Software Engineering

LECTURE 8: Domain Analysis and Modeling

Textbook:Software Engineering Ivan
Marsic

Topics

- ❑ Domain Analysis & Modeling Tools, from Use Cases to Domain Model
  - – Identifying Concepts
  - – Concept Attributes
  - – Concept Associations
  - – Contracts: Preconditions and Postconditions
- ❑ Domain Modeling Beyond Use Cases

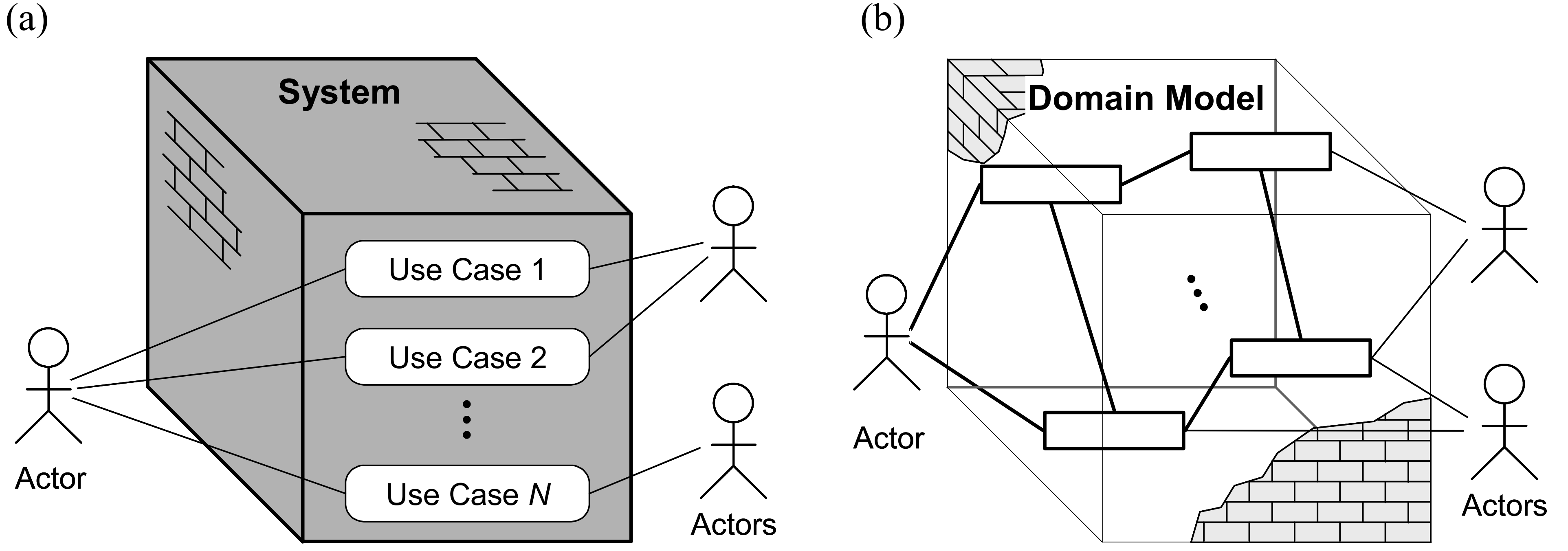
Domain Analysis & Modeling

- ❑ Domain analysis and modeling identifies the system elements needed to solve the problem (i.e., meet the requirements)
- ❑ Why? —The goal of domain modeling is to understand how system-to-be will work
  - – Requirements analysis determined how users will interact with system-to-be (external behavior)
  - – Domain modeling determines how elements of system-to-be interact (internal behavior) to produce the external behavior
- ❑ How? —We do domain modeling based on sources:
  - – Knowledge of how system-to-be is supposed to behave (from requirements analysis, e.g., use cases)
  - – Studying the work domain (or, problem domain)
  - – Knowledge base of software designs
  - – Developer’s past experience with software design

