Software

Engineering

Problem Frames

Decomposition, Modeling & Recombination

何明昕 HE Mingxin, Max

Send your email to c.max@yeah.net with
a subject like: *SE345-Andy: On What …*

Download from c.program@yeah.net
/文件中心/网盘/SoftwareEngineering24s

Topics

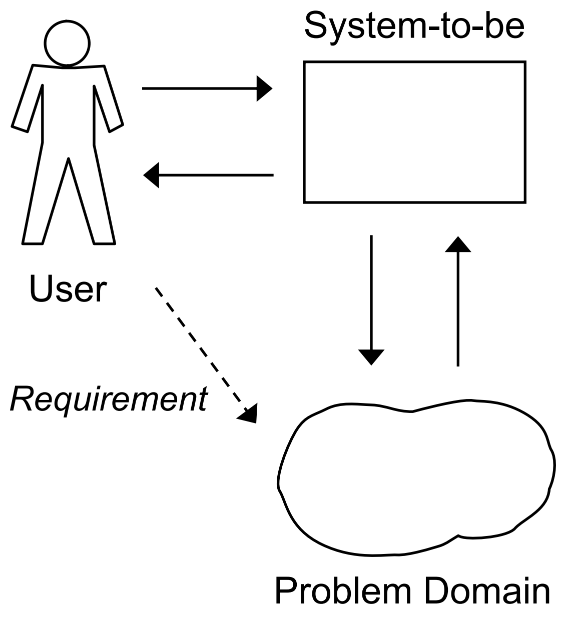
- ❑ Typical System Requirements and Problems
  - → Analysis Reuse
- ❑ Five Problem Frames:
  - Transformation
  - Simple Editing (a.k.a. “Simple Workpieces”)
  - Required Behavior
  - Information Display
  - Commanded Behavior
- ❑ Frame Concern

Why Problem Frames?

- Because the best way to start solving your problem is by understanding it first
  - Problem frames help us analyze and understand the problem that we are to solve
- Problem frames are the building blocks of SE problems
  - They are the most basic types of system requirements
- Purpose: problem frames allow ***reuse*** of existing generic analyses of typical problems
  - Benefits: speed up the analysis and make the result more correct and complete

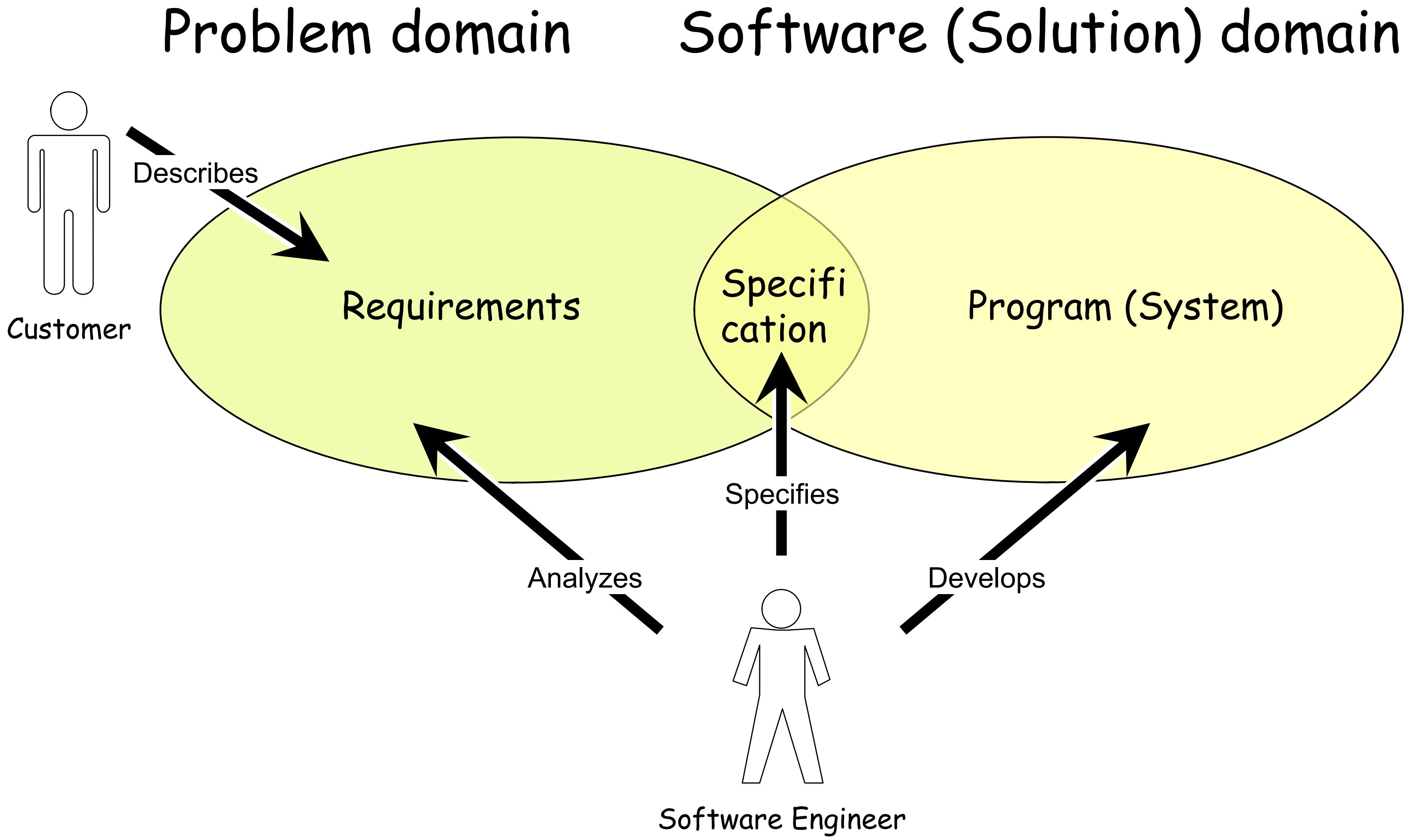
Software Engineering Problem

- ❑ User has business goals in the problem domain
- ❑ System-to-be will help the user achieve these goals
- ❑ Problem domain can be *imagined* or *physical world*

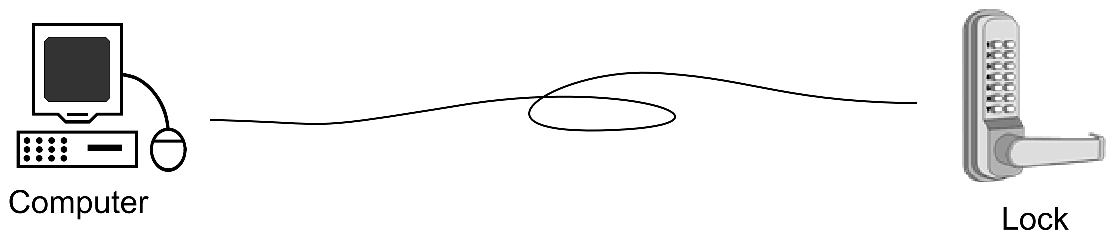


- ❑ User's business goals determine the system requirements

Requirements and Specification

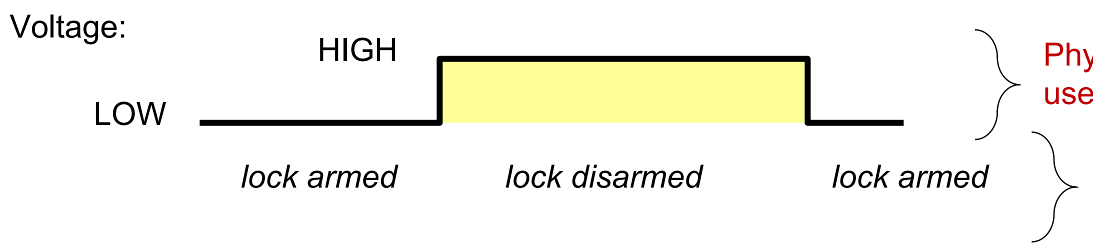


Problem Domain: How Electronic Lock Works



We may need separate descriptions/models of door vs. lock.

Door state is what the user cares about; lock is one way of achieving it.



Physical domain (lock) description; used in specification

Semantic meaning defined by user's goals; used in requirements

The behavior of the system-to-be determined not only by user's actions but also by the context ("situation").

