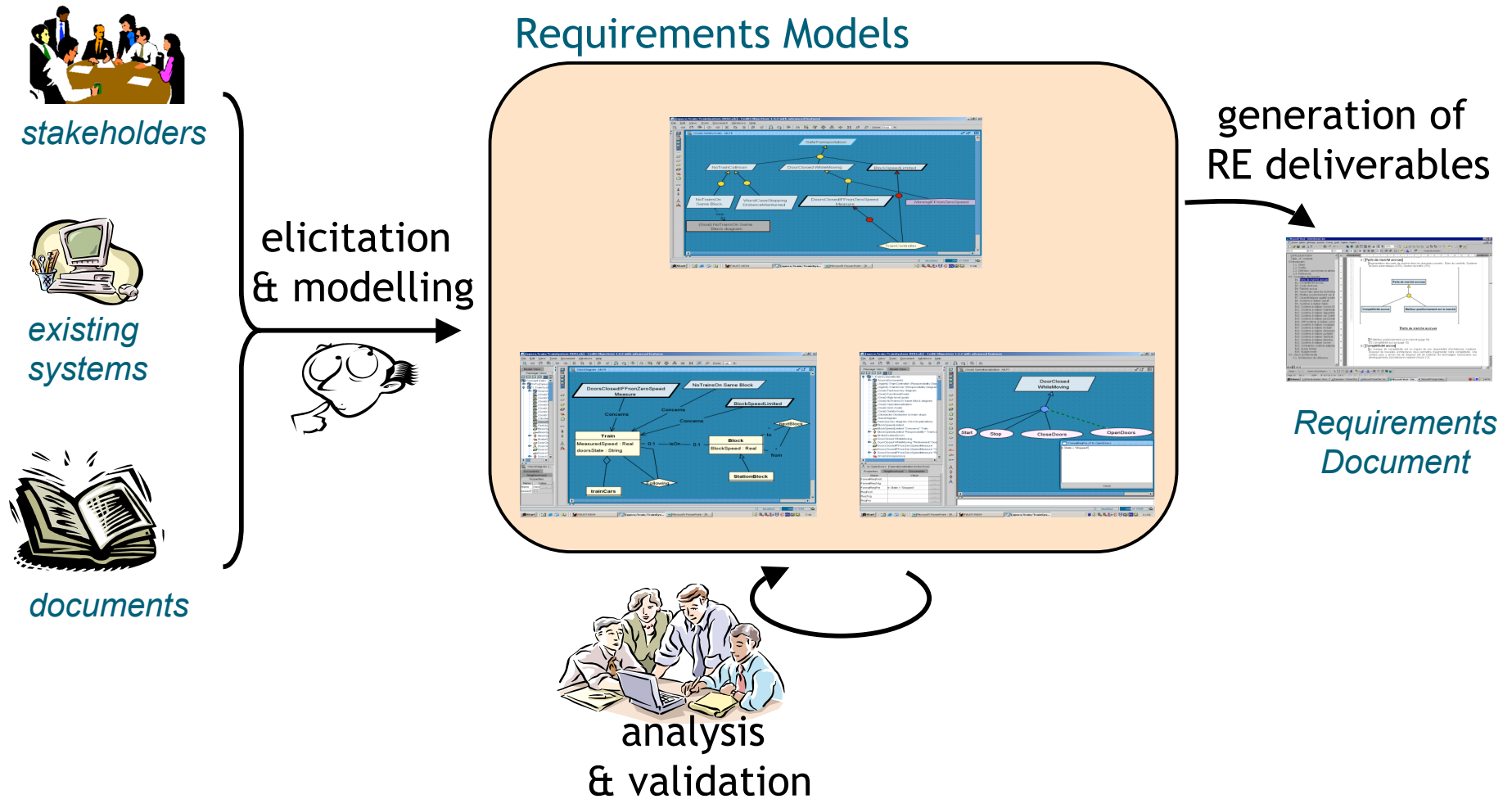


UML for requirements modelling



Giordano Tamburrelli
tamburrelli@elet.polimi.it

UML for requirements modelling



Using UML for requirement modeling

Use cases

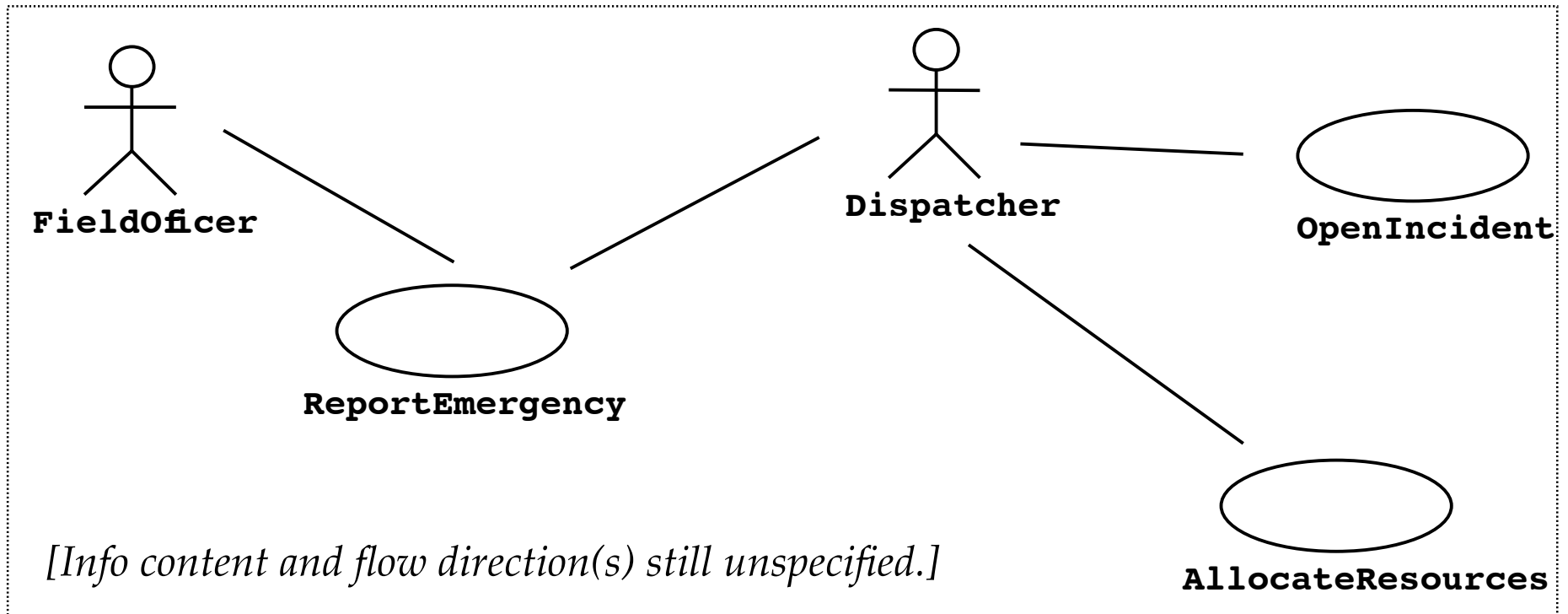
Use Cases

- A use case is a flow of events in the system, including interaction with actors
- It is initiated by an actor
- Each use case has a name
- Each use case has a termination condition
- Graphical Notation: An oval with the name of the use case



***Use Case Model:* The set of all use cases specifying the complete functionality of the system**

Example: Use Case Model for Incident Management

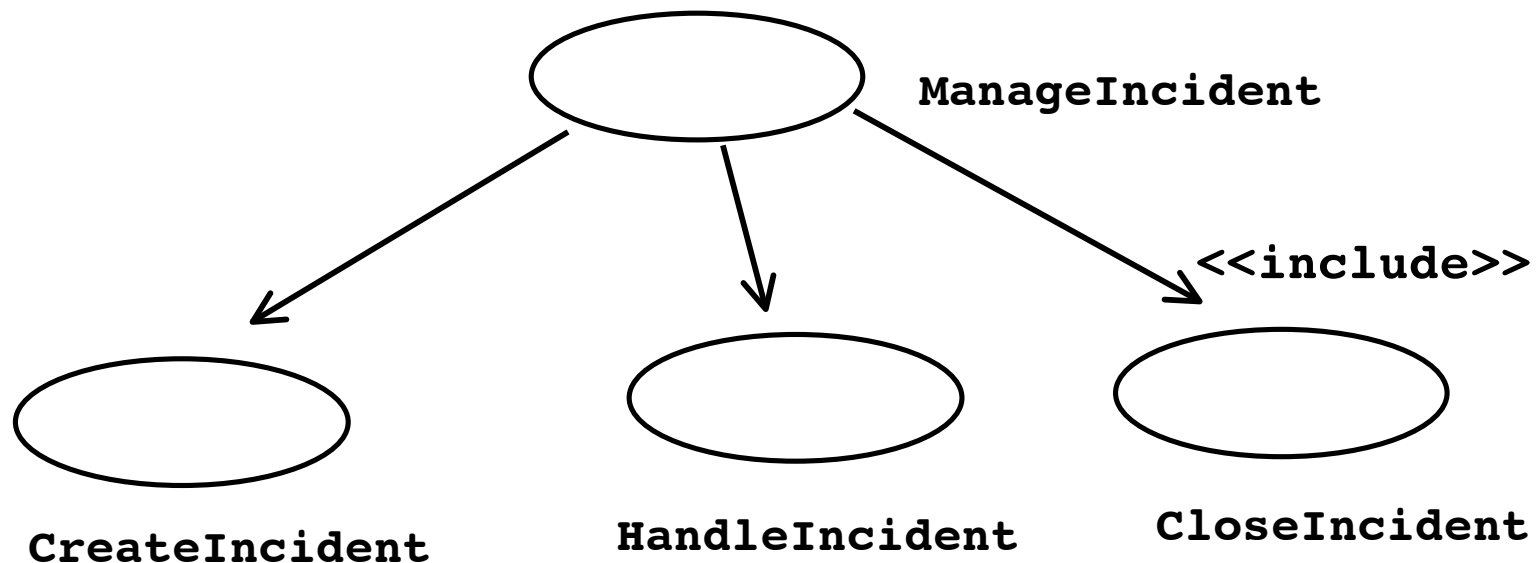


Use Case Associations

- A use case model consists of use cases and use case associations
 - A use case association is a relationship between use cases
- Important types of use case associations: Include, Extends, Generalization
- Include
 - A use case uses another use case (“functional decomposition”)
- Extends
 - A use case extends another use case
- Generalization
 - An abstract use case has several different specializations