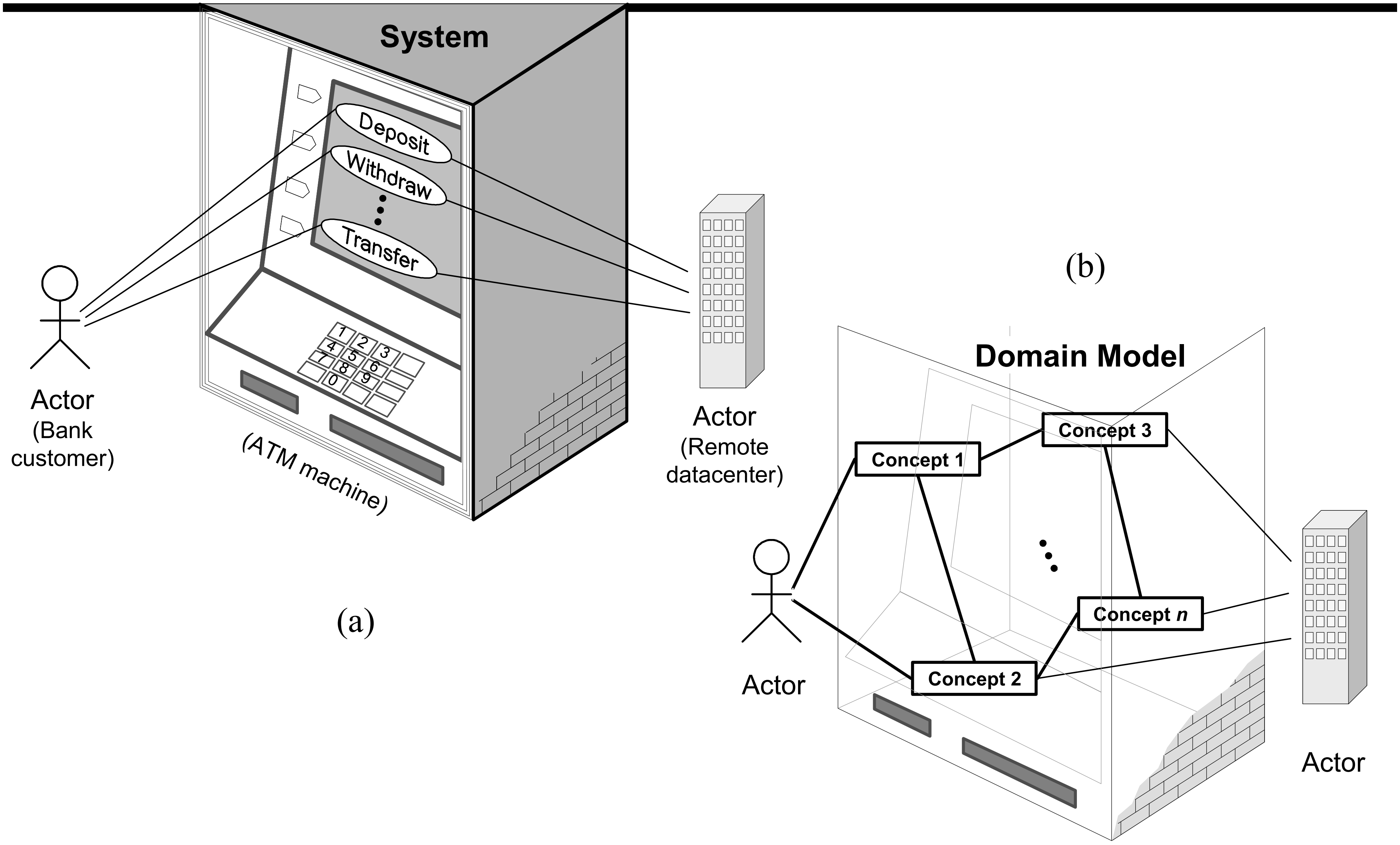
Use Cases vs. Domain Model

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

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

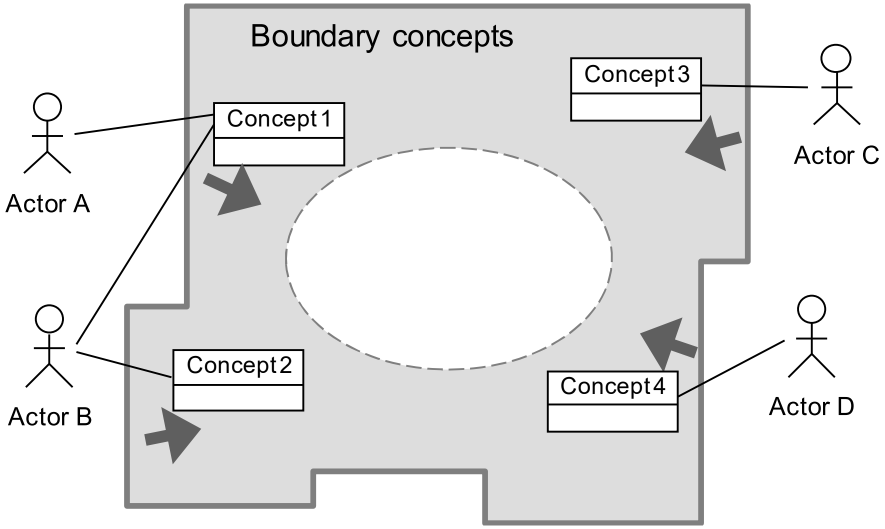


What is Domain Modeling
Example: ATM Machine

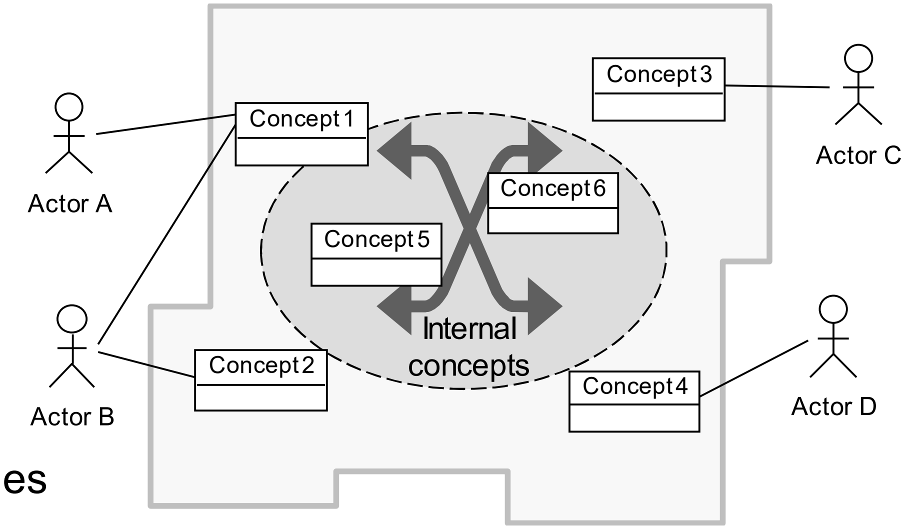


Building Domain Model from Use Cases

Step 1: Identifying the boundary concepts



Step 2: Identifying the internal concepts



Step 2: Identifying the internal concepts

Internal concepts “route” data between boundaries

- Data format conversion
- Data protection policies

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.

