Requirements Elicitation with Use Cases

Jörg Kienzle & Shane Sendall
School of Computer Science
McGill University, Montreal, QC, Canada

Overview

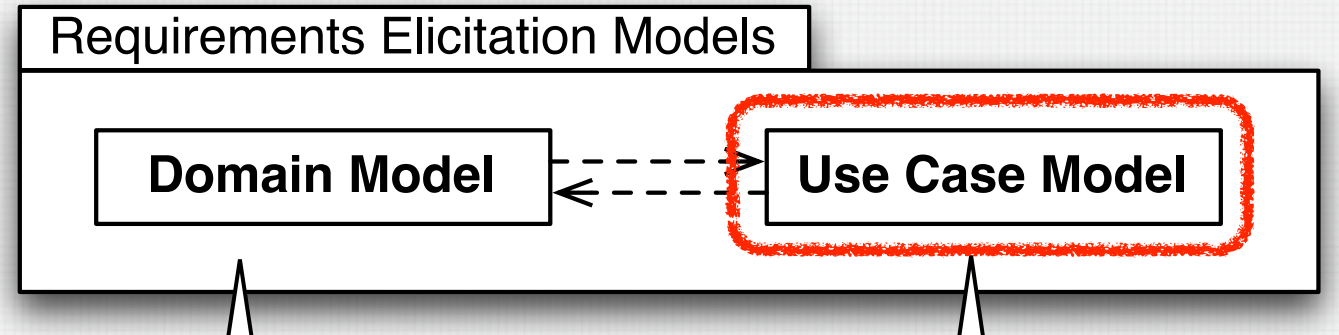
- Use Case Definition
- Actors
- System Boundary
- Textual Template
- Granularity
- UML Use Case Diagram
- Use Case Process
- Use Case Checklist
- Exercises

Requirements Elicitation Activity

- Discover the requirements of the system to develop
  - Users and Stakeholders
  - Goals and Softgoals
  - User expectations
  - Functional requirements
  - Non-functional requirements / qualities
    - Distribution
    - Security
    - Safety
    - Reliability
    - Fault Tolerance
    - Availability

Fondue Models: Requirements Elicitation

Start with either one, or establish them simultaneously



UML Class Diagram,
describing the concepts of the problem domain and their relationships

UML Use Case Diagram + textual template,
describing the different ways users / stakeholders interact with the system

What are Use Cases Good For?

- Discover and document the functional requirements of the desired system
  - In a way that *all important participants of a project can understand*
  - In a way that is clearly *related to the motivation* for the system (e.g., business vision)
  - In a complete, consistent, and verifiable manner

Use Cases

- Use Cases capture interactions between the system and the environment to achieve *user goals*
- Use cases capture who (actor) does what (interaction) with the system, with what purpose (goal), without dealing with system internals.
- A complete set of use cases specifies all the different ways to achieve a goal with the system, and thus defines all behaviour required of the system, bounding the scope of the system.
- Designed to be understood by non-technical parties

Use Case Life Cycle

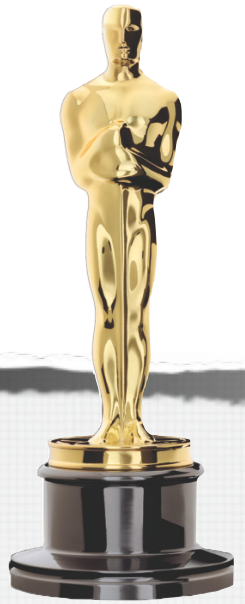
- Jacobson (1992):
  - A use case is a sequence of transactions performed by a system, which yields an observable result of value for a particular actor.
  - A transaction consists of a set of actions performed by a system. A transaction is invoked by a stimulus from an actor to the system, or by a timed trigger within the system.
  - A transaction consists of 4 steps:
    - 1. The primary actor sends the request and the data to the system;
    - 2. The system validates the request and the data;
    - 3. The system alters its internal state;
    - 4. The system replies to the actor with the result.

Use Case Comments

- Being a black-box view of the system, use cases are a good approach for finding the What rather than the How
- A black-box matches users view of the system: things going in and things coming out
- Use cases force one to look at exceptional as well as normal behaviour
- Helps us to surface hidden requirements
- Use cases can help formulate system tests:
  - “Is this use case built into the system?”

Actors: What Are They?

- "The actors represent what interacts with the system." [Jacobson '92]
- An actor represents a role that an external entity such as a user, a hardware device, or another system plays in interacting with the system
- A role is defined by a set of characteristic needs, interests, expectations, behaviors and responsibilities [Wirfs-Brock '94]



Actor Comments

- An actor communicates by sending and receiving messages to/from the system under development.
- A use case is not limited to a single actor.
- Sources, i.e. how to discover actors:
  - People: Workshops, Meetings, etc.
  - Documentation: user manuals and training guides are often directed at roles representing potential actors
  - Domain Model

How to Find Actors

- Look for external entities that interact with the system
  - Which persons interact with the system (directly or indirectly)? Don't forget maintenance staff!
  - Will the system need to interact with other systems or existing legacy systems?
  - Are there any other hardware or software devices that interact with the system?
  - Are there any reporting interfaces or system administrative interfaces?

