Software Engineering

TOPIC: Use Case Analysis

Topics

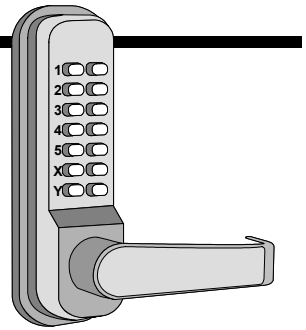
- Actors, Goals
- Sketchy/Summary Use Cases
- Use Case Diagram
- Traceability Matrix
- System Boundary and Subsystems
- Detailed Use Case Specification
- System Sequence Diagrams
- Security and Risk Management

Use Cases

- Used for Functional Requirements Analysis and Specification
- A ***use case*** is a step-by-step description of how a user will use the system-to-be to accomplish business goals
- Three key *elements* of use cases:
  1. Environmental agents or actors that will cooperate with the system
  2. Use case diagram shows relations of the actors and use cases
  3. Use case scenarios are plans for realization of user goals or handling contingencies
- Use cases *scenarios* or *scripts* show an envisioned sequence of cooperative interactions between the external actors and the system-to-be

Deriving Use Cases from System Requirements

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



Initiator	Initiator's Goal	Participants	Use Case Name
Tenant	Unlock and enter home.	Lock, Household Devices, Database	Unlock (UC-1)
Tenant	Lock the door.	Lock, Household Devices, Database	Lock (UC-2)
Landlord	Create a new user account and allow access to home.	Tenant, Database	AddUser (UC-3)
Landlord	Retire an existing user account and disable access.	Database	RetireUser (UC-4)
Tenant	Review the history of home accesses.	Database	ViewHistory (UC-5)
Tenant	Configure the operational preferences for household devices.	Database	SetDevicePrefs (UC-6)
Visitor	Visit a resident’s home.	Lock, Database	VisitHome (UC-7)

- Initiating actors (“users”) are already identified in user stories
- Participating actors identified as part of use case analysis

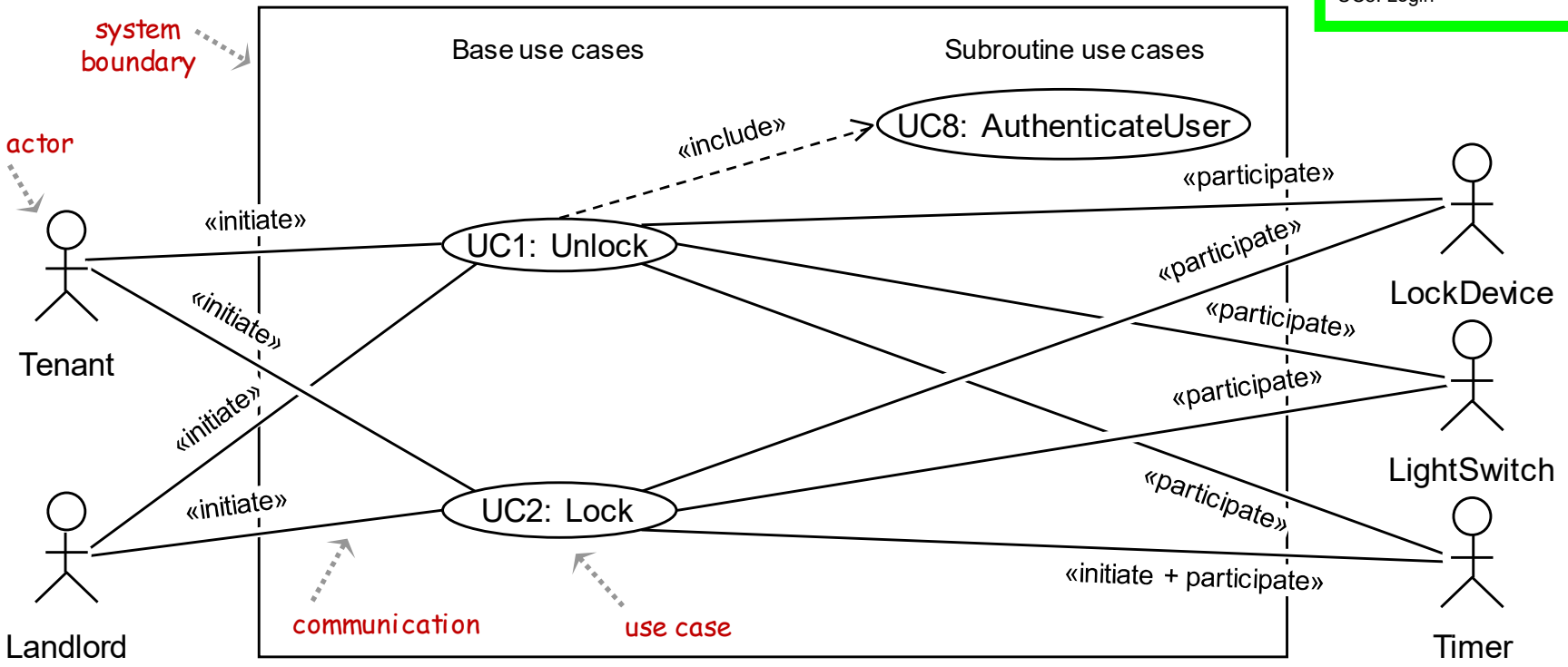
Types of Actors

- ***Initiating actor*** (also called *primary actor* or simply “user”): initiates the use case to achieve a goal
- ***Participating actor*** (also called *secondary actor*): participates in the use case but does not initiate it. Subtypes of participating actors:
  - ***Supporting actor***: helps the system-to-be to complete the use case
  - ***Offstage actor***: passively participates in the use case, i.e., neither initiates nor helps complete the use case, but may be notified about some aspect of it (e.g., for keeping records)

Use Case Diagram

High-level goal: ARRIVE

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



Interestingly, we keep mentioning “home” and “doors,” but they do not appear as actors!
This issue will be discussed in a later lecture (Domain Model)

DRY Use Cases

Remove Redundancies Using Subroutine Use Cases

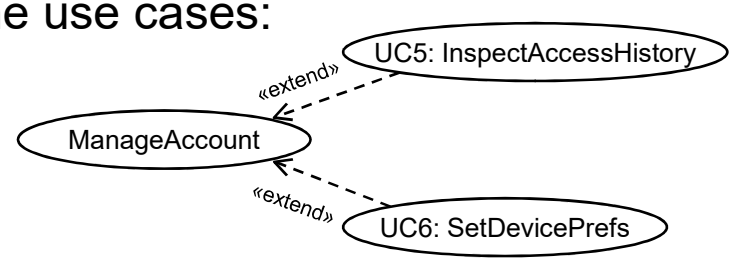
- Don't repeat yourself
  - https://en.wikipedia.org/wiki/Don%27t_repeat_yourself
- In Detailed Use cases or System Sequence Diagrams (described later):
  Remove redundant parts of base use cases that are common to multiple use cases by **parametrizing them as subroutine use cases**

Subroutine Use Cases

Optional Use Cases: «extend»