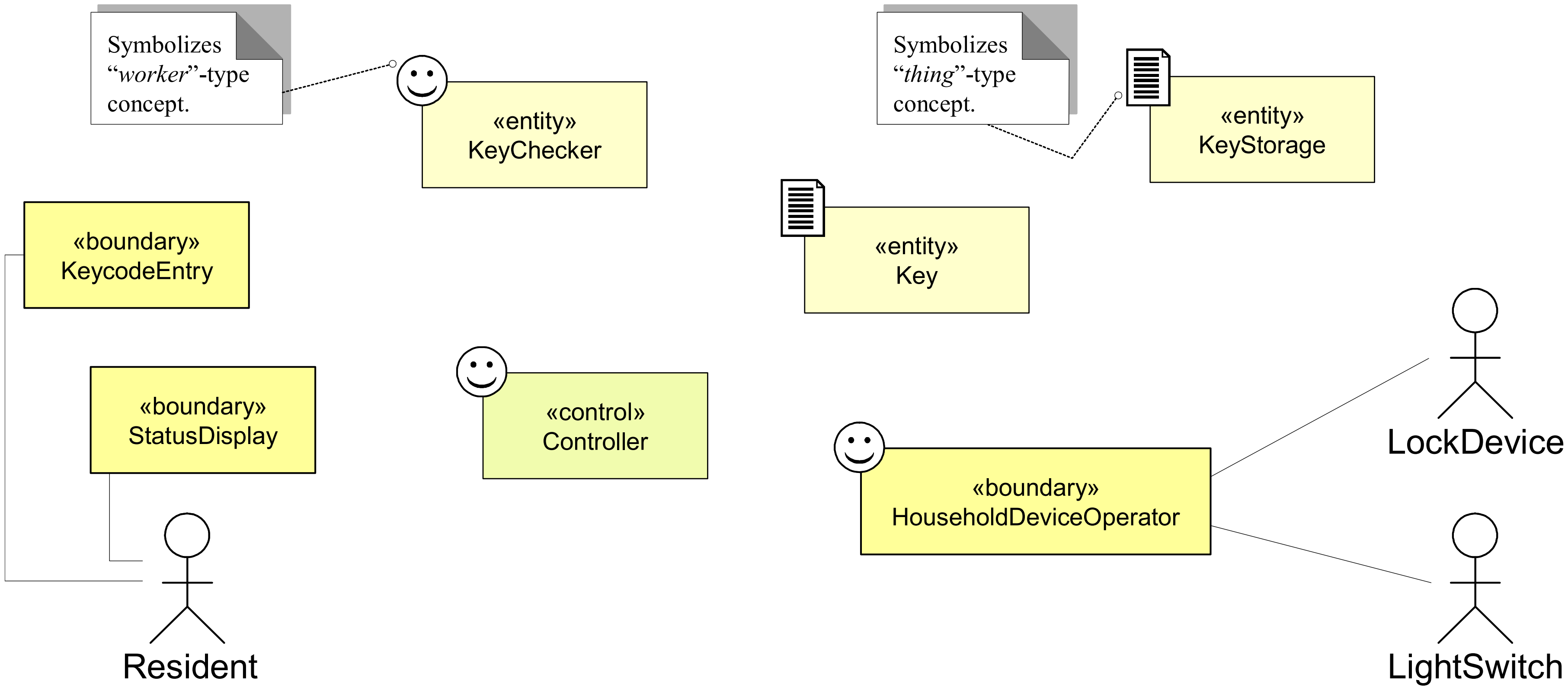
Flow of Events for Main Success Scenario:

→	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

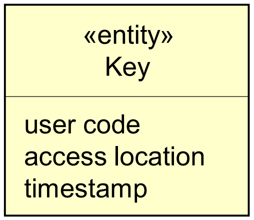
Responsibility Description	Type	Concept Name
Coordinate actions of all concepts associated with a use case, a logical grouping of use cases, or the entire system and delegate the work to other concepts.	D	Controller
Container for user's authentication data, such as pass-code, timestamp, door identification, 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

Domain Model (1)



