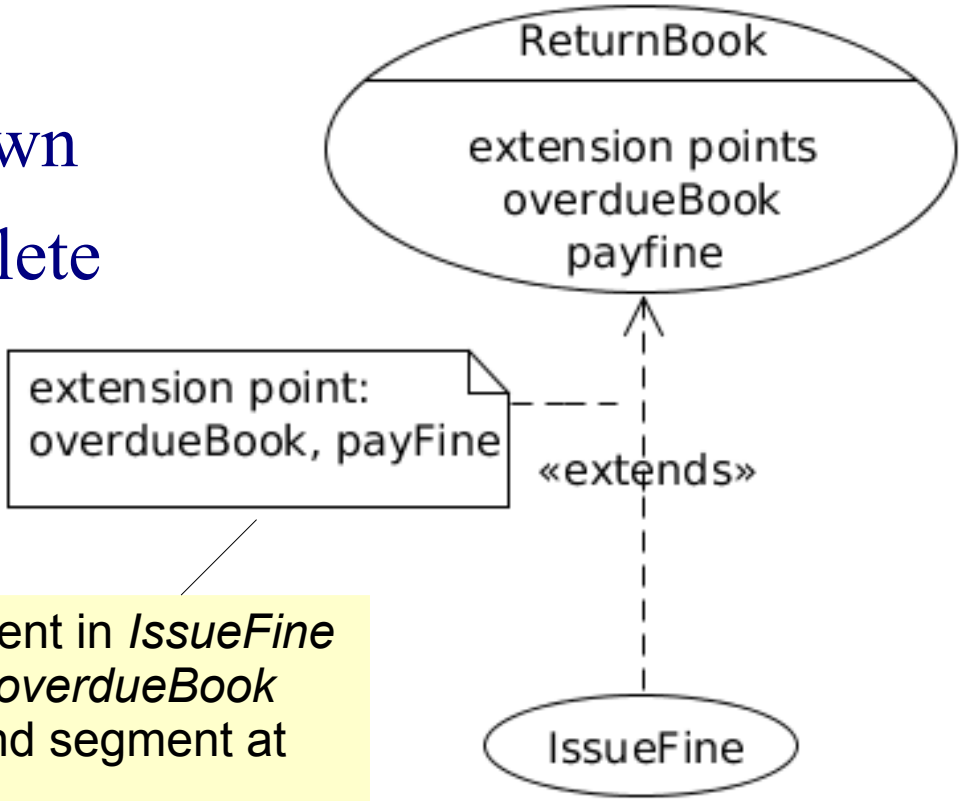
Purpose

- provides a mechanism to **insert new behaviour** into existing (base) use case
  - *base use case* provides extension points (hooks)
  - *extension use case* provides a set of *insertion segments*

Use Case «extend»

Observation

- base use case **does not know** about the extension use case
- extension use cases are **not complete** on their own
- base use cases are complete