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

Example optional subroutine use cases:



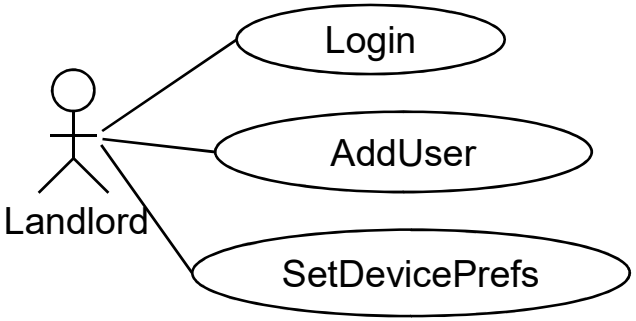
Key differences between **«include»** and **«extend»** relationships

	Included use case	Extending use case
Is this subroutine use case optional?	No	Yes
Is the base use case complete without this use case?	No	Yes
Is the execution of subroutine use case conditional?	No	Yes
Does this subroutine use case change the behavior of the base use case?	No	Yes

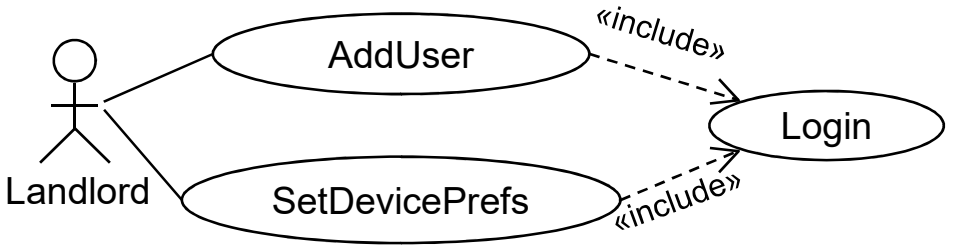
[Source: Robert Maksimchuk & Eric Naiburg: ***UML for Mere Mortals***, Addison-Wesley, 2005.]

Login Use Case?

BAD:

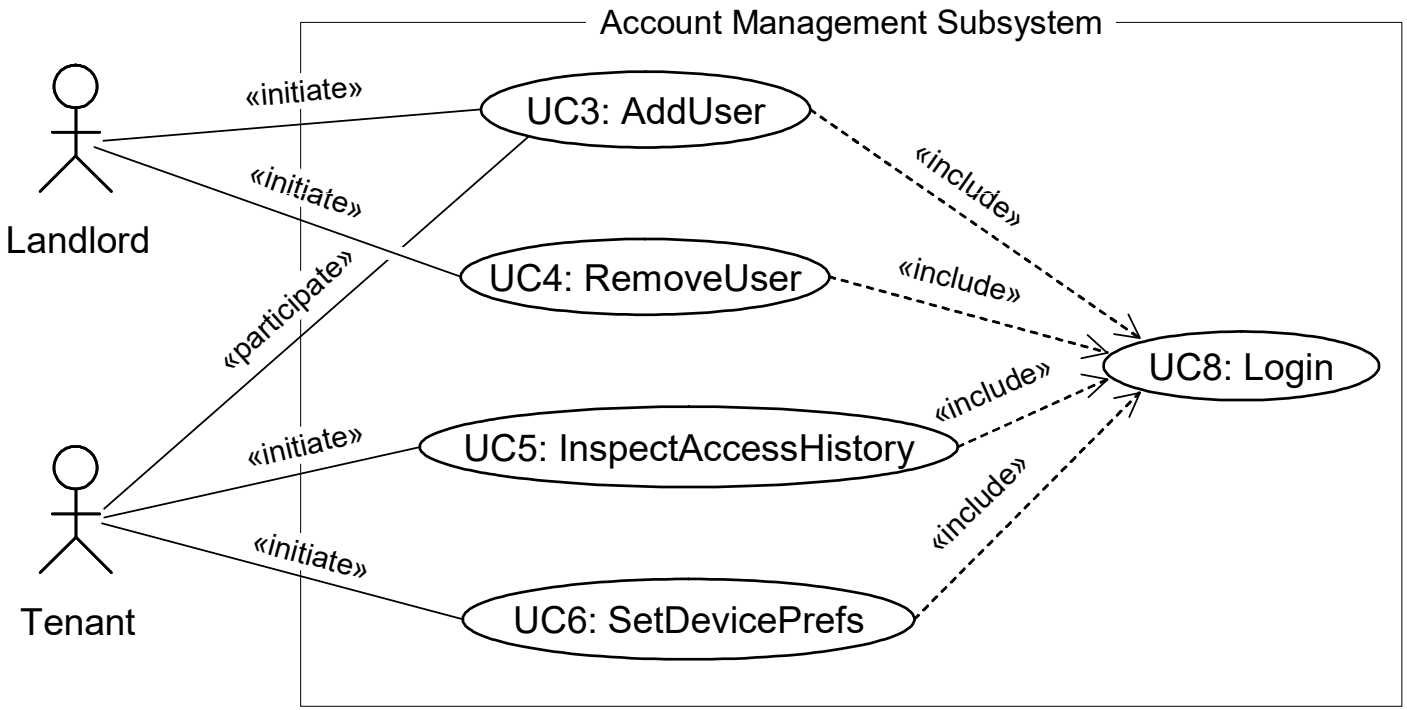


GOOD:



Use Case Diagram: Account Management

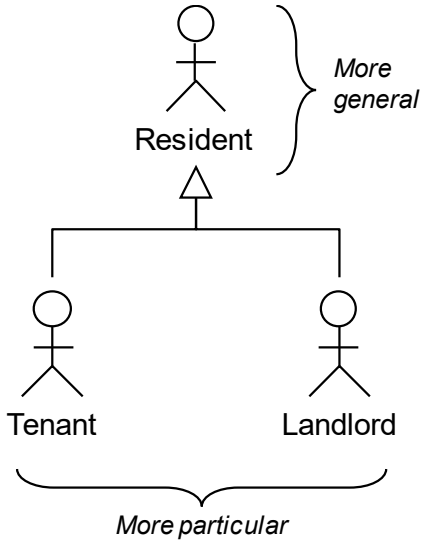
UC1: Unlock
UC2: Lock
UC3: AddUser
UC4: RemoveUser
UC5: InspectAccessHistory
UC6: SetDevicePrefs
UC7: AuthenticateUser
UC8: Login



