MTAT.03.094
Software Engineering

Lecture 04: Analysis

Dietmar Pfahl

email: dietmar.pfahl@ut.ee

Fall 2013

Labs Next Week

- Part of the lab time will be used for assessment
  - your Task 1 solutions
- Remember that each project team must attend with all its members present
  - If not all team members present, penalty applies

Submission of Lab Task 2 is soon coming. Don't miss it!

Schedule of Lectures

Week 01: Introduction to SE

Week 02: Requirements Engineering I

Week 03: Requirements Engineering II

Week 04: Analysis

Week 05: Development Infrastructure I

Week 06: Development Infrastructure II

Week 07: Architecture and Design

Week 08: Refactoring

Week 09: Measurement

Week 10: Agile/Lean Methods

Week 11: Verification and Validation I (incl. SW Quality)

Week 12: Verification and Validation II

Week 13: Process Improvement

Week 14: Course wrap-up, review and exam preparation

Week 15: no lecture

Week 16: no lecture

Acknowledgements

Textbooks/Slides:

- Ian Sommerville: Software Engineering, 9th edition, 2010
  (http://www.softwareengineering-9.com/)
- Ivan Marsic: Software Engineering, 2012
  (http://www.ece.rutgers.edu/~marsic/books/SE/book-SE_marsic.pdf)

Structure of Lecture 04

- Preliminaries
  - Use Cases
  - UML Notations: UC Diagram, Class Diagram, Sequence Chart, State Chart
- Domain Analysis and Modelling Example
  - Identifying Concepts (Responsibilities)
  - Concept Attributes
  - Concept Associations
- Domain Analysis and Modelling Summary

Schema for Detailed Use Cases

Use Case UC-#:	**Name / Identifier [verb phrase]**
Related Requirements:	List of the requirements that are addressed by this use case
Initiating Actor:	Actor who initiates interaction with the system to accomplish a goal
Actor's Goal:	Informal description of the initiating actor's goal
Participating Actors:	Actors that will help achieve the goal or need to know about the outcome
Preconditions:	What is assumed about the state of the system before the interaction starts
Postconditions:	What are the results after the goal is achieved or abandoned; i.e., what must be true about the system at the time the execution of this use case is completed

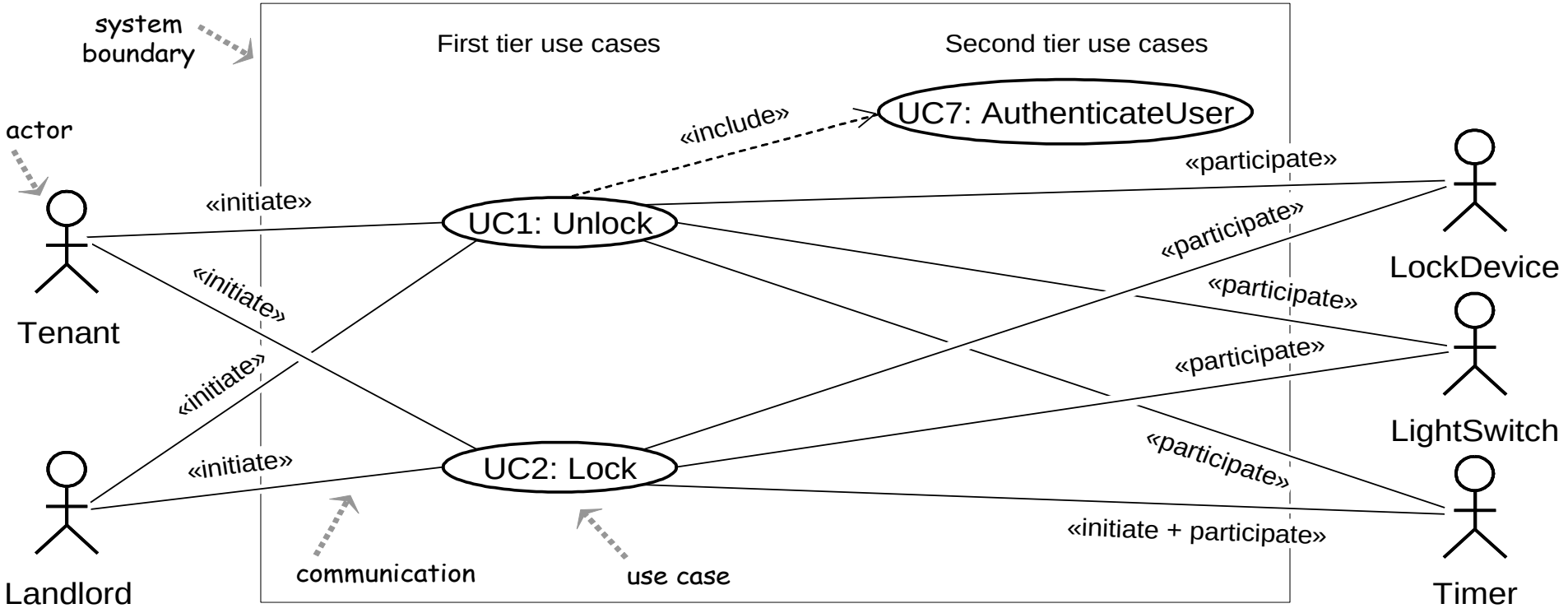
Flow of Events for Main Success Scenario:		
→	1.	The initiating actor delivers an action or stimulus to the system (the arrow indicates the direction of interaction, to- or from the system)
←	2.	The system's reaction or response to the stimulus; the system can also send a message to a participating actor, if any
→	3.	…

Flow of Events for Extensions (Alternate Scenarios): What could go wrong? List the exceptions to the routine and describe how they are handled		
→	1a.	For example, actor enters invalid data
←	2a.	For example, power outage, network failure, or requested data unavailable
		…

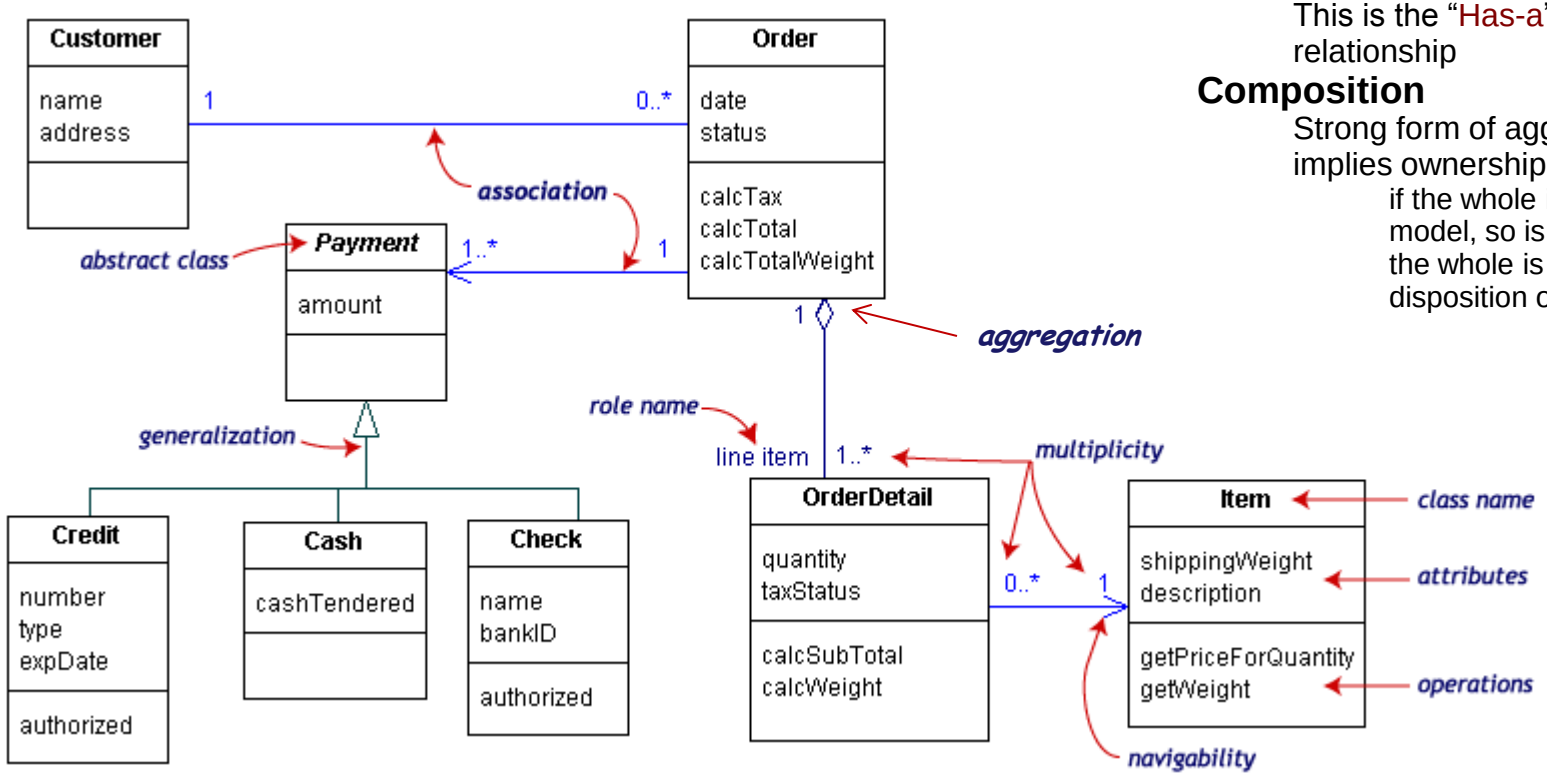
The arrows on the left indicate the direction of interaction: → Actor's action; ← System's reaction

Use Case Diagram Example

UC1: Unlock
UC2: Lock
UC3: AddUser
UC4: RemoveUser
UC5: InspectAccessHistory
UC6: SetDevicePrefs
UC7: AuthenticateUser
UC8: Login