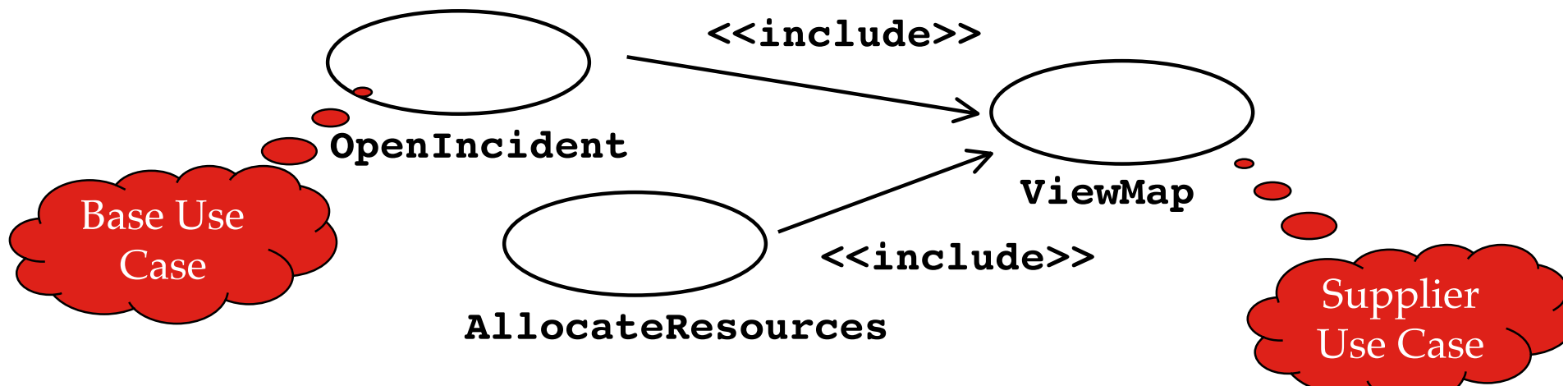
<<Include>>: Functional Decomposition

- Problem:
 - A function in the original problem statement is too complex to be solvable immediately
- Solution:
 - Describe the function as the aggregation of a set of simpler functions. The associated use case is decomposed into smaller use cases



<<Include>>: Reuse of Existing Functionality

- Problem:
 - There are already existing functions. How can we *reuse* them?
- Solution:
 - The include association from a use case A to a use case B indicates that an instance of the use case A performs all the behavior described in the use case B (“A delegates to B”)
- Example:
 - The use case “ViewMap” describes behavior that can be used by the use case “OpenIncident” (“ViewMap” is factored out)

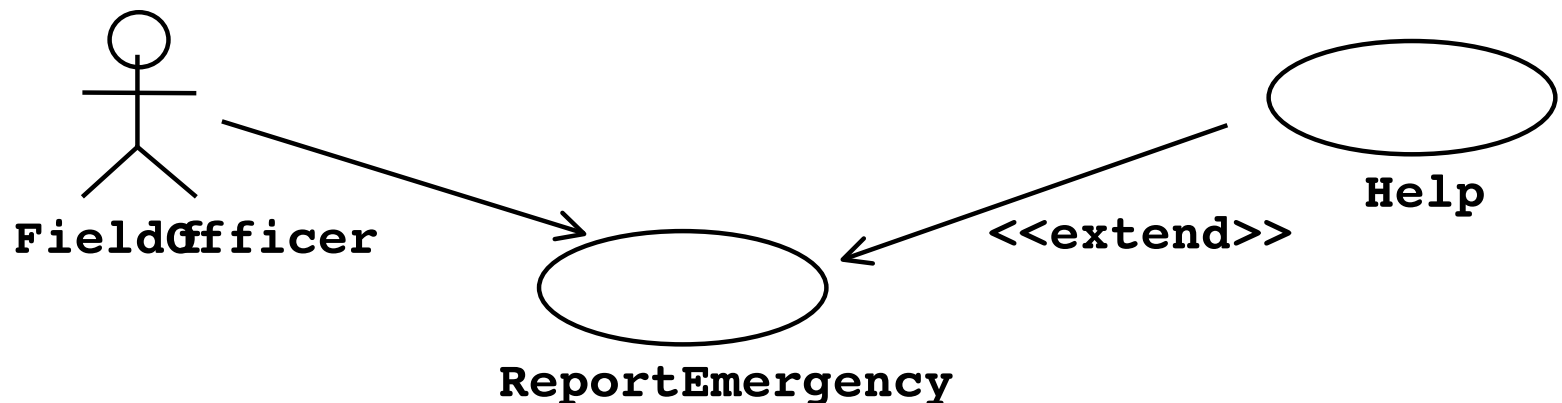


Note: The base case cannot exist alone. It is always called with the supplier use case. [*Base class is a client of the supplier as server*]

SE II 2008/2009

<Extend> Association for Use Cases

- Problem:
 - The functionality in the original problem statement needs to be extended.
- Solution:
 - An *extend association* from a use case A to a use case B indicates that use case A is an extension of use case B.
- Example:
 - The **[base]** use case “ReportEmergency” is complete by itself , but can be extended by the use case “Help” for a specific scenario in which the user requires help



Note: The base use case can be executed without the use case extension in extend associations.

SE II 2008/2009

Generalization association in use cases

- **Problem:**

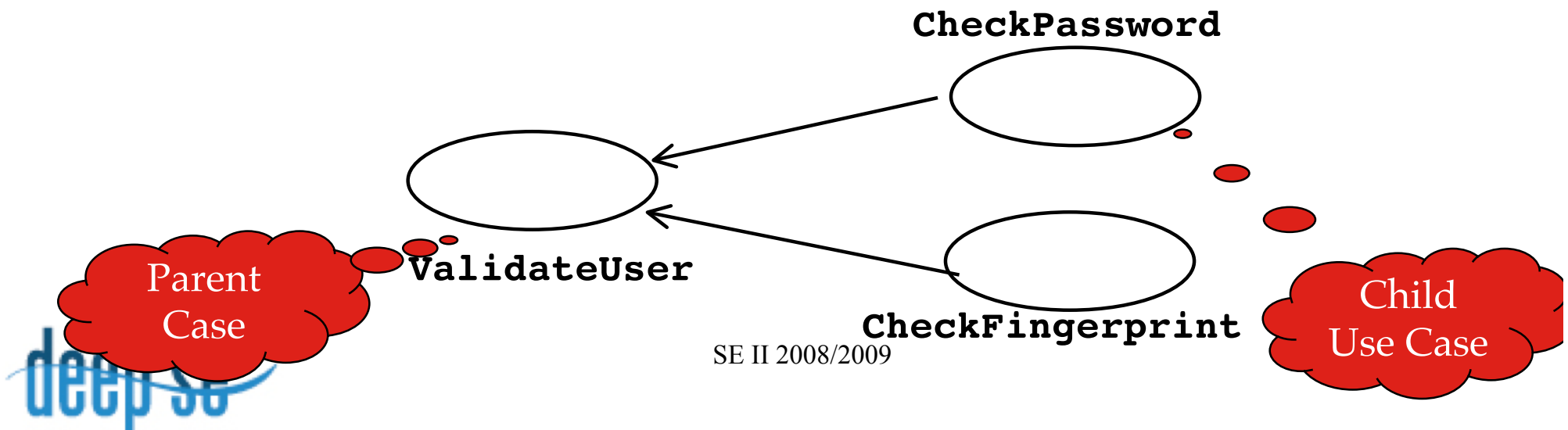
- You have common behavior among use cases and want to factor this out.

- **Solution:**

- The generalization association among use cases factors out common behavior. The child use cases inherit the behavior and meaning of the parent use case and add or override some behavior.

