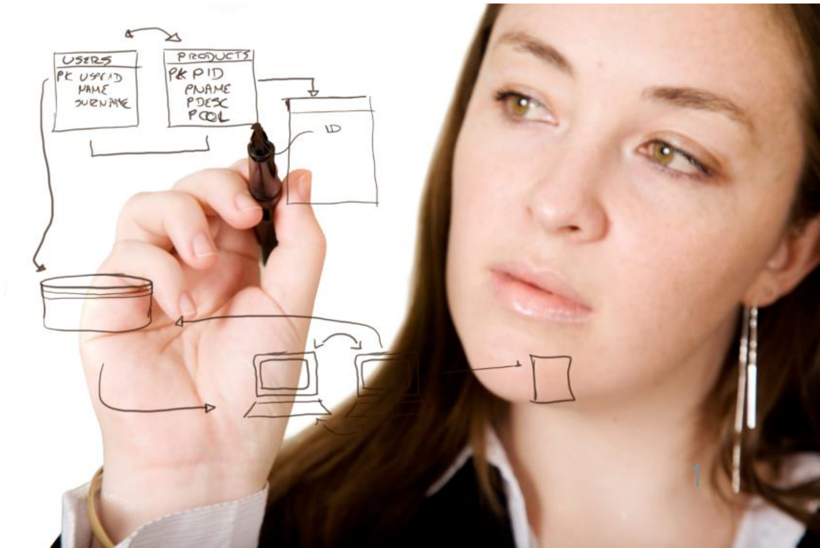
מודלים לפיתוח מערכות תוכנה
Software Systems Modeling

קורס 12003

סמסטר ב' תשע"ה

1. מבוא + UML



ד"ר ראובן יגל
robi@post.jce.ac.il

השבוע

- על הקורס
  - לוגיסטיקה – סילבוס
- מבוא למודלים
- UML
  - ריענון
  - תרחישי שימוש
- תרגיל 1 מידול בסיסי – חלק א'

המרצה

אתם?

- רקע וניסיון
- במודלים?
- מה מעניין אתכם במודלים לתוכנה?

הקורס

- חדש ובבניה! מבוסס מקורות שונים
- כולל למידה עצמית והתנסות
- אתם שותפים!
- מקורות עיקריים (כרגע):
  - Wei Le RIT Class
  - http://mdse-book.com/
- סילבוס

תכנית (ראשונית)

- UML
- MDA/MDE
- Modeling Tools
- Generating Models
- Formal Methods?
- Modeling Research
- Modeling and..
- Modeling Project

מקורות להיום

- Modeling
- UML
  - Fowler, "UML Distilled: A Brief Guide to the Standard Object Modeling Language“
  - Ambler, Introduction to Object-Orientation and the UML
  - Cockburn Writing Effective Use Cases
  - se-class requirement lecture http://jce-il.github.io/se-class/lecture/se03-requirements.pdf

מבוא למודלים

- Are Models Useful?
- From Coursera: Model Thinking https://www.coursera.org/course/modelthinking
  - One to many and many to one 1:55m

- RIT:

Overview

What is a model?

Why software modeling?

What to model?

How to obtain a model?

Why modeling?

- Modeling is a tool for design, verification and testing
- Modeling and simulation
- Not only software, but any systems
- address more challenging problems, such as parallel computing and distributed systems.

Why Software Modeling?

- Schedule and divide tasks
- Collaboration and communication (contract)
- Decomposing complexity for coding
- Checking for software (correctness, security)
- Refactoring code
- Reuse and automatic coding

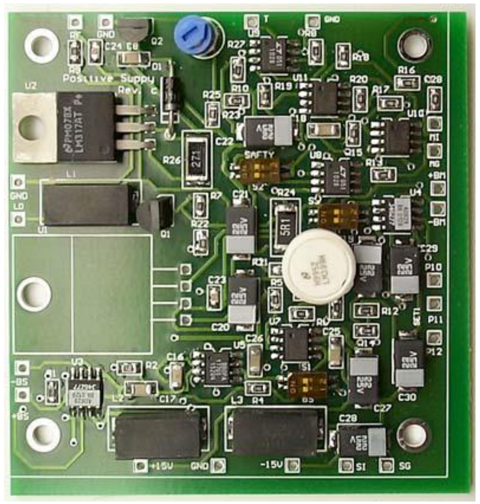
……

Other Questions

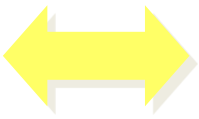
- Is Software modeling in real use?
- What about process development modeling?
- Modeling in Agile?