Class Diagram



Aggregation

This is the "Has-a" or "Whole/part" relationship

Composition

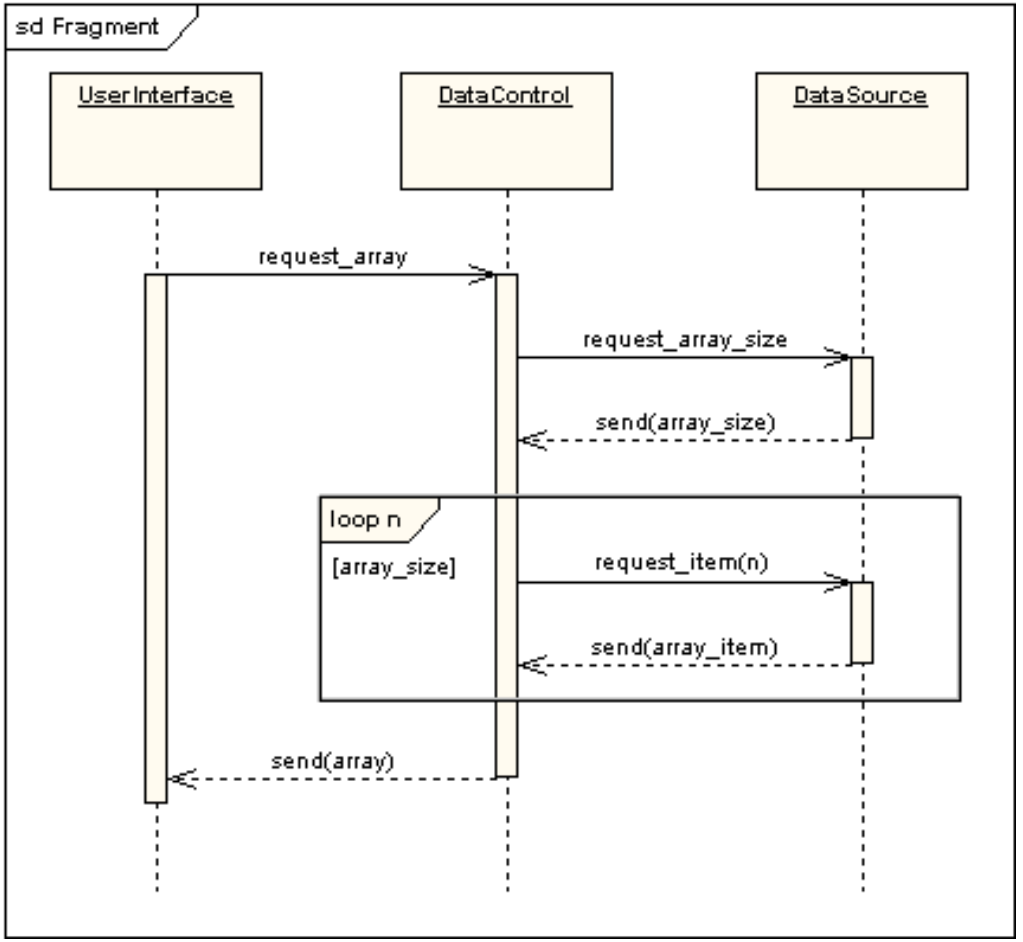
Strong form of aggregation that implies ownership:

- if the whole is removed from the model, so is the part
- the whole is responsible for the disposition of its parts

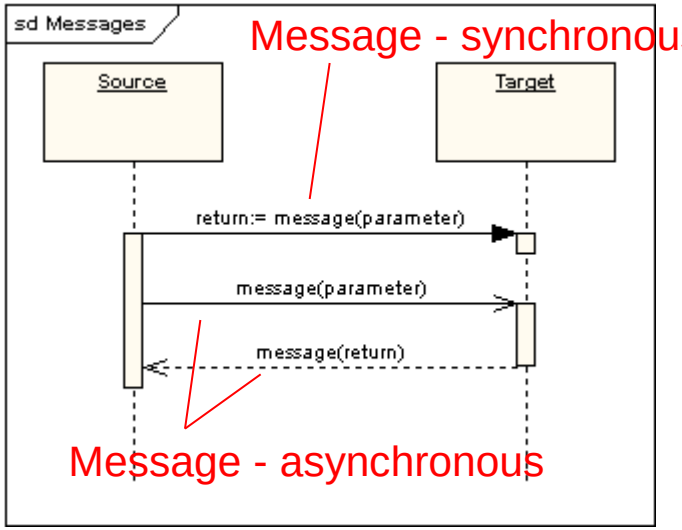
Can be used to model the domain structure, i.e. concepts and their properties and relationships

Sequence Charts

- A sequence chart is a form of interaction diagram which shows objects as lifelines running down the page, with their interactions over time represented as messages drawn as arrows from the source lifeline to the target lifeline.
- Sequence charts are good at showing which objects communicate with which other objects; and what messages trigger those communications.
- Sequence charts are not intended for showing complex procedural logic.

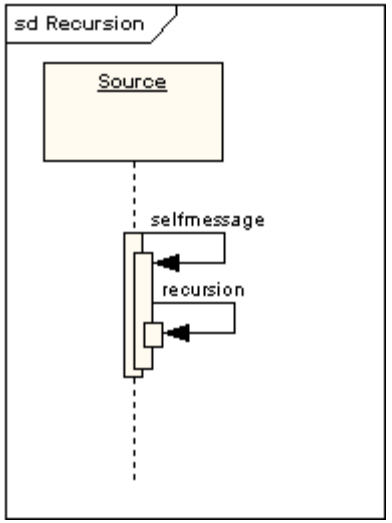


Sequence Charts



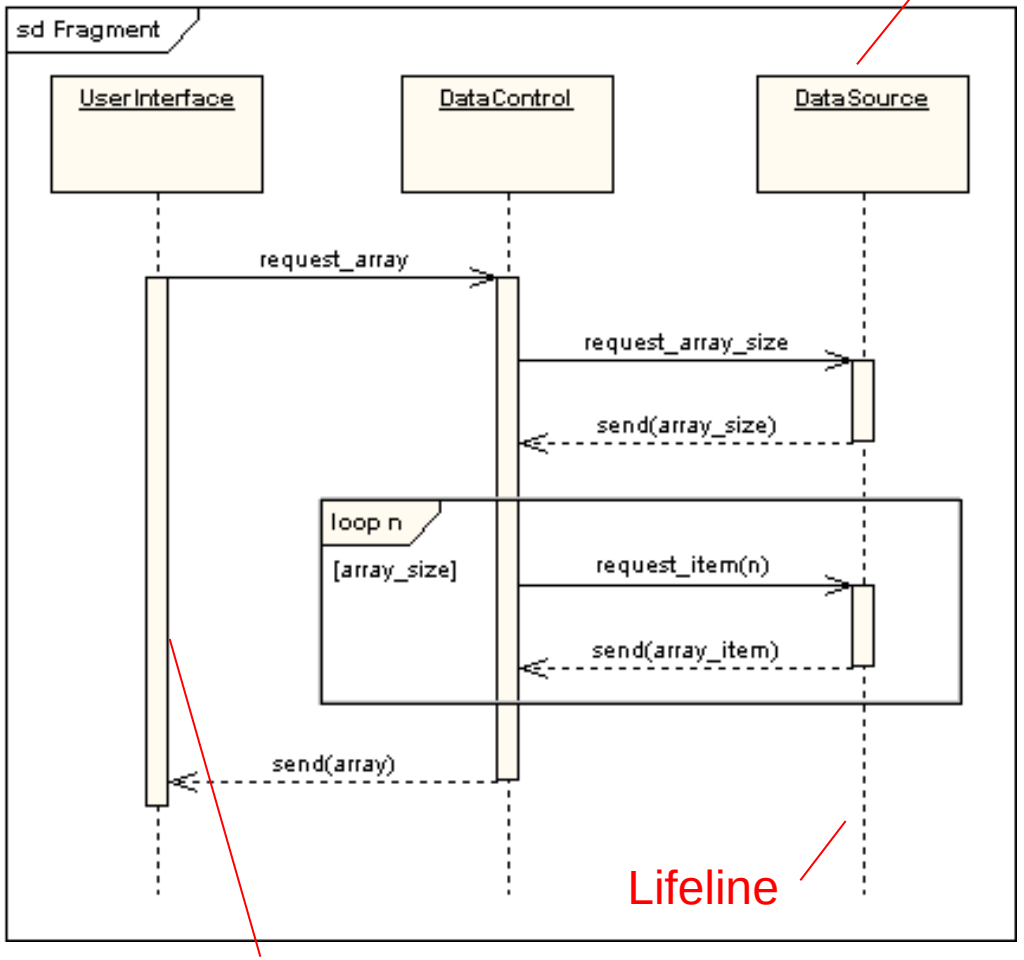
Message - synchronous

Message - asynchronous



Self-message
&
Recursion

Object (or Actor)