E.g., what in case of power failure?
- By default armed
- By default disarmed (e.g., fire exit)

Analyst’s Task: Three Descriptions

The ***requirement***

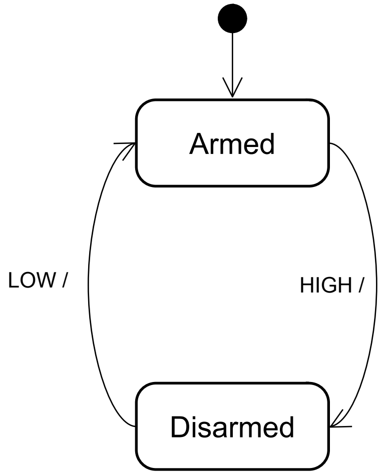
What user wants:

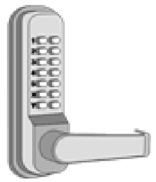
When valid keycode entered + Unlock pressed, open the lock;

Automatically lock after a period of time.

The ***problem domain***

How problem domain behaves:




Electromechanical lock

The ***specification***

What software-to-be will do (at interface):

If entered number matches one of stored numbers + Button-1 pressed, put HIGH voltage on Output-port-1;

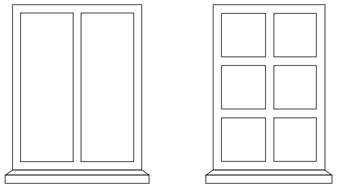
Start a timer countdown;

When the timer expires,
put LOW voltage on Output-port-1.

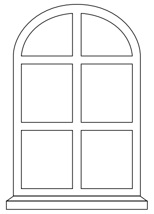
Concern:

It is not obvious that this is the only or even “correct” solution to the requirement-posed problem.

Problem Frames tell us what each description should contain and how to verify the concern.

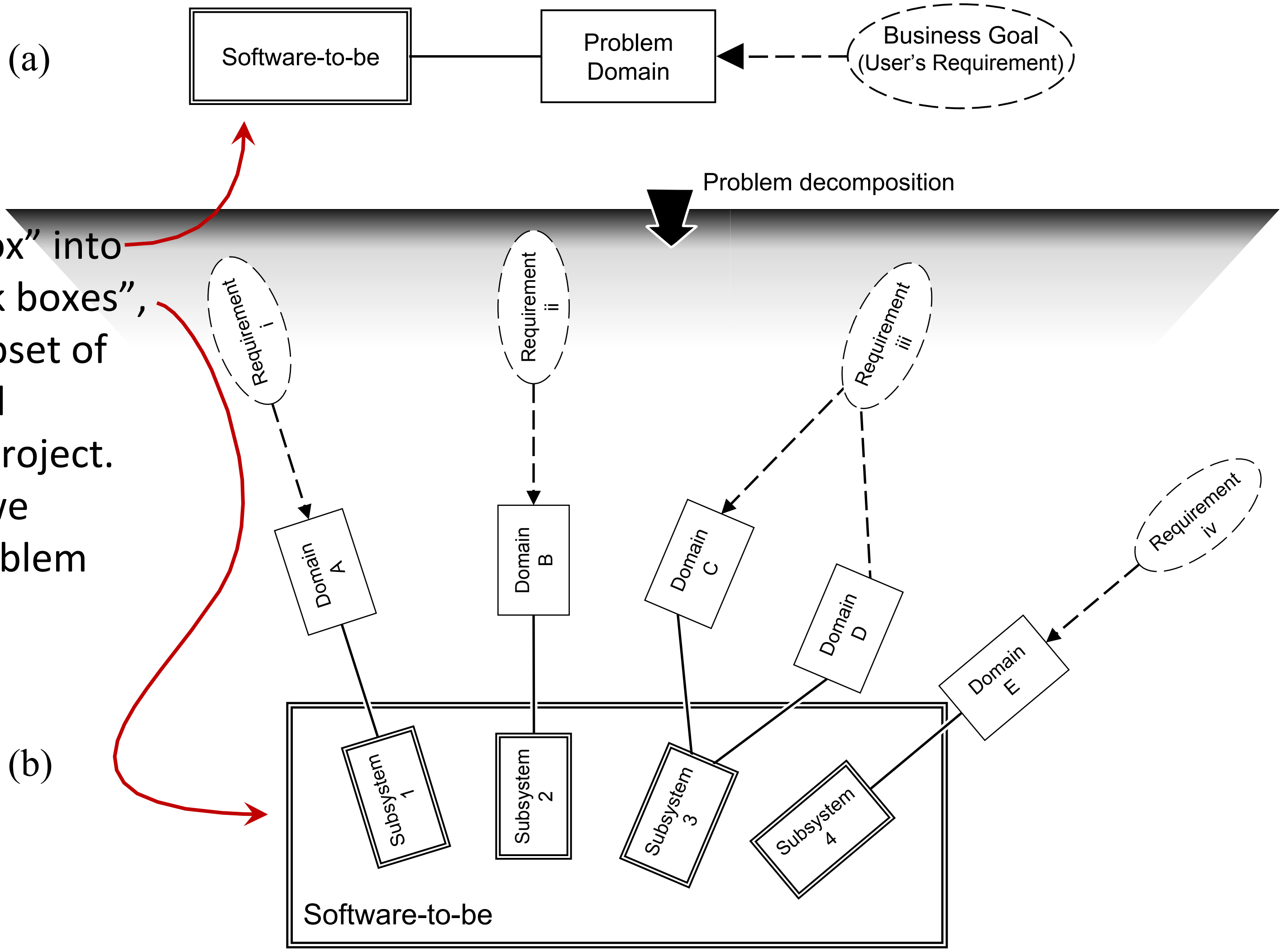


Problem Framing



- Problem framing means dividing the problem at its “seams” into smaller sub-problems
  - The difficulty is in recognizing the “seams”
- **Problem frames** are derived from commonly occurring software requirements
  - They afford ready-made **templates** for requirements analysis and system specification
  - We look for such basic problems to help us discover the “seams” of our problem

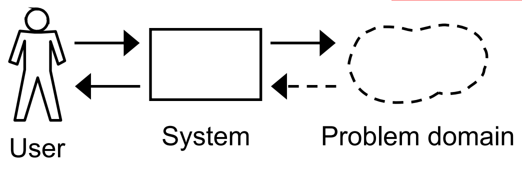
Complex Problem Decomposition



Frame Approach:
Split the big "black box" into several smaller "black boxes", each addressing a subset of the requirements and representing a mini-project. During the analysis, we consider each subproblem separately.

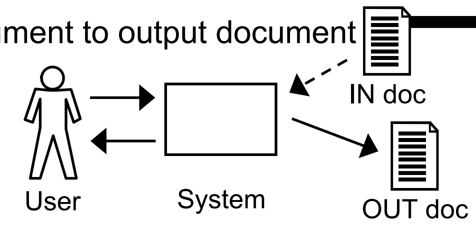
Typical Software Eng. Problems

1. User works with computer system (problem domain is "virtual", not physical)



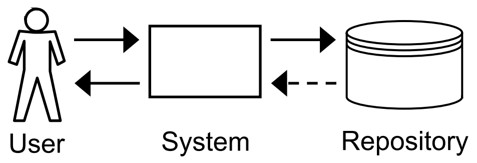
1.a) System transforms input document to output document

REQ-1: Map input data to output data as said by given rules

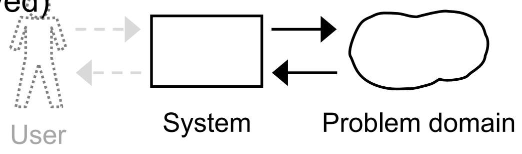


1.b) User edits information stored in a repository

REQ-2: Allow repository editing, where "repository" is a collection of data

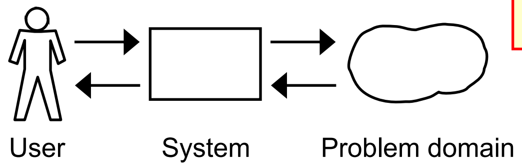


2. Computer system controls the (physical) problem domain (user not involved)



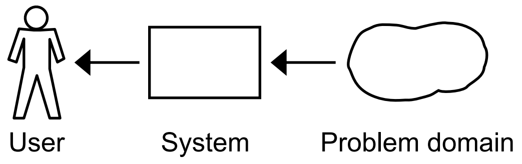
REQ-3: Autonomously control a physical object/device