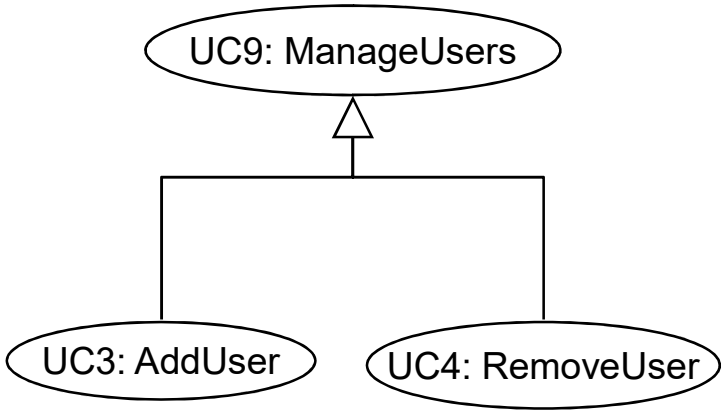
Use Case Generalizations

- More abstract representations can be derived from particular representations

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



Actor Generalization



Use Case Generalization

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

Continued for domain model, design diagrams, …

Traceability Matrix Purpose

- To check that all requirements are covered by the use cases
- To check that none of the use cases is introduced without a reason (i.e., created not in response to any requirement)
- To prioritize the work on use cases

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 1: Unlock

Use Case UC-1:	**Unlock**
Related Requirem'ts:	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]

Subroutine «include» Use Case

Use Case UC-7:	**AuthenticateUser** (sub-use case)
Related Requirements:	REQ3, REQ4 stated in Table 2-1
Initiating Actor:	Any of: Tenant, Landlord
Actor's Goal:	To be positively identified by the system (at the door interface).
Participating Actors:	AlarmBell, Police
Preconditions:	• The set of valid keys stored in the system database is non-empty. • The counter of authentication attempts equals zero.
Postconditions:	None worth mentioning.

Flow of Events for Main Success Scenario:

← 1. **System** prompts the actor for identification, e.g., alphanumeric key

→ 2. **Tenant/Landlord** supplies a valid identification key

← 3. **System** (a) verifies that the key is valid, and (b) signals to the actor the key validity

Flow of Events for Extensions (Alternate Scenarios):

2a. **Tenant/Landlord** enters an invalid identification key

← 1. **System** (a) detects error, (b) marks a failed attempt, and (c) signals to the actor

← 1a. **System** (a) detects that the count of failed attempts exceeds the maximum allowed number, (b) signals to sound **AlarmBell**, and (c) notifies the **Police** actor of a possible break-in

→ 2. **Tenant/Landlord** supplies a valid identification key

3. Same as in Step 3 above

Acceptance Test Case for UC-7 Authenticate User

Test-case Identifier:	TC-1
Use Case Tested:	UC-1, main success scenario, and UC-7
Pass/fail Criteria:	The test passes if the user enters a key that is contained in the database, with less than a maximum allowed number of unsuccessful attempts
Input Data:	Numeric keycode, door identifier

Test Procedure:	**Expected Result:**
Step 1. Type in an incorrect keycode and a valid door identifier	System beeps to indicate failure; records unsuccessful attempt in the database; prompts the user to try again
Step 2. Type in the correct keycode and door identifier	System flashes a green light to indicate success; records successful access in the database; disarms the lock device

Use Case 2: Lock

Use Case UC-2:	**Lock**
Related Requirements:	REQ1, REQ2, and REQ5 stated in Table 2-1
Initiating Actor:	Any of: Tenant, Landlord, or Timer
Actor's Goal:	To lock the door & get the lights shut automatically (?)
Participating Actors:	LockDevice, LightSwitch, Timer
Preconditions:	The system always displays the menu of available functions.
Postconditions:	The door is closed and lock armed & the auto-lock timer is reset.

Flow of Events for Main Success Scenario:

→ 1. **Tenant**/**Landlord** selects the menu item "Lock"

← 2. **System** (a) signals affirmation, e.g., "lock armed," (b) signals to **LockDevice** to arm the lock (if not already armed), (c) signal to **Timer** to reset the auto-lock counter, and (d) signals to **LightSwitch** to turn the light off (?)

Flow of Events for Extensions (Alternate Scenarios):

2a. System senses that the door is not closed, so the lock cannot be armed

← 1. **System** (a) signals a warning that the door is open, and (b) signal to **Timer** to start the alarm counter

→ 2. **Tenant**/**Landlord** closes the door

← 3. **System** (a) senses the closure, (b) signals affirmation to the **Tenant**/**Landlord**, (c) signals to **LockDevice** to arm the lock, (d) signal to **Timer** to reset the auto-lock counter, and (e) signal to **Timer** to reset the alarm counter

Use Case 3: Add User

Use Case UC-3:	**AddUser**
Related Requirements:	REQ6 stated in **Error! Reference source not found.**
Initiating Actor:	Landlord
Actor's Goal:	To register new or remove departed residents at runtime.
Participating Actors:	Tenant
Preconditions:	None worth mentioning. (But note that this use case is only available on the main computer and not at the door keypad.)
Postconditions:	The modified data is stored into the database.

Flow of Events for Main Success Scenario:

→ 1. **Landlord** selects the menu item "ManageUsers"

2. **Landlord** identification: Include *Login* (UC-8)

← 3. **System** (a) displays the options of activities available to the Landlord (including "Add User" and "Remove User"), and (b) prompts the **Landlord** to make selection

→ 4. **Landlord** selects the activity, such as "Add User," and enters the new data

← 5. **System** (a) stores the new data on a persistent storage, and (b) signals completion

Flow of Events for Extensions (Alternate Scenarios):

4a. Selected activity entails adding new users: Include *AddUser* (UC-3)

4b. Selected activity entails removing users: Include *RemoveUser* (UC-4)

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 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

Example Use case: Wash Hands

- How a beginner writes the use case:

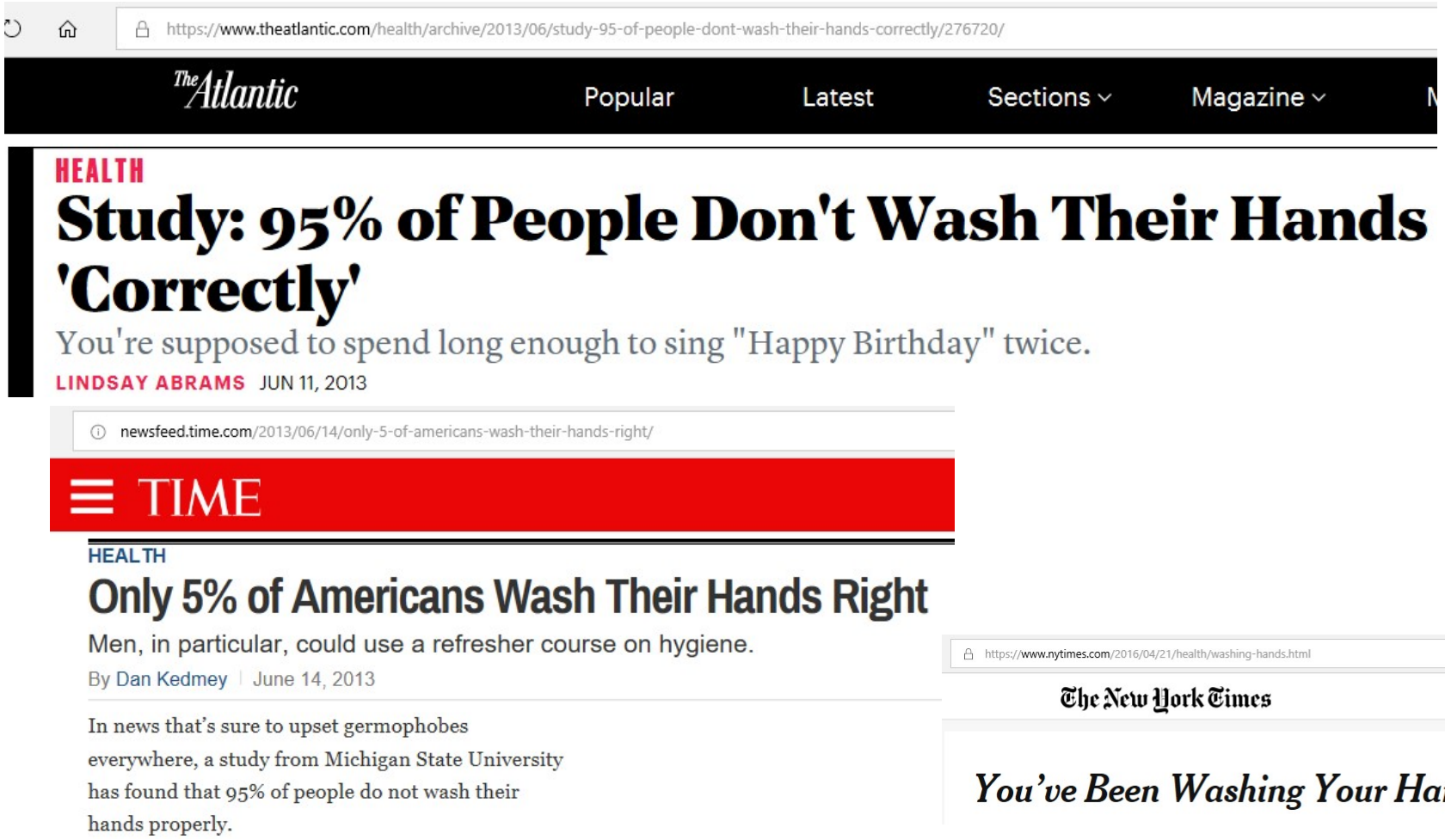
Example Use case: Wash Hands

- How an experienced developer writes the use case:

- But, that's ***obvious***!

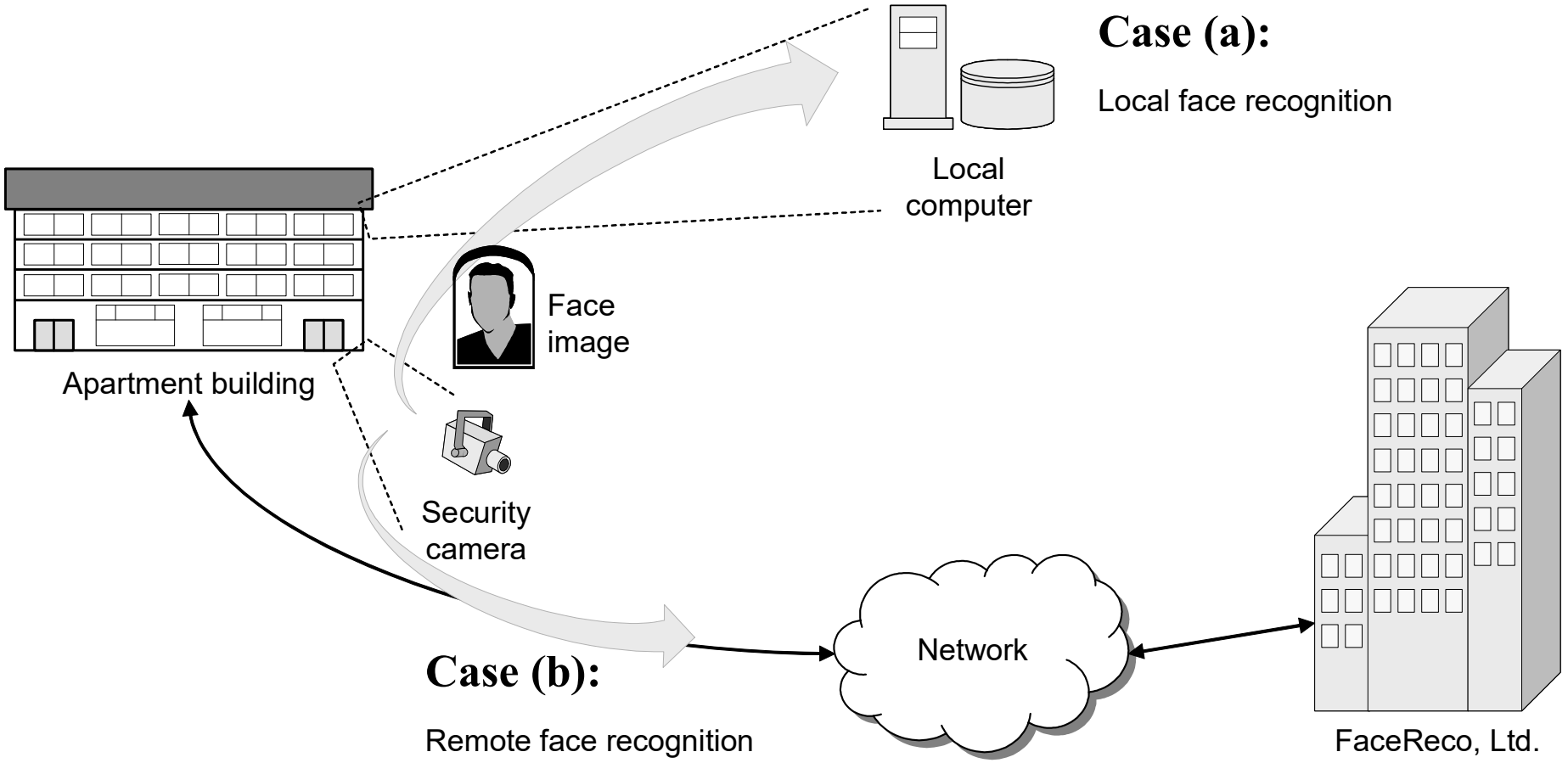
Example Use case: Wash Hands

- Well, maybe not!