Domain concepts for subsystem #1 of safe home access

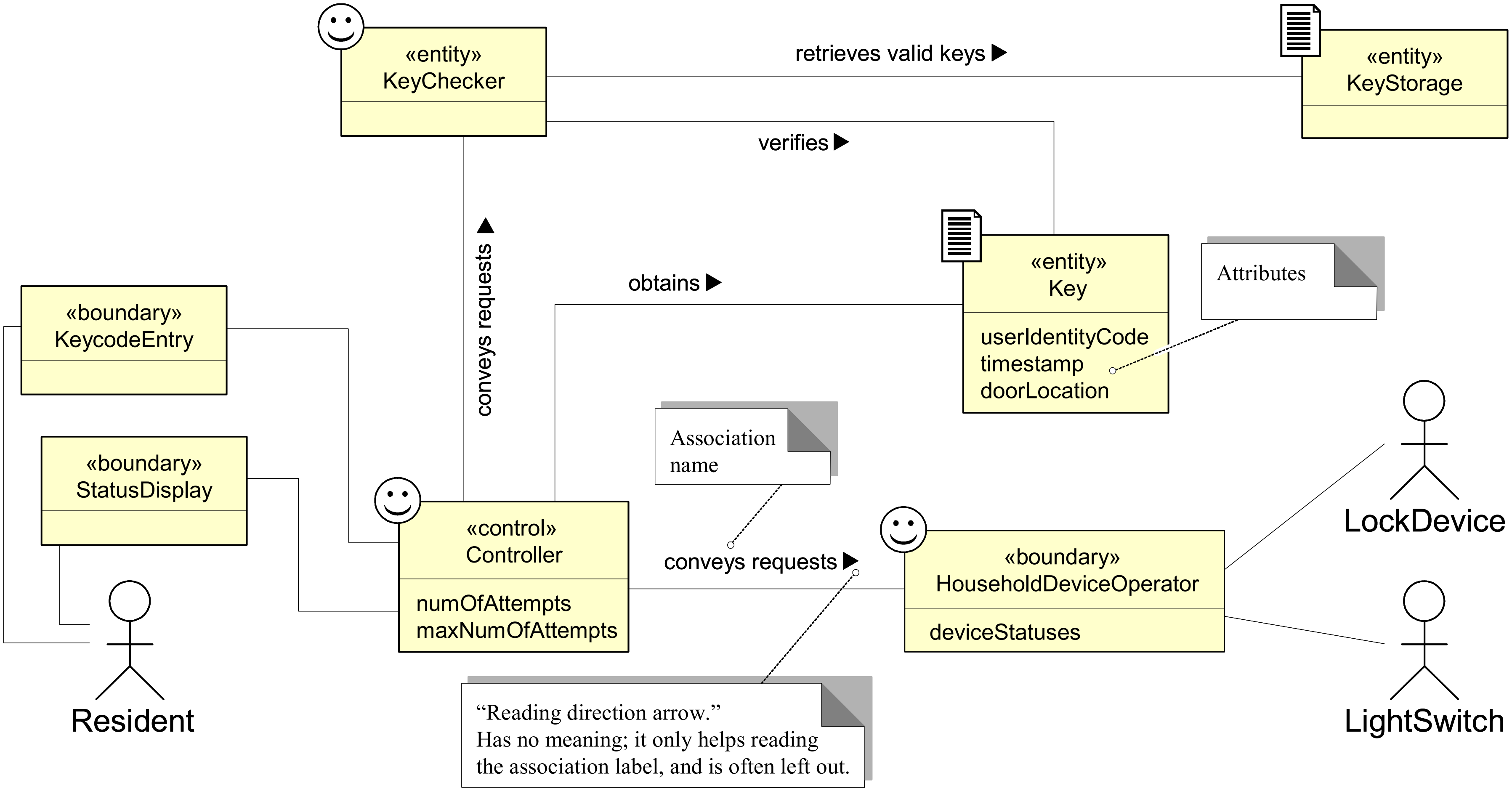
Domain Model (2)