- **Example:**

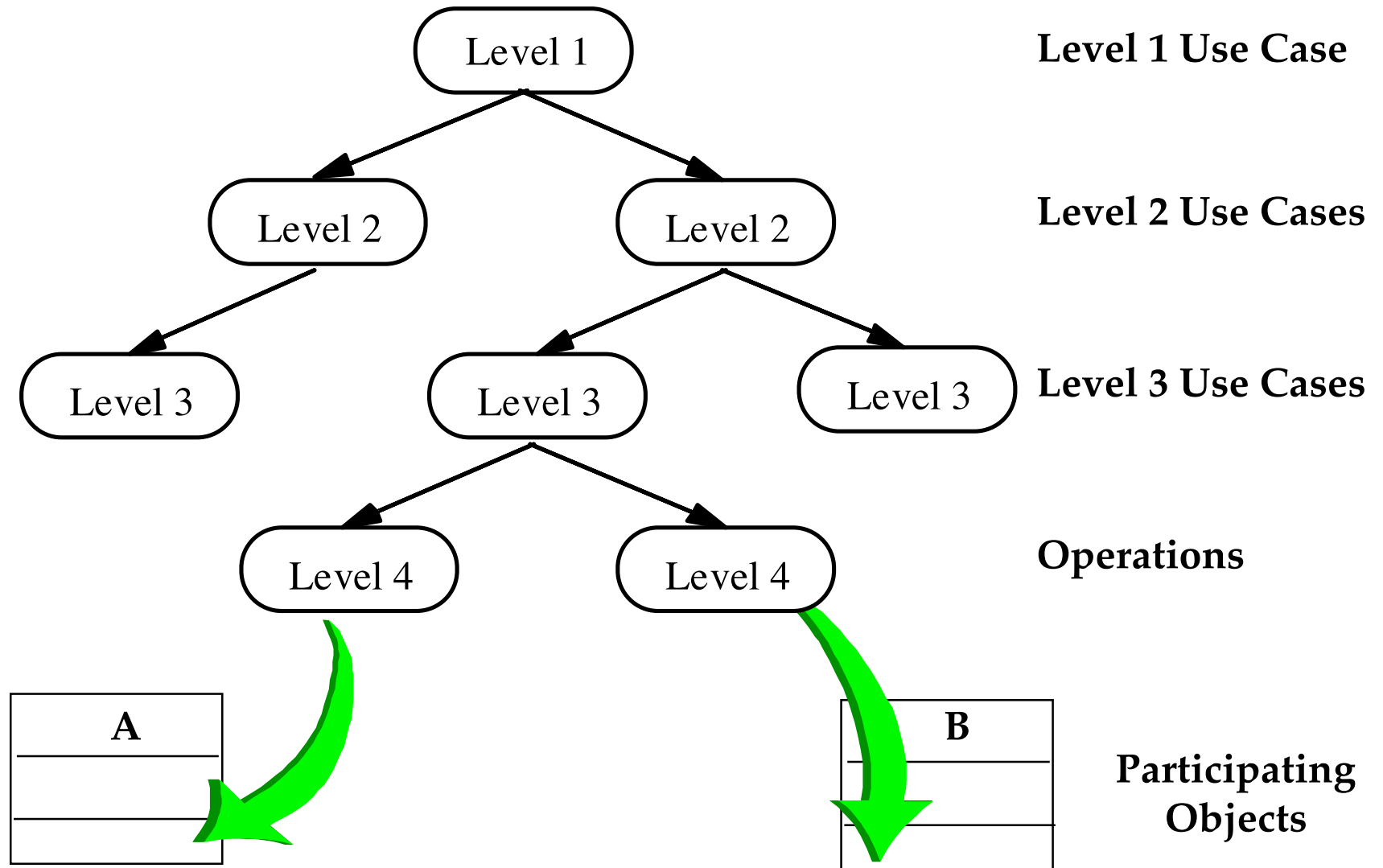
- Consider the use case “ValidateUser”, responsible for verifying the identity of the user. The customer might require **[one of]** two realizations: “CheckPassword” or “CheckFingerprint” **[depending on the context]**.



Using UML in requirement modelling

Object modeling

From Use Cases to Objects



Requirements-level class diagrams

- are conceptual models for the application domain
 - different from OO software design models
- may model objects that will not be represented in the software-to-be
 - because this is not known at the start of the requirements modelling effort
- do not attach operations (methods) to objects
 - it's best to postpone this kind of decisions until software design

Association Multiplicities

- **multiplicity** : the minimum and maximum times an object can be *simultaneously* involved in the association



at any given time, an ambulance may be mobilized to **at most one** incident

at any given time, an incident may have **zero or more** ambulances mobilized to it

- Note:
 - multiplicities do not distinguish between domain properties and goals
 - most assertions are not expressible by multiplicities

Textual Specification

Entity Incident

Def. *An incident is any situation that may require an urgent intervention from the ambulance service.*

Has

location: Location

{the actual incident location}

time: Time

{the time at which the incident occurred}

pending: Bool

{an incident is pending if it has occurred and no ambulance has arrived at the incident scene yet}

Domain Invariants

There is at most one pending incident per location, i.e. two incidents are considered to be the same if they have the same location and are both pending.

Every class must have a natural language definition that precisely define conditions for an object to be among the class' instances in current state

Every attribute must have a natural language definition that define the real-world meaning of the attribute

Domain invariants specify **domain properties** for the class

How do you find classes?

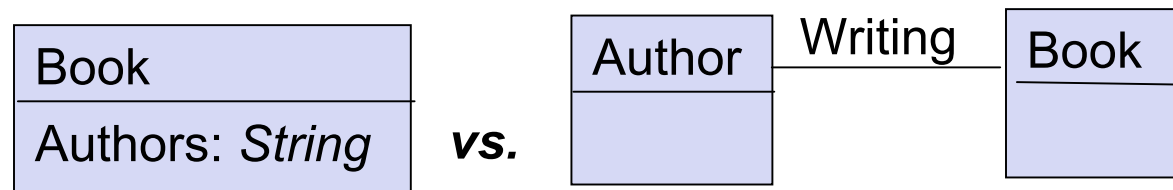
- Finding objects is the central piece in object modeling
 - Learn about problem domain: Observe your client
 - Apply general world knowledge and intuition
 - Take the flow of events and find participating objects in use cases
 - Try to establish a taxonomy
 - Do a syntactic analysis of *problem statement*, *scenario* or *flow of events*
 - Abbott Textual Analysis, 1983, also called noun-verb analysis
 - Nouns are good candidates for classes
 - Verbs are good candidates for operations
 - Apply design knowledge:
 - Distinguish different types of objects
 - Apply design pattern

Finding Participating Objects in Use Cases

- Pick a **use case** and look at its **flow of events**
 - Find terms that developers or users need to clarify in order to understand the flow of events
 - Look for recurring nouns (e.g., Incident),
 - Identify real world entities that the system needs to keep track of (e.g., FieldOfficer, Dispatcher, Resource),
 - Identify real world procedures that the system needs to keep track of (e.g., EmergencyOperationsPlan),
 - Identify data sources or sinks (e.g., Printer)
 - Identify interface artifacts (e.g., PoliceStation) [**e.g., telecomm link?**]
- Be prepared that some objects are still missing and need to be found:
 - Model the flow of events with a sequence diagram
- Always use the user's terms

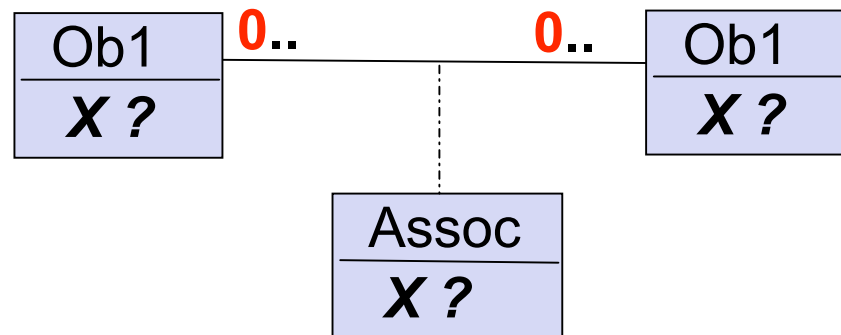
Modeling objects: tips & heuristics (1)

- For X a structural element appearing in use case formulations...
 - Make X an attribute when...
 - instances of X are non-distinguishable
 - X is a function (yielding one single value when applied to some conceptual instance e.g. age)
 - you do not want to attach attributes/associations to X

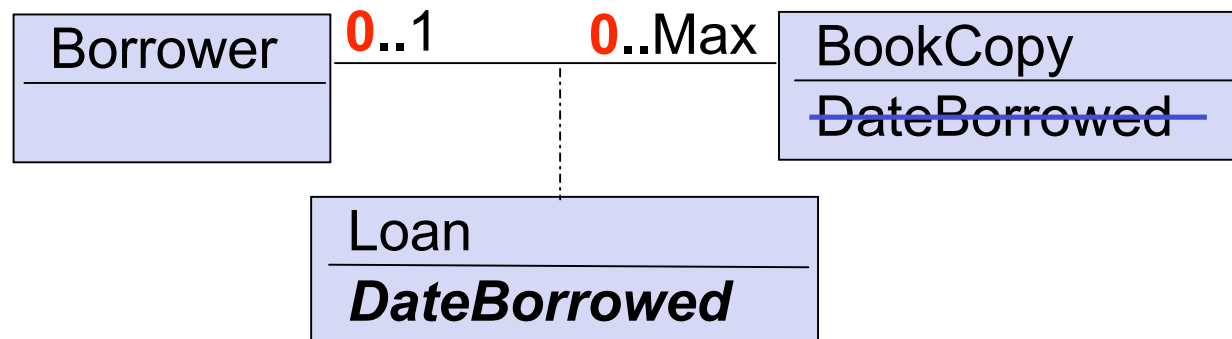


Modeling objects: tips & heuristics (2)

- Should I attach an attribute *X* to an *object* in association or to the *association*?