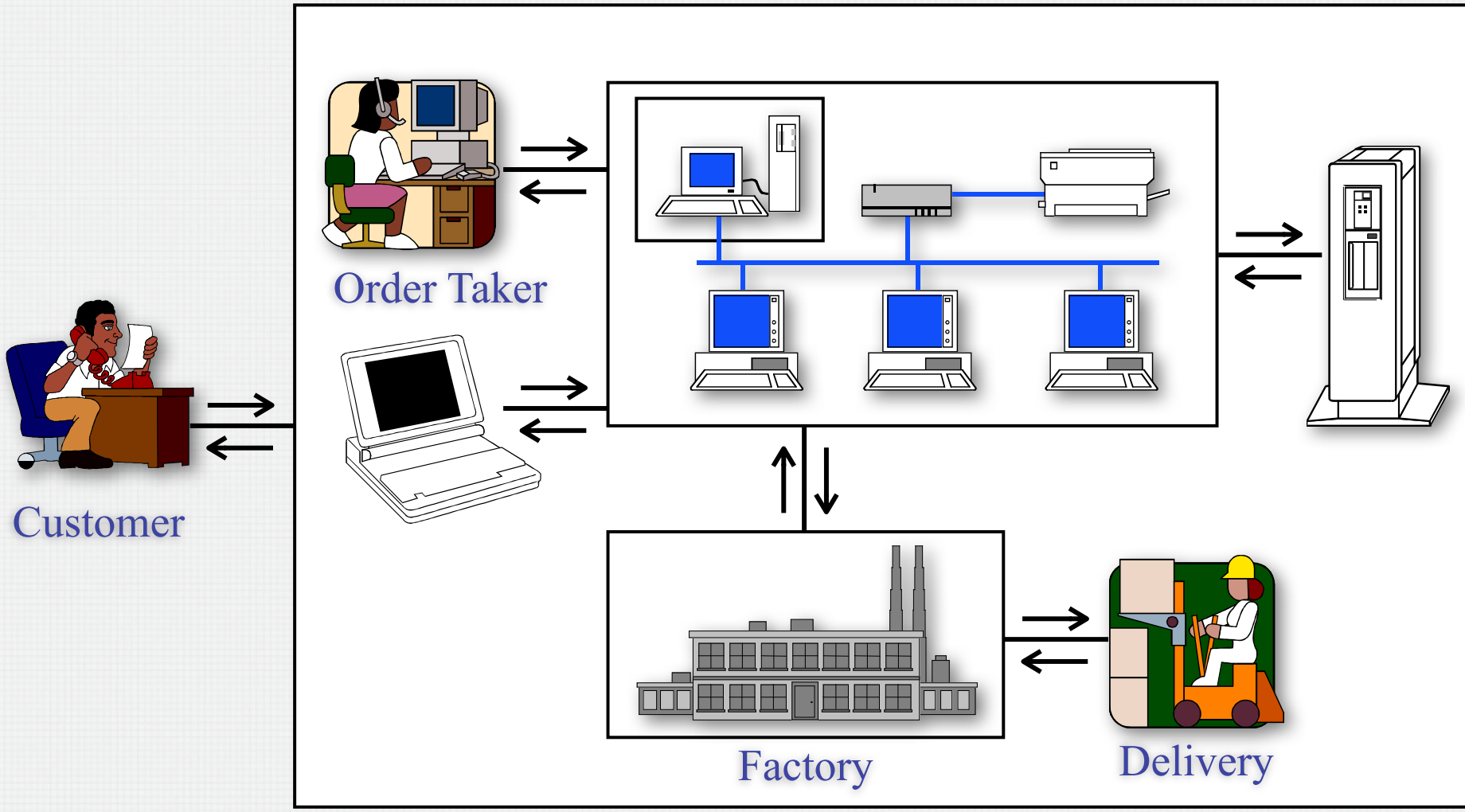
Actor Categories

- Jacobson (1992) categorized actors into two types:
  - Primary Actor
    - Actor with goal on system (sometimes off-stage)
    - Obtains value from the system
    - Sometimes, primary actors interact with the system through *facilitator actors*
  - Secondary Actor
    - Actor with which the system has a goal
    - Supports “creating value” for other actors
  - Facilitator Actor
    - Actor or device that is used by a primary actor or secondary actor to communicate with the system

System Boundary

- The system boundary defines the separation between the system and its environment
  - Clear definition is extremely important
- Movement of the system boundary often has a large effect on what should be built
- A common area of conflict between stakeholders arises when they assume different system boundaries, and hence refer to different systems!

System Boundary Example



Use Case Description

- Use cases are primarily textual descriptions
- Use case steps are written in an easy-to-understand structured narrative using the vocabulary of the application domain
- A use case description includes
  - How the use case starts and ends
  - The context of the use case
  - The actors that interact with the system
  - All the circumstances in which the primary actor's goal is reached and not reached
  - What information is exchanged

Structured Use Cases

- Fondue provides a use case template
  - Use Case Name
  - Scope
  - Level
  - Intention in Context
  - Multiplicity
  - Primary Actor
    - Secondary Actors
  - Main Success Scenario
    - Sequence of Interaction Steps
  - Extensions & Exceptions
    - Additional / Alternative Interaction Steps

Interaction Steps

- An interaction step either
  1. Refers to a lower level use case
  2. Describes a base interaction step between the system and the environment
     - A base interaction step must always contain the word *System* and (at least) an actor and
       - Describes an *input interaction*, when an actor sends an input event to the system, or
       - Describes an *output interaction*, when the system sends an output event to an actor
  3. Describes an optional system processing step or communication happening in the environment that is included for clarity, or to properly describe the control flow of actions.
- Interaction steps are numbered to reflect their sequencing
  - The "." notation, i.e. "3.1", denotes sequential "sub"steps
  - Letters, i.e. "3a", denotes alternatives to a step
  - The "||" symbol denotes parallelism

Single-Cabin Elevator Example

Use Case: TakeElevator
Scope: Elevator Control System
Primary Actor: User
Intention: The intention of the *User* is to take the elevator to go to a destination floor.
Level: User Goal
Multiplicity: Many *Users* can take the elevator simultaneously.
Main Success Scenario:
1. *User* **Call[s]Elevator**
2. *User* **Ride[s]Elevator**

Extensions:
1a. Cabin is already at *User's* floor…
1b. *User* is already inside…

Use Case: CallElevator
Primary Actor: User
Intention: User wants to call...
Level: Subfunction
Main Success Scenario:
1. *User* pushes button, indicating to *System* in which direction she wants to go.
2. *System* acknowledges *User's* request.
3. *System* **schedules ElevatorArrival** for the floor the *User* is currently on.