- Domain model for UC-1: Unlock

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

Concept pair | Association description | Association name



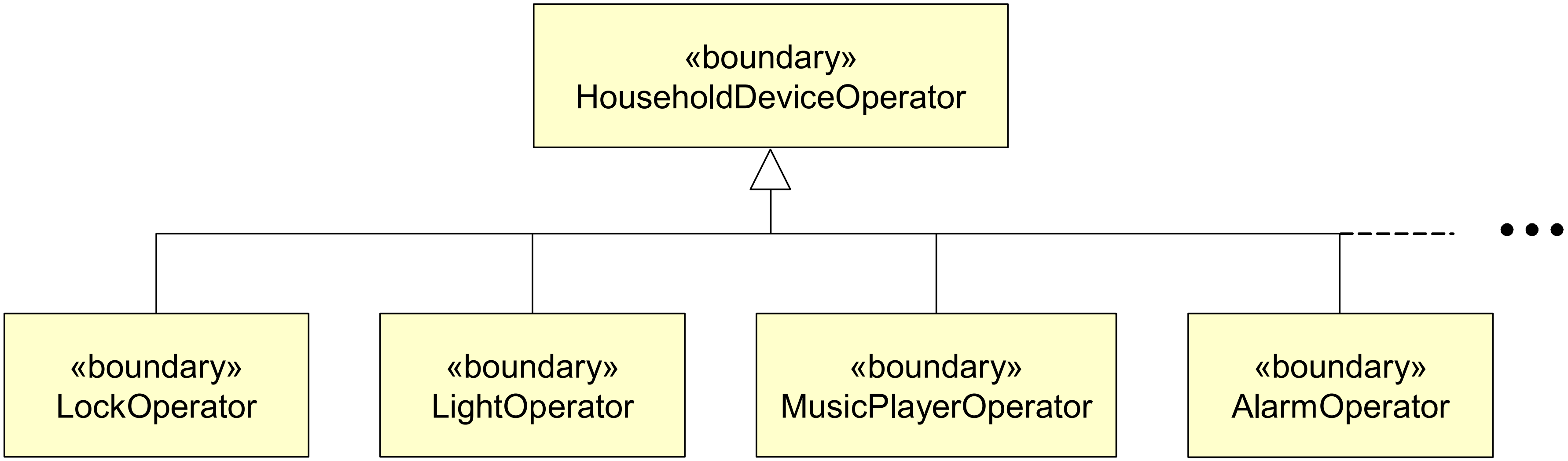
Attributes

Association name

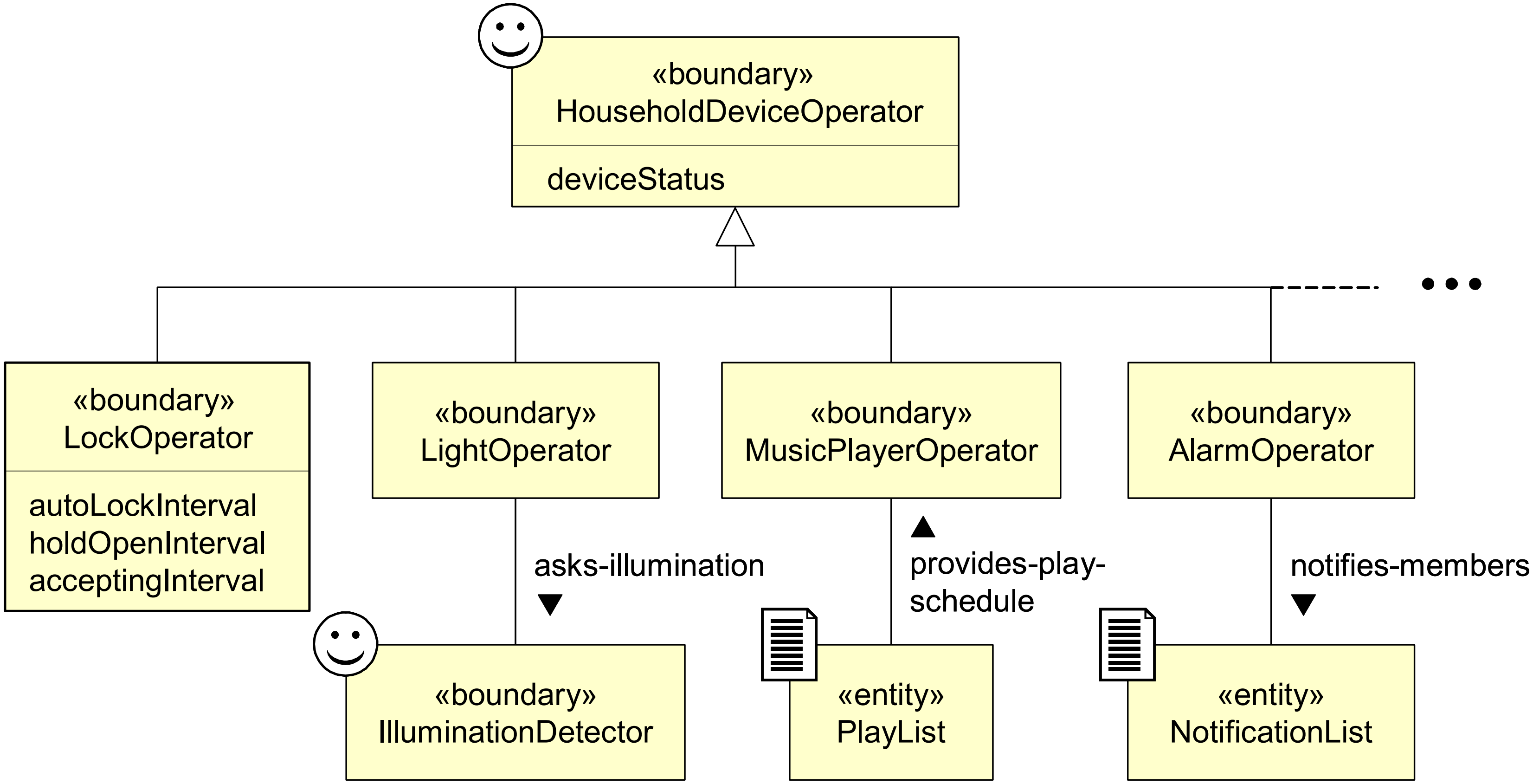
"Reading direction arrow."
Has no meaning; it only helps reading
the association label, and is often left out.