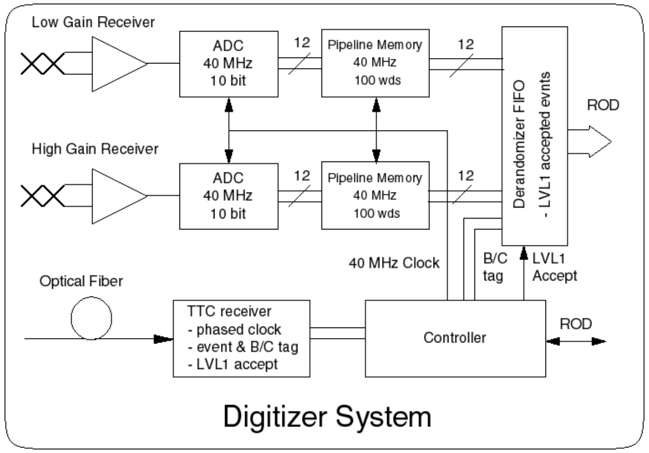
What is a model?

- Engineering model: abstraction
  A <u>reduced representation</u> of some system that highlights the properties of interest <u>from a given viewpoint</u>



system





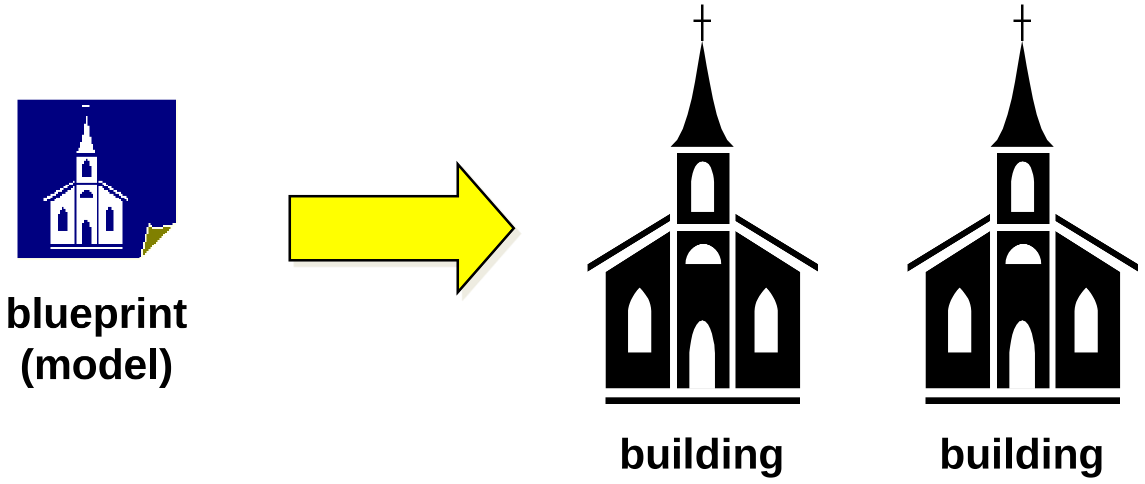
Low Gain Receiver
ADC 40 MHz 10 bit
12
Pipeline Memory 40 MHz 100 wds
12
Derandomizer FIFO - LVL1 accepted evnts
ROD
High Gain Receiver
ADC 40 MHz 10 bit
12
Pipeline Memory 40 MHz 100 wds
12
40 MHz Clock
B/C tag
LVL1 Accept
Optical Fiber
TTC receiver - phased clock - event & B/C tag - LVL1 accept
Controller
ROD
Digitizer System

Functional Model

- We don't see everything at once
- We use a representation (notation) that is easily understood for the purpose on hand

Intuitive Understanding

- A *model* is a description of something
  - *"a pattern for something to be made"* (Merriam-Webster)



- model ≠ thing that is modeled
  - The Map is Not The Territory

Levels of Abstraction and Reasons

- Business model
- Requirement
- Design and Algorithm
- Architecture
- Code

- Tracibility

Modeling Maturity Level

- Level 0: No specification
- Level 1: Textual
- Level 2: Text with Diagrams
- Level 3: Models with Text
- Level 4: Precise Models
- Level 5: Models only

What to Model?
- Structures, Behaviors, Requirement

- Overall architecture of the system
- System dependencies
- Complexity
- Flow of information through a system
- Business requirements
- Database organization and structure
- Security features (attack models)
- Configuration and environment

….

How to Obtain Models?

- Manually construct
- Automatically transform from one model to another
- Automatically recover from the code

…….