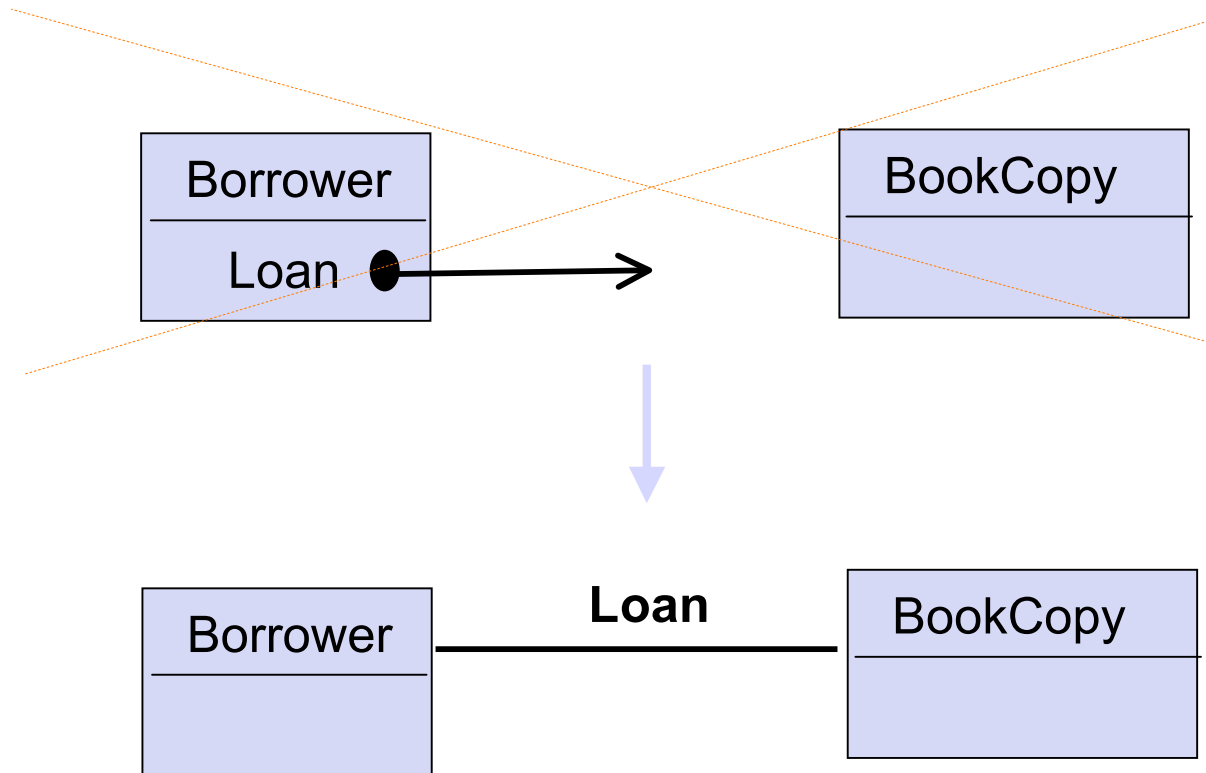


To the association if object instance can be unassociated to avoid losing info
e.g. who is borrower at date d



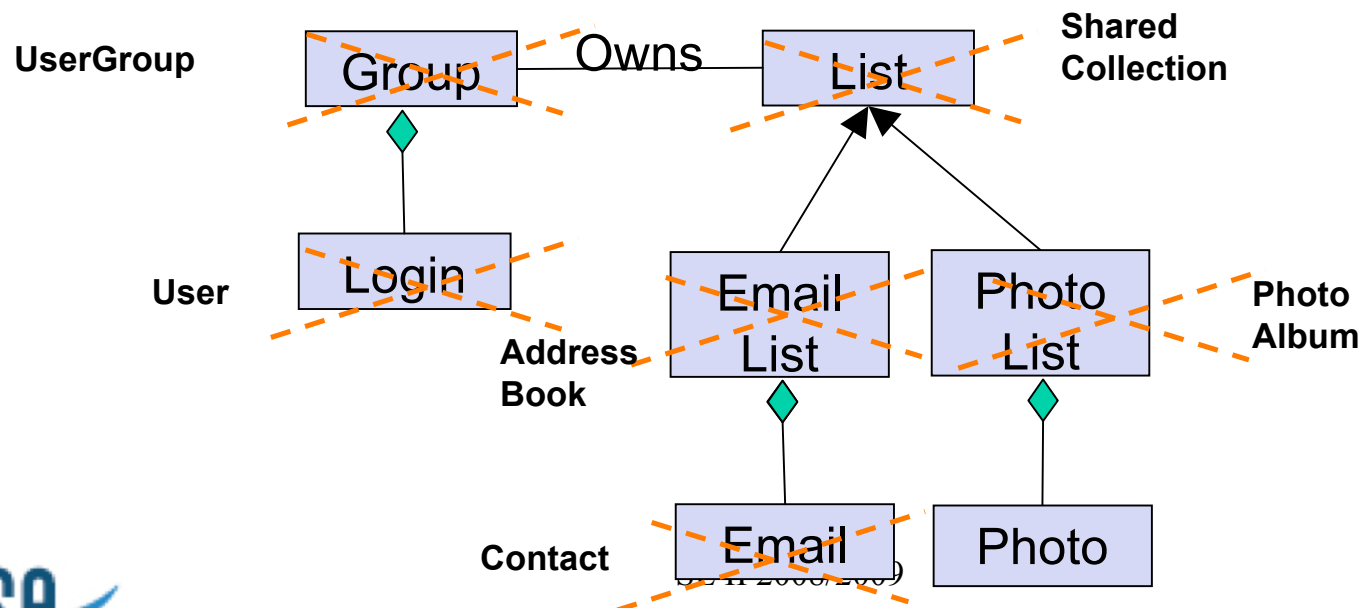
Modeling objects: tips & heuristics (3)

- Avoid frequent flaws in conceptual modeling ...
 - object attribute as "pointer" to another object



Modeling objects: tips & heuristics (7)

- Naming objects and attributes
 - important for understandability
 - chose usual words in application domain, avoid CS jargon
 - avoid unspecific names (List, Form, Thing, ...)
- Model the application domain, not the software
 - avoid implementation concepts



Object modeling: an Example

Flow of events:

- The customer enters the store to buy a toy.
- It has to be a toy that his daughter likes and it must cost less than 50 Euro.
- He tries a videogame, which uses a data glove and a head mounted display. He likes it.
- An assistant helps him.
- The suitability of the game depends on the age of the child
- His daughter is only 3 years old.
- The assistant recommends another type of toy, namely the boardgame "Monopoly". *[At age 3?]**

Is this a good use Case?

Not quite!

* "Monopoly" is probably a left over from the scenario

The use case should terminate with the customer leaving the store

Textual Analysis using Abbot's technique of Natural Language Analysis [OOSE 5.4.1]

<i>Example</i>	<i>Grammatical construct</i>	<i>UML Component? *</i>
"Monopoly"	Concrete Person, Thing	
"toy"	noun	?
"3 years old"	Adjective	?
"enters"	verb	?
"depends on...."	Intransitive verb	?
"is a", "either..or", "kind of..."	Classifying verb	?
"Has a ", "consists of"	Possessive Verb	?
"must be", "less than..."	modal Verb	?

* {aggregation|attribute|class|constraint|inheritance|method|method(event)|object?}

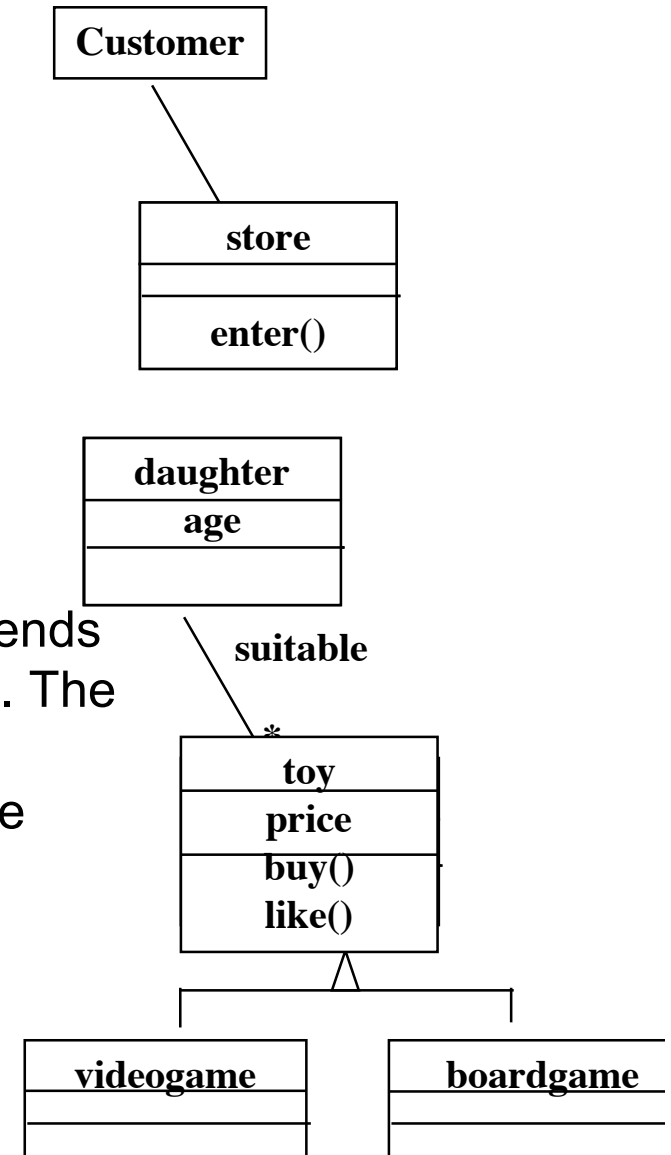
Textual Analysis using Abbot's technique of Natural Language Analysis [OOSE 5.4.1]

<i>Example</i>	<i>Grammatical construct</i>	<i>UML Component</i>
"Monopoly"	Concrete Person, Thing	Object
"toy"	noun	class
"3 years old"	Adjective	Attribute
"enters"	verb	Operation
"depends on...."	Intransitive verb	Operation (Event)
"is a" , "either..or", "kind of..."	Classifying verb	Inheritance
"Has a ", "consists of"	Possessive Verb	Aggregation
"must be", "less than..."	modal Verb	Constraint

Generation of a class diagram from flow of events

The customer enters the store to buy a toy. It has to be a toy that his daughter likes and it must cost less than 50 Euro. He tries a videogame, which uses a data glove and a head-mounted display. He likes it.