System Boundary & Subsystems

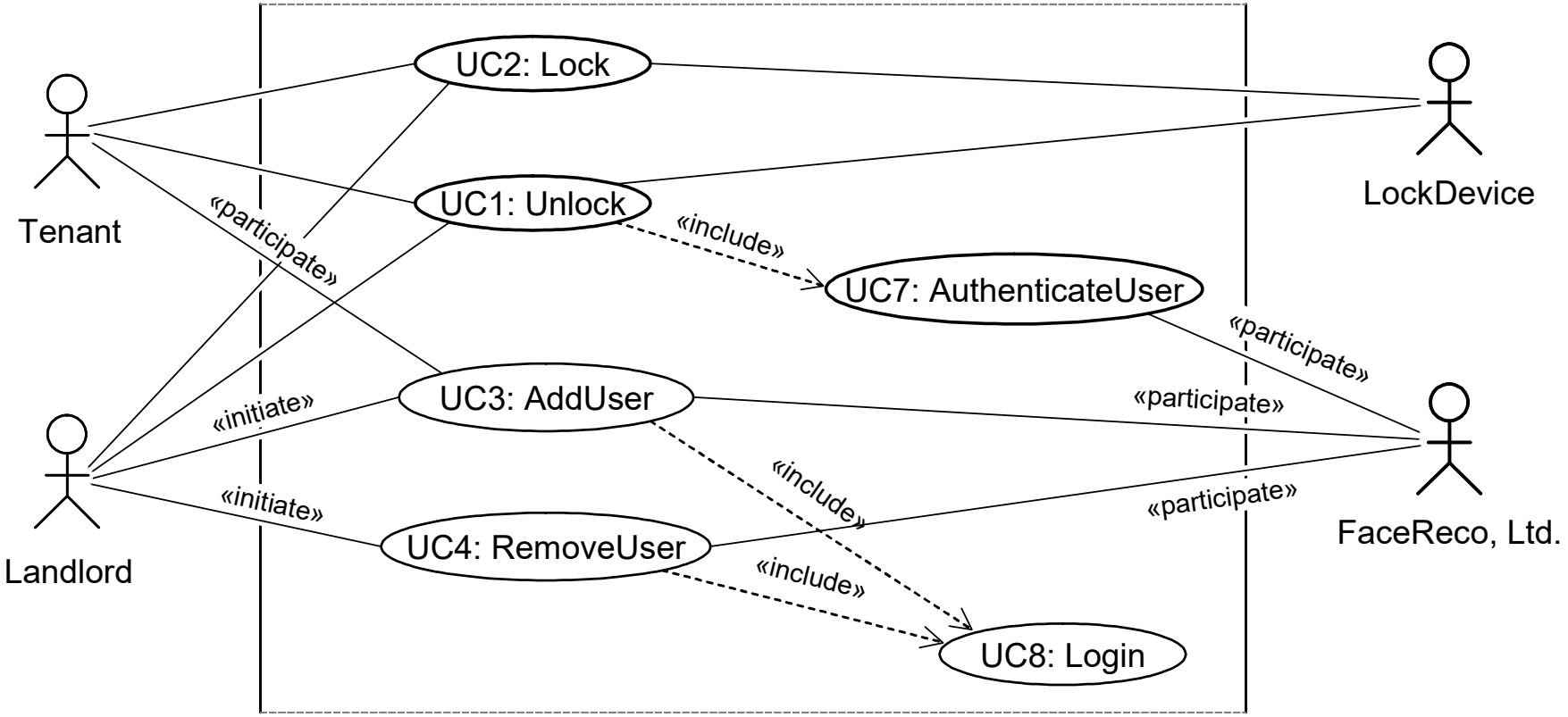
Use Case Variations Example:



Modified Use Case Diagram

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

Authentication subsystem (FaceReco, Ltd.)
is externalized from the system-to-be:



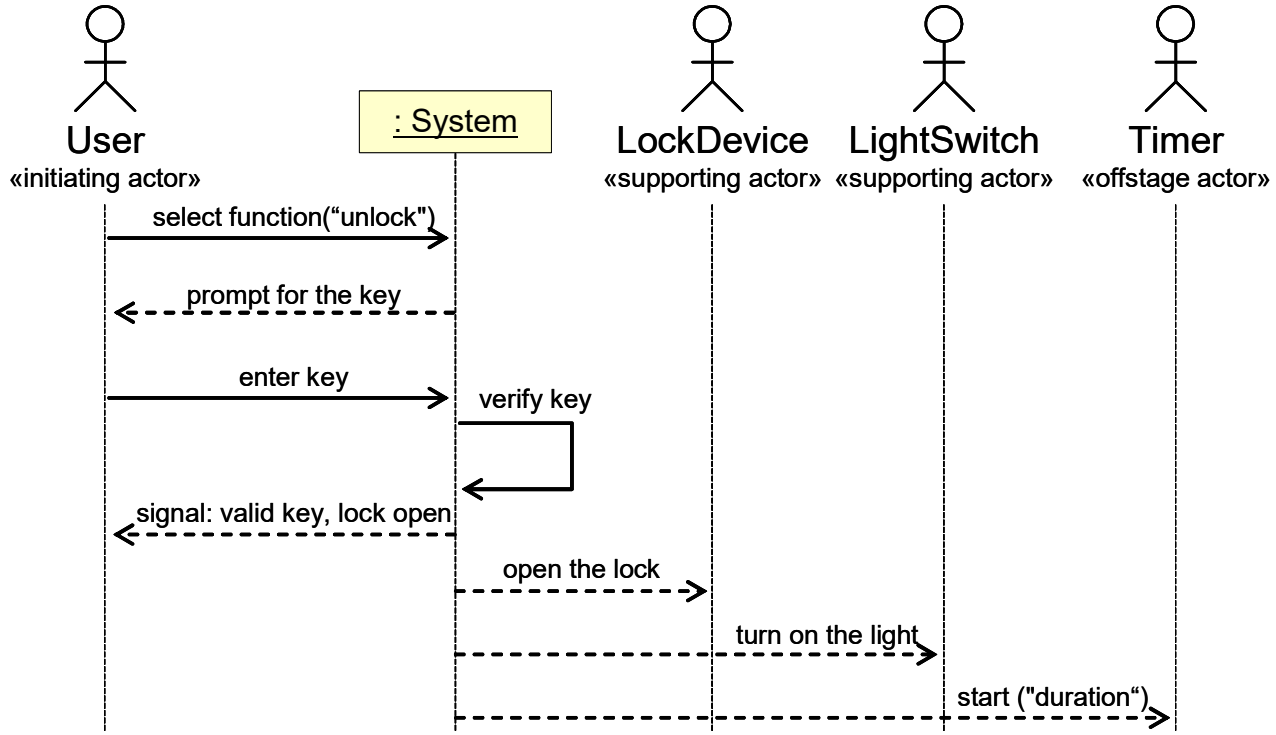
Security and Risk Management

- Identifying and preempting the serious risks to system's safe and successful functioning
- Risk types:
  - Intolerable
  - As low as reasonably practical (ALARP)
  - Acceptable
- Example **abuse case** input sequence:
  - invalid-key, invalid-key, … ≤ maxNumOfAttempts ; wait maxAttemptPeriod ; invalid-key, invalid-key, …

System Sequence Diagram

[Modeling System Workflows]