3. Computer system intermediates between the user and the problem domain



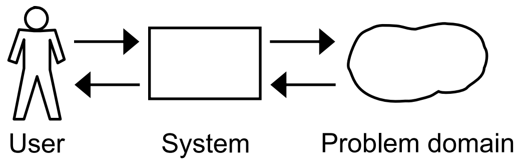
3.a) System observes the problem domain and displays information

REQ-5: Monitor and display information about an object

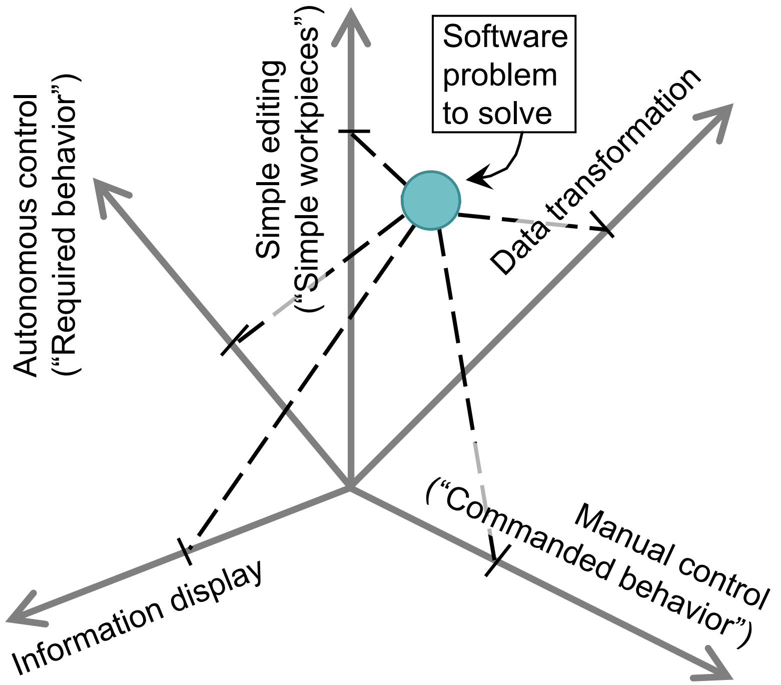


3.b) System controls the problem domain as commanded by the user

REQ-4: Interactively control a physical object/device



5-dimensional Problem Space



- The five elementary problem types represent the coordinate system of the problem space
- The "axis" projections represent the degree to which the whole problem contains a subproblem of this type
- Each subproblem can be analyzed independently and eventually recombined into the whole problem
- The structure of the solution should be selected to fit the problem structure

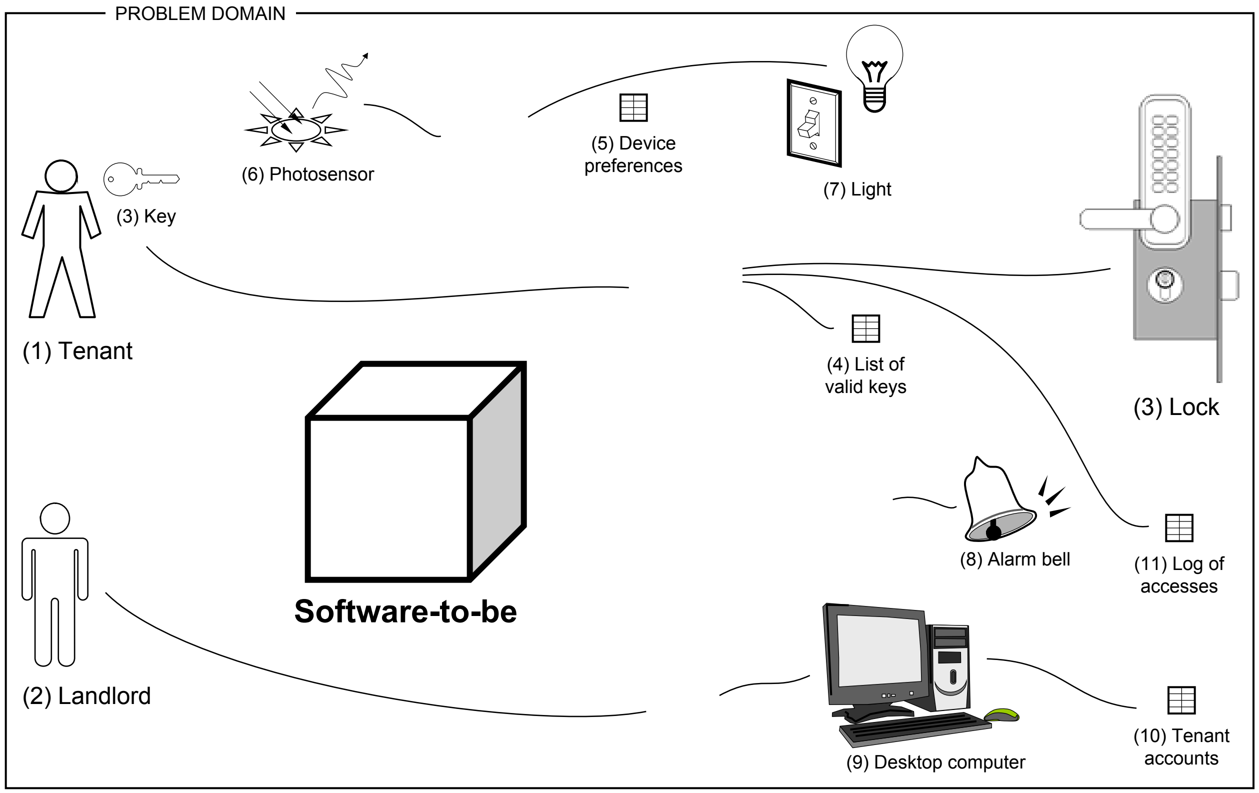
Typical System Requirements

- **REQ-1**: Map input data to output data as said by given rules
- **REQ-2**: Allow repository (or document) editing, where “repository” is a collection of data
- **REQ-3**: Autonomously control a physical object/device
- **REQ-4**: Interactively control a physical object/device
- **REQ-5**: Monitor and display information about an object

Critical insight: there are known approaches for analyzing each of these requirement types, including the information that should be specified for each type