An assistant helps him. The suitability of the game depends on the age of the child. His daughter is only 3 years old. The assistant recommends another type of toy, namely a boardgame. The customer buy the game and leaves the store



Who uses class diagrams?

- Purpose of Class diagrams :
 - The description of the static properties of a system (main purpose)
 - [Has a lot to do with the semantic meaning of the parts]
- Who uses class diagrams?
- The application domain expert uses class diagrams to model the application domain
- The developer uses class diagrams during the development of a system, that is, during analysis, system design, object design and implementation.
- The customer and the end user are often not interested in class diagrams. They usually focus more on the functionality of the system.

In-Class Exercise

- A major drug distributor wants to reduce its delivery delays so that every morning order (received before 11AM) is delivered in the afternoon before 5PM and every afternoon order (received between 11 AM and 20PM) is delivered the next morning before 10AM.
- The current system operates as follows:
 - When a pharmacist calls, an order form is completed with the pharmacist's ID, a delivery address, and a list of required products and their quantities
 - a package containing the products in the required quantities is then prepared and a label with the client's delivery address is printed and attached to the package. If a product stock becomes low, an order must be sent to the furnisher immediately.
 - the prepared package is then picked up by the delivery department that organize its rounds so as to minimize costs. Any package ready before 4PM or 9AM is delivered before 5 PM or 10AM, respectively.
- The manual procedure for preparing package was found to be inefficient, mainly due to the number of errors during package preparation.
- The delivery department is considered to work efficiently and its procedures will not be changed.

Using UML in requirement modelling

Dynamic modeling

Dynamic Modeling

- Purpose:
 - Supply methods to model interactions, behaviors of participants and workflow
- How do we do this?
 - Start with use case or scenario
 - Model interaction between objects => sequence diagram
 - Model dynamic behavior of a single object => statechart diagram
 - Model workflow => activity diagram

Start with Flow of Events from Use Case

- Flow of events from “Dial a Number” Use case:
 - Caller lifts receiver
 - Dial tone begins
 - Caller dials
 - Phone rings
 - Callee answers phone
 - Ringing stops
 -

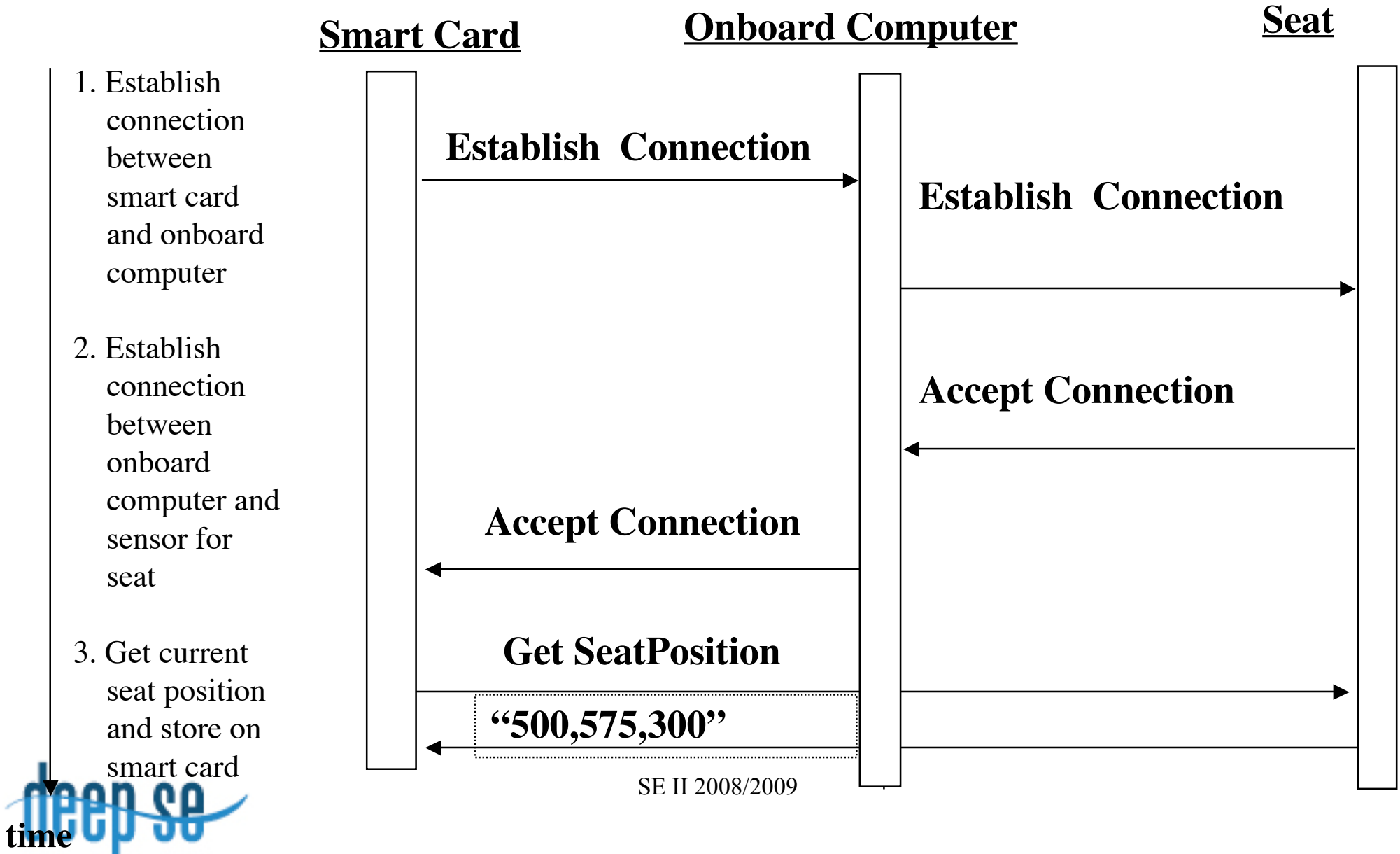
Sequence Diagram

- From the flow of events in the use case or scenario, proceed to the sequence diagram
- A sequence diagram is a graphical description of objects participating in a use case or scenario using a DAG (directed acyclic graph) notation
- Relation to object identification:
 - Objects/classes have already been identified during object modeling
 - Other objects are identified as a result of dynamic modeling
- Heuristic:
 - A event always has a sender and a receiver.
 - The representation of the event is sometimes called a message
 - Find sender and receiver for each event => These are the objects participating in the use case

An Example

- Flow of events in a “Get SeatPosition” use case :
 1. Establish connection between smart card and onboard computer
 2. Establish connection between onboard computer and sensor for seat
 3. Get current seat position and store on smart card
- Which are the objects?

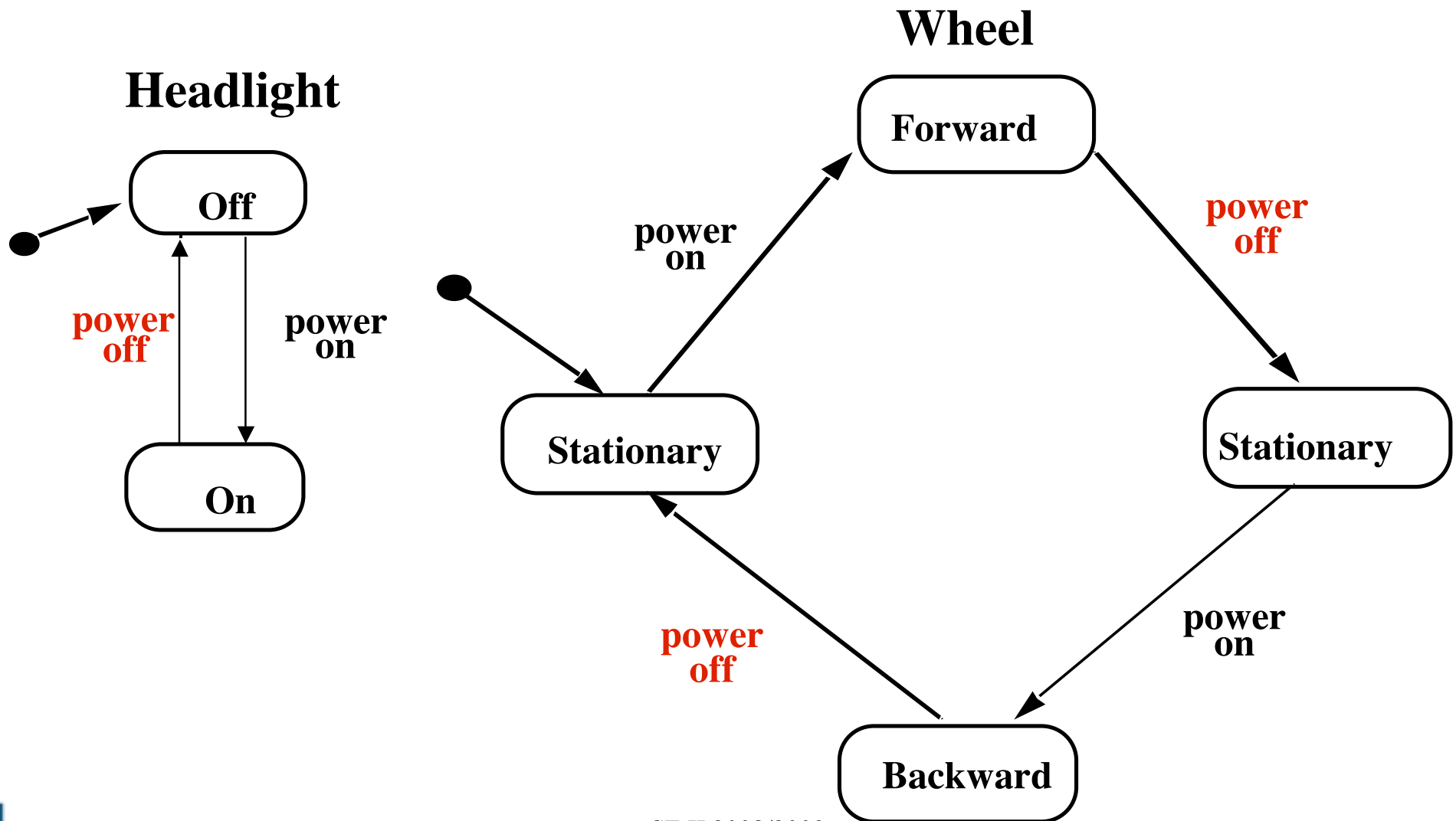
Sequence Diagram for “Get SeatPosition”