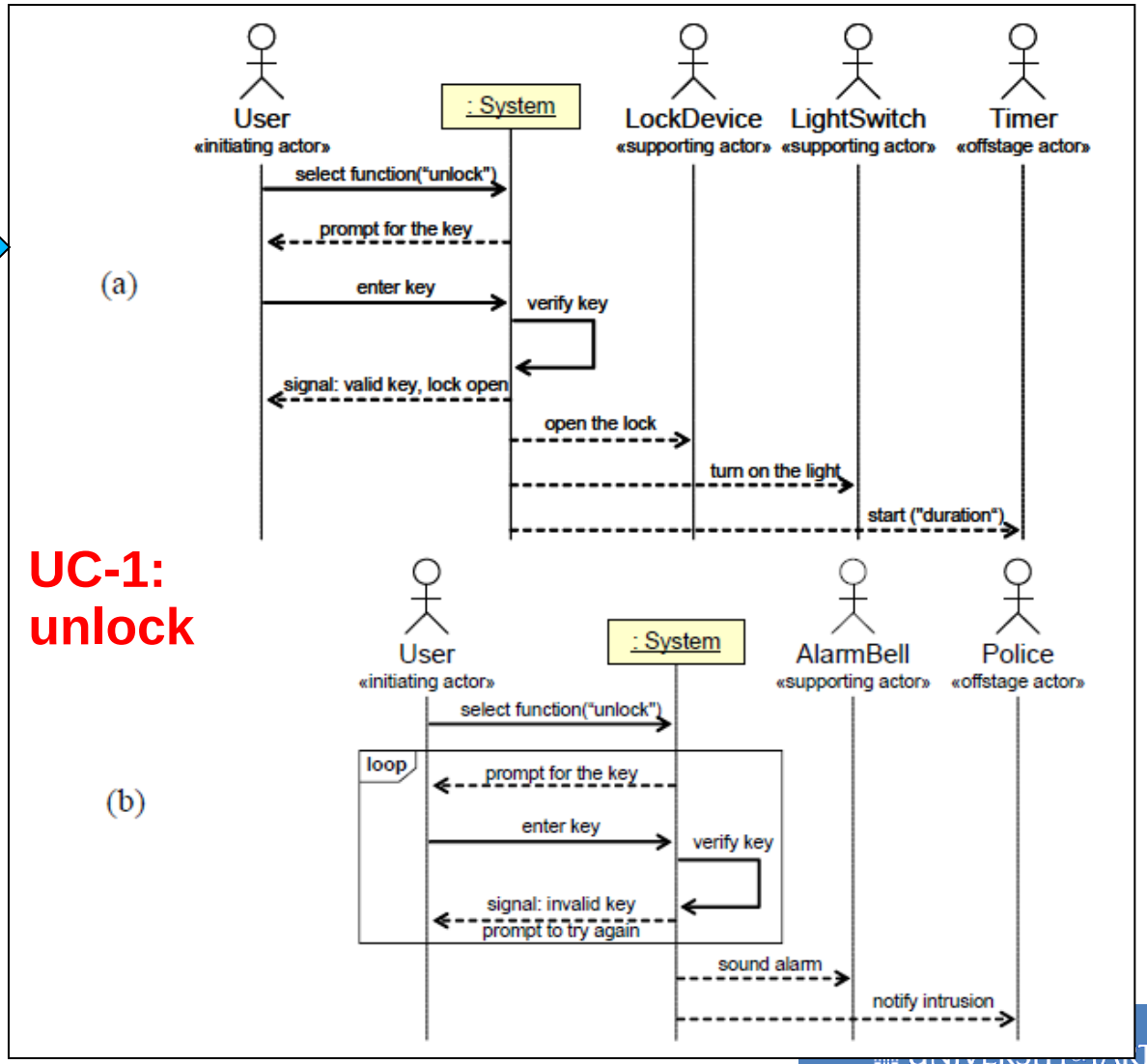
Lifeline

Execution Occurrance

Sequence Charts

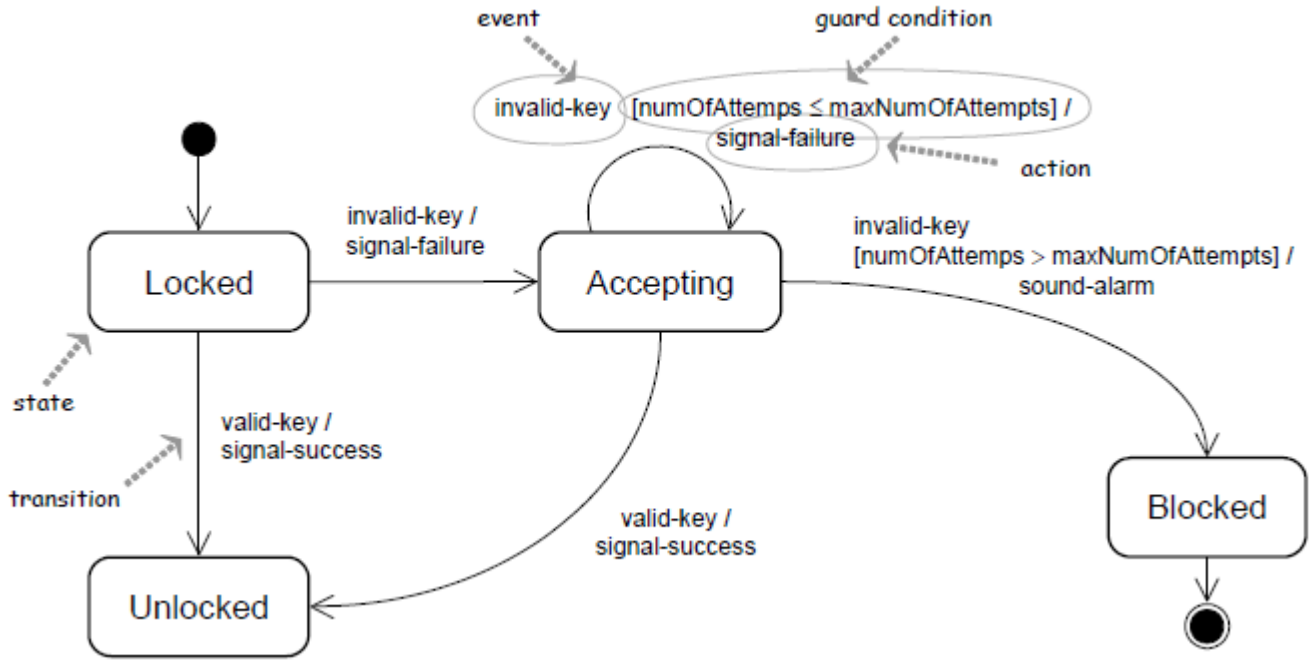
Can be used to:

- Represent usage scenarios of actors interacting with the system-to-be (analysis)
- Represent interaction between objects within the system (design)



State Charts

- SCs are used to detail the transitions or changes of state an object can go through in the system.
- SCs show how an object moves from one state to another and the rules that govern that change.
- SCs typically have a start and end condition.



Example: State Chart of class 'Controller'

Useful mainly in design and for testing

Structure of Lecture 04

- Preliminaries
  - Use Cases
  - UML Notations: UC Diagram, Class Diagram, Sequence Chart, State Chart
- Domain Analysis and Modelling Example
  - Identifying Concepts (Responsibilities)
  - Concept Attributes
  - Concept Associations
- Domain Analysis and Modelling Summary

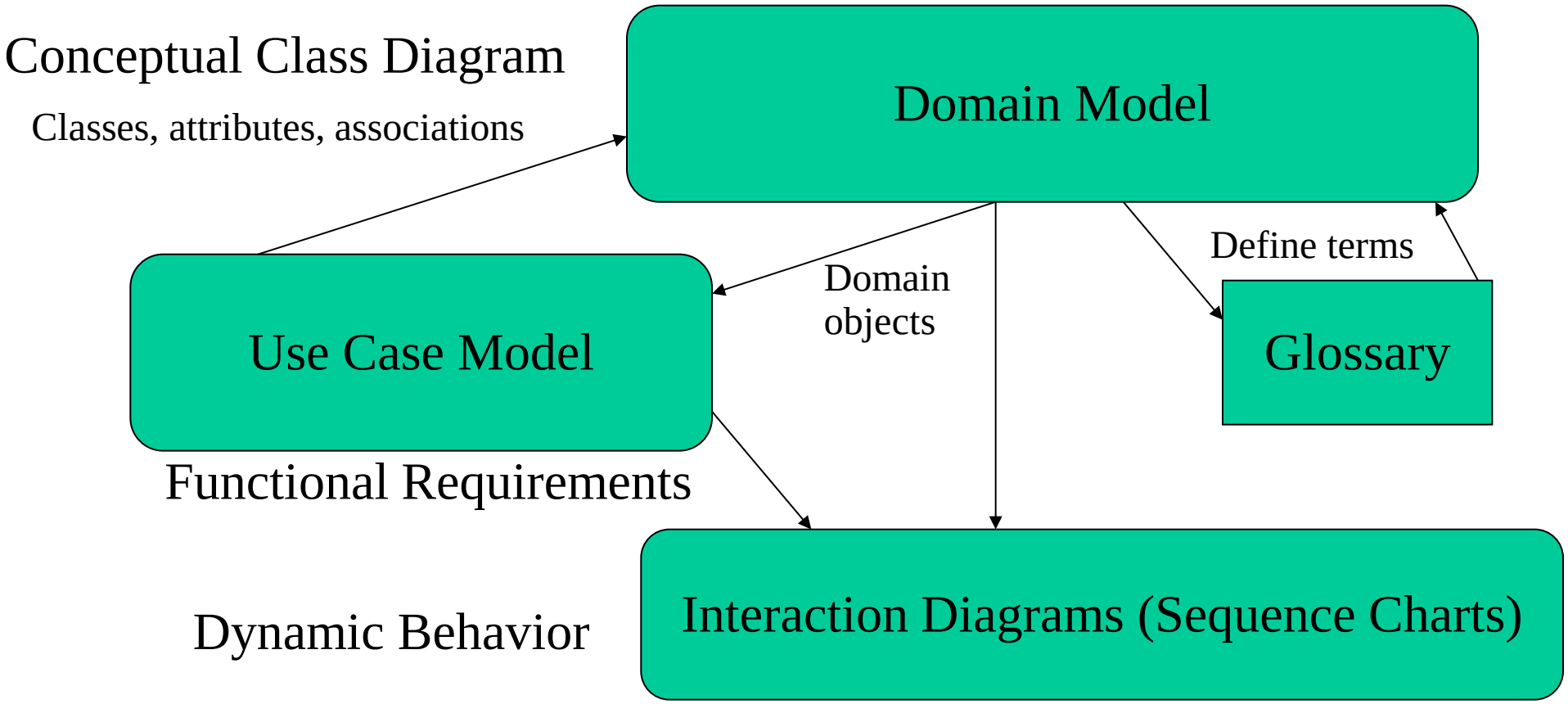
What is a Domain Model?

"A domain model captures the most important types of objects in the context of the business. The domain model represents the 'things' that exist or events that transpire in the business environment." – I. Jacobsen

Why do a Domain Model?

- Gives a conceptual framework of the things in the problem space
- Helps you think – focus on semantics
- Provides a glossary of terms – noun based
- It is a static view - meaning it allows us convey time invariant business rules
- Intertwined with use case/workflow modelling
- Based on the defined structure, we can describe the state of the problem domain at any time.

Domain Model Relationships

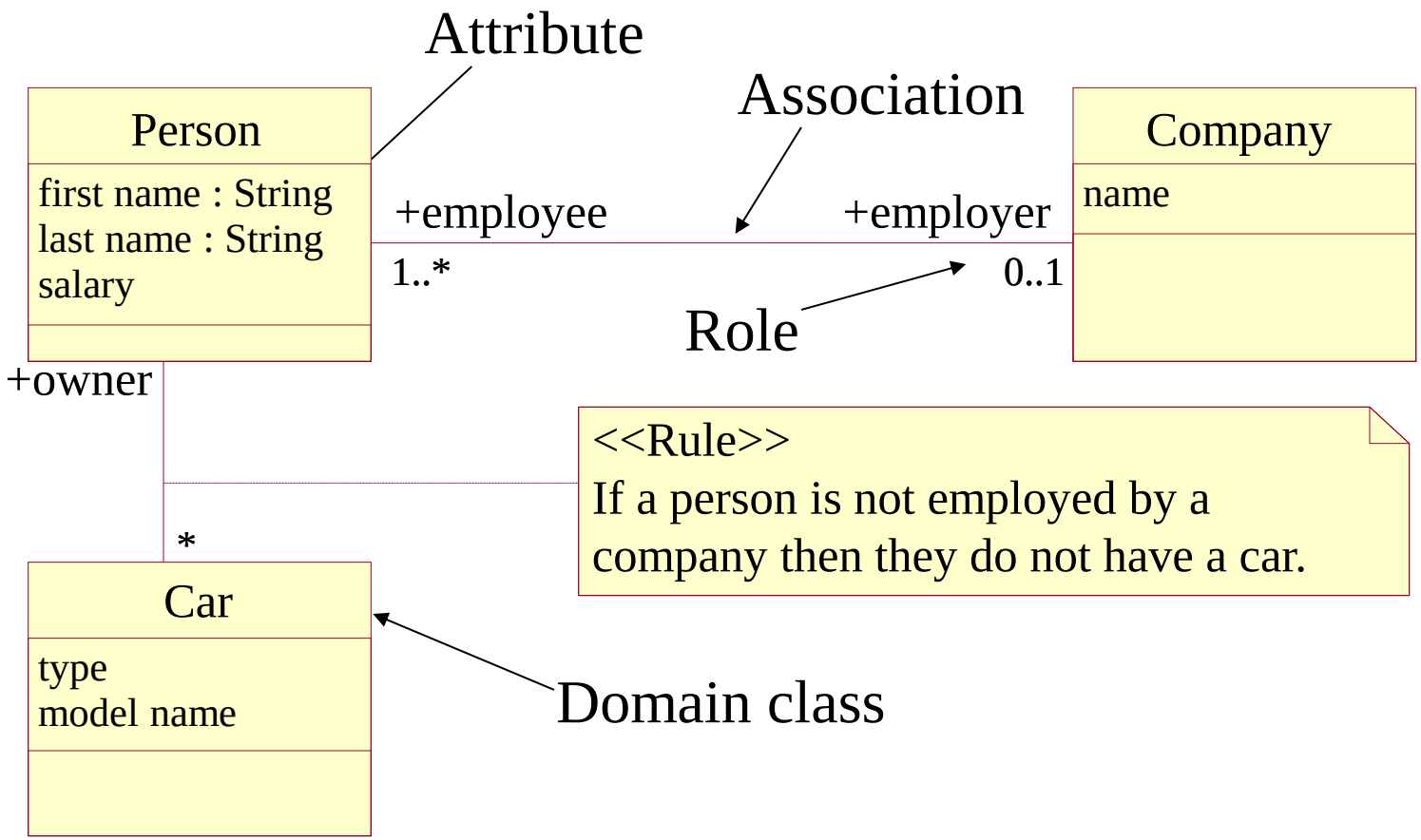


What do you learn about when and how to create these models?