Degrees of Domain Model Refinement

- ❑ Simplest case: all household devices are conceptually the same—just an on/off switch to activate or deactivate
- ❑ If each device will provide additional functionality, use different conceptual objects
- ❑ The correct approach depends on the requirements



Domain Model (3)



Use Case 5: Inspect Access History

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

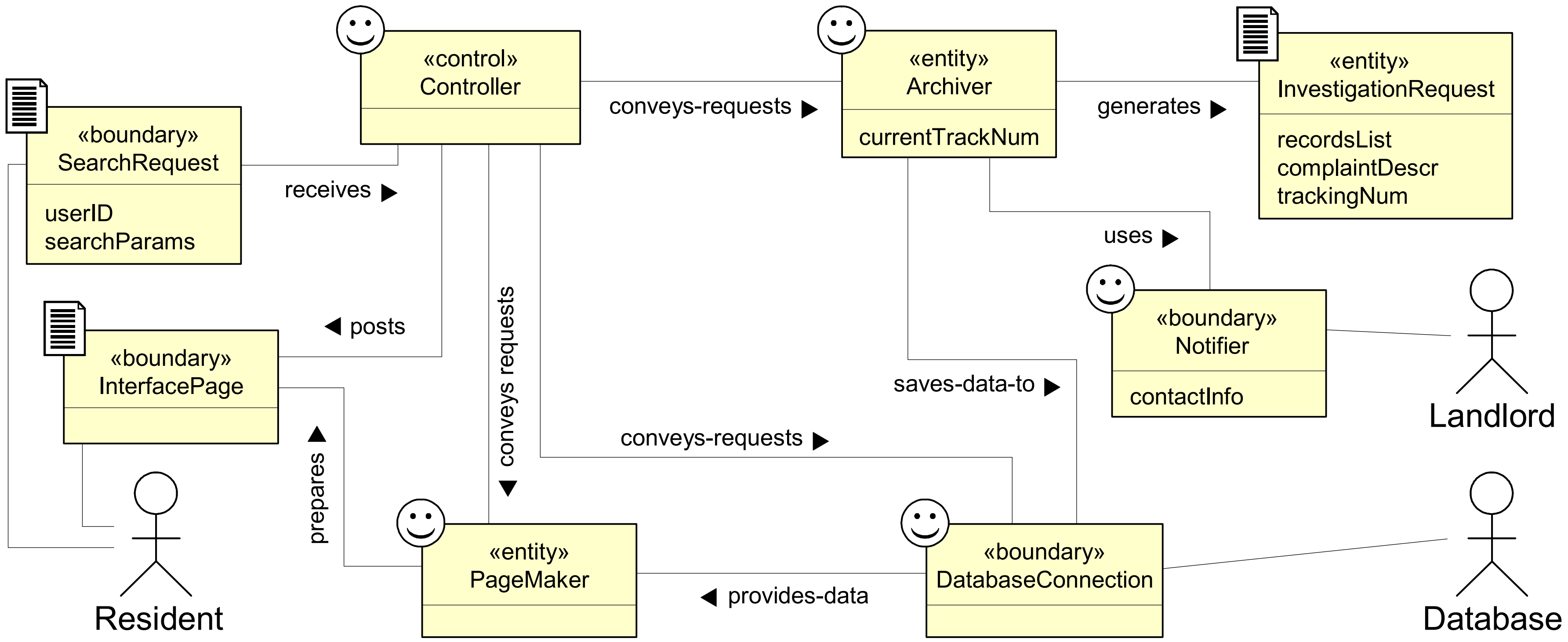
Concept pair	Association description	Association name
Controller ↔ Page Maker	Controller passes requests to Page Maker and receives back pages prepared for displaying	conveys requests
Page Maker ↔ Database Connection	Database Connection passes the retrieved data to Page Maker to render them for display	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 (4)

Domain model
for UC-5: Inspect Access History



Traceability Matrix (1)

Mapping: System requirements to Use cases

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

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

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	2	2	2	1	5	2	1
Total PW		15	3	2	2	3	9	2	3

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	15	X	X	X	X			X								
UC2	3	X	X					X								
UC3	2								X		X	X		X		
UC4	2								X		X	X		X		
UC5	3								X	X	X	X	X	X	X	X
UC6	9								X		X	X		X		
UC7	2	X	X		X	X	X									
UC8	3								X		X	X		X		

Contracts: Preconditions and Postconditions

Operation	Unlock
Preconditions	• set of valid keys known to the system is not empty
	• numOfAttempts ≤ maxNumOfAttempts
	• numOfAttempts = 0, for the first attempt of the current user
Postconditions	• numOfAttempts = 0, if the entered Key ∈ Valid keys
	• current instance of the Key object is archived and destroyed

Operation	Lock
Preconditions	None (that is, none worth mentioning)
Postconditions	• lockStatus = “armed”, and
	• lightStatus remains unchanged (see text for discussion)

Typical Problems with Domain Models

- Unaware that requirements are **not simply a wish list**
  - Ignoring real-world constraints and problems
    - Physical I/O devices, networks, sensors, etc., are failure prone
    - Economic, legal, cultural, etc., constraints
  - Results in one requirement (or even use case!) being mapped to one concept/module that acts as
    a trivial input-to-output “connector”
- Omitting *input data* for modules (if any) and *output data* for modules (if any)