Challenges

Create Software Models

- Modeling languages: https://en.wikipedia.org/wiki/Modeling_language
  - General purpose and domain-specific languages
  - Formalism
  - Level of abstraction
- Models for software running in different platforms
  - Model-driven architecture
  - Views: PIM (computation), CIM (environment), PSM
- Models for software consistently changing at runtime (agent)
- Modularity, separate concerns

Manage Software Models

- Find information from the models (query)
- Correctness of the models:
  - Model consistencies
  - Model checking models
- Transformations
  - Decomposition
  - Composition
  - Between models
- Evolutions of models

Use Software Models

- Generate code
- Monitor runtime software behavior (interacting with environments, adaptation)
- Testing (model-based testing criteria and test input generation)

UML Modeling – Overview

UML Modeling

- A language: syntax and semantics
- Capture ideas, relations, decisions, requirements in a well-defined notations

AgileData.org: … all developers should have a basic understanding of the industry-standard Unified Modeling Language (UML). A good starting point is to understand what I consider to be the core UML diagrams – use case diagrams, sequence diagrams, and class diagrams – although as I argued in An Introduction to Agile Modeling and Agile Documentation you must be willing to learn more models over time.

UML Diagrams

- Structural : relations of objects (class diagram, component diagram)
- Behavioral : sequence of actions (activity diagram, sequence diagram)

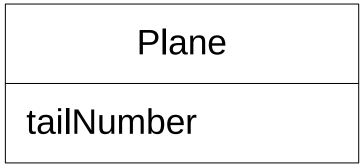
UML Views

One concept can be expressed in different graphs, choose one to express for your purposes, for your audiences

- Design – class, structure
- Deployment – configure, install,
- Implementation – state chart, interaction
- Process – performance, runtime behavior
- Use case

Domain concept
Design representation
Code representation

domain concept



visualization of domain concept

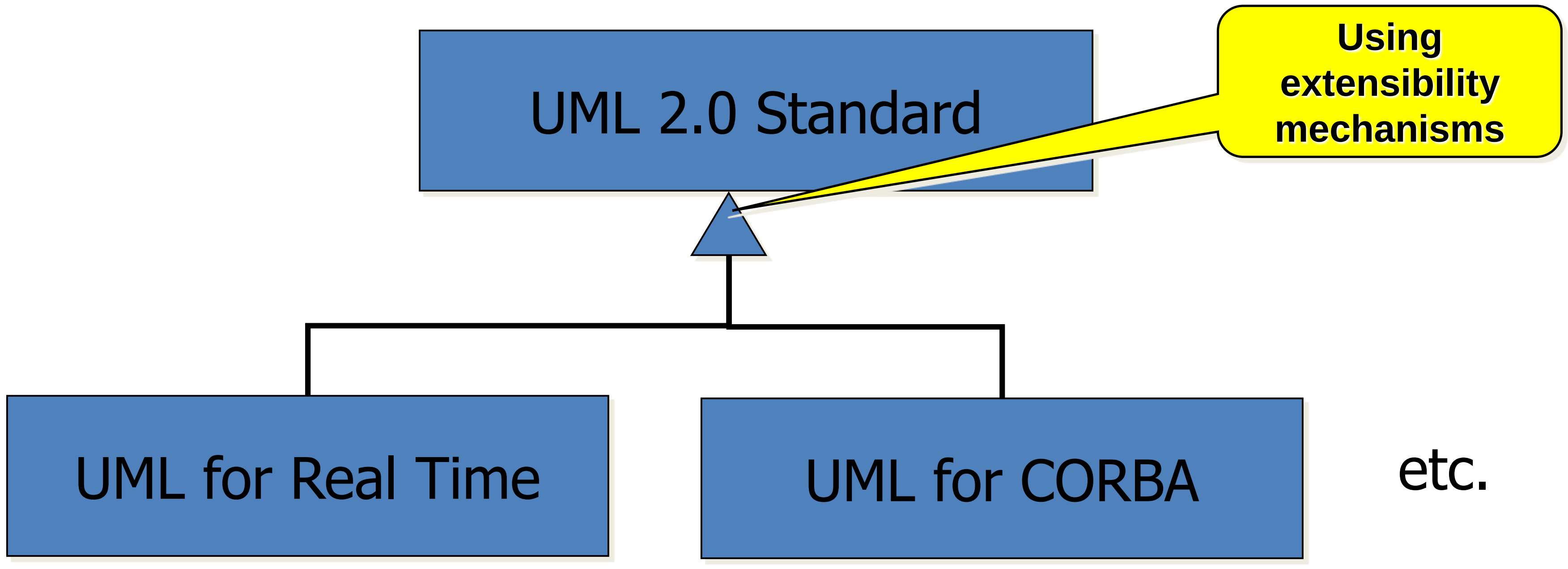
representation in an object-oriented programming language

```
public class Plane
{
private String tailNumber;

public List getFlightHistory() {...}
}
```