State Chart Diagram vs Sequence Diagram

- State chart diagrams help to identify:
 - *Changes* to an individual object over time
- Sequence diagrams help to identify
 - The *temporal relationship* of between objects over time
 - *Sequence of operations* as a response to one ore more events
- *State diagrams are class-level documentation; sequence diagrams are instance-level documents.*
 - ***We can infer many possible event-order-dependent behaviors from an STD;***
 - ***A sequence diagram shows one behavioral history, for one specific ordering of input events.***




An example: toy car statecharts



Practical Tips for Dynamic Modeling

- Construct dynamic models only for classes with significant dynamic behavior
 - Avoid “analysis paralysis”
- Consider only relevant attributes
 - Use abstraction if necessary ***[Partition the data state space into control states (equivalence classes of data state values that have the same dynamic behavior)]***
- Look at the granularity of the application when deciding on actions and activities
- Reduce notational clutter
 - Try to put actions into state boxes (look for identical actions on events leading to the same state)

Summary: Requirements Analysis

- What are the transformations?  **Functional Modeling**
 - Create scenarios and use case diagrams
 - Talk to client, observe, get historical records, do thought experiments
- What is the structure of the system?  **Object Modeling**
 - Create class diagrams [static information models]
 - Identify objects.
 - What are the associations between them? What is their multiplicity?
 - What are the attributes of the objects?
 - What operations are defined on the objects?
- What is its behavior?  **Dynamic Modeling**
 - Create sequence diagrams [Dynamic object behavior instance examples]
 - Identify event senders and receivers
 - Show sequence of events exchanged between objects. Identify event dependencies and event concurrency.
 - Create state diagrams [Dynamic class method behavior models]
 - Only for the dynamically interesting objects.

Running example

We want to build a system like the poliself system: it should be used to submit study plans, register to exams, register grades and (for instructors) to decide the date of an exam.

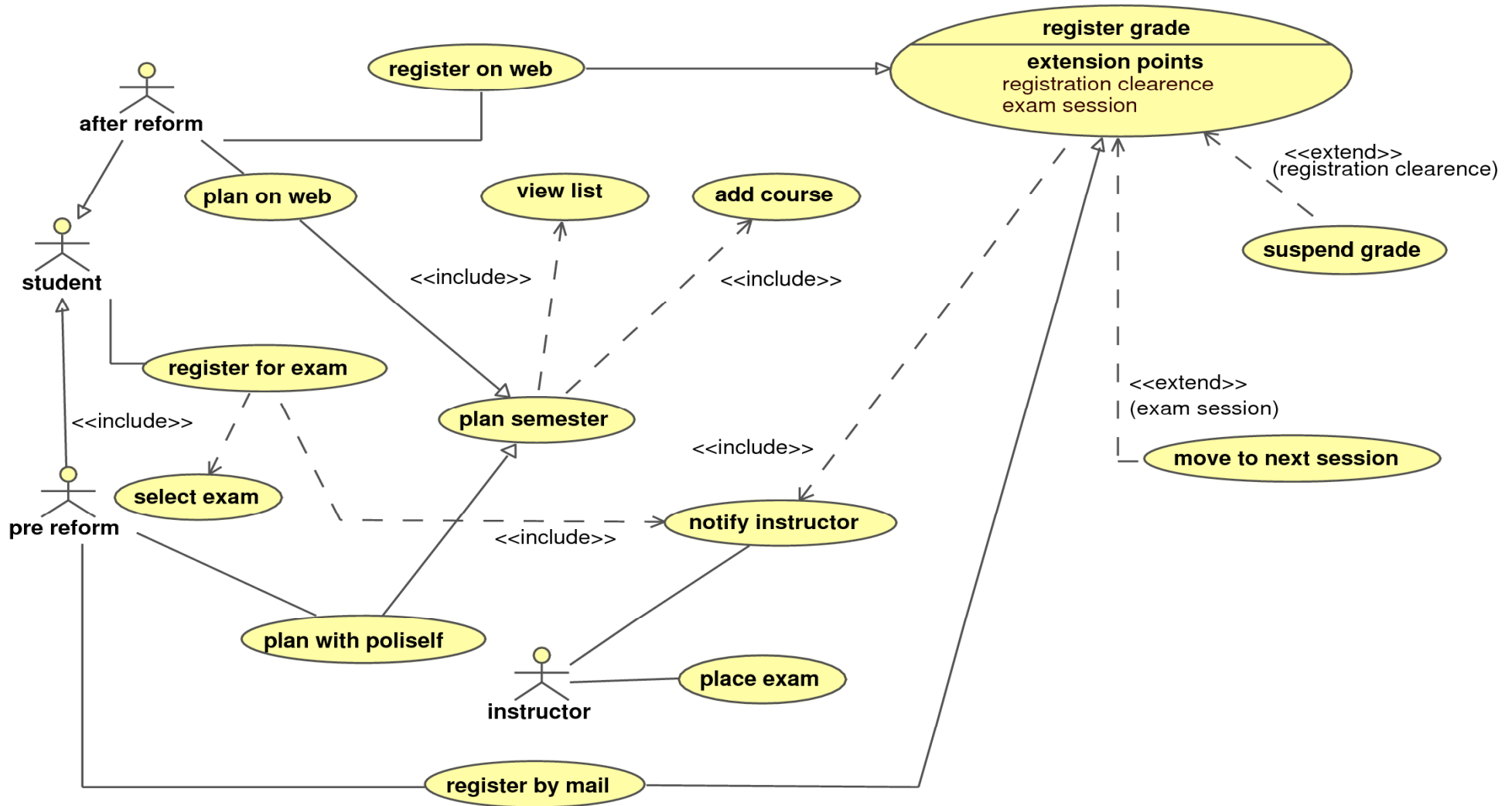
It works differently for pre reform students and after reform students: pre reform students must register a grade by mail and must present the study plan with a poliself terminal, while after reform students can do both things on the web

Instructors are enabled to decide the date of an exam, and to be notified both when a student register or drops a grade and when a student registers for a course.

UML2 in requirements elicitation

- Use cases/ scenario identification
 - allows to decompose the system into macro functionalities
 - associates user to roles
 - defines relationships between functionalities
 - Defines flow of events in an informal way (with textual scenarios)
- Conceptual model
 - defines the concepts that are part of the application domain (coming from use cases)
 - Formalizes interaction between concepts

Poliself use case diagram



Some use cases description

Use case	register on web
Type	Primary
Pre-condition	Have a grade to be registered
Normal flow	The user executes the login, then he chooses an exam from the list of the taken exams. At this point he sees the grade for that exam and he can choose to register it or to drop it. At the end the instructor is notified
Post-condition	The grade is registered if the student wants

Use case	register for an exam
Type	Primary
Pre-condition	Have a grade to be registered
Normal flow	The student logs into the system, the studyplan is presented. Now the student chooses a course from the list, and chooses an exam for a specific course. At this point the student can register the exam he has chosen and the instructor is notified.
Post-condition	the student is registered and the instructor are notified

Some use cases description

Use case	Notify the instructor
Type	Primary
Pre-condition	An exam has been registered or dropped
Normal flow	The instructor logs into the system, then it sees a list of notifications coming from registration events. He can see for every event, the name of the student, the proposed grade and the action chosen by the student
Post-condition	seen notifications are deleted