Elements of a Domain Model

- The following elements enable us to express time invariant static business rules for a domain:-
  - **Domain classes** – each domain class denotes a concept (type of object).
  - **Attributes** – an attribute is the description of a named slot of a specified type in a domain class; each instance of the class separately holds a value.
  - **Associations** – an association is a relationship between two (or more) domain classes that describes links between their object instances. Associations can have roles, describing the multiplicity and participation of a class in the relationship.
  - **Additional rules** – complex rules that cannot be shown with symbols can be shown with attached notes.

Simple Domain Model Example



What are Domain Classes?

- Each domain class denotes a concept. It is a descriptor for a set of things that share common properties.
- Domain Classes can be:
  - *Business objects* - things that are manipulated in the business e.g. *Order*.
  - *Real world objects* - things that the business keeps track of e.g. *Contact*, *Site*.
  - *Actors/Workers/Persons* - e.g. *Controller* and *Customer*.
  - *Events that transpire* - e.g. *Sale* and *Payment*.
- A domain class has attributes and associations with other classes.

How to create a Domain Model?

Perform the following in very short iterations:

- Make a list of candidate domain classes.
- Draw these classes in a UML class diagram.
- If possible, add brief descriptions for the classes.
- Identify any associations that are necessary.
- Decide if some domain classes are really just attributes.
- Where helpful, identify role names and multiplicity for associations.
- Add any additional static rules as UML notes that cannot be conveyed with UML symbols.
- Group diagrams/domain classes by category into packages.
- Concentrate more on just identifying domain classes in early iterations!

How to identify Domain Classes?

- An obvious way to identify domain classes is to identify nouns and phrases in textual descriptions of a domain.

Consider a use case description as follows:

1. **Customer** arrives at a **checkout** with **goods** and/or **services** to purchase.
2. **Cashier** starts a new **sale**.
3. **Cashier** enters **item identifier**.
4. System records the **sale line item** and presents the **item description, price** and running **total**.

Identifying Concepts (Domain Classes) from noun phrases

- Vision and Scope, Glossary and Use Cases are good for this type of linguistic analysis

- However:
  - Words may be ambiguous or synonymous
  - Noun phrases may also be attributes or parameters rather than classes:
    - If it stores state information or it has multiple behaviors, then it's a class
    - If it's just a number or a string, then it's probably an attribute

How to identify Attributes ?

A domain class sounds like an attribute if …

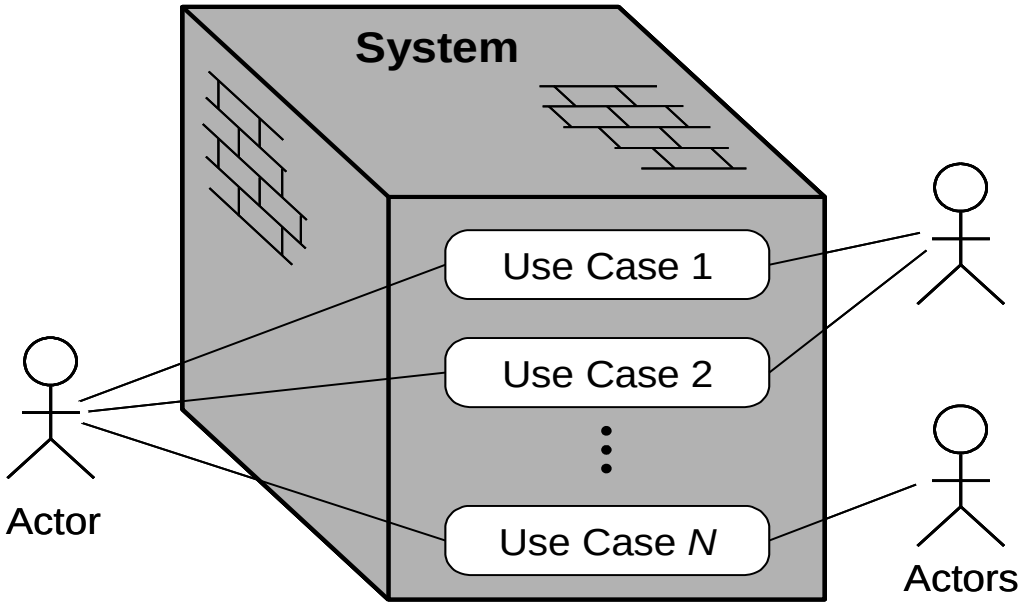
- It relies on an associated class for it's identity – e.g. 'order number' class associated to an 'order' class. The 'order number' sounds suspiciously like an attribute of 'order'.
- It is a simple data type – e.g. 'order number' is a simple integer. Now it really sounds like an attribute!

Use Cases vs. Domain Model

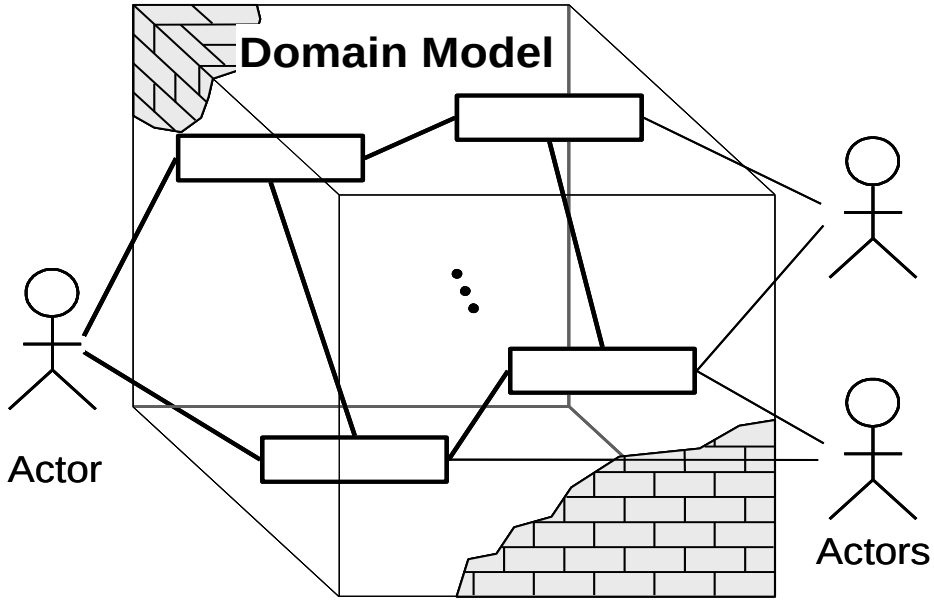
In **use case analysis**, we consider the system as a "**black box**"

In **domain analysis**, we consider the system as a "**transparent box**"

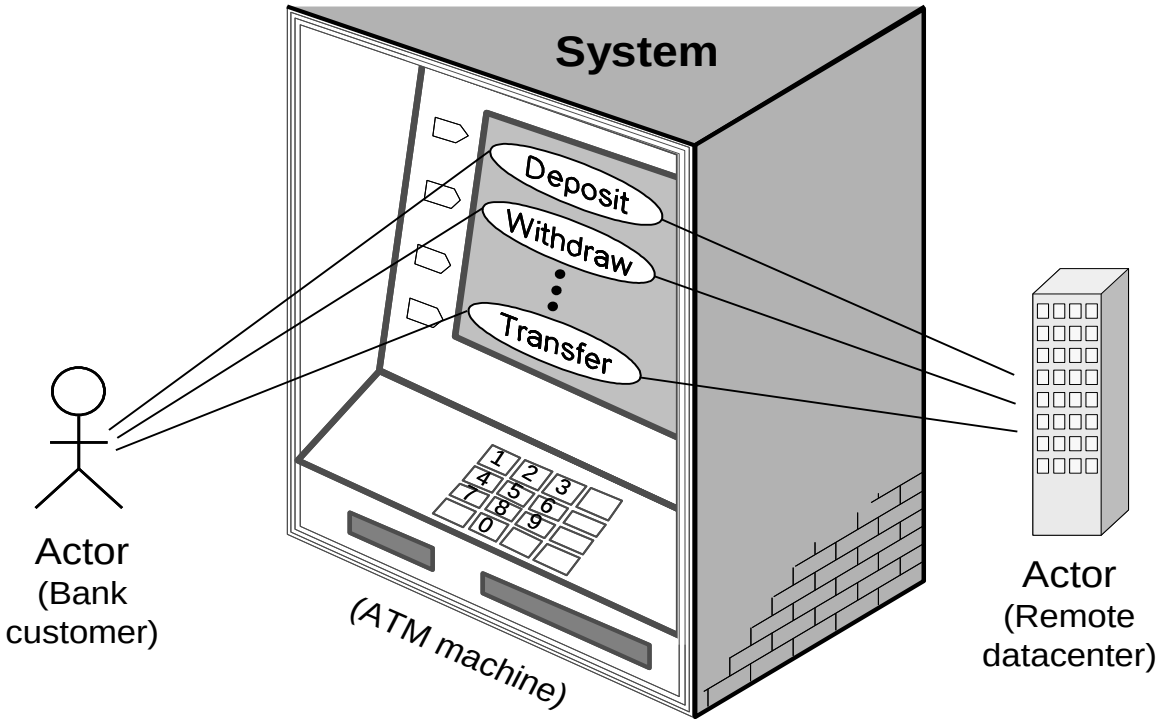
(a)



(b)