Example: Problem Decomposition

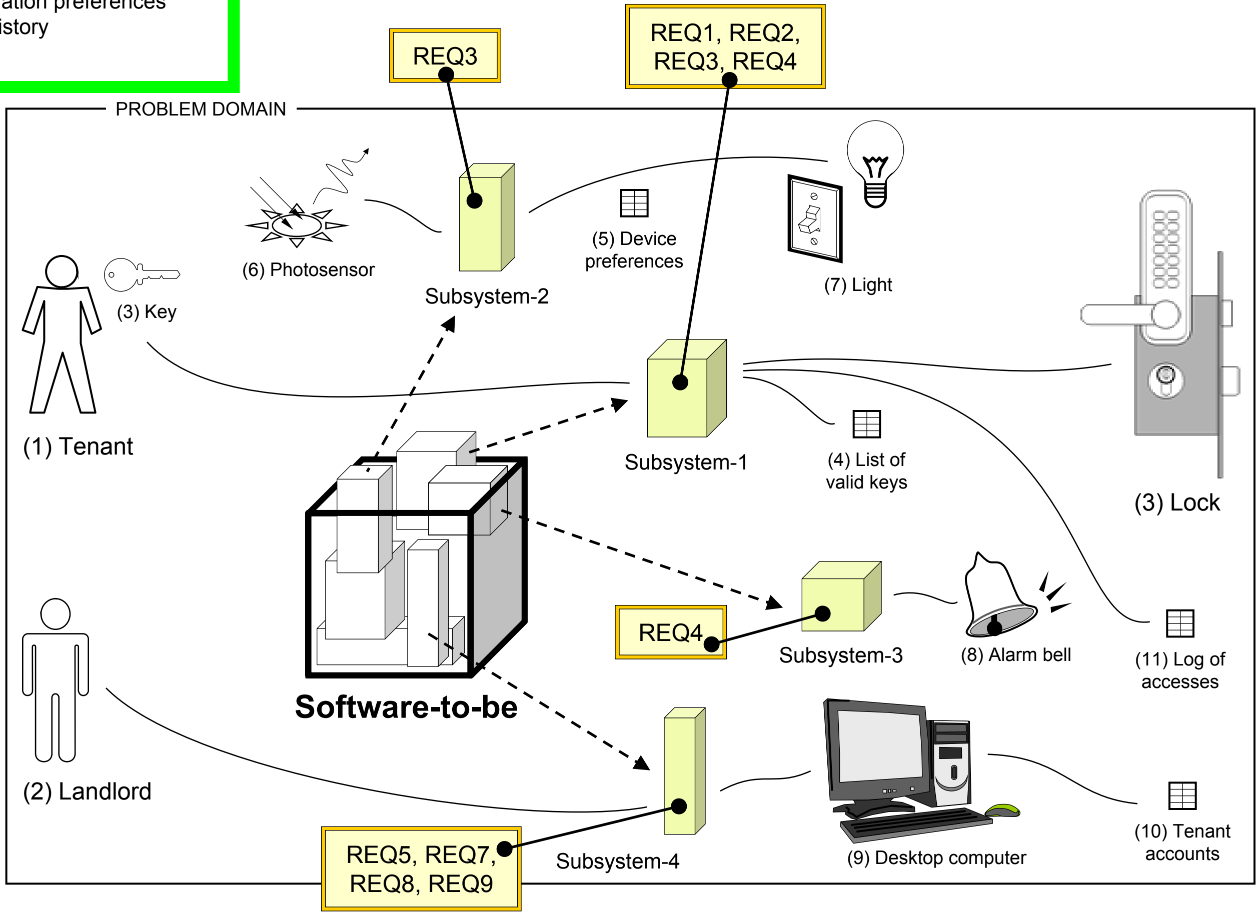
REQ1: Keep door locked and auto-lock
REQ2: Lock when "LOCK" pressed
REQ3: Unlock when valid key provided
REQ4: Allow mistakes but prevent dictionary attacks
REQ5: Maintain a history log
REQ6: Adding/removing users at runtime
REQ7: Configuring device activation preferences
REQ8: Inspecting the access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



Difficult to consider the whole system at once…

Example: Problem Decomposition

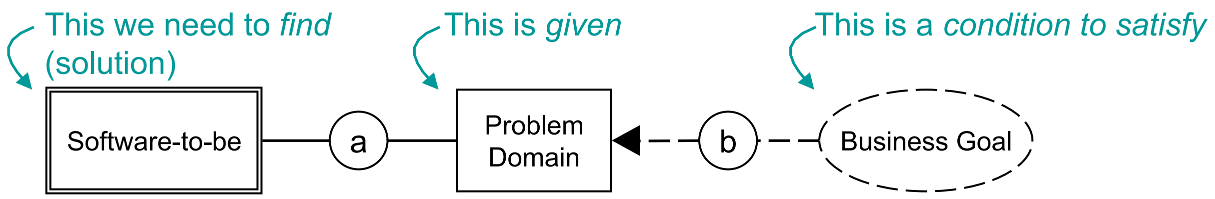
REQ1: Keep door locked and auto-lock
REQ2: Lock when "LOCK" pressed
REQ3: Unlock when valid key provided
REQ4: Allow mistakes but prevent dictionary attacks
REQ5: Maintain a history log
REQ6: Adding/removing users at runtime
REQ7: Configuring device activation preferences
REQ8: Inspecting the access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



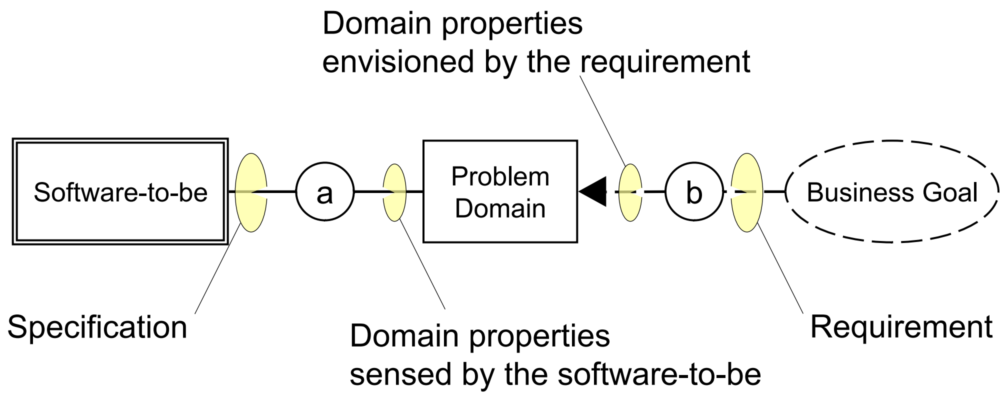
⇒ Decompose the system based on its requirements

Problem and Solution Domain

(a)


This we need to *find* (solution)
This is *given*
This is a *condition to satisfy*
Software-to-be
a
Problem Domain
b
Business Goal

(b)


Domain properties envisioned by the requirement
Software-to-be
a
Problem Domain
b
Business Goal
Specification
Domain properties sensed by the software-to-be
Requirement

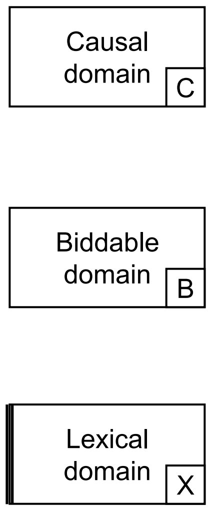
a: specification interface phenomena (system inputs/outputs)
b: requirement interface phenomena (user goals and interaction)

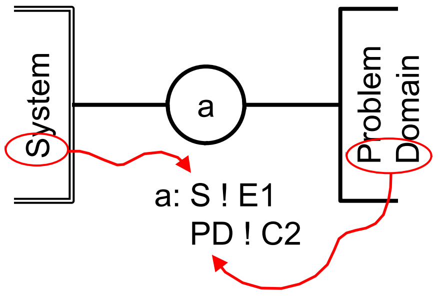
Description of what actually happens in the problem domain versus in what form it will appear as system input; as well as what output will the system issue versus what should happens in the problem domain.

Description of what the user is trying to achieve in the problem domain and how is the user supposed to interact with the system to make it happen.

Notation Syntax for Problem Frames

- $\boxed{C}$ – causal domain
  - predictable causal relationships among its causal phenomena such as physical laws or business contracts or social norms
- $\boxed{B}$ – biddable domain
  - usually people: unpredictable, incoercible
- $\boxed{X}$ – lexical domain
  - a physical representation of data (i.e., symbolic phenomena)
- [C·] - **causal phenomena**
  - events, states; directly produced or controlled by an entity; can give rise to other phenomena in turn
- [E·] - **events**
- [Y·] – **symbolic requirement phenomena**
  - values, and truths and states relating only values; symbolize other phenomena and relationships among them





Example: Problem Decomposition

[Case Study 1: Safe Home Access]

- REQ1: keep door locked and auto-lock

Required Behavior

- REQ2: lock when "LOCK" pressed
- REQ3: unlock when valid key provided
- REQ4: allow mistakes but prevent dictionary attacks

Commanded Behavior

- REQ5: maintain a history log

Information Display (database is the "display")

- REQ6: adding/removing users at runtime

Simple Editing

- REQ7: configuring device activation preferences

Simple Editing

- REQ8: inspecting the access history

Information Display

- REQ9: filing inquiries

Simple Editing

Example: Problem Decomposition

[Case Study 2: Investment Fantasy League]

- REQ1: view market prices and volumes
  - Domains: exchange, display

Information Display

- REQ2: place a trade
  - Domains: investor, tradable-securities, pending-trades

Simple Editing

- REQ3: execute trades when conditions matched
  - Domains: exchange, pending-trades