Extensions:
2a. The same request already exists. System ignores the request…

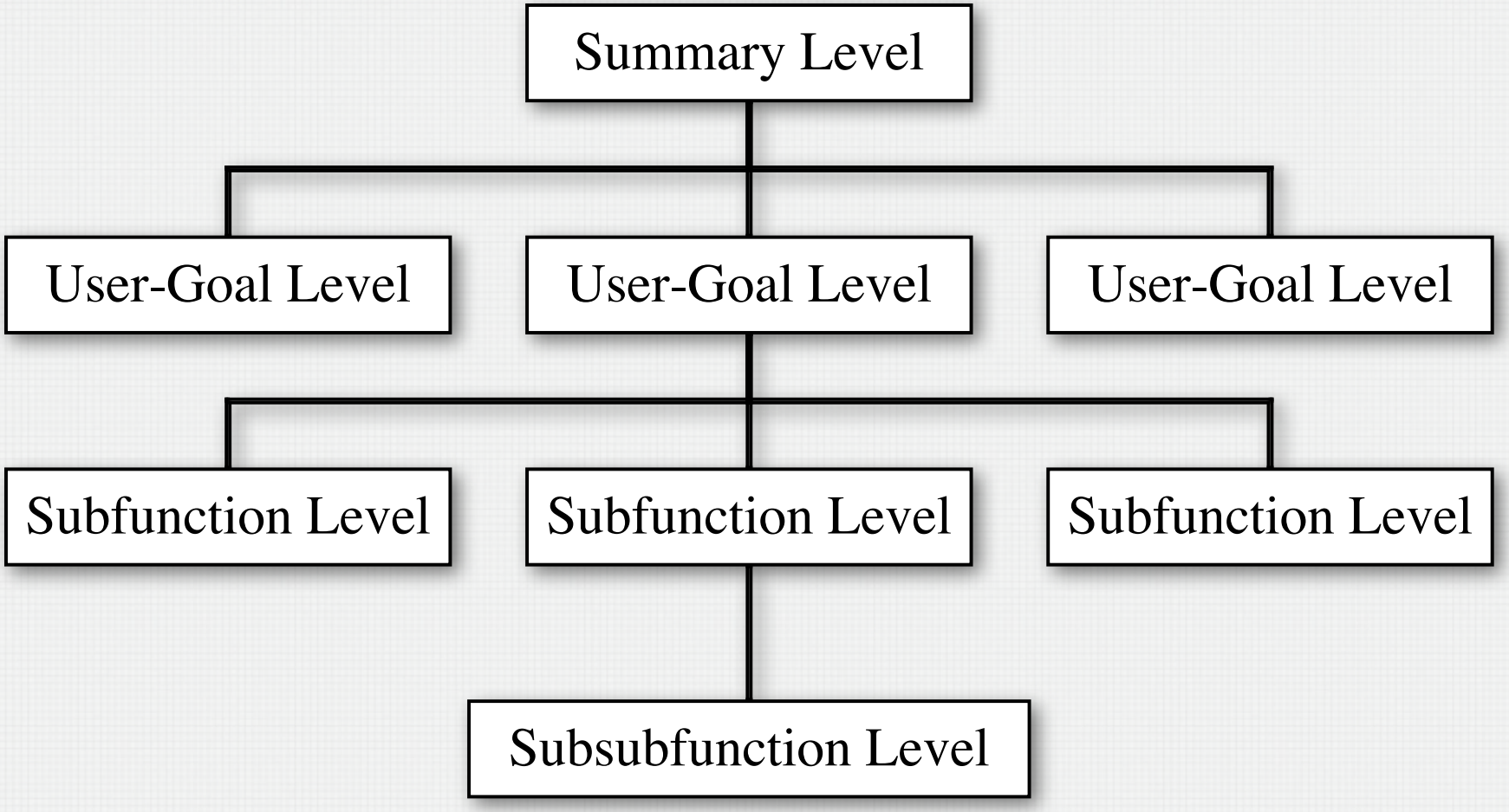
Elevator Arrival Example

Use Case: ElevatorArrival
Intention: System wants to move the elevator to the User's destination floor.
Level: Subfunction
Multiplicity: The elevator system having only one cabin, there can only be one instance of ElevatorArrival executing at a given time.
Main Success Scenario:
1. *System* asks *Motor* to start moving in the direction of the destination floor.
2. *Floor Sensor* informs *System* that elevator is approaching destination floor.
3. *System* requests *Motor* to stop.
4. *System* requests *Door* to open.

Granularity of Use Cases

- Summary Level
  - Are large grain use cases that encompass multiple lower-level, user goal use cases; they provide the context (lifecycle) for those lower-level use cases
  - They can act as a table of contents for user goal level use cases
- User Goal Level
  - Are usually done by one actor, in one place, at one time; the (primary) actor can normally go away happy as soon as this goal is completed
  - Achieve a single, discrete, complete, meaningful, and well-defined task of interest to an actor
- Subfunction Level
  - Provide “execution support” for user-goal level use cases; they are low-level and need to be justified, either for reasons of reuse or necessary detail
  - Often it is not clear who the primary actor of a subfunction-level use case is
- The interaction step at one level of abstraction forms the “Why” for the next level down

Use Case Hierarchy



Summary Level Example

Use Case: Manage Funds By Bank Account
Scope: Bank Accounts and Transactions System
Level: Summary
Intention in Context: The intention of the *Client* is to manage his/her funds by way of a bank account. *Clients* do not interact with the System directly; instead all interactions go through either: a Teller, a Web Client, or an ATM, which one depends also on the service.
Multiplicity: Many Clients may be performing transactions and queries at any one time. Each Client performs its transactions sequentially.
Primary Actor: Client
Main Success Scenario:
1. *Client* opens an account.
2. *Client* identifies with *System*.

Step 3 can be repeated according to the intent of the Client

3. *Client* performs task on account:

    deposit money, withdraw money, transfer money, get balance.

4. *Client* closes his/her account.

Extensions:
3a. *System* fails to identify the client; use case continues at step 2.

User-Goal Use Case Example

Use Case: Deposit Money
Scope: Bank Accounts and Transactions System
Level: User Goal
Intention in Context: The intention of the *Client* is to deposit money on an account. *Clients* do not interact with the System directly; instead, for this use case, a *Client* interacts via a *Teller*.
Multiplicity: Many *Clients* may be performing deposits at any one time. A *Client* only requests one deposit at a given time.
Primary Actor: Client
Facilitator Actor: Teller
Main Success Scenario:
Client requests Teller to deposit money on an account, providing sum of money.
1. *Teller* requests *System* to perform a deposit, providing deposit transaction details.
2. *System* validates the deposit, credits account with the requested amount, records details of the transaction.
3. *System* informs *Teller* that deposit was successful.
Extensions:
2a. *System* ascertains that it was given incorrect information:
    2a.1 *System* informs *Teller* about error; use case continues at step 2.

Subfunction Level Use Case Example (1)

Use Case: Identify Client
Scope: Automatic Teller Machine (ATM for short)
Level: Sub-Function
Intention in Context: The intention of the Client is to identify him/herself to the System. A project (operational) constraint states that identification is made with a card and a personal identification number (PIN).
Multiplicity: Only one client can use the ATM for identification at a given time.
Primary Actor: Client
Secondary Actors: Card Reader, Bank Server, Numeric Keyboard
Main Success Scenario:
Client inserts card into Card Reader.
1. *Card Reader* informs *System* of card details.
2. *System* validates card type.
3. *Client* provides PIN to *System* using the *Numeric Keyboard*.
4. *System* requests *Bank Server* to verify identification information.
5. *Bank Server* informs *System* that identification information is valid.