UML as a “Family of Languages”

- The standard can be specialized for different domains



Use case and use case diagram

Steps Before Coding

Phase	Action	Results
Initiation	Raise a business need	domain model, business use cases
Requirement	What to accomplish (abstract)	use case, activity diagrams
Design	How the system works (more details: software architecture, components, data types, algorithms)	component, class, sequence diagrams, formal specifications

Source of Requirements

- Initial requirements come from the customer, by:
  1. Documents, such as RFI/RFP
  2. Meetings, reports

- Advanced requirements come from the analysts, after studying scope and price
  1. Feasibility (technological, organizational etc)
  2. Prototypes

- Final requirements are stabilized in an iterative process.

Types of Requirements

Visible Functional Requirements

“The system will deliver cash to the customer”

“Cash will be delivered after card was taken out”

Qualitative Requirements

“The authorization process will take no more than 1 sec”

“The user interface will be easy to use”

Hidden Requirements

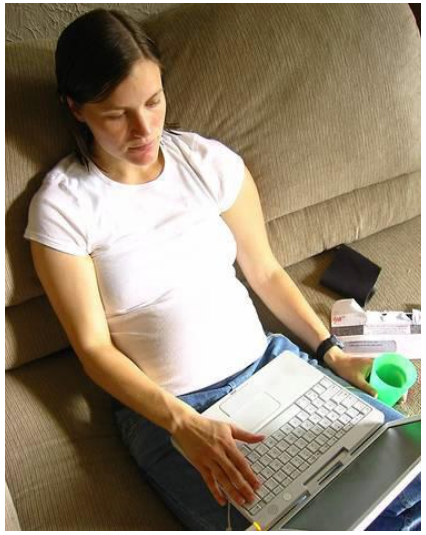
“Database maintenance processes will occur every night”

Intro: Use Case and Use Case Diagram

Use Cases as Means of Communication

Customer

Designer


User

The use case should stimulate a discussion about **what the system should do, mainly with people who are outside of the development team.**

Use Case

A use case is a **contract of an <u>interaction</u> between the system and an <u>actor</u>.**

Use Case Diagram: an integration of use cases

Use Case Diagram

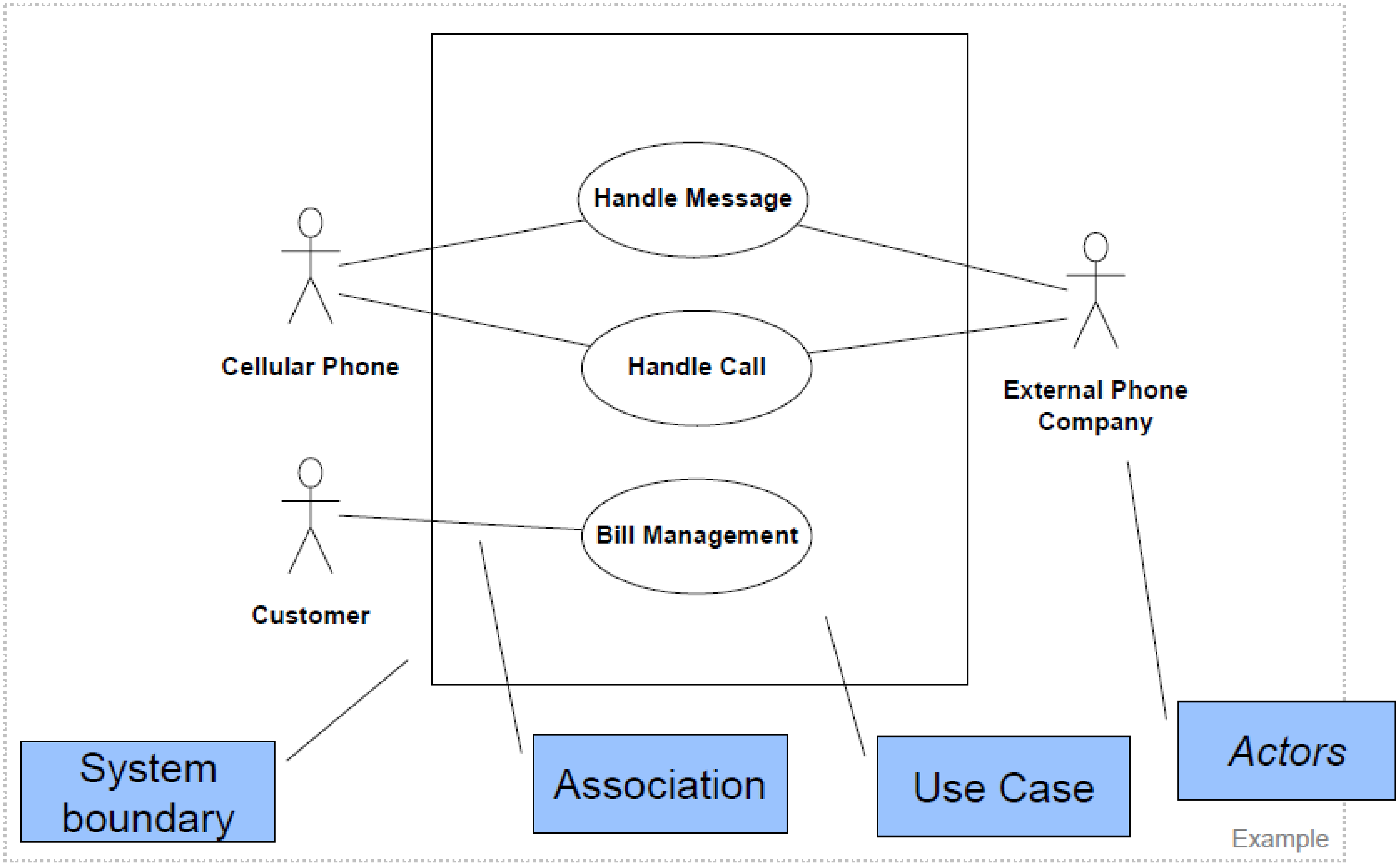
A use case diagram illustrates a set of use cases for a system, the actors, and the interactions between actors and use cases.

A graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases.

Use Case Diagram Objectives

1. Create a semi-formal model of the functional requirements

2. Analyze and define:
   - Scope
   - External interfaces
   - Scenarios and reactions


Handle Message
Cellular Phone
Handle Call
External Phone Company
Customer
Bill Management
System boundary
Association
Use Case
Actors
Example

What makes a good Use Case Diagram?

Lack of ambiguity

- Each requirement must be interpreted in a single manner.

Completeness

- The collection of all use cases is everything that can be done to/with the system.

Consistency

- Requirements should not conflict with each other. If there are, tradeoffs must be detected and discussed.

Avoid design

- Requirements should raise a need, not answer it.

Construct a Use Case Diagram

Finding actors

External objects that produce/consume data:

1. Must serve as sources and destinations for data
2. Must be external to the system
3. Actors vs. Stakeholders

   Humans

   Machines

   External systems

   Sensors

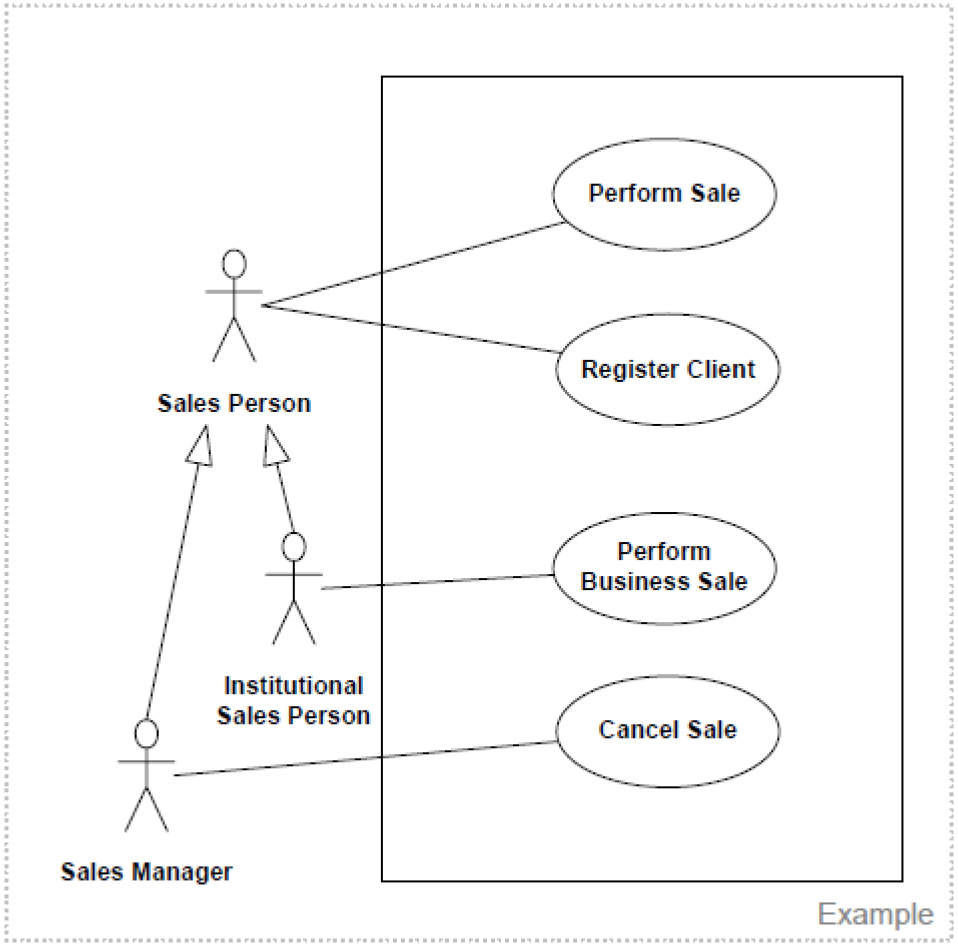
Actor Relationships – Generalization/Specialization