Use case: Unlock



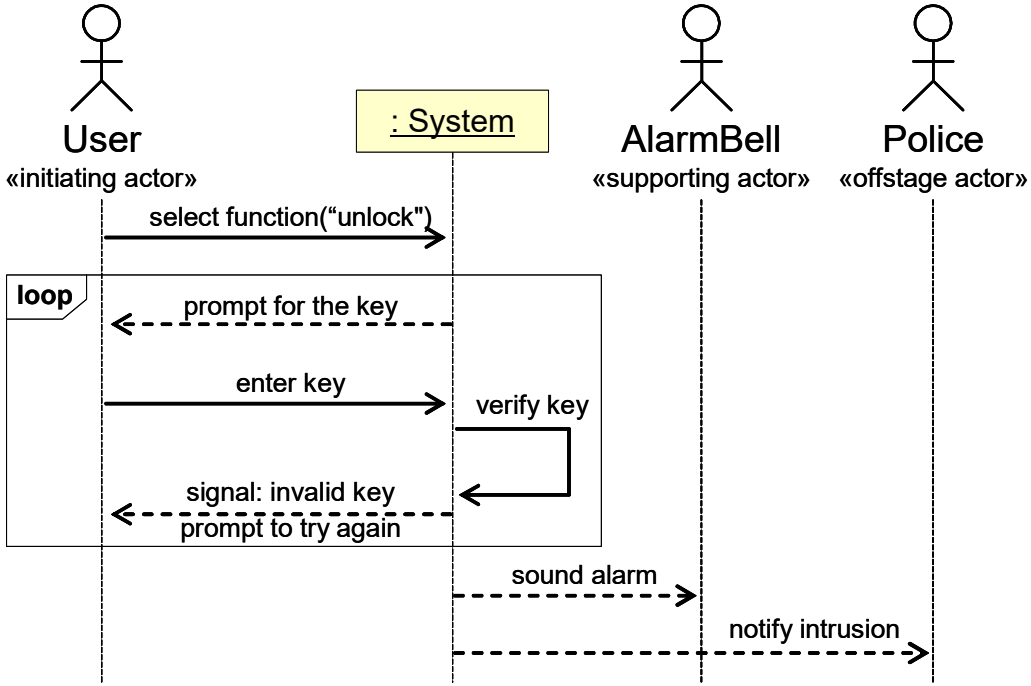
Main success scenario

Similar to UML sequence diagrams,
but for ***actor interactions*** instead of software ***object interactions***

System Sequence Diagram

[Modeling System Workflows]

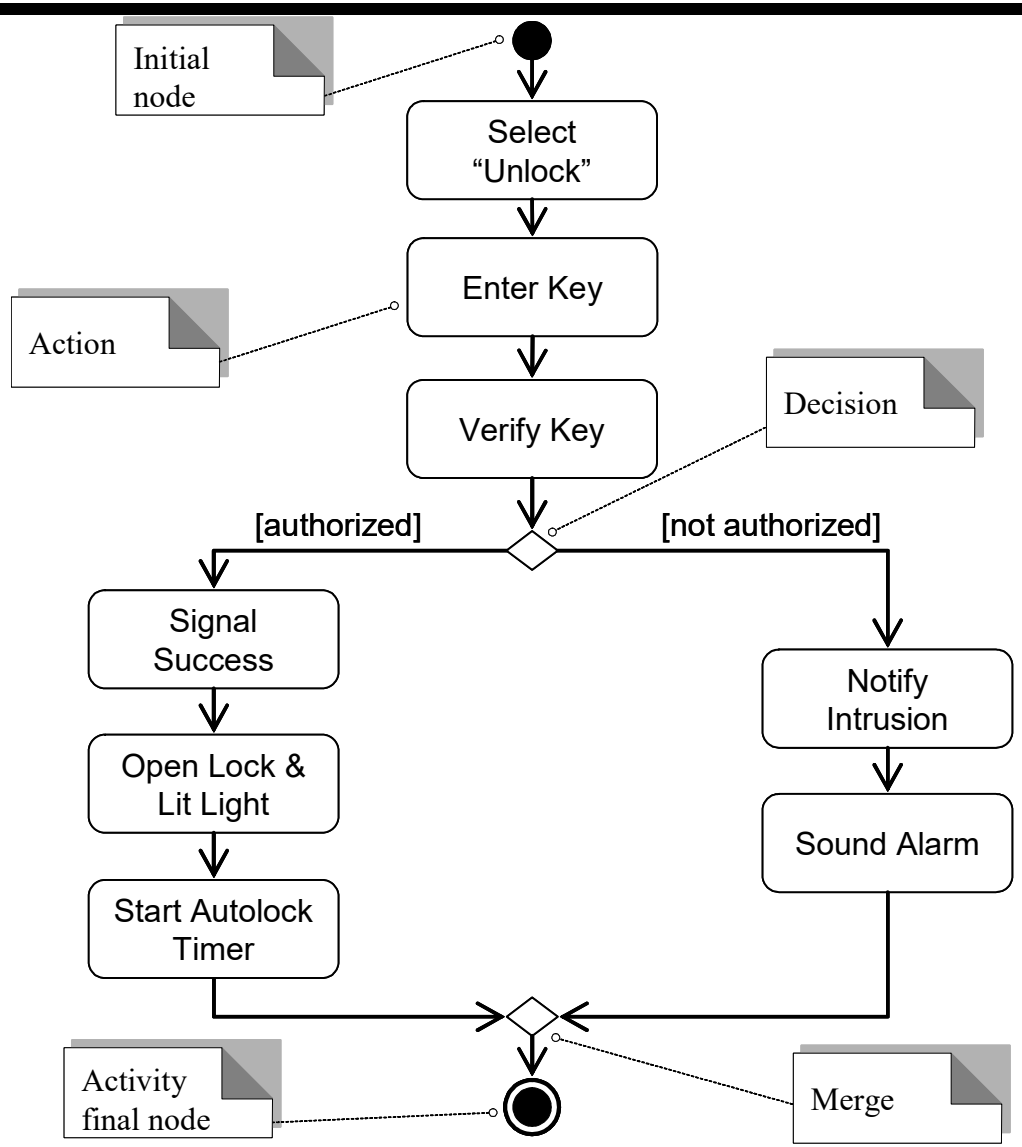
Use case: Unlock



Alternate scenario (burglary attempt)

Activity Diagram

[Modeling System Workflows]



Limitations of Use Cases

NOTE: User stories suffer the same limitations

- Often the user is using the system on behalf of the actual beneficiary. A store, agency, or bank clerk, lawyer, doctor, ... interacting with the system when providing service to a customer
- Whose business goals should we consider for these use cases: direct user's or actual beneficiary's?
  - Arguably there are merits in considering both, because otherwise we get only a partial picture.
    - E.g., in the restaurant example, we discuss what the wait staff may want to do, but we often mention the guest: the guest may order this or require that.
- The key importance of the end beneficiary is highlighted by the fact that automation systems often replace the intermediary in a business scenario and allow the end beneficiary to directly interact with the system.
  - When the intermediary cannot be completely replaced by automation, the intermediary becomes another user (supporting actor?) instead of being interposed between the system and the actual beneficiary.