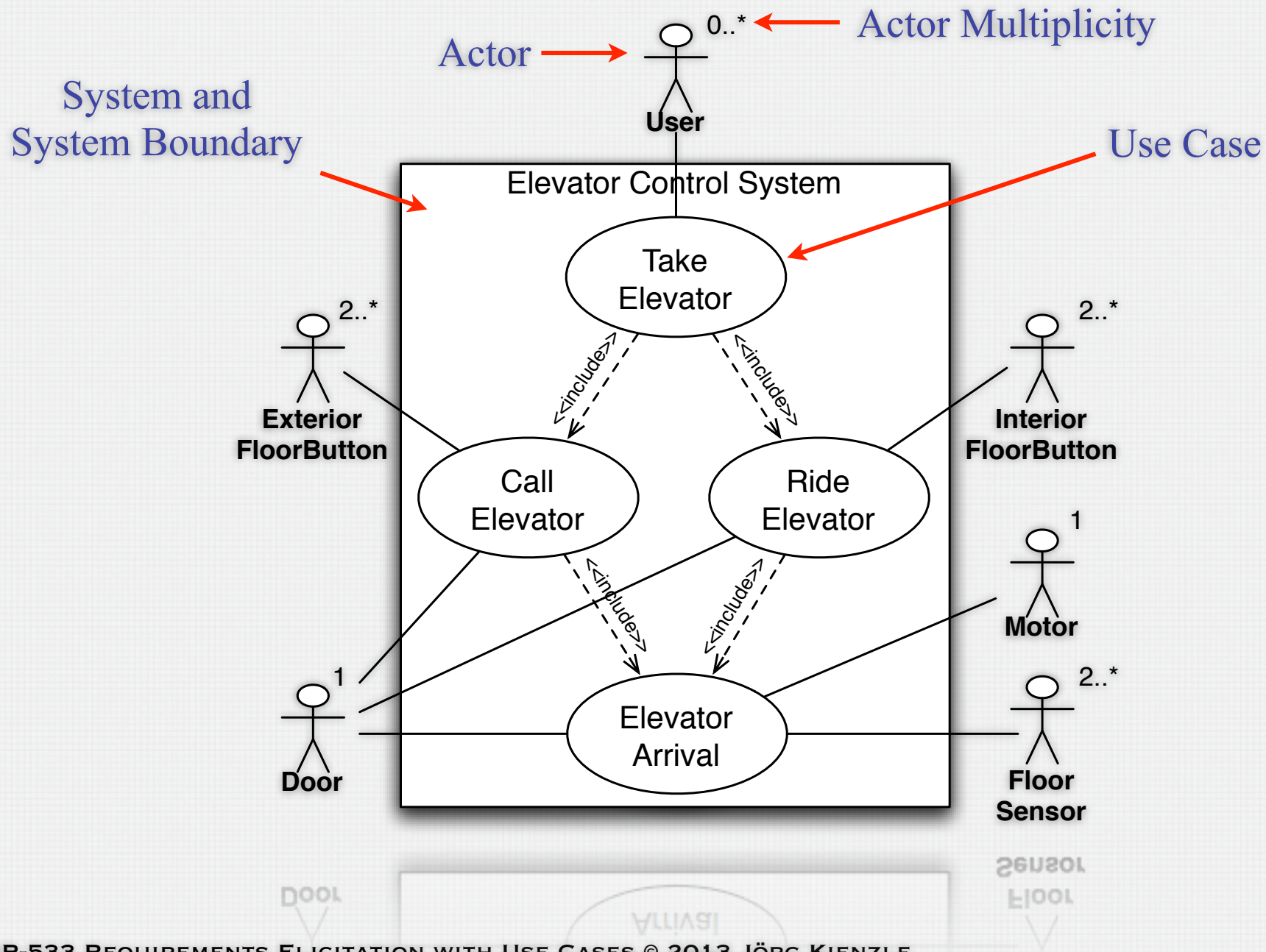
Subfunction Level Use Case Example (2)

Extensions:
(1-5)a. *Client* informs *System* that he wants to cancel the identification process.
(1-5)a.1. *System* requests *Card Reader* to eject card; use case ends in failure.
2a. *System* ascertains that card type is unknown:
2a.1. *System* informs *Client* and requests the *Card Reader* to eject the card; the use case ends in failure.
3a. *System* times out on waiting for *Client* to provide PIN:
3a.1. *System* requests *Card Reader* to eject card; use case ends in failure.
5a. *Bank Server* informs *System* that password is incorrect:
5a.1a. *System* informs *Client* and prompts him/her to retry; use case continues at step 3.
5a.1b. *System* ascertains that *Client* entered an incorrect PIN for the third time:
5a.1b.1. *System* instructs *Card Reader* to swallow the card.
5a.1b.2. *System* informs *Client* to contact bank for further details; use case ends in failure.
5b. *System* is unable to communicate with *Bank Server*:
5b.1. *System* requests *Card Reader* to eject card.
5b.2. *System* informs *Client* that it is now out of service; use case ends in failure.

Use Cases in UML

- UML provides a graphical representation for use cases called the *use case diagram*
- It allows one to graphically depict:
  - actors,
  - use cases,
  - associations,
  - dependencies,
  - generalizations,
  - packages,
  - and the system boundary.

Example Use Case Diagram



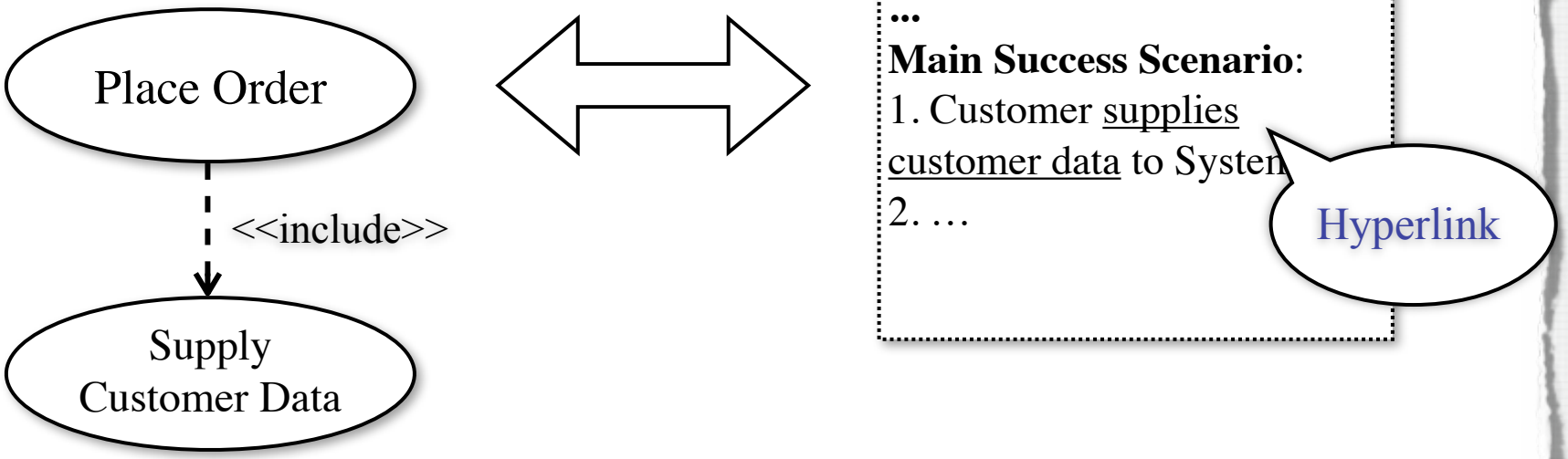
Use Case Model

- A Use Case Model consists of:
  - (At least one) use case diagram and
  - Use case textual template for each “ellipse”

Use Case: Open Account
Scope: Bank System
Level: User Goal
Intention in Context: The intention of the Client is to...