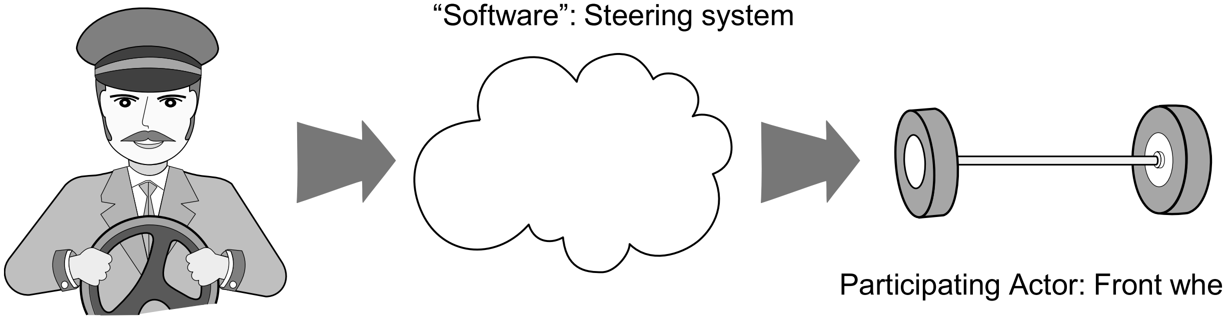
  “things” for “worker” concepts
- Unaware of *dependencies* between requirements (or use cases)
- Unaware of *incompatible data* across concepts/modules
  - Different concepts/modules may receive or output different data formats

Why Domain Modeling?

- To achieve *N* different things, we need *N* different tasks
  - Formulated by W.R. Ashby as the *law of requisite variety*
    - https://en.wikipedia.org/wiki/Variety_(cybernetics)
    - https://en.wikipedia.org/wiki/Good_regulator
  - It’s basically like ensuring that every row in the **traceability matrix** crosses at least one column (assuming the complete requirements)!
- The problem for the beginners is that they do not know *what* needs to be achieved
  - Example: car steering problem—the beginner is not aware of differences between steering at low and high vehicle speeds
- Experienced developers will at least know or guess some things that are common to many problems
  - such as: generic issues for networks or I/O devices
- but the only way to know what is needed is to study the problem domain and get help from domain experts

Example: Car Steering System

<u>Requirement</u>: As a driver, I will be able to steer the car left or right to follow the road.

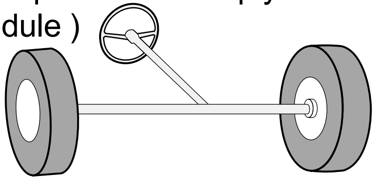


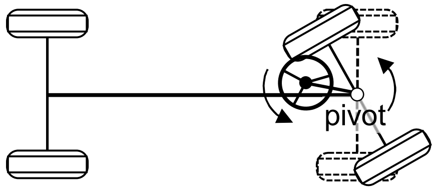
"Software": Steering system

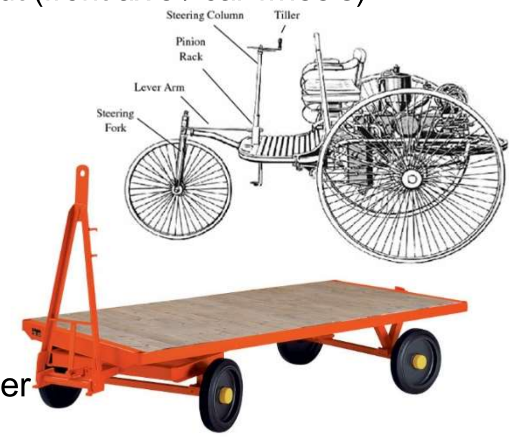
Participating Actor: Front wheels

Initiating Actor: Driver + Steering wheel

<u>Solution #1</u> is derived from the requirement: Simply connect the input (steering wheel) to the output (front axle / car wheels)
(1 requirement ➔ 1 concept/module)







Works!**?** (called "turntable steering," a design in which a rigid axle is turned around its center and both front wheels turn around a common pivot)

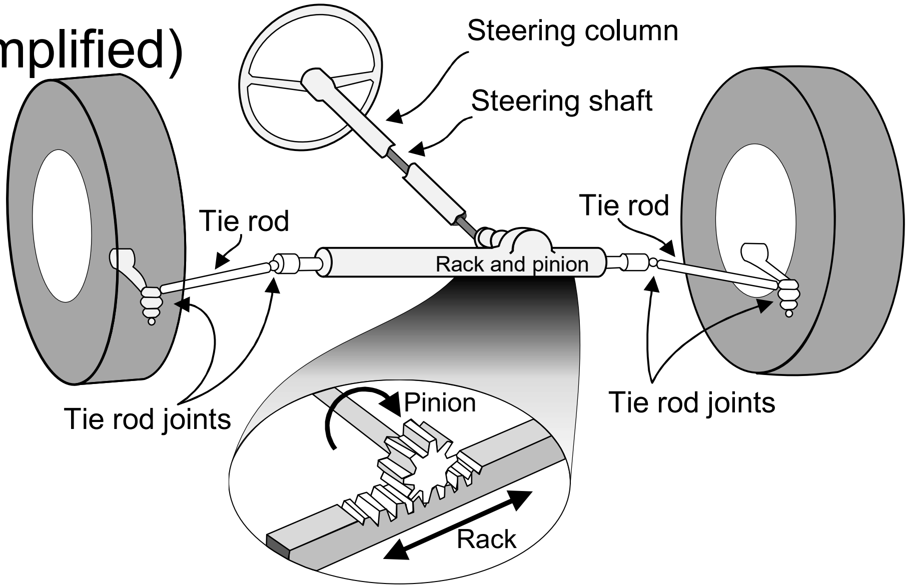
Problem: At higher vehicle speeds, wheels on the inside and outside of a turn need to trace out circles of different radii.