Required Behavior

- REQ4: generate periodic report
  - Domains: portfolio, report

Transformation

- REQ5: generate leaderboard
  - Domains: all-portfolios, leaderboard

Transformation

- REQ6: view leaderboard
  - Domains: leaderboard, display

Information Display

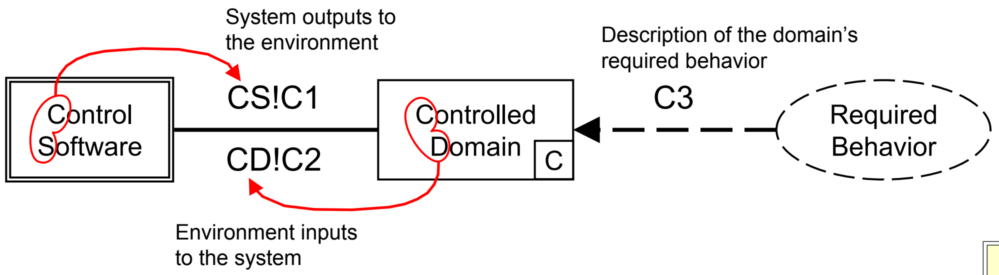
- REQ7: start a trading group/clique
  - Domains: investor, group

Simple Editing

- REQ8: invite to a trading group/clique
  - Domains: investor, friends, group

Commanded Behavior

Basic Frame 1: Required Behavior

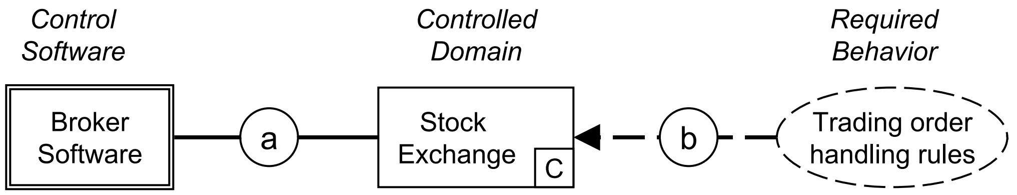


System outputs to the environment
Description of the domain's required behavior
CS!C1
C3
Control Software
Controlled Domain
C
Required Behavior
CD!C2
Environment inputs to the system

Key:

C	Causal domain
X	Lexical domain
[C·]	Causal phenomena
[E·]	Events
[Y·]	Symbolic requirement phenomena

Example: Execute a Trading order



a: BS! {Execute[i], Cancel[i]} [C1]
SE! {PriceQuotes, Ack[i], Failed[i]} [C2]

b: SE! {Submit[i], Revoke[i], Fulfilled[i], Failed[i], Expired[i]} [C3]

Required Behavior
- Frame Concern Checklist

Required Behavior Frame Checklist

- ❑ Frame requirement: Autonomously control an object ("domain")
  - ☐ Describe the business goals to be achieved
- ❑ Description of the controlled domain:
  - ☐ Describe the (relevant) behavioral rules of the domain
- ❑ Specification of the system-to-be:
  - ☐ What inputs the system will receive and outputs produce
- ❑ Capabilities of the available "sensors" (inputs):
  - ☐ Sensed parameters: type, sampling rate, precision, …
- ☐ Capabilities of available "actuators" (outputs):
  - ☐ Actuation parameters: type, acting rate, strength, …

What the three descriptions of a "Required Behavior" Frame should contain

[Case Study 1: Safe Home Access — REQ1: keep door locked and auto-lock]

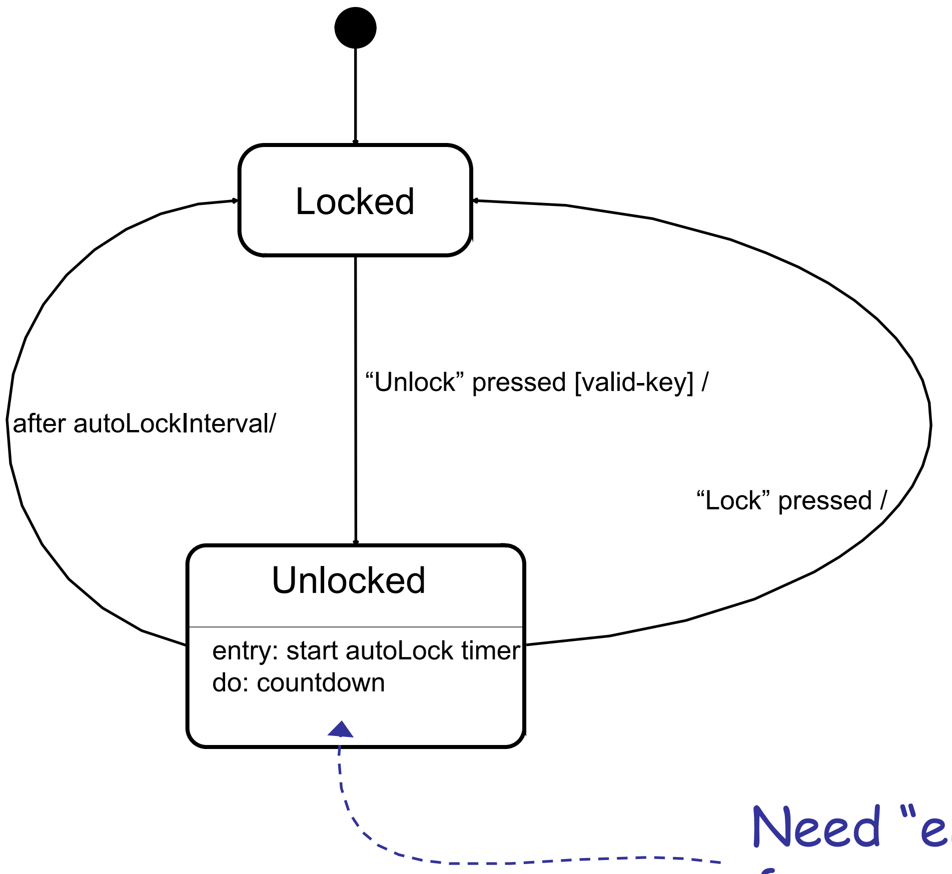
Business goal for REQ1:
The lock will be armed electronically and periodically checked that it remains so until explicitly disarmed
If the user unlocks, but forgets to lock, the lock will be automatically disarmed after a specified interval

Description of the electromechanical lock (problem domain):
A low or none voltage keeps the latch extended ("armed");
A high voltage causes mechanical retraction of the latch ("disarmed") and remains retracted until low voltage.

Capabilities of available "sensors":
The lock device will report current status when queried via USB protocol, using a command

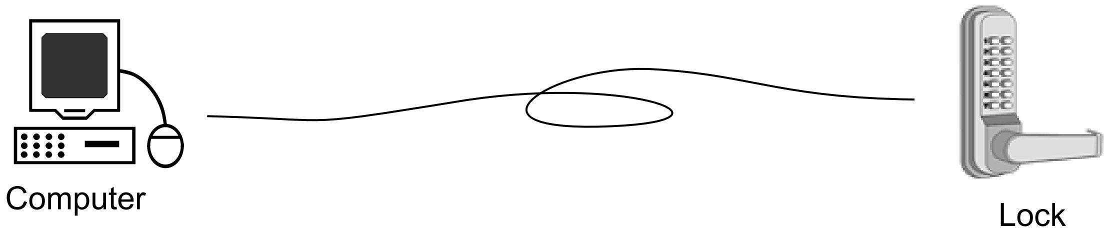
Capabilities of available "actuators":
The lock device's latch will extend or retract, as commanded via USB protocol, using a command

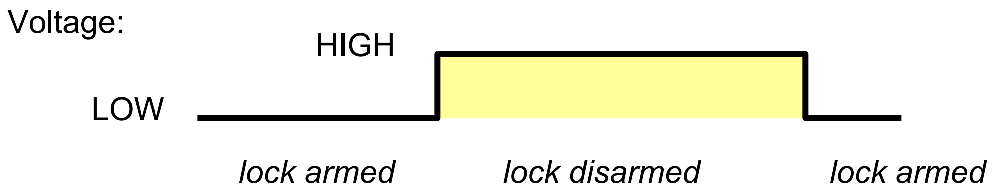
Requirement:
REQ1 - keep door locked and auto-lock