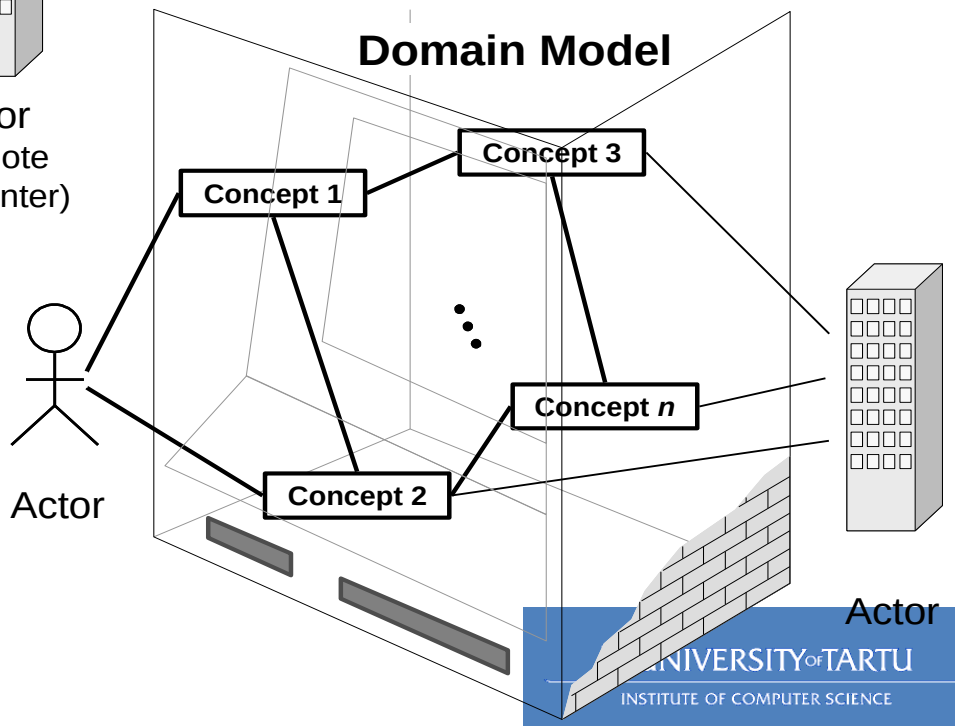


Example: ATM Machine



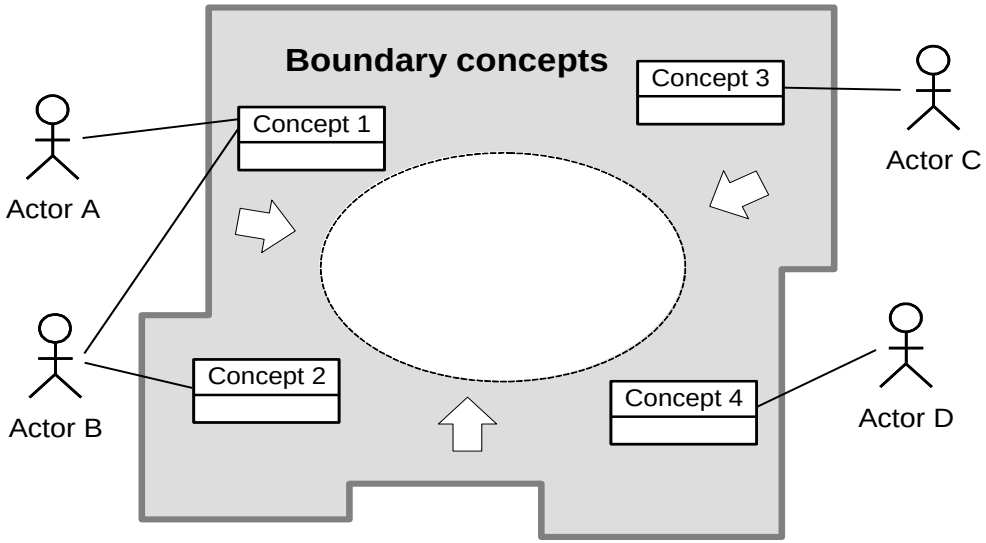
(a)

(b)

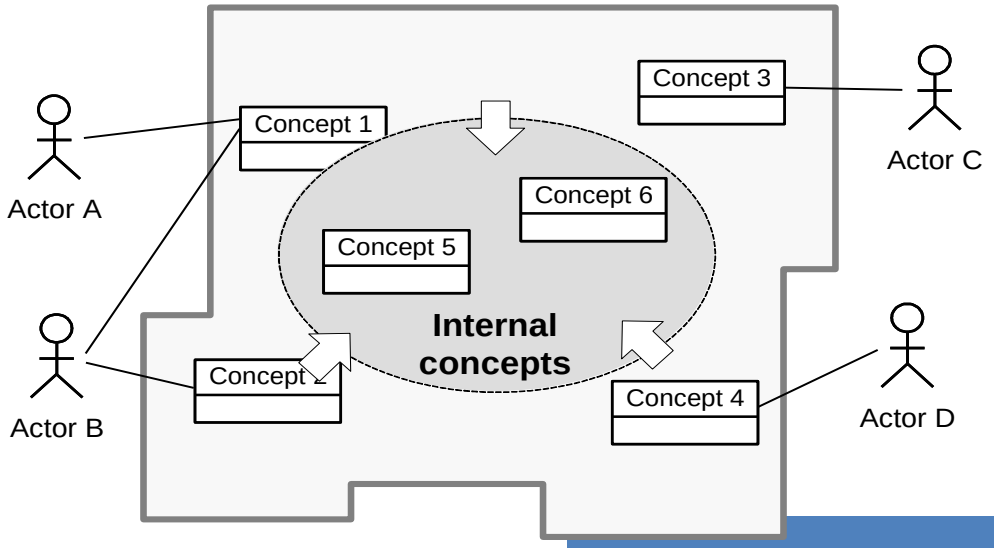


Building a Domain Model

Step 1: Identifying the boundary concepts



Step 2: Identifying the internal concepts

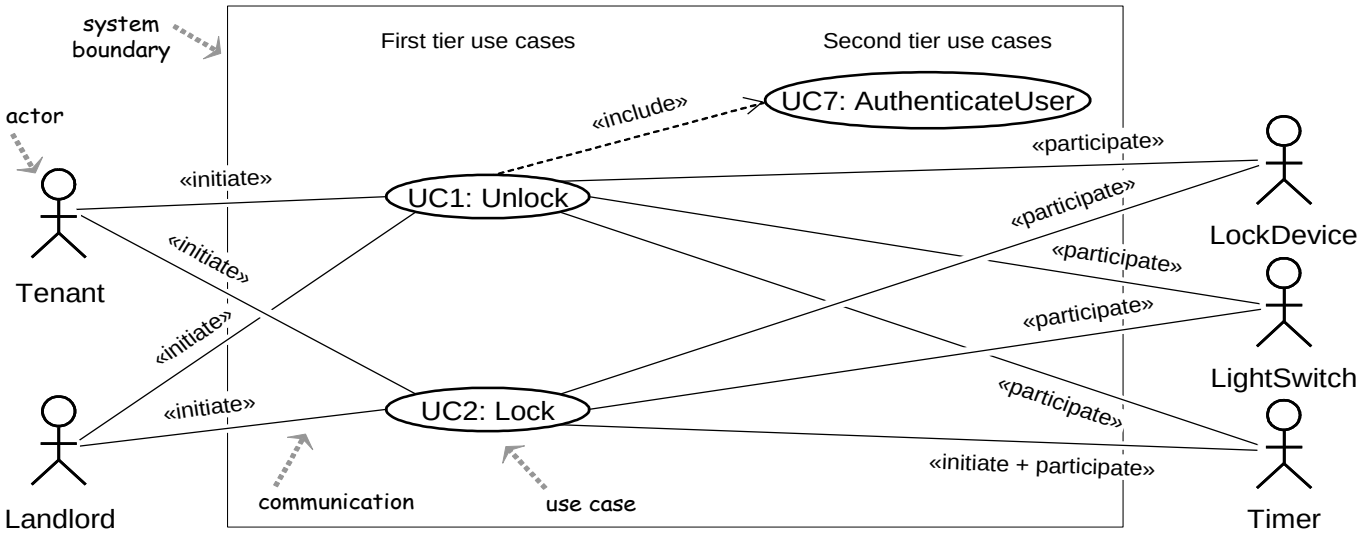


Domain Modelling Strategies for Concept Identification

- ’Outside-In’ Approach: First identify boundary concepts, then internal concepts
  - Internal concepts might be further classified into control and entity concepts
- ’Setting-up-an-enterprise’ Approach: What workers need to be hired and what things acquired?
  - Start with ’worker’ concepts and their responsibilities
    - Usually (at least) one ’Controller’
  - Distinguish between ’doing’ (D) and ’knowing’ (K) responsibilities
    - Usually: D=worker and K=thing; but not always clear

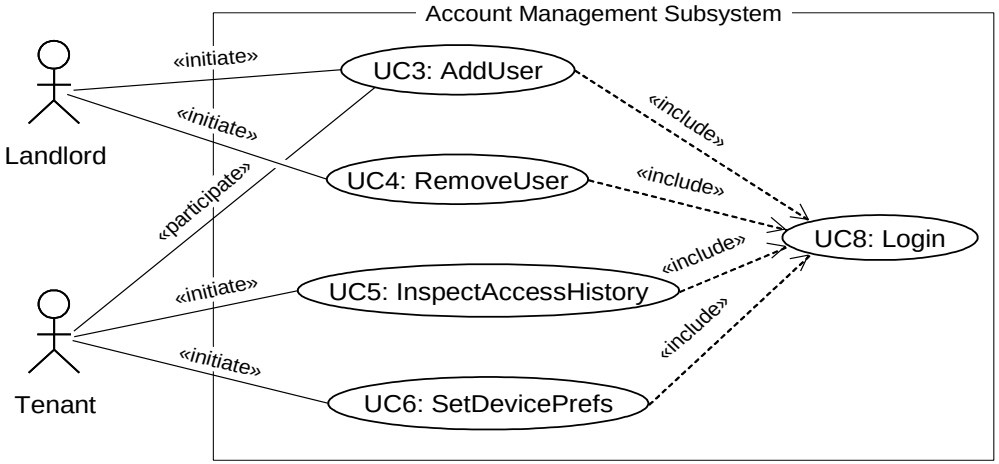
Use Case Diagrams

UC1: Unlock
UC2: Lock
UC3: AddUser
UC4: RemoveUser
UC5: InspectAccessHistory
UC6: SetDevicePrefs
UC7: AuthenticateUser
UC8: Login



Subsystem 1: Device Control

Subsystem 2: Account Management



Use Case 1: Unlock

Use Case UC-1:	**Unlock**
Related Requirements:	REQ1, REQ3, REQ4, and REQ5 stated in Table 2-1
Initiating Actor:	Any of: Tenant, Landlord
Actor's Goal:	To disarm the lock and enter, and get space lighted up automatically.
Participating Actors:	LockDevice, LightSwitch, Timer
Preconditions:	• The set of valid keys stored in the system database is non-empty. • The system displays the menu of available functions; at the door keypad the menu choices are "Lock" and "Unlock."
Postconditions:	The auto-lock timer has started countdown from autoLockInterval.