Example: Car Steering System

Problem: wheels on the inside and outside of a turn need to trace out circles of different radii ➔
➔ Incompatible components—steering wheel moves *rotationally*, wheels need to turn *linearly* in lockstep.

“Ackermann steering” (simplified)

(Note how many new “concepts” we got!)



Agile approach:

But what better way to figure out that a solution is wrong than by trying to implement it!

Yes, if one has good acceptance tests …
… which are hard to create without a systematic and thorough domain analysis.

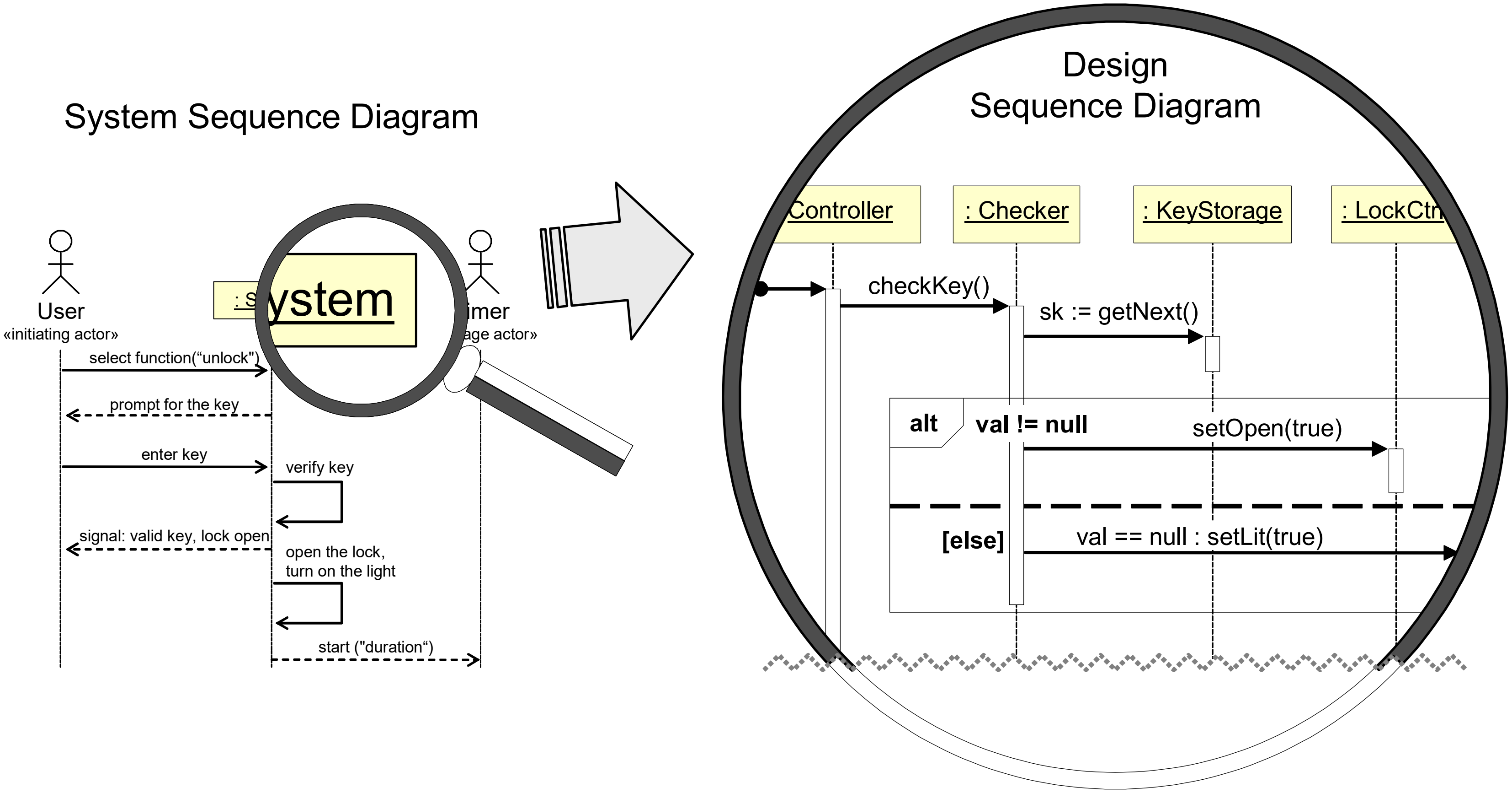
It gets more complicated— Another problem:
At high vehicle speeds, a tire needs a slip angle to transfer the lateral forces…

Domain Modeling: Looking from Inside Out

- ❑ The developer should *not* engage in unconstrained construction of models of real world
- ❑ Instead, identify only the concepts relevant for the problem at hand
  - – Looking from inside out: What the computer needs to know about the world to solve the current problem
- ❑ The resulting model should be as parsimonious as possible

Next Lecture:

Design of Object Interactions


System Sequence Diagram
User
«initiating actor»
: System
Timer
«offstage actor»
select function(“unlock")
prompt for the key
enter key
verify key
signal: valid key, lock open
open the lock,
turn on the light
start ("duration“)
Design
Sequence Diagram
: Controller
: Checker
: KeyStorage
: LockCtrl
checkKey()
sk := getNext()
alt
val != null
setOpen(true)
[else]
val == null : setLit(true)