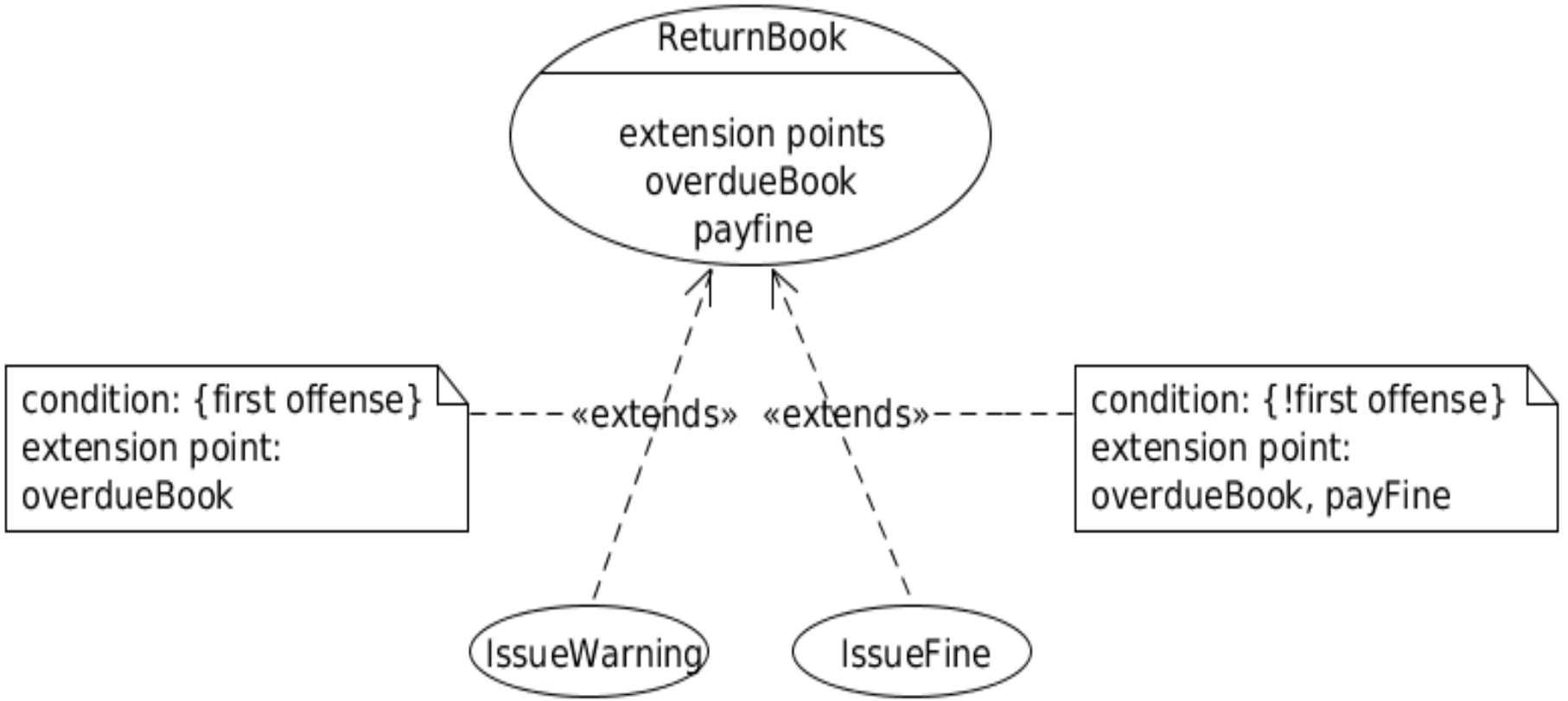


Use Case «extend»

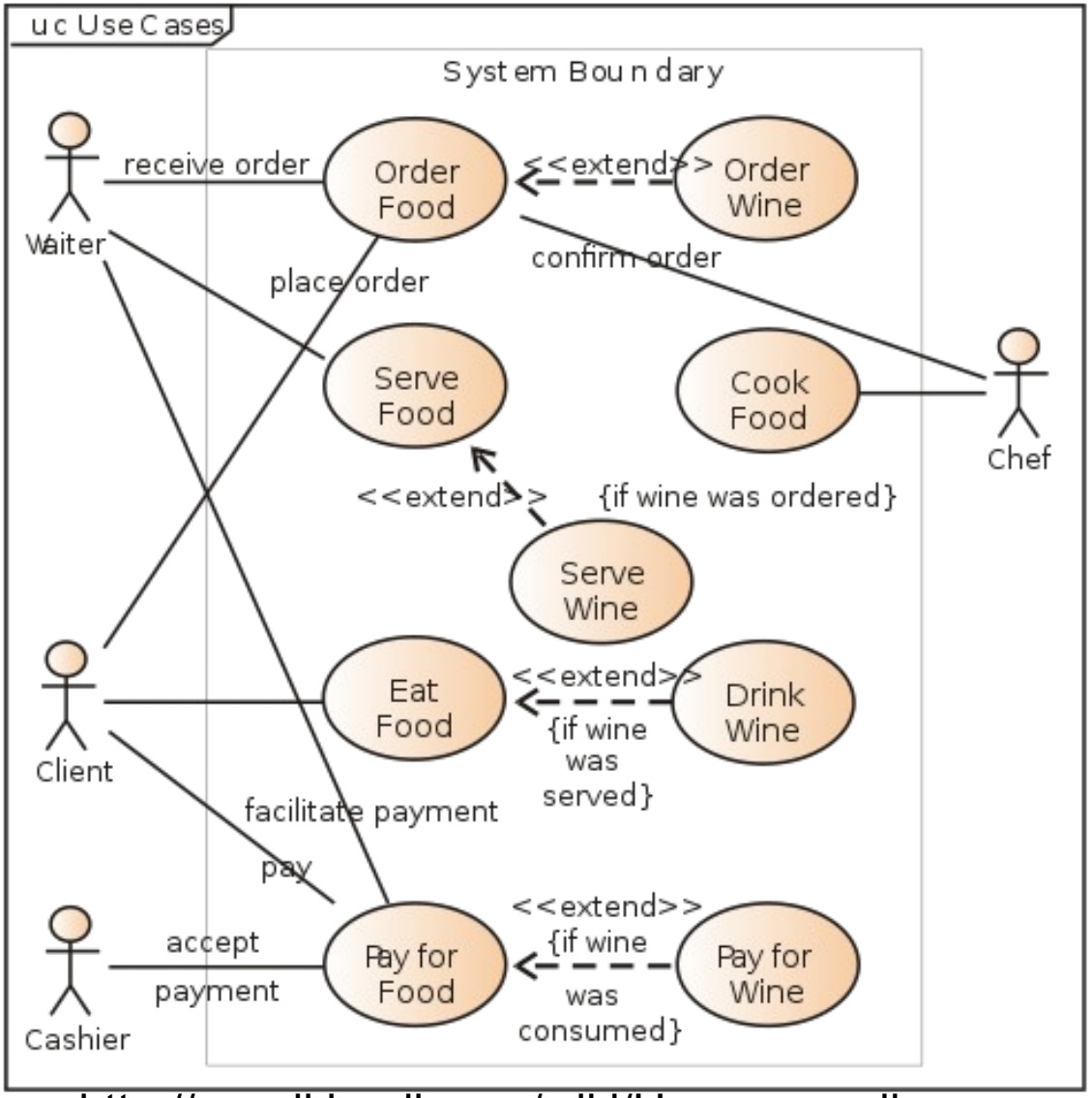
Rules

- «extend» relationship must identify one or more extension points in the base use case
  - otherwise the relationship refers to all extension points
- extension use case must have the same number of insertion segments as there are extension points in the «extend» relationship
- it is legal for legal for two extension use cases to «extend» the same base use case at the same extension point