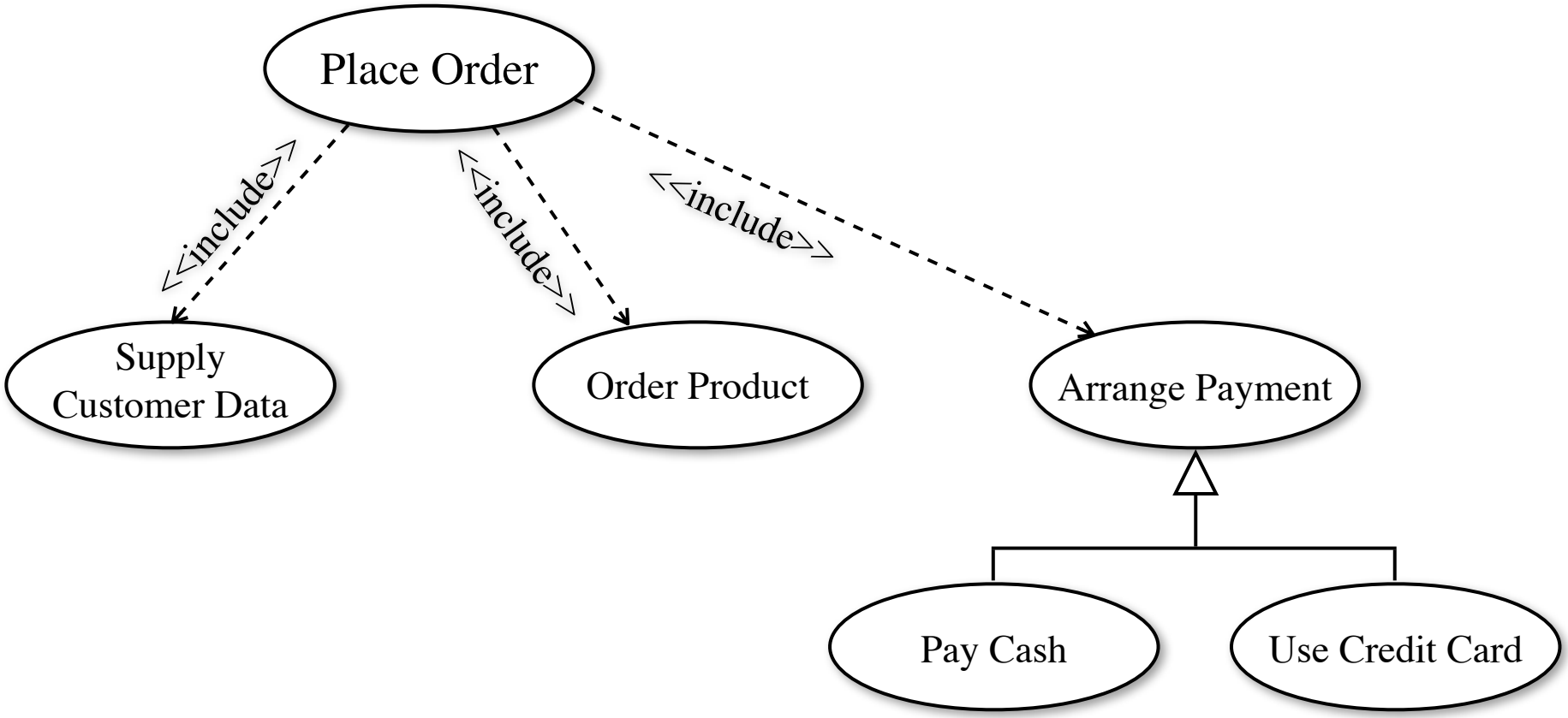
<<include>> Relationship

- An <<include>> relationship means that the base use case *explicitly* incorporates the behaviour of another use case *at a location specified in the base*.



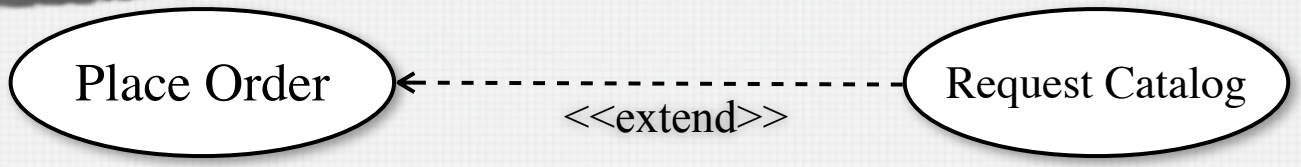
Generalization/ Specialization

- Explicitly incorporating a “super use case” results in incorporating any child use case



<<extend>> Relationship

- An <<extend>> relationship means that the extending use case *adds new interaction steps to the base* use case *at locations specified in the extending use case*.



Use Case: Place Order
...
Main Success Scenario:
1. ...
Step 2 can be repeated.
2. Customer orders a product.
3. Customer arranges payment.

Use Case: Request Catalog
extends Place Order
...
Main Success Scenario:
2.1 Customer informs System that he would like to receive a catalog with the order.

Requirements Elicitation Process (1)

- Task 1: Actors
  - Brainstorm actors and primary actor goals taking into account the questions for identifying Primary and Secondary actors, and Initiators, and Facilitators.
  - To decide on how much technical details should be considered when defining actors ask yourself:
    - Is this actor / facilitator part of the requirements? If yes, then it should be included.
    - Is this actor / facilitator a possible solution of interaction with the system? If yes, then it should be omitted.
  - Make a list with each primary actor and its goals.

Actor Descriptions

Actor	Role	Brief Description
Client	Primary	A customer of the bank that will use the system to perform transactions and queries on his/her accounts.
Bank	Primary	...
Printer	Secondary	...
Teller	Facilitator	...

Actor	Goals
Client	Open Account Deposit Money Withdraw Money Transfer Money Close Account
Bank Manager	...

Requirements Elicitation Process (2)

- Task 2: Use Case Outlines
  - Construct (summary and/or user-goal) use cases briefs for each actor goal on the system, making the actor the primary one.
  - Always ask “why” in order to find the next level up!
  - Questions:
    - What measurable value/service is needed by the actor?
    - What are the actors intentions?
    - Why do the actors do what they do?