Need "entry" and "do" state activities for countdown timers

Problem Domain Properties:
How Electronic Lock Works



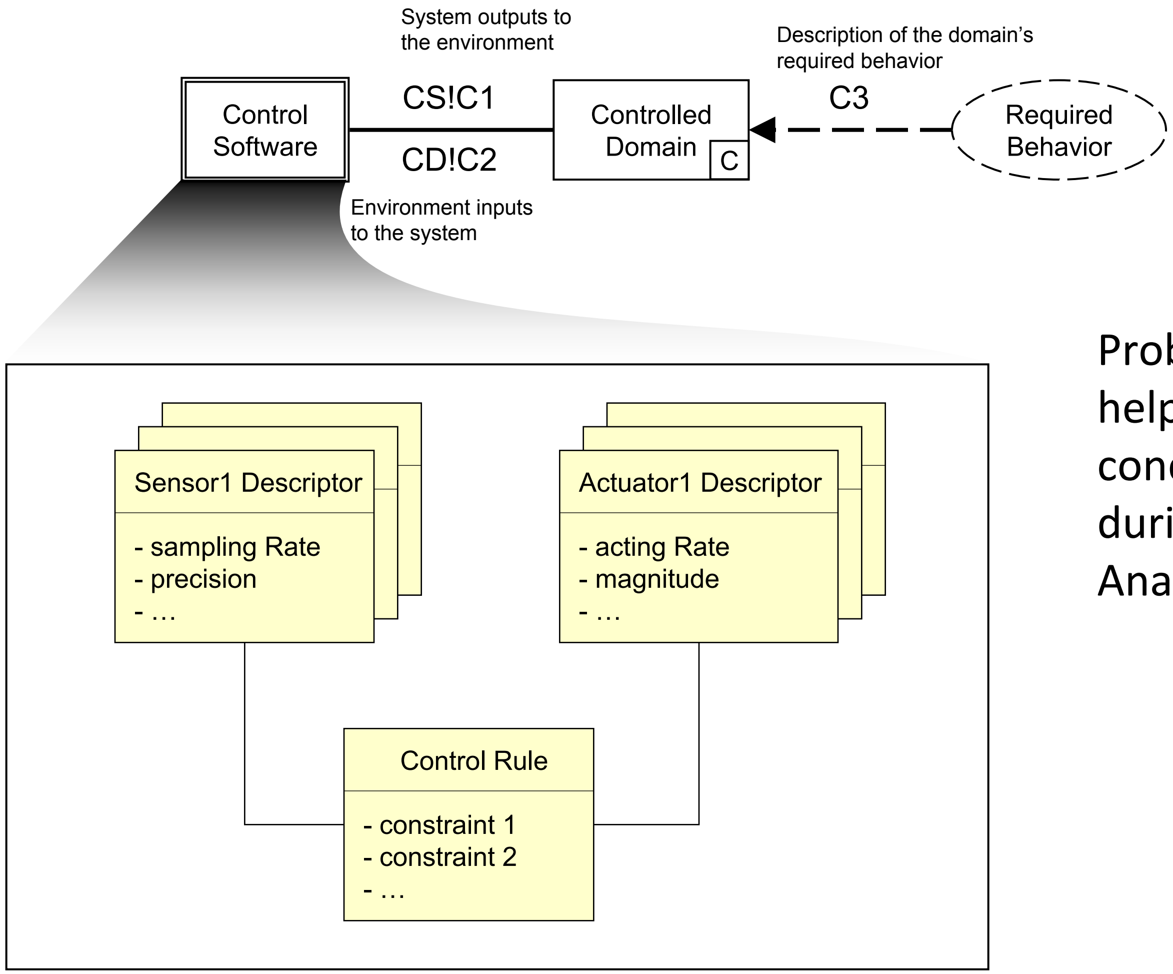


The system-to-be will set one of its output lines to “high voltage”, but only the human user will know the semantic meaning of this action (i.e., “disarm the lock”)

What in case of power failure?
- By default armed
- By default disarmed (e.g., fire exit)

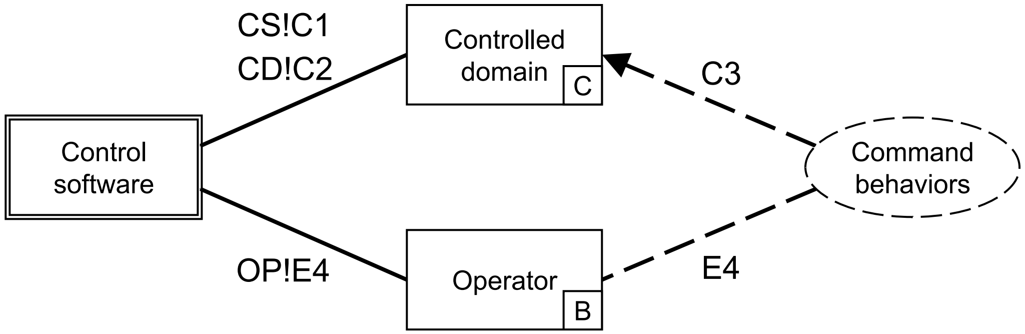
This information is not given.
These are deliberate decisions that the developer needs to make, based on understanding of the problem domain.
It is not enough to consider how the physical device behaves; we may also need to consider state laws, public safety policies, etc.

Required Behavior
- Domain Model -

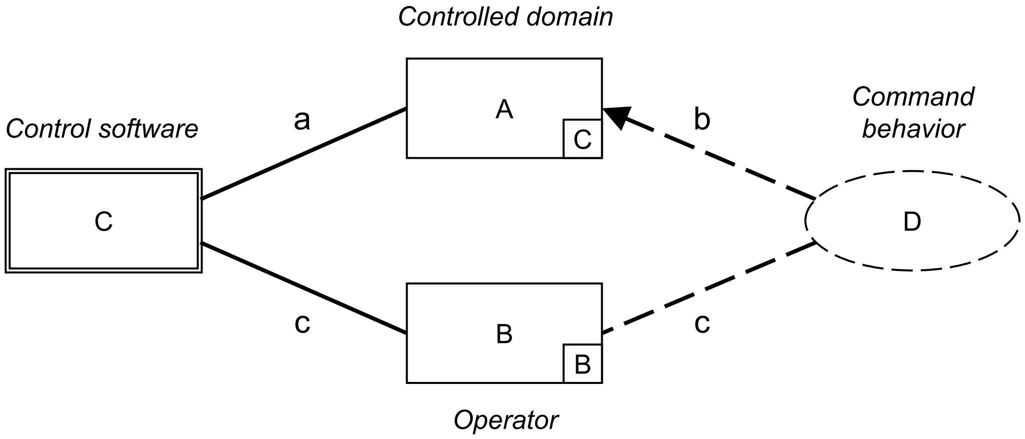


Problem Frame checklist helps with identifying the conceptual software objects during Object-Oriented Analysis

Basic Frame 2: Commanded Behavior



Example: Submit a Trading order



a: TS! {Create[i]} [E1]

b: TR! {PriceQuotes, Submit[i]} [Y2]

c: TR! {Submit[i], Cancel[i], Executed[i], Expired[i]} [Y3]

Commanded Behavior
- Frame Concern Checklist

Commanded Behavior Frame Checklist

- ❑ Requested user commands:
  - ◻ List of desired user commands and their parameters
- ❑ Command applicability under different scenarios:
  - ◻ Description of scenarios for which each command is applicable
- ❑ Consequences of unacceptable commands:
  - ◻ What should happen if the user tries to execute a command that is not supported or not allowed in the current scenario

Document what the user will be able to do and how the system will help them do, or what the user should not do and how the system will prevent these actions

[Case Study 1: Safe Home Access — REQ2: lock when "LOCK" pressed
REQ3: unlock when valid key provided
REQ4: allow mistakes but prevent dictionary attacks]