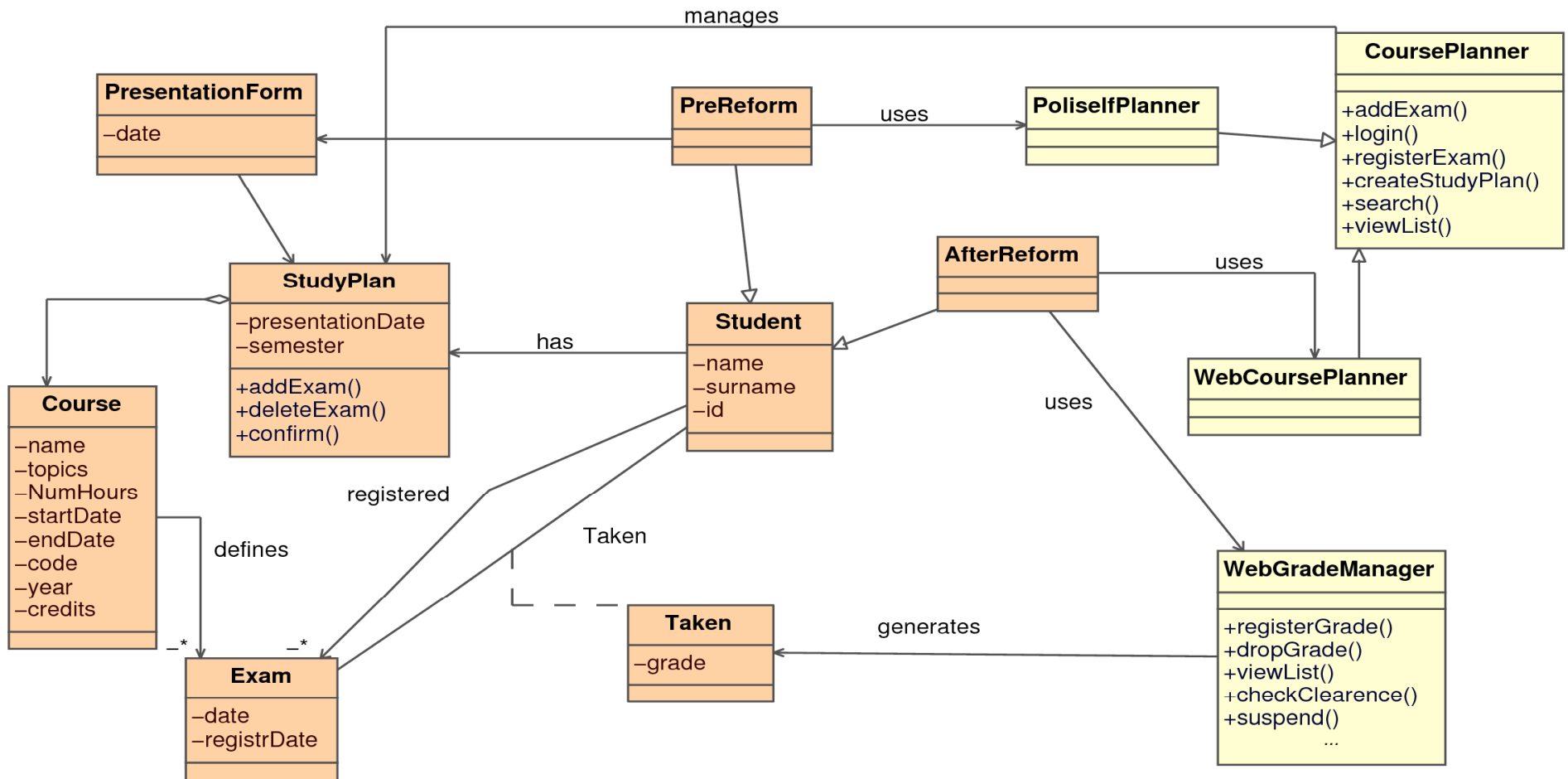
Evaluation

- This model is rather informal
 - Not enough to describe the design of the system
- We need a more precise description of the system
 - Need an analysis phase to define the concepts which are part of the systems
 - Need to define the behaviour of these elements
 - This part goes before implementation or architectural decision but drives them

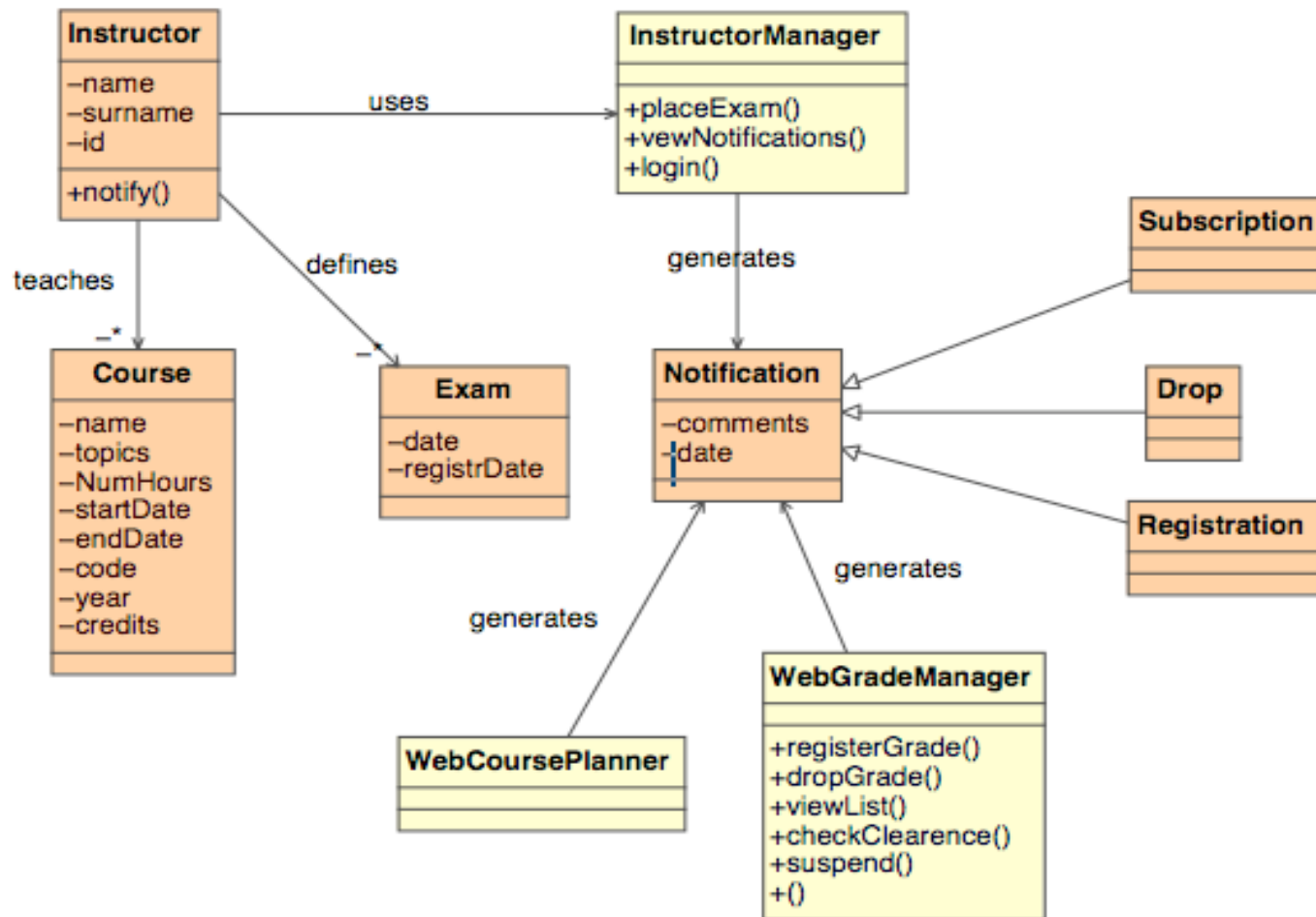
Conceptual model

- Conceptual model:
 - structural (classes)
 - *dynamic (sequence diagrams and statecharts)*
- *These elements define elements viewed by the user*
 - *Yes: studyplan, exam, registrationSystem*
 - *No: courseDatabase, network,....*