Prioritized Use Case List

- Prioritize use cases to:
  - Determine on which functionality to work on first, especially when doing iterative development / prototyping
  - Support decision making when tradeoffs have to be made

Actor	Goal	Description	Business Need	Priority	#
Client	Open a savings account	…	Medium	3	1
	Open a high transaction account	…	Top	1	2
	Deposit money on an accout	…	Top	2	3

Requirements Elicitation Process (3)

- Task 3: Use Case Bodies
  - Capture each actor's intent and responsibility— from trigger to goal delivery.
  - For each use case, fill in the main success scenario before the extensions.
  - Extensions
    - Think of alternate ways of achieving the user goal.
    - Think of exceptional situations
      - What situation prevents the successful completion of a user goal?
      - What situations take priority over the user goal?
      - What would happen if this step fails / is omitted?
    - Make sure that you clearly state if at the end of an extension the use case fails or continues (at step x).

Requirements Elicitation Process (4)

- Task 4: Use Case Structuring
  - For each use case:
    - If the main success scenario of the use case is greater than 9 steps, collect steps that encapsulate a sub-goal of the primary actor and create a new lower-level use case with the steps.
    - Inversely, if the use case is smaller than 3 steps, think about expanding it or putting it back in the calling use case.
    - The most important thing is that the steps of the use case have a consistent level of description, no matter what the level!
    - Iteration/refactoring is the key: use cases are always better next time around.

Requirements Elicitation Process (5)

- Task 5: Checks
  - Review each use case:
    - Does it achieve a single, discrete, complete, meaningful, and well-defined task of interest to an actor?
    - Is it written in a clear and precise way?
    - Is it written using the vocabulary of the application domain (see domain model) and abstracts away from technology and solution?
    - Is it complete, correct, consistent, verifiable?
  - For each section of the template:
    - **Title**: Is the title an active-verb goal phrase, that expresses the goal of the primary actor? Can the system deliver that goal?
    - **Scope**: Is the system boundary clear?
    - **Context**: Is the use case's purpose and intent clear?
    - **Multiplicity**: Has it been clearly stated how many instances of this use case the system needs to handle concurrently?

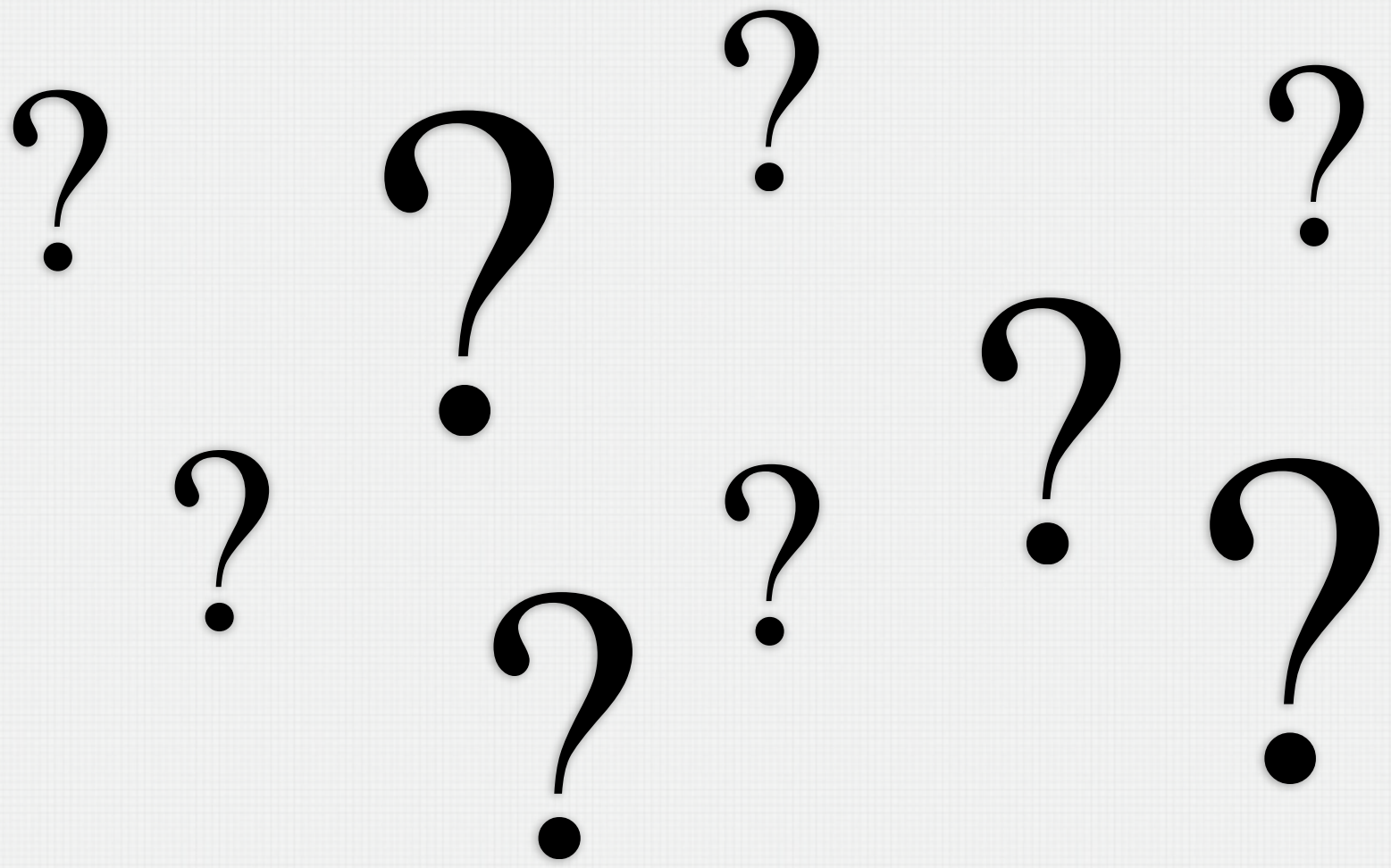
Requirements Elicitation Process (6)

- Task 5 continued:
  - **Main Success Scenario**: Does it run from trigger to success? Is it written using the vocabulary of the application domain (see domain model) and abstracts away from technology and solution?
  - Does it achieve a single, discrete, complete, meaningful, and well-defined task of interest to an actor?
  - Does it have the right amount of steps (no more than 10)?
  - **Extensions**: Is this extension necessary? Can the system detect the condition under which the extension becomes relevant?
  - **For each step**:
    - Is it either an input interaction, an output interaction, or a call to a subfunction level use case?
    - Make sure that input or output interaction steps contain the word "System"!
    - Does the process of achieving the primary actor's goal move distinctively forward after successful completion of this step?