Flow of Events for Main Success Scenario:

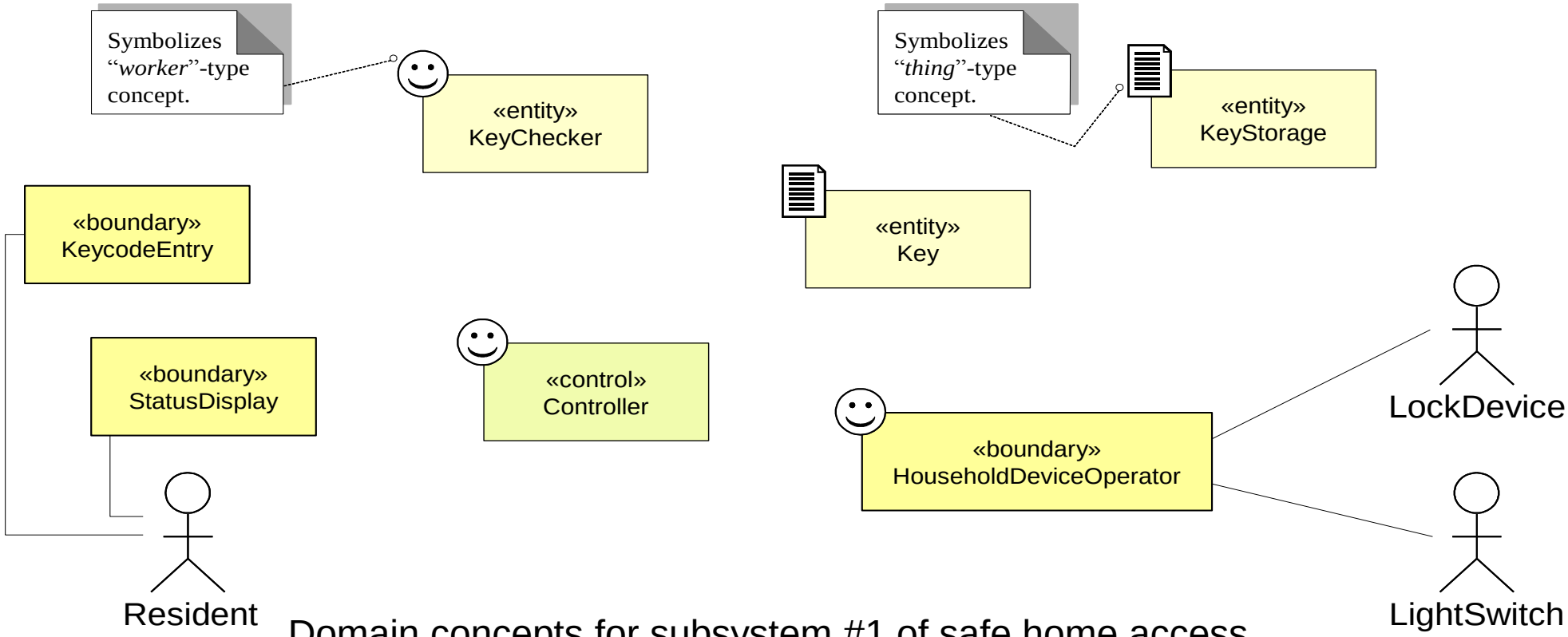
→	1.	Tenant/Landlord arrives at the door and selects the menu item "Unlock"
	2.	include::*AuthenticateUser* (UC-7)
←	3.	System (a) signals to the Tenant/Landlord the lock status, e.g., "disarmed," (b) signals to LockDevice to disarm the lock, and (c) signals to LightSwitch to turn the light on
←	4.	System signals to the Timer to start the auto-lock timer countdown
→	5.	Tenant/Landlord opens the door, enters the home [and shuts the door and locks]

Extracting the Responsibilities

Responsibility Description	Type	Concept Name
Coordinate actions of all concepts associated with a use case, a logical grouping of use cases, or the entire system and delegate the work to other concepts.	D	Controller
Container for user's authentication data, such as pass-code, timestamp, door identification, etc.	K	Key
Verify whether or not the key-code entered by the user is valid.	D	KeyChecker
Container for the collection of valid keys associated with doors and users.	K	KeyStorage
Operate the lock device to armed/disarmed positions.	D	LockOperator
Operate the light switch to turn the light on/off.	D	LightOperator
Operate the alarm bell to signal possible break-ins.	D	AlarmOperator
Block the input to deny more attempts if too many unsuccessful attempts.	D	Controller
Log all interactions with the system in persistent storage.	D	Logger

[Note: Incomplete,e.g., 'Timer' missing ...]

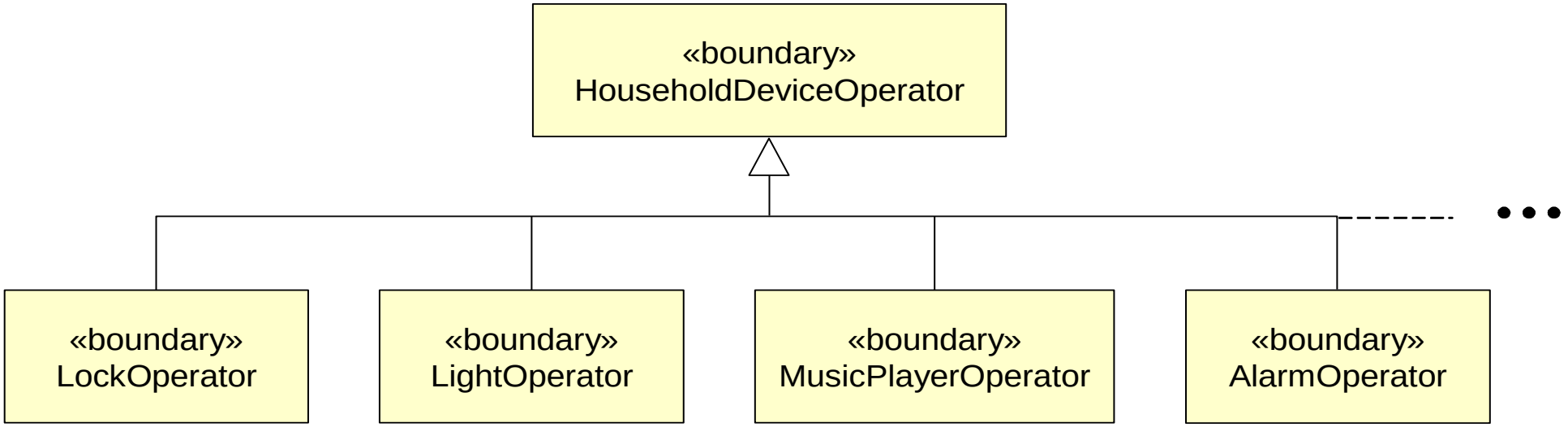
Domain Model (1a)



Domain concepts for subsystem #1 of safe home access

[Note: Incomplete, e.g., Logger, Timer missing ...]

Domain Model (1b)

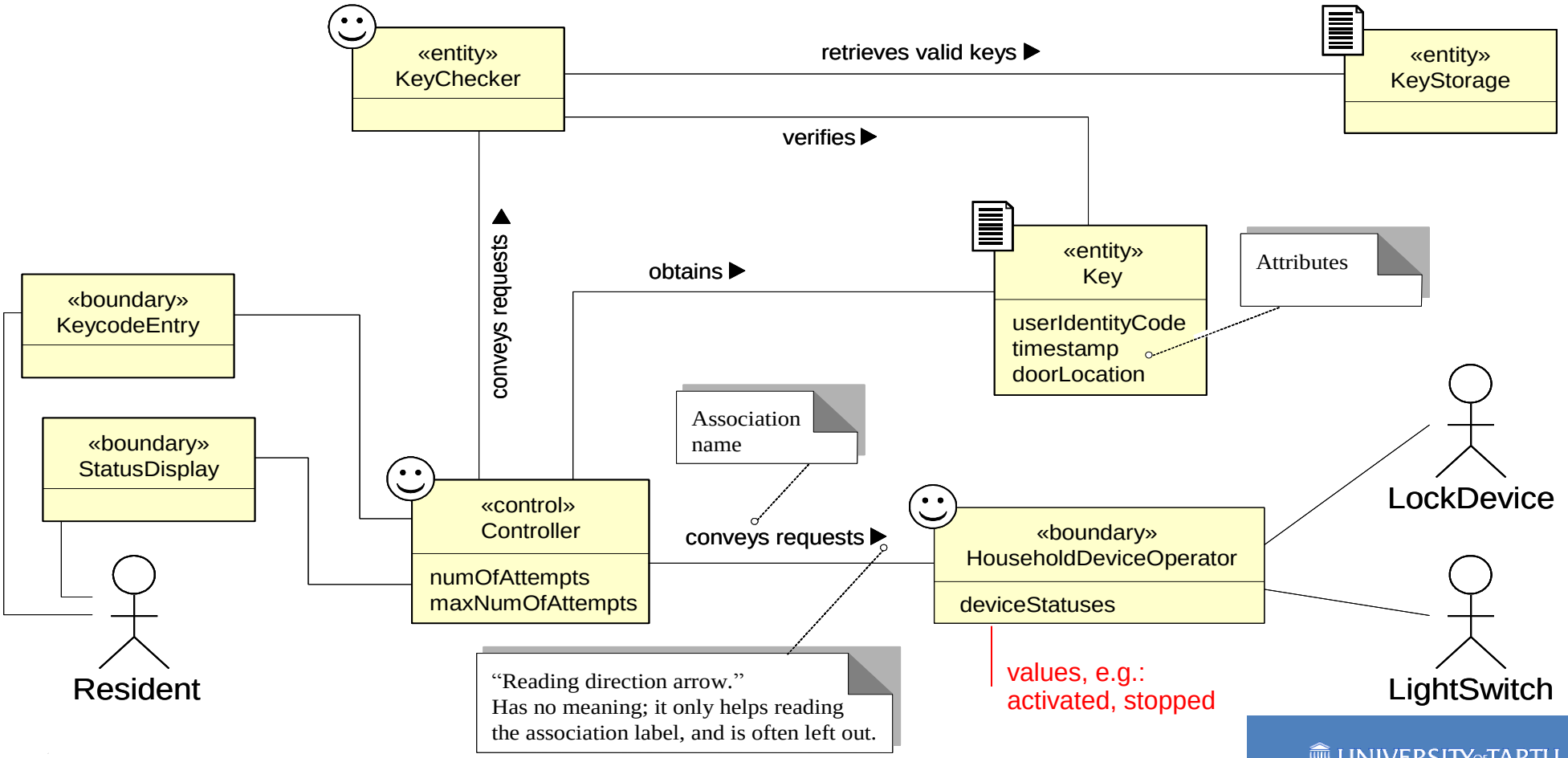


Domain Model (2a)