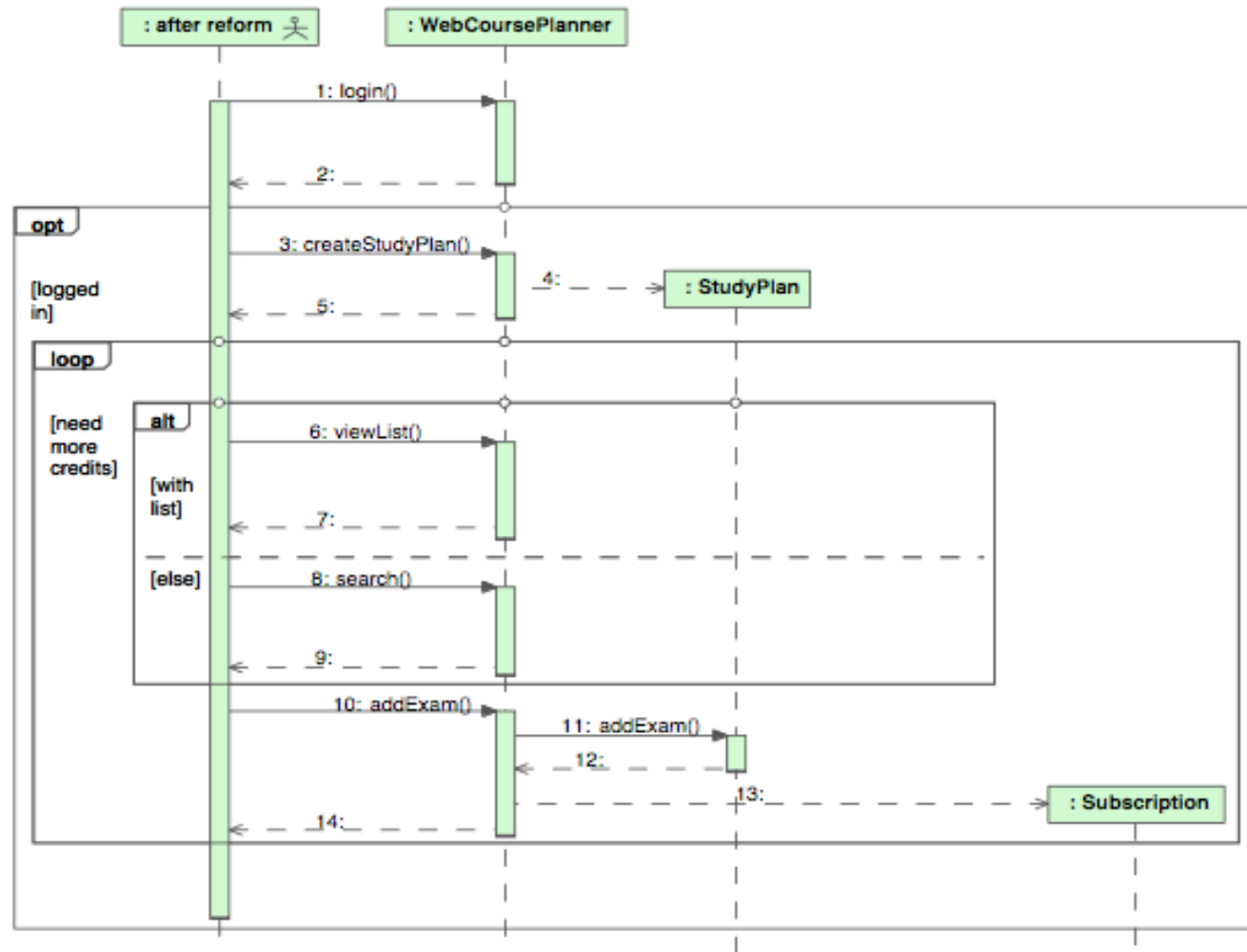
Conceptual model



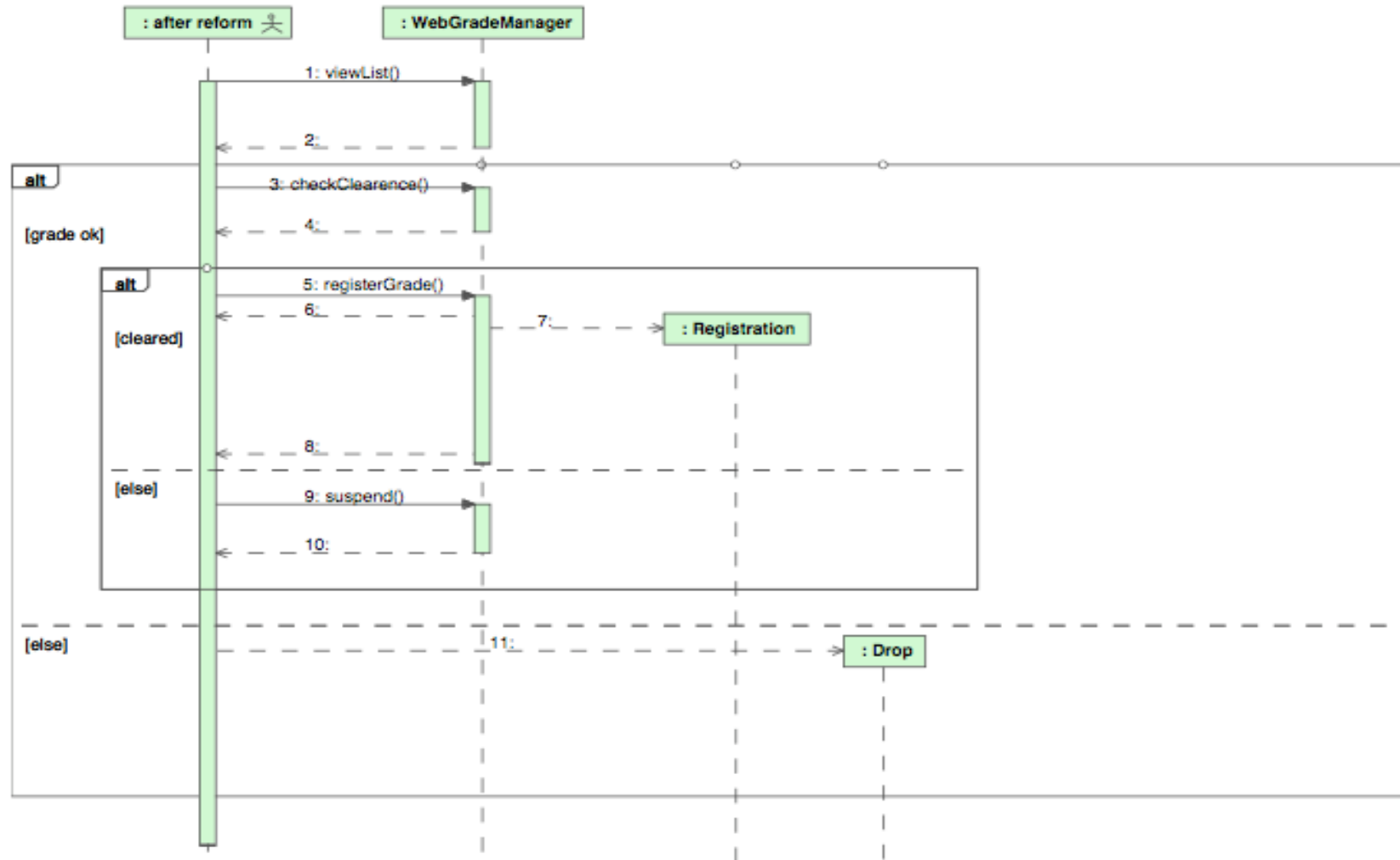
Conceptual model (contd.)



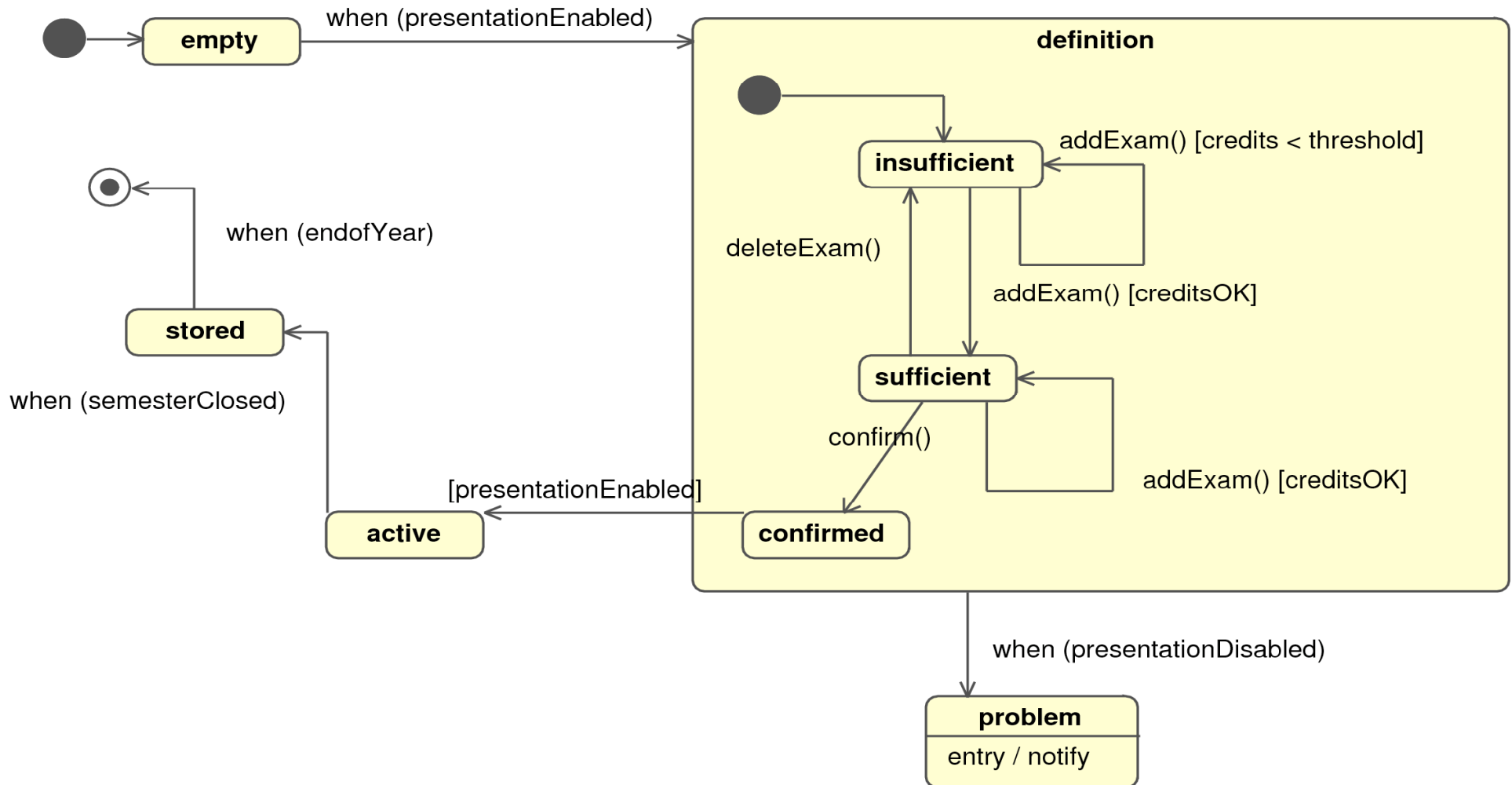
Conceptual model (interaction)



Conceptual model (interaction)



Conceptual model (state based)



Bibliography

- B. Bruegge & A.H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns, and Java, 2nd Edition, Prentice Hall, Upper Saddle River, NJ, September 25, 2003.
- A. van Lamsweerde, Requirements Engineering, Wiley and Sons (draft on the web).
- E. Letier, Lectures on Requirement Engineering, UCL.