Define hierarchy for actors

Notation

The child actor inherits all use-cases associations

Should be used if (and only if), the specific actor has more responsibility than the generalized one (i.e., associate with more use-cases)



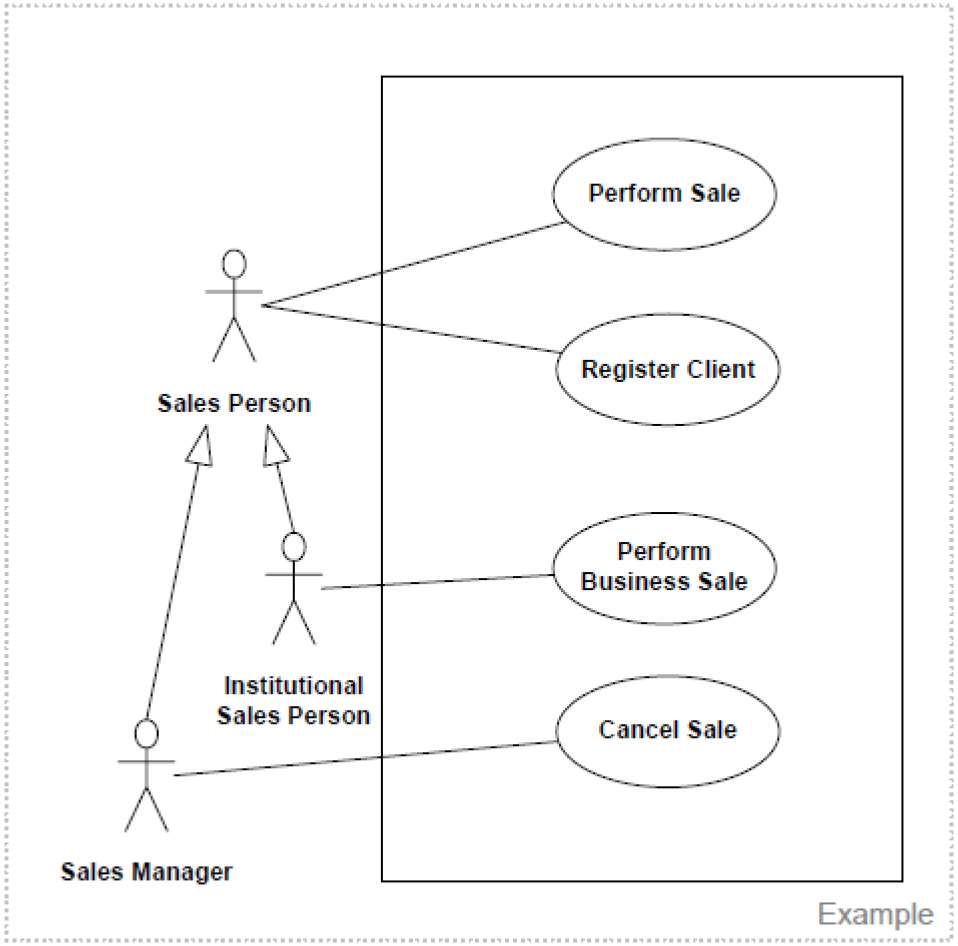
Association: Actor and Use Case

Solid line:

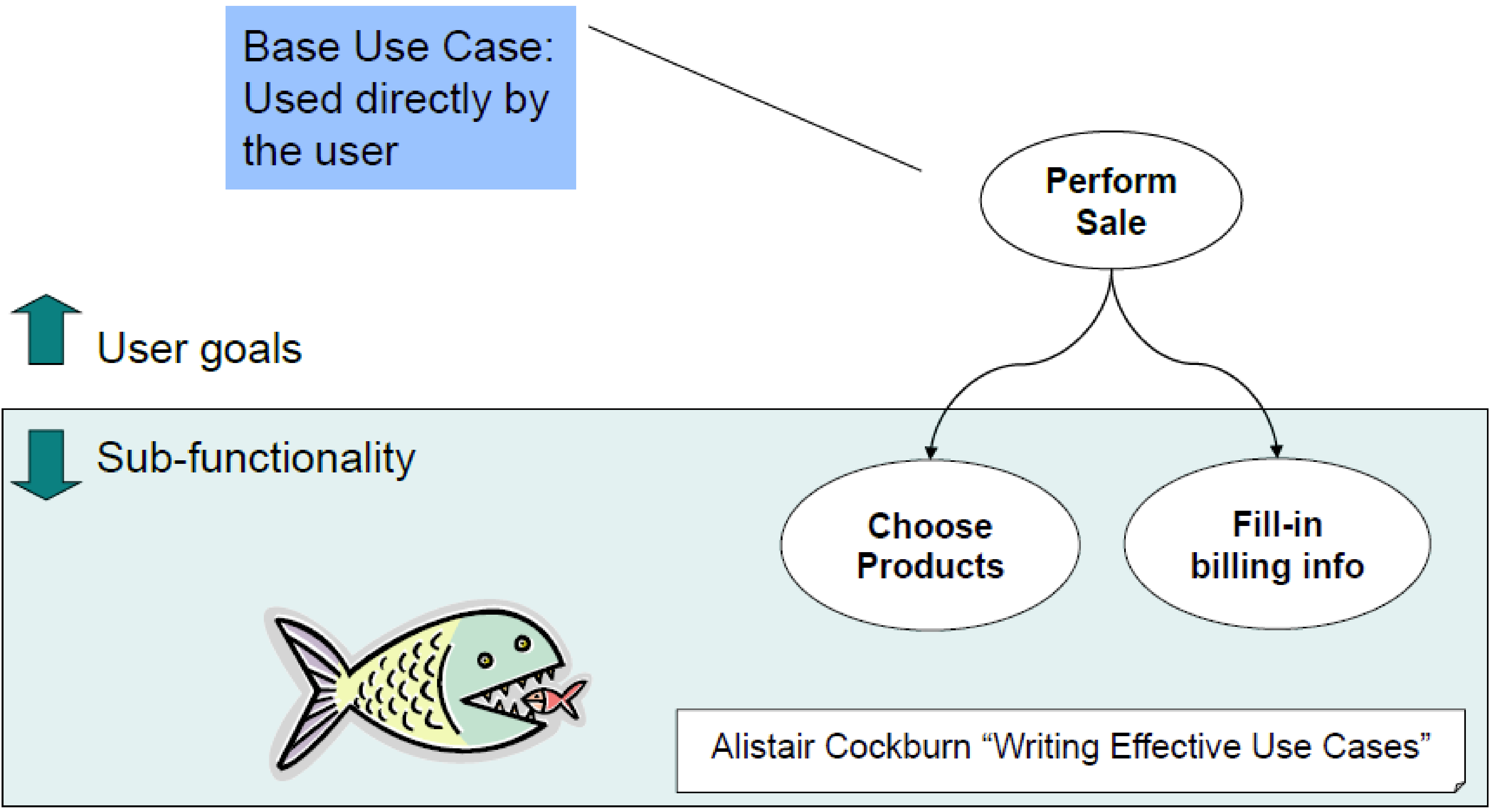
Interaction between actors and use case

Arrowhead (optional)

- Control flow
- Initial invocation, primary actor



Use Case Levels


Base Use Case: Used directly by the user
Perform Sale
User goals
Sub-functionality
Choose Products
Fill-in billing info
Alistair Cockburn “Writing Effective Use Cases”

Use Case Relationships

- Goal: enable flexibility in requirements specification
  1. Isolating functionality
  2. Enabling functionality sharing
  3. Breaking functionality into manageable chunks