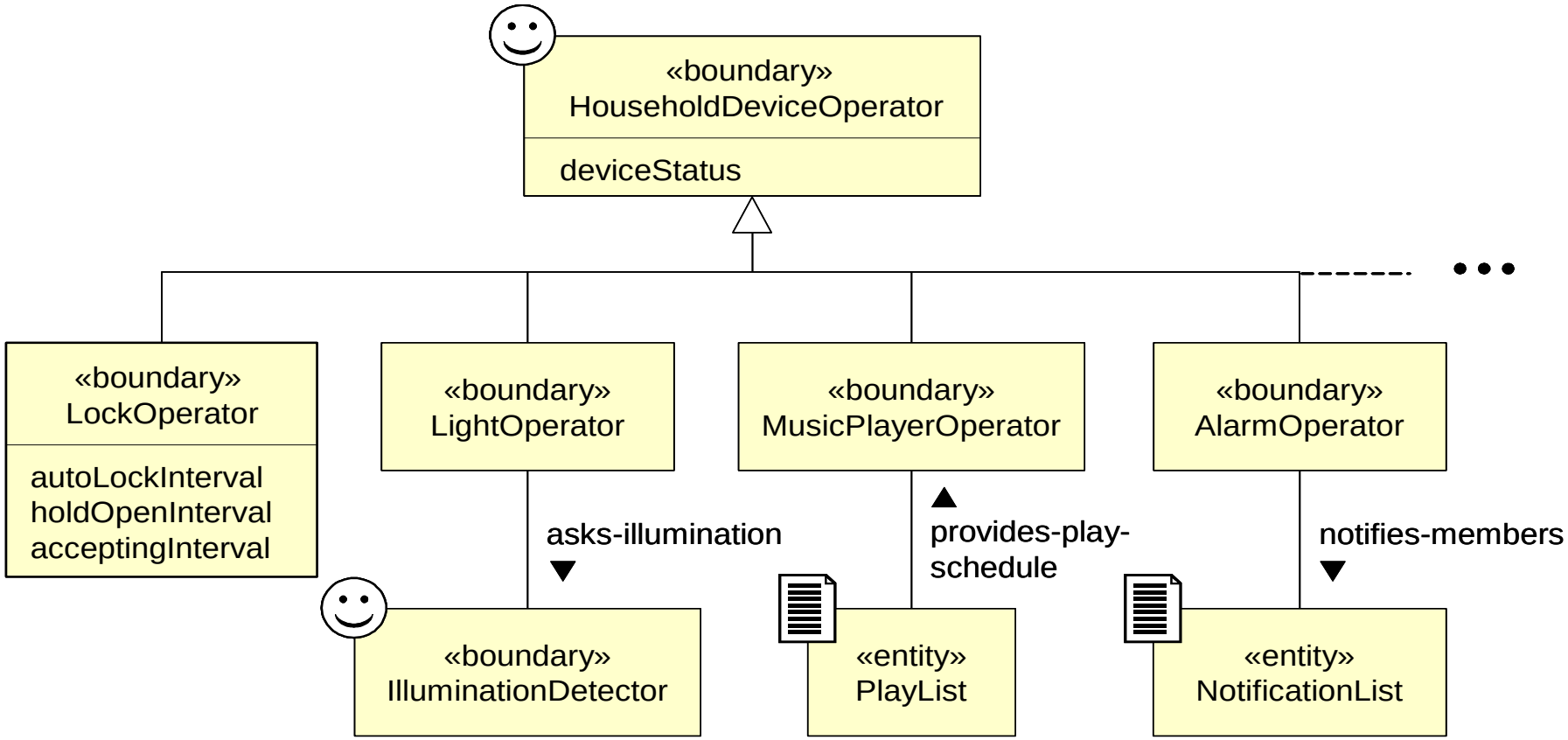
Domain model for
UC-1: Unlock

Associations: who needs to work together, *not how* they work together

Concept pair | Association description | Association name



Domain Model (2b)



Recommendations

- It is more important to identify the domain concepts than their associations and attributes
  - Every concept that the designer can discover, should be mentioned.
- Conversely, for an association (or attribute), in order to be shown it should pass the “does it need to be mentioned?” test.
  - If the association in question is obvious, it should be omitted from the domain model.
  - For example, the association <<Controller–obtains–Key>> is fairly redundant.
  - Several other associations could as well be omitted, because the reader can easily infer them, and this should be done particularly in schematics that are about to become cluttered.
- Remember, clarity should be preferred to accurateness, and, if the designer is not sure, some of these can be mentioned in the text accompanying the schematic, rather than drawn in the schematic itself.

Use Case 5: Inspect Access History

REQ8	**The system should allow searching the history log by specifying one or more of these parameters: the time frame, the actor role, the door location, or the event type (unlock, lock, power failure, etc.). This function shall be available over the Web by pointing a browser to a specified URL.**
REQ9	**The system should allow filing inquiries about "suspicious" accesses. This function shall be available over the Web.**

Use Case UC-5:	**Inspect Access History**
Related Requirements:	REQ8 and REQ9 stated in Table 2-1
Initiating Actor:	Any of: Tenant, Landlord
Actor's Goal:	To examine the access history for a particular door.
Participating Actors:	Database, Landlord
Preconditions:	Tenant/Landlord is currently logged in the system and is shown a hyperlink "View Access History."
Postconditions:	None.

Flow of Events for Main Success Scenario:

→	1.	Tenant/Landlord clicks the hyperlink "View Access History"
←	2.	System prompts for the search criteria (e.g., time frame, door location, actor role, event type, etc.) or "Show all"
→	3.	Tenant/Landlord specifies the search criteria and submits
←	4.	System prepares a database query that best matches the actor's search criteria and retrieves the records from the Database
→	5.	Database returns the matching records
↵ ←	6.	System (a) additionally filters the retrieved records to match the actor's search criteria; (b) renders the remaining records for display; and (c) shows the result for Tenant/Landlord's consideration
→	7.	Tenant/Landlord browses, selects "interesting" records (if any), and requests further investigation (with an accompanying complaint description)
↵ ←	8.	System (a) displays only the selected records and confirms the request; (b) archives the request in the Database and assigns it a tracking number; (c) notifies Landlord about the request; and (d) informs Tenant/Landlord about the tracking number

Extracting the Responsibilities

Responsibility Description	Type	Concept Name
Rs1. Coordinate actions of concepts associated with this use case and delegate the work to other concepts.	D	Controller
Rs2. Form specifying the search parameters for database log retrieval (from UC-5, Step 2).	K	Search Request
Rs3. Render the retrieved records into an HTML document for sending to actor's Web browser for display.	D	Page Maker
Rs4. HTML document that shows the actor the current context, what actions can be done, and outcomes of the previous actions.	K	Interface Page
Rs5. Prepare a database query that best matches the actor's search criteria and retrieve the records from the database (from UC-5, Step 4).	D	Database Connection
Rs6. Filter the retrieved records to match the actor's search criteria (from UC-5, Step 6).	D	Postprocessor
Rs7. List of "interesting" records for further investigation, complaint description, and the tracking number.	K	Investigation Request
Rs8. Archive the request in the database and assign it a tracking number (from UC-5, Step 8).	D	Archiver
Rs9. Notify Landlord about the request (from UC-5, Step 8).	D	Notifier

Extracting the Associations

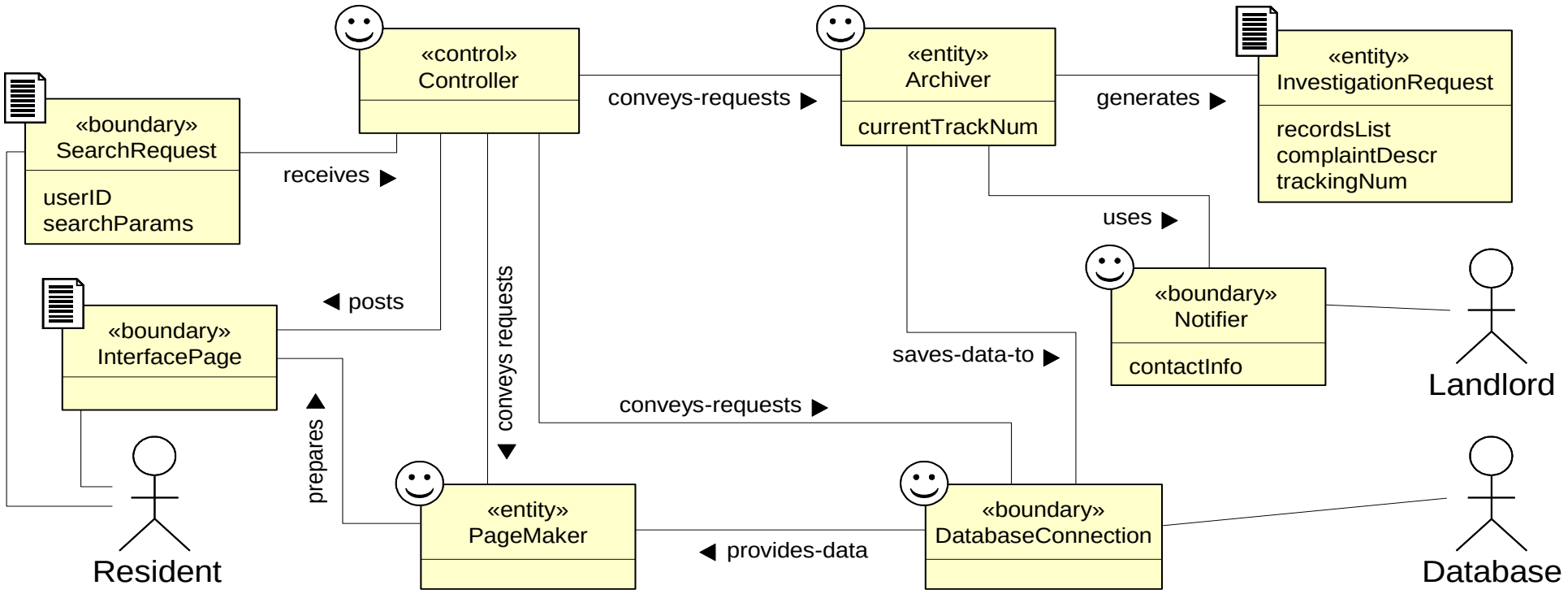
Concept pair	Association description	Association name
Controller ↔ Page Maker	Controller passes requests to Page Maker and receives back pages prepared for displaying	conveys requests
Page Maker ↔ Database Connection	Database Connection passes the retrieved data to Page Maker to render them for display	provides data
Page Maker ↔ Interface Page	Page Maker prepares the Interface Page	prepares
Controller ↔ Database Connection	Controller passes search requests to Database Connection	conveys requests
Controller ↔ Archiver	Controller passes a list of “interesting” records and complaint description to Archiver, which assigns the tracking number and creates Investigation Request	conveys requests
Archiver ↔ Investigation Request	Archiver generates Investigation Request	generates
Archiver ↔ Database Connection	Archiver requests Database Connection to store investigation requests into the database	requests save
Archiver ↔ Notifier	Archiver requests Notifier to notify Landlord about investigation requests	requests notify

Extracting the Attributes

Concept	Attributes	Attribute Description
Search Request	user's identity	Used to determine the actor's credentials, which in turn specify what kind of data this actor is authorized to view.
	search parameters	Time frame, actor role, door location, event type (unlock, lock, power failure, etc.).
Postprocessor	search parameters	Copied from search request; needed to Filter the retrieved records to match the actor's search criteria.
Investigation Request	records list	List of "interesting" records selected for further investigation.
	complaint description	Describes the actor's suspicions about the selected access records.
	tracking number	Allows tracking of the investigation status.
Archiver	current tracking number	Needed to assign a tracking number to complaints and requests.
Notifier	contact information	Contact information of the Landlord who accepts complaints and requests for further investigation.