Requirements Elicitation Process (7)

- Obtain feedback from client, stakeholders and users
- Ask:
  - Is this what you want?
  - Will you be able to tell, upon delivery, whether you got this?

Questions?



Books on Use Cases

- F. Armour, and G Miller; Advanced Use Case Modeling. Object Technology Series, Addison-Wesley 2001.
- A. Cockburn; Writing Effective Use Cases. Addison-Wesley 2000.
- D. Kulak and E. Guiney; Use Cases: Requirements in Context. ACM Press, Addison-Wesley, 2000.
- D. Lefffingwell, and D. Widrig; Managing Software Requirements: A Unified Approach. Object Technology Series, Addison-Wesley 2000.
- S. Robertson, and J. Robertson; Mastering the Requirements Process. Addison-Wesley 2000.

References

- A. Cockburn; Structuring Use Cases with Goals. Journal of Object-Oriented Programming (JOOP Magazine), Sept-Oct and Nov-Dec, 1997.
- E. Ecklund, L. Delcambre and M. Freiling; Change cases: use cases that identify future requirements. OOPSLA '96 - Proceedings of the eleventh annual conference on Object-oriented programming systems, languages, and applications, 1996. pp. 342 - 358.
- M. Fowler; Use and Abuse Cases. Distributed Computing Magazine, 1999. Available at http://www.martinfowler.com/articles.html
- M. Glinz; Problems and Deficiencies of UML as a Requirements Specification Language. Proceedings of the Tenth International Workshop on Software Specification and Design, San Diego, 2000, pp. 11-22.
- T. Korson; The Misuse of Use Cases. Object Magazine, May 1998.
- R. Malan and D. Bredemeyer; Functional Requirements and Use Cases. June 1999. Available at http://www.bredemeyer.com/papers.htm
- J. Mylopoulos, L. Chung and B. Nixon; Representing and Using Nonfunctional Requirements: A Process-Oriented Approach. IEEE Transactions on Software Engineering, Vol. 23, No. 3/4, 1998, pp. 127-155.
- A. Pols; Use Case Rules of Thumb: Guidelines and lessons learned. Fusion Newsletter, Feb. 1997.
- S. Sendall and A. Strohmeier; From Use Cases to System Operation Specifications. UML 2000 - The Unified Modeling Language: Advancing the Standard, Third International Conference, York, UK, October 2-6, 2000, S. Kent, A. Evans and B.Selic (Ed.), LNCS (Lecture Notes in Computer Science), no. 1939, 2000, pp. 1-15.
- R. Wirfs-Brock; The Art of Designing Meaningful Conversations. Smalltalk Report, February, 1994.

Use Case Exercises

- Elevator Control System
- Auction System
- Drink Vending Machine

Elevator Control System (1)

- Develop an elevator control system that is aligned with the following descriptions:
  - There is only one elevator cabin, which travels between the floors.
  - There is a single button on each floor to call the lift.
  - Inside the elevator cabin, there is a series of buttons, one for each floor.
  - Requests are definitive, i.e., they cannot be cancelled, and they persist; thus they should eventually be serviced.
  - The arrival of the cabin at a floor is detected by a sensor.

Elevator Control System (2)

- The system may ask the elevator to go up, go down or stop. In this example, we assume that the elevator's braking distance is negligible.
- The system may ask the elevator to open its door. The system will receive a notification when the door is closed. This simulates the activity of letting people on and off at each floor.
- The door closes automatically after a predefined amount of time. However, neither this function of the elevator nor the protection associated with the door closing (stopping it from squashing people) are part of the system to realize.

Elevator Control System Questions

1. Given the description of the Elevator Control system, propose some fixes to version 1 of the "Take Lift" user-goal level use case.
2. The problems that were found with the first version have been corrected in version 2. Is this now an effective use case?
   If not, explain where this version could be improved.

TakeLift Use Case

Use Case: TakeLift

Scope: Single-Cabin Elevator Control System

Level: User Goal

Intention in Context: The User intends to go from one floor to another.

Multiplicity: Several Users can take the lift simultaneously. A given User can only take one lift at a time.

Primary Actor: User

TakeLift Use Case - Version 1

Main Success Scenario:

1. The user requests a floor.
2. The lift goes to pick-up the user.
3. The lift pick-ups the user.
4. The user asks to go to destination floor.
5. The lift drops off the user.

Extensions:

(2-3)IIa. The user leaves and takes the stairs.

3a. The lift never reaches the source floor or the door does not open:

3a.1 The user unhappily takes the stairs; the use case ends.

4a. The user enters the lift but does not make a request: the use case ends.

5a. The user changes his/her mind on the destination by requesting another destination.

5a.1 The lift goes to the other destination as well.

6a. The lift does not drop off the user at the destination floor, either because the door doesn't open or the lift never reaches the destination floor.

6a.1 The user, by mobile phone, sues the company who made the lift system (not automated); the use case ends.

TakeLift Use Case - Version 2 (1)

Main Success Scenario:

1. The *User* presses the call button, requesting the *System* for the lift from a particular floor.
2. The *System* switches the floor button light on; the *System* commands the lift to go to the floor where the *User* called the lift to do a pick-up; and the *System* places the request on the pending FIFO queue.
3. The *Floor Sensor* informs the *System* that the lift has reached the *User's* calling floor; the *System* commands the lift to stop and open its *Door*; the *System* deletes the external request of the *User* from the pending FIFO request queue; and the *System* turns the corresponding floor light off.
4. The *User* enters the lift.

(cont'd)

TakeLift Use Case - Version 2 (2)

Main Success Scenario (cont'd)

Steps 5 and 6 can happen in parallel (also implying any order)

5. The *System* times-out on the door opening time and the *System* commands the *Door* to close.
6. From inside the cabin, the *User* presses a floor button, requesting the *System* to go to a particular (destination) floor.
7. The *System* turns on the light button; the *System* commands the lift to go to the destination floor; and the *System* places the internal request on the pending FIFO queue.
8. The *Floor Sensor* informs the *System* that the lift has reached the User's destination floor; the *System* commands the lift to stop and open its *Door*; the *System* deletes the internal request of the *User* from the pending FIFO request queue; and the *System* turns the corresponding light off.
9. The *User* exits the lift.

TakeLift Use Case - Version 2 (3)

Extensions:

2a. The *System* ascertains that the lift is already at the source floor with the door open:

2a.1. The *System* requests the *Door* to stay open; and the use case continues at step 4.

2b. The *System* ascertains that the lift is already at the source floor with the door closed and it is currently idle (not servicing another request):

2b.1. The *System* commands the lift to open its *Door*; the use case continues at step 4.