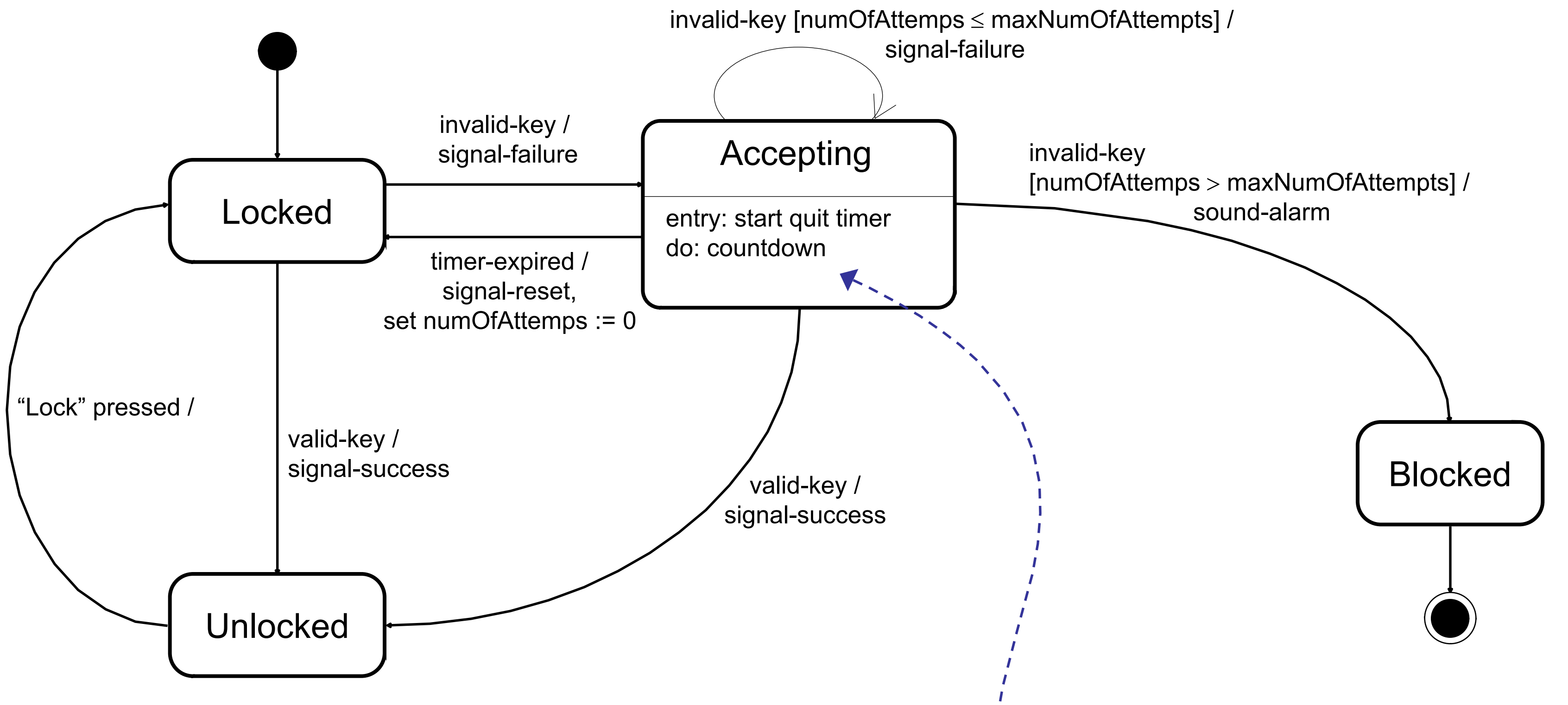
Requested user commands (REQ2, REQ3):
Lock, Unlock(key, door-ID)

Command applicability under different scenarios (REQ2 - REQ4):
Lock has no restrictions (always applicable)
Unlock applicable only if numOfAttemps ≤ maxNumOfAttempts

Consequences of unacceptable commands (REQ4):
When entered key does not correspond to the door-ID, increment numOfAttemps (block if > maxNumOfAttempts)
When not applicable, Unlock will be ignored

Requirement:

REQ3, REQ4, REQ5 - unlock when valid key provided but allow few mistakes



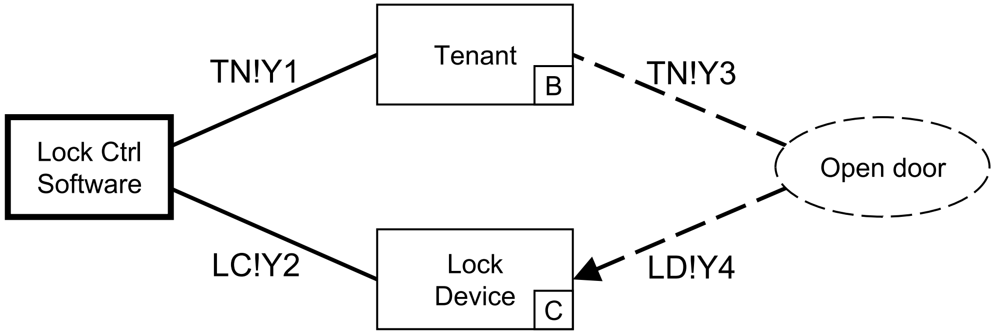
invalid-key [numOfAttemps ≤ maxNumOfAttempts] / signal-failure
invalid-key / signal-failure
Accepting
entry: start quit timer
do: countdown
invalid-key [numOfAttemps > maxNumOfAttempts] / sound-alarm
Locked
timer-expired / signal-reset, set numOfAttemps := 0
"Lock" pressed /
valid-key / signal-success
valid-key / signal-success
Blocked
Unlocked

Need "entry" and "do" state activities for countdown timer

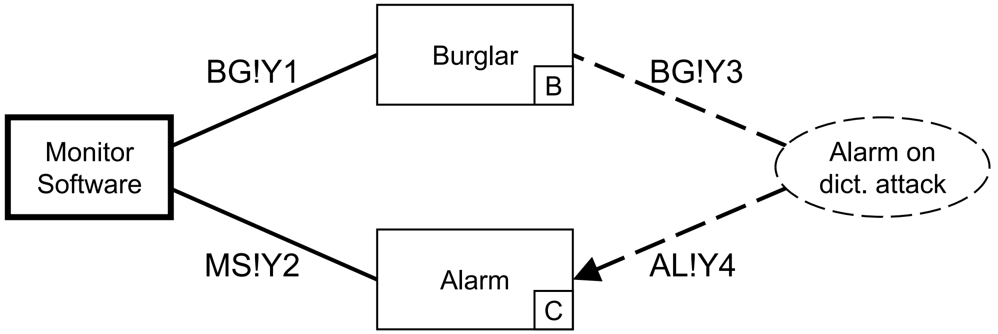
Safe Home Access

Requirement: Unlock the door for tenant and alarm on dictionary attack

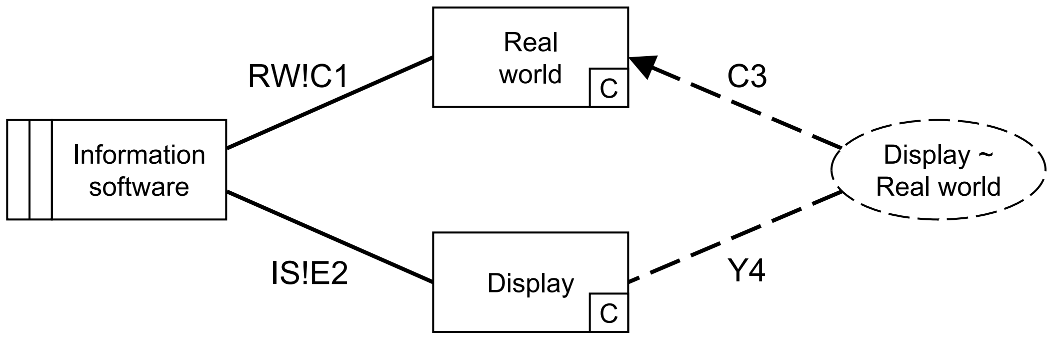
Requirement 1: Unlock the door for tenant



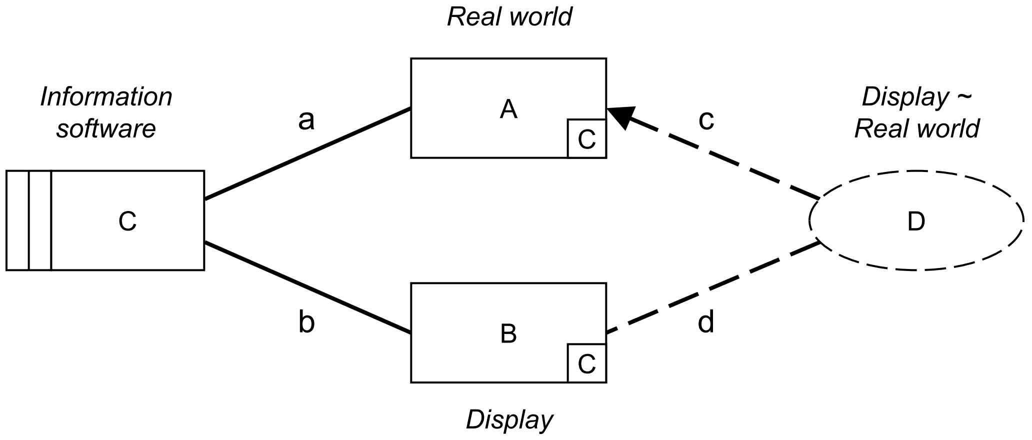
Requirement 2: Alarm on dictionary attack