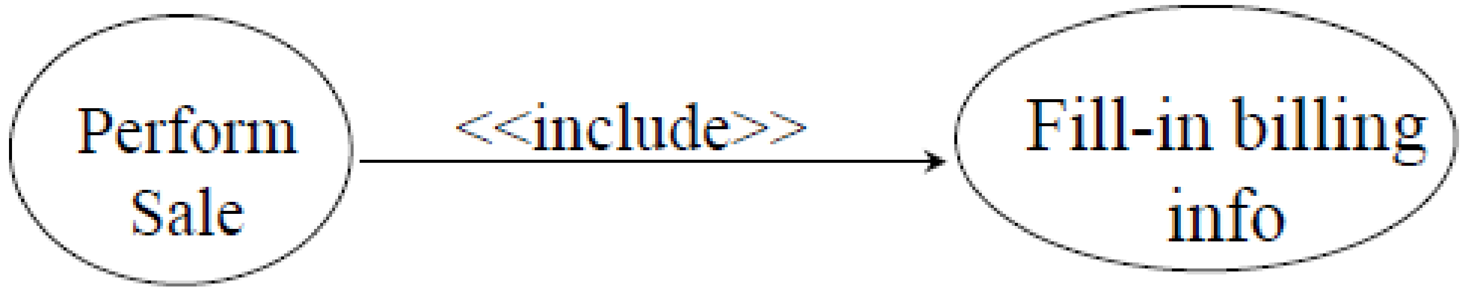
- Relationships
  1. Include
  2. Extend
  3. Generalization

Include

Goal:

1. Decomposing complicated behavior
2. Centralizing common behavior

the behavior of the included use case is inserted into the behavior of the including use case - The first use case often depends on the outcome of the included use *case*.

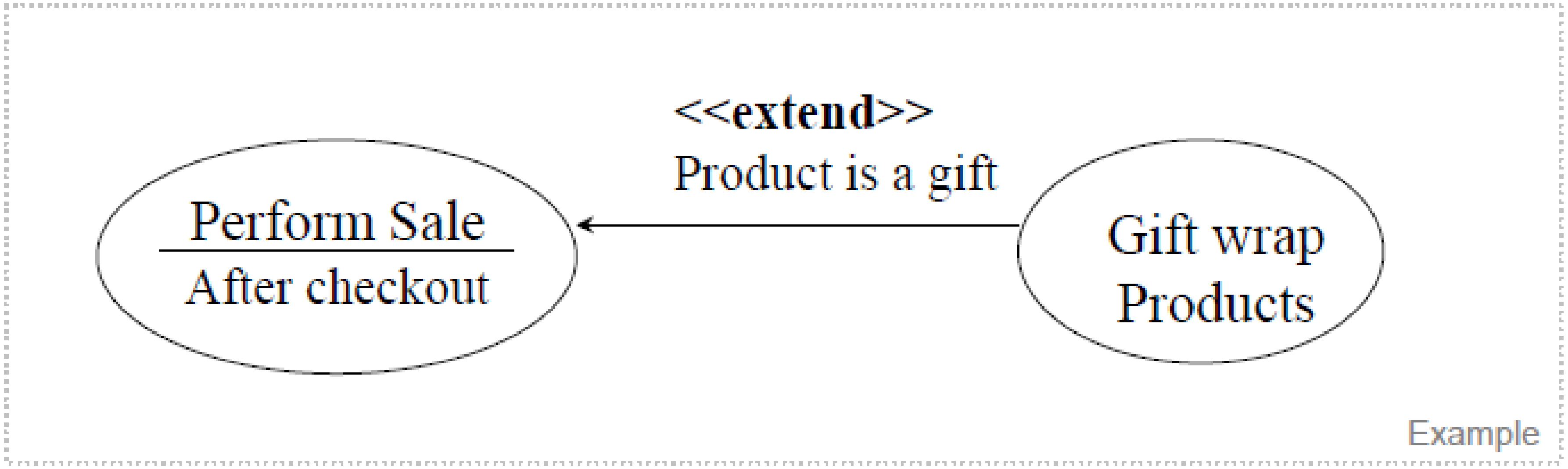


Example

Extend

the behavior of the extension use case may be **inserted** in the extended use case under some conditions

Note the direction of the arrow
The base use-case does not know which use-case extends it



Detailed Use-case

See se-class…

Example: Amazon

Actors?

Base Use Cases?

Include?

Extend?

תרגיל 1- חלק א'

- תרחיש שימוש לאמזון
- שימוש ב- CASE ליצירת קובץ בתבנית סטנדרטית XMI? (ArgoUML, Eclipse/EMF)
- שמירה במאגר git + ויקי להסברים (Readme)
  - מומלץ מאגר פרטי לתוצרי הקורס
- הגשת קישור למאגר (עד ההרצאה עוד שבועיים)

סיכום

- הקורס
- מבוא
  - מידול
- UML
  - תרחישי שימוש