Conditional Extensions



Use Case Modelling

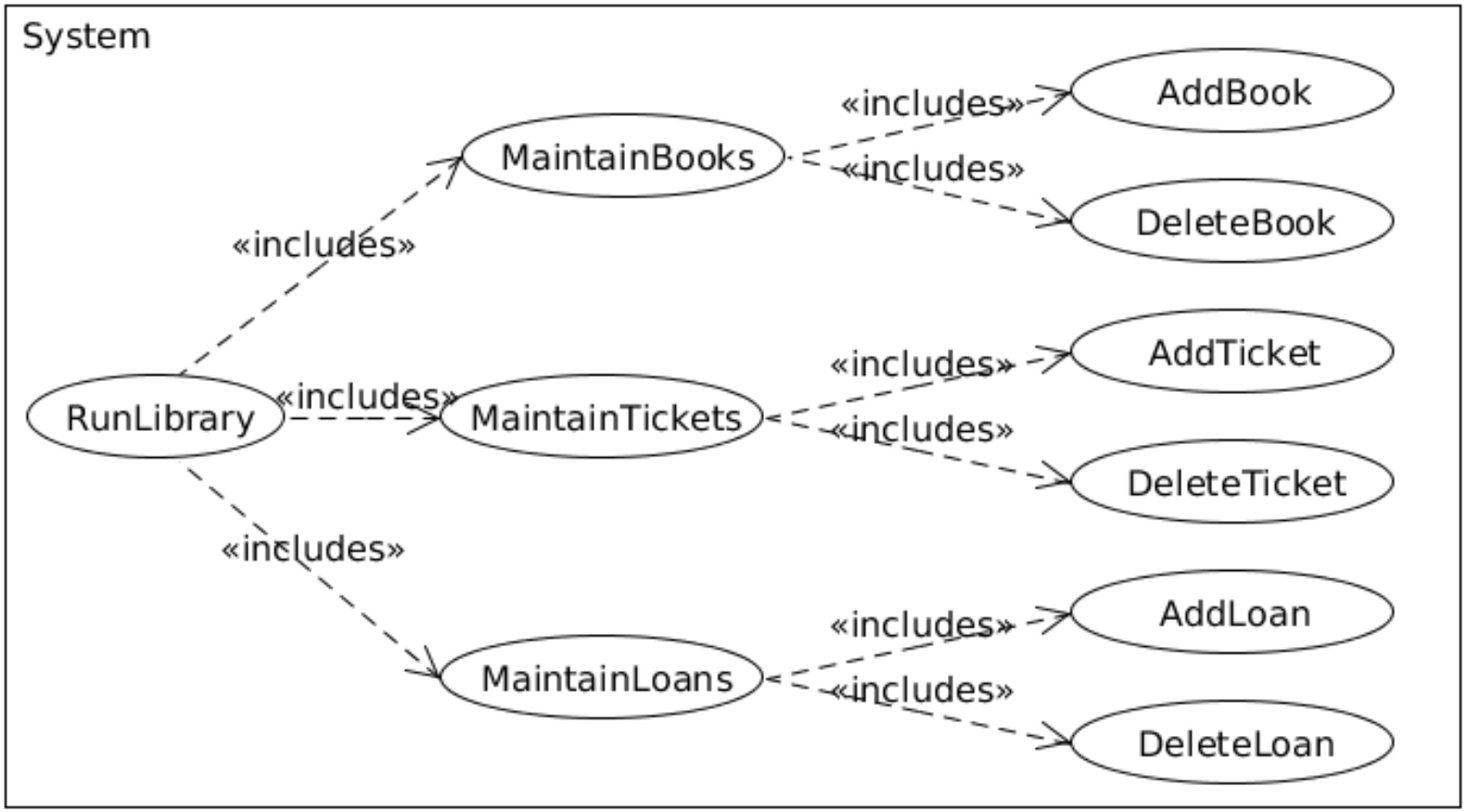


http://en.wikipedia.org/wiki/Use_case_diagram

Functional Decomposition

Good or Bad?

- what is functional decomposition?



Functional Decomposition

Observations

- focus on capturing the requirements
- it is not object design
- the higher level use cases might not be of interest
- model is complicated
- usually indicates that the analyst is viewing the system in a procedural way rather the OO paradigm

Verdict

- bad

Use Case & Requirements Tracing

Purpose

- link requirements to use cases
- many-to-many relationship

Requirement Tracing Matrix

- validate consistency
  - missing use cases
  - missing requirements

use cases

requirements	UC1	UC2	UC3	UC4
R1	X			
R2		X	X	
R3				X
R4	X			