Domain Model (3)

Domain model (sketch)
for UC-5: Inspect Access History



Traceability Matrix (1)

Mapping: System requirements to Use cases

REQ1: Keep door locked and auto-lock
REQ2: Lock when “LOCK” pressed
REQ3: Unlock when valid key provided
REQ4: Allow mistakes but prevent dictionary attacks
REQ5: Maintain a history log
REQ6: Adding/removing users at runtime
REQ7: Configuring the device activation preferences
REQ8: Inspecting the access history
REQ9: Filing inquiries

UC1: Unlock
UC2: Lock
UC3: AddUser
UC4: RemoveUser
UC5: InspectAccessHistory
UC6: SetDevicePrefs
UC7: AuthenticateUser
UC8: Login

Req’t	PW	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8
REQ1	5	X	X						
REQ2	2		X						
REQ3	5	X						X	
REQ4	4	X						X	
REQ5	2	X	X						
REQ6	1			X	X				X
REQ7	2						X		X
REQ8	1					X			X
REQ9	1					X			X
Max PW		5	5	1	1	1	2	5	2
Total PW		18	9	1	1	2	2	9	5

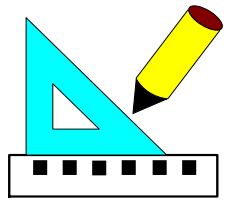
Traceability Matrix (2)

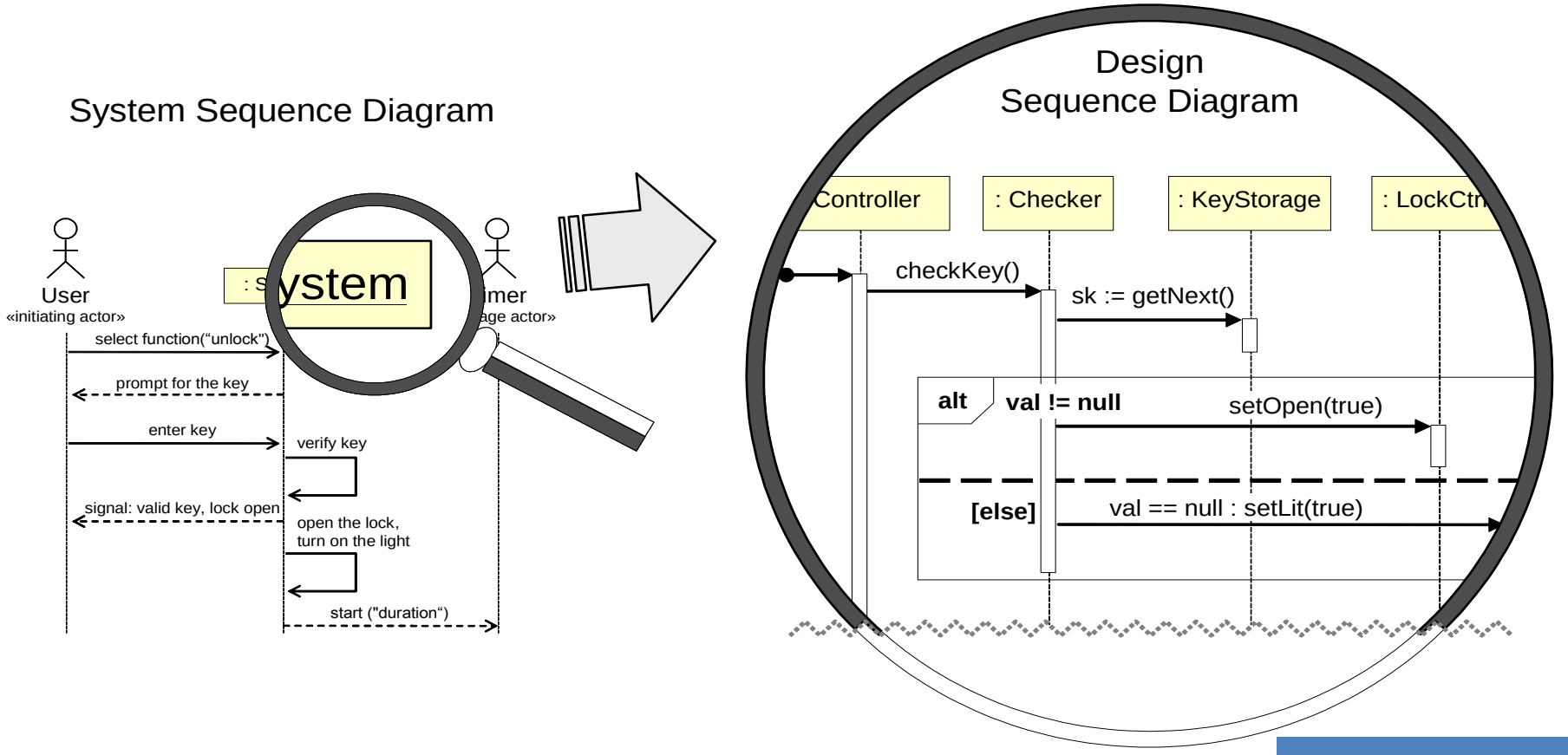
Mapping: Use cases to Domain model

UC1: Unlock
UC2: Lock
UC3: AddUser
UC4: RemoveUser
UC5: InspectAccessHistory
UC6: SetDevicePrefs
UC7: AuthenticateUser
UC8: Login

		Domain Concepts														
Use Case	**PW**	Controller-SS1	StatusDisplay	KeycodeEntry	Key	KeyStorage	KeyChecker	HouseholdDeviceOperator	Controller-SS2	SearchRequest	InterfacePage	PageMaker	Archiver	DatabaseConnection	Notifier	InvestigationRequest
UC1	18	X	X	X	X			X								
UC2	9	X	X					X								
UC3	1								X		X	X		X		
UC4	1								X		X	X		X		
UC5	2								X	X	X	X	X	X	X	X
UC6	2								X		X	X		X		
UC7	9	X	X		X	X	X									
UC8	5								X		X	X		X		

Design: Object Interactions





System Function 'Enter Key'

Much more detail added in each iteration

Alternative solutions are possible and must be discussed

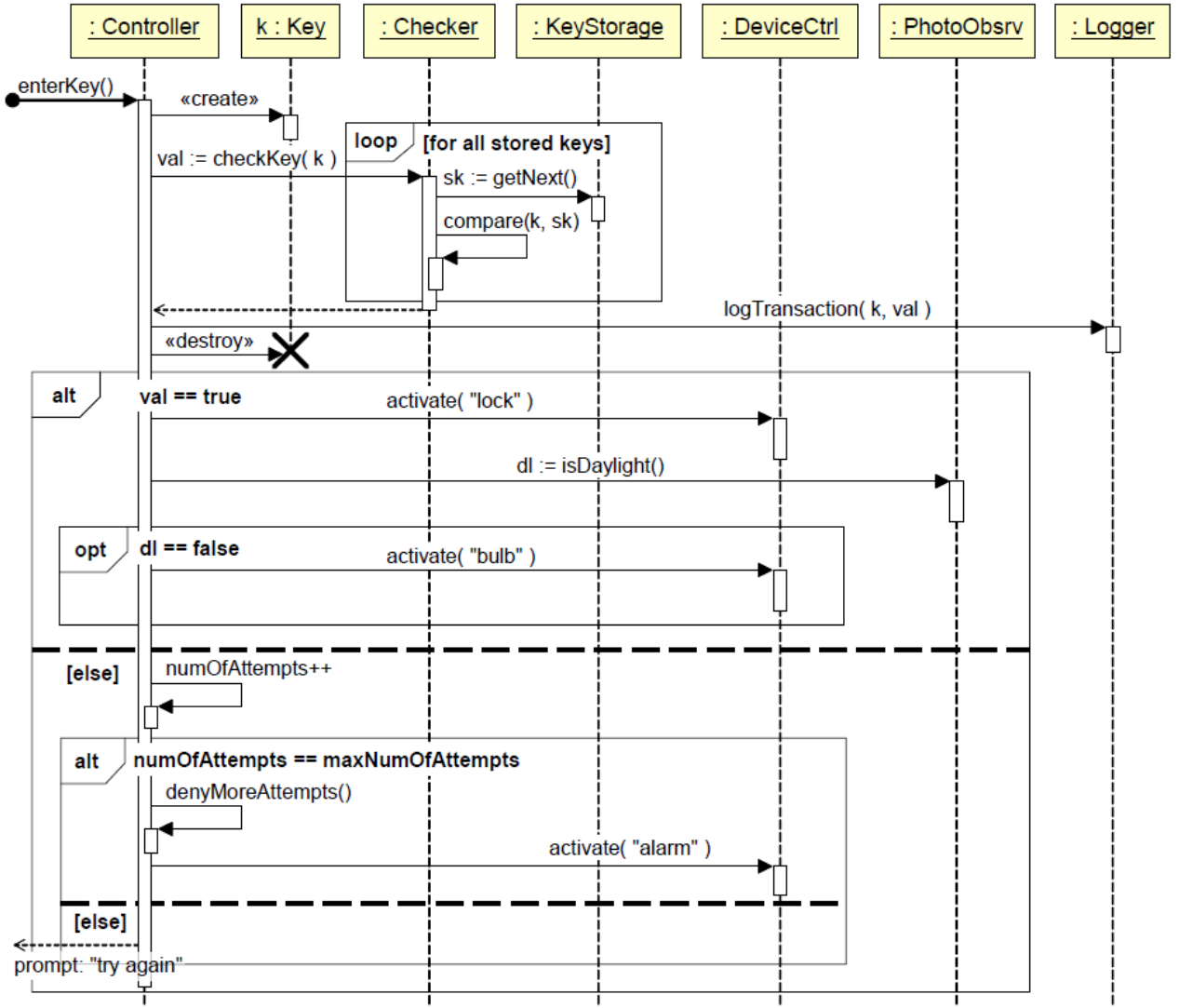


Figure 2-33: Sequence diagram for the system function "enter key" (Figure 2-20). Several UML interaction frames are shown, such as "loop," "alt" (alternative fragments, of which only the one with a condition true will execute), and "opt" (optional, the fragment executes if the condition is true).

Structure of Lecture 04

- Preliminaries
  - Use Cases
  - UML Notations: UC Diagram, Class Diagram, Sequence Chart, State Chart
- Domain Analysis and Modelling Example
  - Identifying Concepts (Responsibilities)
  - Concept Attributes
  - Concept Associations
- Domain Analysis and Modelling Summary

Key Points

- A domain model is an abstract view of a system and its context that ignores system details.
  - Complementary system models can be developed to show the system's context, interactions, structure and behaviour.
- Use case diagrams and sequence diagrams are used to describe the interactions between users and systems in the system being designed.
  - Use cases describe interactions between a system and external actors;
  - Sequence diagrams add more information to these by showing interactions between system objects.
- Structural models show the organization and architecture of a system.
  - Class diagrams are used to define the static structure of classes in a system and their associations.

Key Points

- Behavioral models help describe the dynamic behavior of a system.
  - This behavior can be modeled from the perspective of the data processed by the system, or by the events that stimulate responses from a system.
  - State diagrams are used to model a system's behavior in response to internal or external events.
  - Activity diagrams may be used to model the processing of data, where each activity represents one process step. (NOT COVERED IN LECTURE!)
- More info on UML: http://en.wikipedia.org/wiki/Unified_Modeling_Language
- Model-driven engineering (MDE) is an approach to software development in which a system is represented as a set of models that can be automatically transformed to executable code.
  - Executable UML (xUML)

Next Lecture

- Date/Time:
  - Friday, 04-Oct, 10:15-12:00
- Topic:
  - Development Infrastructure I (by Ivo Mägi)
- For you to do:
  - Finalise and submit Lab task 2 solutions
  - Remember that all team members must be present in next week's lab sessions (assessment of Lab task 1 solutions)!