User Interface Requirements

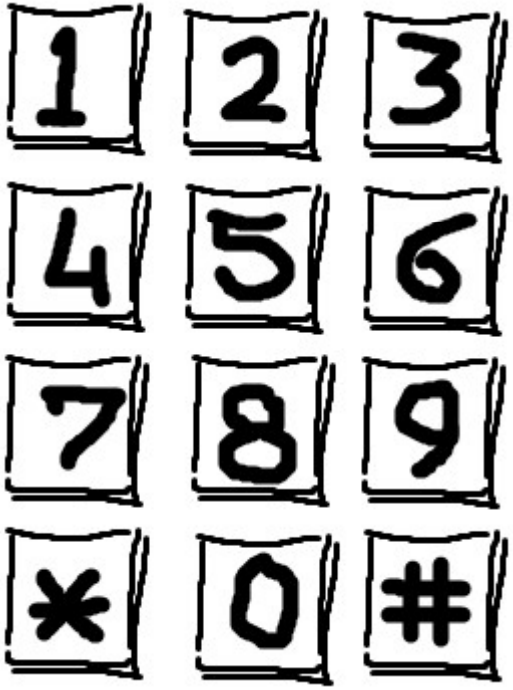
- Do not waste your time and your customer's time by creating elaborate screen shots with many embellishments, coloring, shading, etc., that only distract attention from most important aspects of the interface
- Hand-drawing the proposed interface forces you to ***economize*** and focus on the most important features
- Only when there is a consensus that a good design is reached, invest effort to prototype the interface

https://arstechnica.com/gadgets/2018/01/with-ink-to-code-microsoft-is-turning-back-of-napkin-sketches-into-software/

Example: Safe Home Access

User interface requirement:
The UI must be simple for occasional visitors who will not have time to familiarize or to study user's documentation.

Initial design of the door keypad:

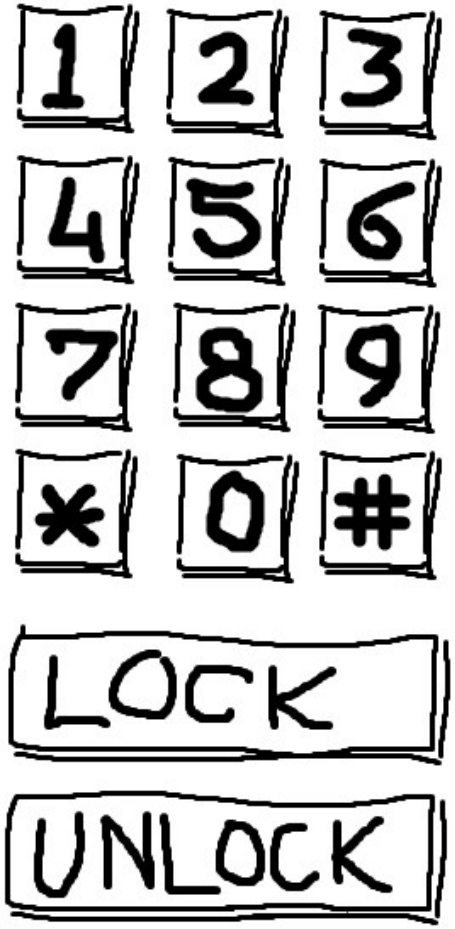


Analysis:

- If "Unlock" button is *not* available, then the system could simply take the **first** *N* digits and validate as the key.
- If "Unlock" button *is* available, then the system could take the **last** *N* digits and validate as the key (and ignore any number >*N* of previously entered digits).
- The advantage of the latter approach is that it allows correcting for unintentional mistakes during typing, so the *legitimate* user can have more opportunities to attempt.
- Note that this feature will not help the burglar trying a dictionary attack.

Example: Safe Home Access

Redesigned door keypad: includes the “Unlock”&“Lock” buttons

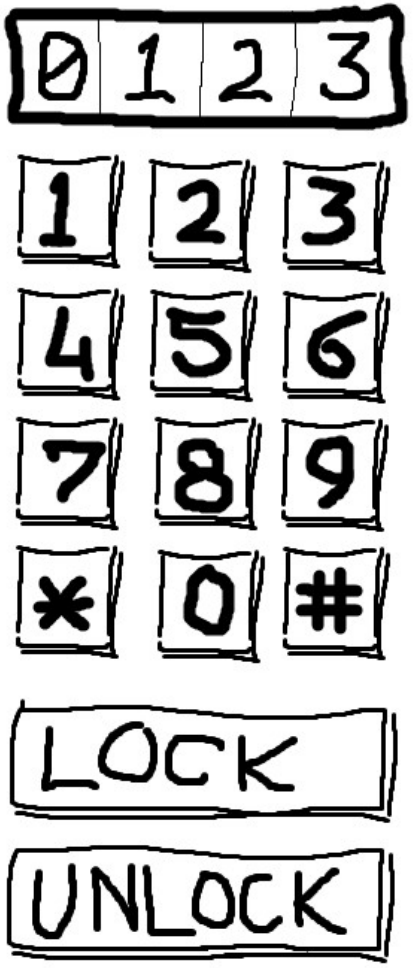


Analysis:

- When a user types in the key and the system reports an invalid key, the user may not know whether the problem is in his fingers or in his memory: “*did I mistype the correct key, or I forgot the key?*”
- To help the user in such a situation, we may propose to include a numerical display that shows the digits as the user types.

Example: Safe Home Access

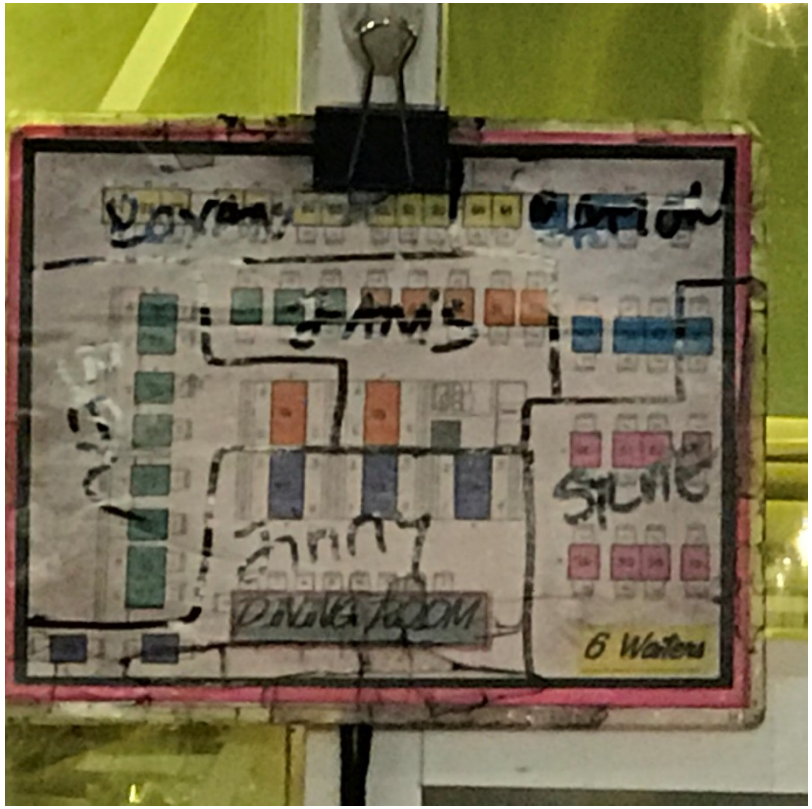
Re-redesigned door keypad: includes the keycode display