2c. The *System* ascertains that the lift is currently busy:

2c.1. The *System* turns the floor button light on and the *System* places the external request on the pending FIFO queue.

2c.2a. The *System* ascertains that the lift has arrived at the floor of the requesting *User*:

2c.2a.1. The *System* commands the lift to stop and open its *Door*, and the *System* deletes the external request of the *User*; the use case continues at step 4.

2c.2b. The *System* ascertains that the lift has become available to service the request:

2c.2b.1 The *System* commands the lift to go to the destination floor; the use case continues at step 3.

(cont'd)

TakeLift Use Case - Version 2 (4)

Extensions (cont'd):

5a. The *System* commands the lift to close its *Door*, but it determines that something is obstructing the *Door* from closing:

5a.1. The *System* commands the lift to reopen its *Door* to secure the safety of its users.

6a. The *User* does not make a request: the use case ends.

6b. The *User* presses numerous buttons for different floors:

6b.1. The *System* turns on the light for each button; the *System* places all the internal requests on the pending FIFO queue; and the *System* commands the lift to go to the closest floor of the ones that was requested; the use case continues at step 8.

6c. From inside the cabin, the *User* presses a button for a (destination) floor, but the lift *Door* is still open:

6c.1. The *System* turns the light on and the *System* places the internal request on the pending FIFO queue.

6c.2. The *System* determines that the *Door* should be closed and it commands the *Door* to close.

6c.3. The *System* commands the lift to go to the requested floor; the use case continues at step 8.

(cont'd)

TakeLift Use Case - Version 2 (5)

Extensions (cont'd):

7ll. The *User* presses another button to go to a floor:

7ll.1. The *System* turns the light on and the *System* places the internal request on the pending FIFO queue; the use case continues at step 8.

7a. The *System* determines that the lift is already at the destination floor:

7a.1. The *System* commands the lift to open its *Door*; the use case continues at step 9.

7b. The *System* acknowledges the internal request, but it continues to service the current request, storing this request.

7b.1. The use case continues at step 8.

9a. The *System* ascertains that the *User* has other requests pending:

9a.1. The *System* commands the lift to go to the next most suitable floor requested.

Auction System (1)

Your team has been given the responsibility to develop an online auction system that allows people to negotiate over the buying and selling of goods in the form of English-style auctions (over the Internet). The company owners want to rival the Internet auctioning sites, such as, eBay (www.ebay.com), and uBid (www.ubid.com). The innovation with this system is that it guarantees that all bids are solvent.

All potential users of the system must first enroll with the system; once enrolled they have to log on to the system for each session. Then, they are able to sell, buy, or browse the auctions available on the system. Customers have credit with the system that is used as security on each and every bid. Customers can increase their credit by asking the system to debit a certain amount from their credit card.

Auction System (2)

A customer that wishes to sell initiates an auction by informing the system of the goods to auction, together with a minimum bid price and reserve price for the goods, the start period of the auction, and the duration of the auction, e.g., 30 days. The seller has the right to cancel the auction as long as the auction's start date has not been passed, i.e., the auction has not already started.

Customers that wish to follow an auction must first join the auction. Note that it is only possible to join an active auction. Once a customer has joined the auction, he/she may make a bid, or post a message on the auction's bulletin board (visible to the seller and all customers who are currently participants in the auction). A bid is valid if it is over the minimum bid increment (calculated purely on the amount of the previous high bid), and if the bidder has sufficient funds, i.e. the customer's credit with the system is at least as high as the sum of all pending bids.

Auction System (3)

Bidders are allowed to place their bids until the auction closes, and place bids across as many auctions as they please. Once an auction closes, the system calculates whether the highest bid meets the reserve price given by the seller (English-style auction reserve price), and if so, the system deposits the highest bid price minus the commission taken for the auction service into the credit of the seller (credit internal with the system).

The auction system is highly concurrent — clients bidding against each other in parallel, and a client placing bids in different auctions and increasing his/her credit in parallel.

Auction System Questions

1. Given the description of the auction system, identify actors, stakeholders and their principal interests.
2. Propose a summary goal level use case for “buy and sell goods by auction”. Make use, when appropriate, of the following sub-”use cases” (e.g. user-level or subfunction level use cases):
   - enrol with system
   - buy item on auction
   - sell item by auction
   - transfer credit
   - search for auction item
   - close auction
   - identify user
3. Draw the UML use case diagram that contains all the use cases mentioned in question 2.
4. Write the complete use case “buy item under auction”.

Drink Vending Machine (1)

A drink vending machine is a simple but non-trivial kind of reactive system. The physical machine consists of several components: drink shelves, a sensor that detects when a drink has been delivered, drink selector buttons, lights indicating drink availability, a display, a coin slot, a cancel button, and a money box that stores the coins inside the machine. Inside the machine, there is also a small terminal for use by the service person. Human interaction takes place between a customer or service person and the physical components when, for example, the customer selects a drink, or when the service person replenishes a shelf or changes the price of a drink. Finally, software coordinates the physical components, receiving messages from them and sending commands to them.

Drink Vending Machine (2)

During an informal talk with the vending company it has been determined that the interaction between a customer and the machine should be as follows: the customer first selects a drink, then inserts coins, and when the inserted amount reaches or exceeds the price for the drink, the drink is provided together with the change.

Also, due to the constraints of the hardware, some decisions had to be made. They are summarized below:

Drink Vending Machine (3)

The money box is actually composed of three different collectors: the first one keeps coins until the sale is complete or cancelled, the second one receives the coins from the first one once the drink has been distributed, and the third one contains coins used for change. The money box accepts 25 cents, 1 dollar and 2 dollar coins. It is assumed that the first collector is big enough to hold as many coins as needed to buy the most expensive drink and more. The second collector, although very big, has a limited capacity and must be emptied by the service person now and then. If the moneybox is full, the vending machine goes out of service. The third collector contains a fixed amount of 25 cent coins used for change. If there is no change left and a client does not insert the right amount, “no change” is displayed and the money is returned to the client.

Drink Vending Machine (4)

Drinks are stored on shelves. Each shelf is associated with a beverage kind and a price. Prices are all multiples of 25 cents. The number of shelves of the machine is fixed, and the capacity of each shelf is fixed as well. Initially, there are no drinks in the shelves.

If for some reason a selected drink can not be delivered, the sale is cancelled.

Only authorized personnel is allowed to service the machine.

Drink Vending Machine Questions

1. Write the complete use case *BuyDrink*.
2. Write the complete use case *ServiceMachine*.
3. Propose a use case diagram that includes the two main use cases of the system: *BuyDrink* and *ServiceMachine*.
   (You do not have to add more use cases...)