Basic Frame 3: Information Display



Example: Place a Trading order



a: TS! {Create[i]} [E1]

b: TR! {PriceQuotes, Place[i]} [Y2]

c: TR! {Place[i], Cancel[i], Executed[i], Expired[i]} [Y3]

Information Visualization
- Frame Concern Checklist

Information Display Frame Checklist

- ❑Required information to observe:
  - ☐ Capabilities of available "sensors"
- ❑Required information to visualize:
  - ☐ Visualization description
- ❑Rules for visualization of the observed information:
  - ☐ The transformations needed to process the raw observed information to obtain displayable information

[Case Study 1: Safe Home Access — REQ5: maintain a history log (database is the "display")]

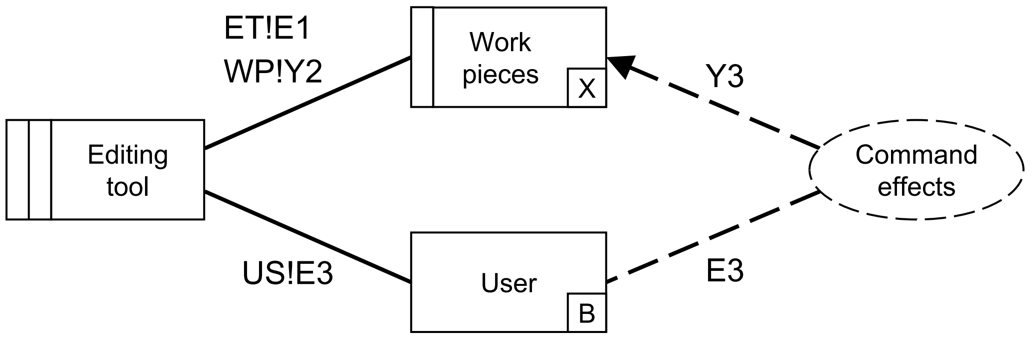
[Case Study 1: Safe Home Access — REQ8: inspecting the access history]

Information to observe for REQ8:
Database records of past accesses

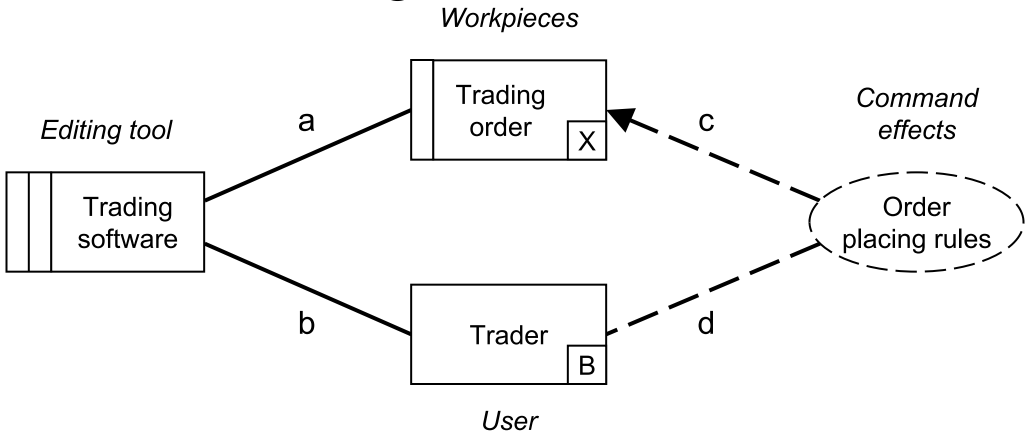
Required information to visualize for REQ8:
Access information will be displayed as stored, without post-processing

Rules for visualization for REQ8:
Render the result of the query as an HTML table

Basic Frame 4: Simple Editing



Example: Place a Trading order



a: TS! {Create[i]} [E1]

b: TR! {PriceQuotes, Place[i]} [Y2]

c: TR! {Place[i], Cancel[i], Executed[i], Expired[i]} [Y3]

Simple Editing
- Frame Concern Checklist

Simple Editing Frame Checklist

- ❑ Data structures:
  - ☐ Data types of elements ("workpieces") of the document
- ❑ Requested commands:
  - ☐ List of all possible user commands and their parameters
- ❑ Command applicability under different scenarios:
  - ☐ For each command, exactly describe the preconditions for execution
- ❑ Consequences of unacceptable commands:
  - ☐ What should system do if the user tries to execute a command that is not supported/allowed under the current scenario

[Case Study 1: Safe Home Access — REQ9: filing inquiries]

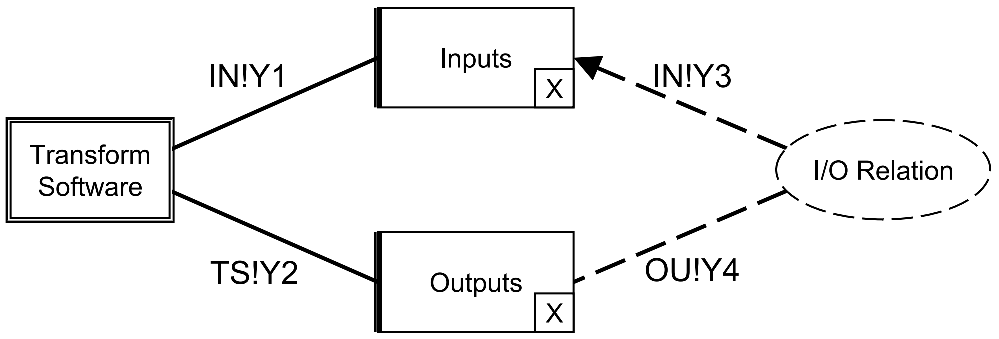
[Case Study 1: Safe Home Access — REQ6: adding/removing users at runtime]

Data structures for REQ6:
Database tables for each user, containing name, demographics, apartment number, keycode, …

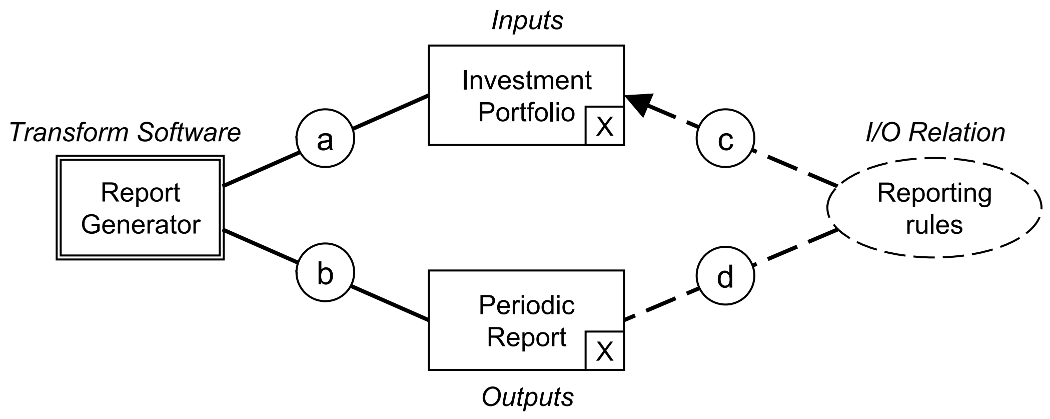
Requested commands for REQ6:
Add new tenant
Modify information of existing tenant
Move tenant to a past-tenants table (no permanent deletion allowed)

Command applicability under different scenarios:
Applicable for a user type Landlord

Basic Frame 5: Transformation



Example: REQ4 - generate periodic report



a: IP! {Shares[i], Price[i], Balance}	[Y1]	c: IP! {Stock[i], Balance}	[Y3]
b: RG! {Report Line, Char}	[Y2]	d: PR! {Line Data}	[Y4]

Transformation
- Frame Concern Checklist

Transformation Frame Checklist

- ❑ Input & output data structures:
    - ☐ Data types of elements of the input document & of output doc
- ❑ Traversal rules for data structures:
    - ☐ E.g., breadth-first or depth-first
- ❑ Mapping rules for elements of data structures:
    - ☐ How an input element is mapped/transformed to an output element