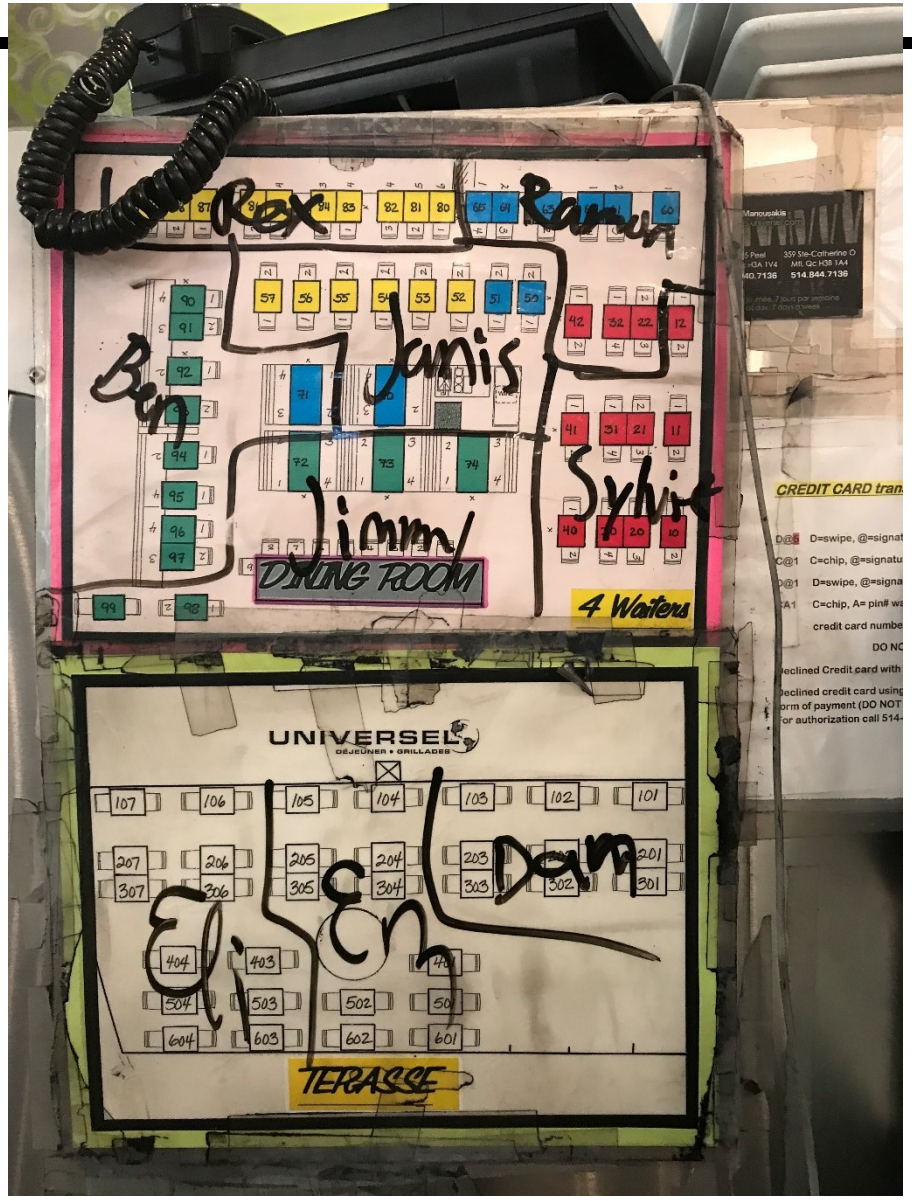


Analysis:

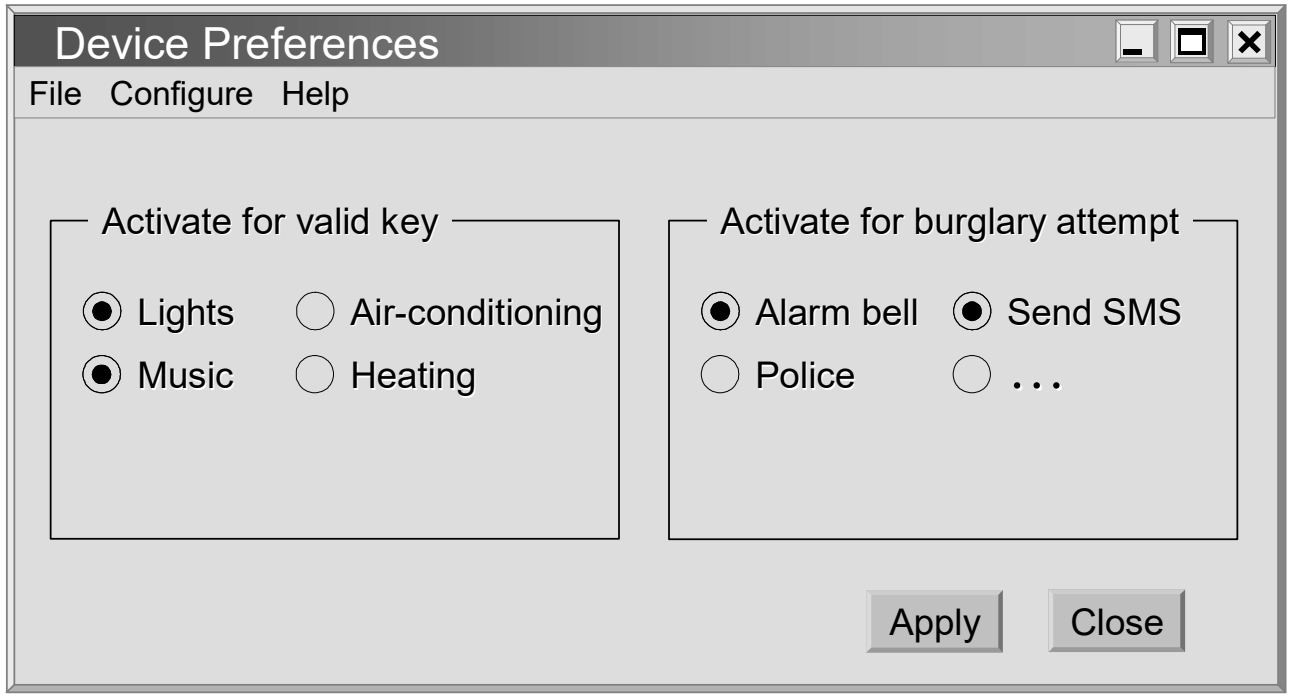
- There are several issues to consider about the display feature:
  - How much this feature would increase the overall cost?
  - Would the display make the door device bulky and inconvenient to install?
  - Would it be significantly more sensitive to rain and other elements?
  - Would the displayed information be helpful to an intruder trying to guess a valid key?

Restaurant Table Assignments





GUI for UC6: Set Device Pref's



(NOTE: Lock device is mandatory, not an option)