Requirements

Types:

user requirements

- what tasks the user can do with the system

functional requirements (features)

- what behaviors the system does or supports

non-functional requirements (qualities)

- how well the system should do what it does
- e.g., response time, resource usage, availability

Requirements

Types:

external interfaces

- e.g., interfaces to other hardware and software, data sources and sinks, formats, protocols

physical setting

- e.g., location, workspace, lighting, noise, temperature

developer constraint

- e.g., implementation technology, documentation

Requirements

Types:

business requirements

- why the system is needed

business constraint

- what the system or process must comply with
- e.g., corporate policy, industry standard, government regulation

Requirements

Requirements should be:

correct

- requirements properly represent user needs

complete

- all possible scenarios are described

consistent

- requirements do not contradict each other

clear

- no ambiguities

realistic

- can be achieved by “mere mortals”

Requirements

Also desired:

traceable

- can trace functionality and tests to the requirement being satisfied

verifiable

- repeatable test(s) can be designed to show that the system fulfills the requirement

Requirements Activities

Done iteratively:

requirements elicitation

- discover user needs

requirements analysis

- decide scope and priorities
- study feasibility, create mockups

requirements specification

- detail the requirements in terms the *users* can *understand*

Users

Who is the "user"?

- primary
  - end user
  - with frequent hands-on use
- secondary
  - manager of end users
  - with occasional use, or via an assistant
- tertiary
  - owner of the system
  - uses output, influences or makes funding decisions

Users

Some characteristics to consider:

background

- literacy and language
- motivation to learn

- domain knowledge
- task familiarity

- computer skills
- attitude to computers and technology

Users

Some characteristics to consider:

perceptual, motor, and tactile abilities

- seeing and hearing difficulties
- fine motor skills with input devices

physical

- height and strength (for kiosk design)
- hand/finger size (for mobile device design)
- health, age, and gender

social

- relationships with peers
- culture

Users

Kids of use:

infrequent use / novice user

- need wizards
- need clear prompts, error handling

frequent use / expert user

- need keyboard shortcuts
- need customization, programmability

Users

Some issues to consider:

users cannot always express what they want

- but they often know what they do not like

users may not know what is possible

- what is technically and economically feasible?

users stick to what they …

- know already works, or have always done

users may fear job losses

- leads to non-constructive participation

Users

“Innovator’s dilemma”:

as the user base for an application grows, there is a tendency for developers to focus on this increasingly expert (and vocal) group of users

the system becomes more sophisticated

development becomes “optimized” for them

Users

"Innovator's dilemma":

- potential new users need "less"
- experts don't want their app "dumbed down"
- competitor attracts the new users with a simpler "good enough" app
- original app loses market share due to disruption from the low end

Understanding

Tips:

manage expectations

- be clear and honest about claims
- avoid surprises, disappointments, hype

involve the user

- build tangible prototypes to gain feedback
- more likely to forgive problems if they are involved

establish a glossary

- terminology used in the application domain (not programming domain)

- **User Requirements**

Identifying Tasks

Study what tasks users do:

- what is the goal and context?
- what information is needed?
- what are the steps?
- who does the user work with?
- why is it done this way?

Identifying Tasks

Scenario:

an informal narrative

personal and concrete, but not particularly general

use the scenario to understand existing goals, task flow, and possible irritants

Scenario

Example:

> "I want to track the calories for a meal, so I consult the USDA Nutrient database. I want to look up 'Pacific salmon' so I enter that as the keywords. Item not found! So I enter 'salmon' and try again. That works, but I get 46 items, including salmonberries and even cloudberries. Why? I choose 'fish, salmon, sockeye, cooked, dry heat', then figure 2.5 x 100 g units for my item, and scan the table to see 422 kcal in the energy row."

Specifying Tasks

Use Cases:

capture the goal, conditions, and steps of a coherent interaction between the actor(s) and the software system

more general than a specific scenario

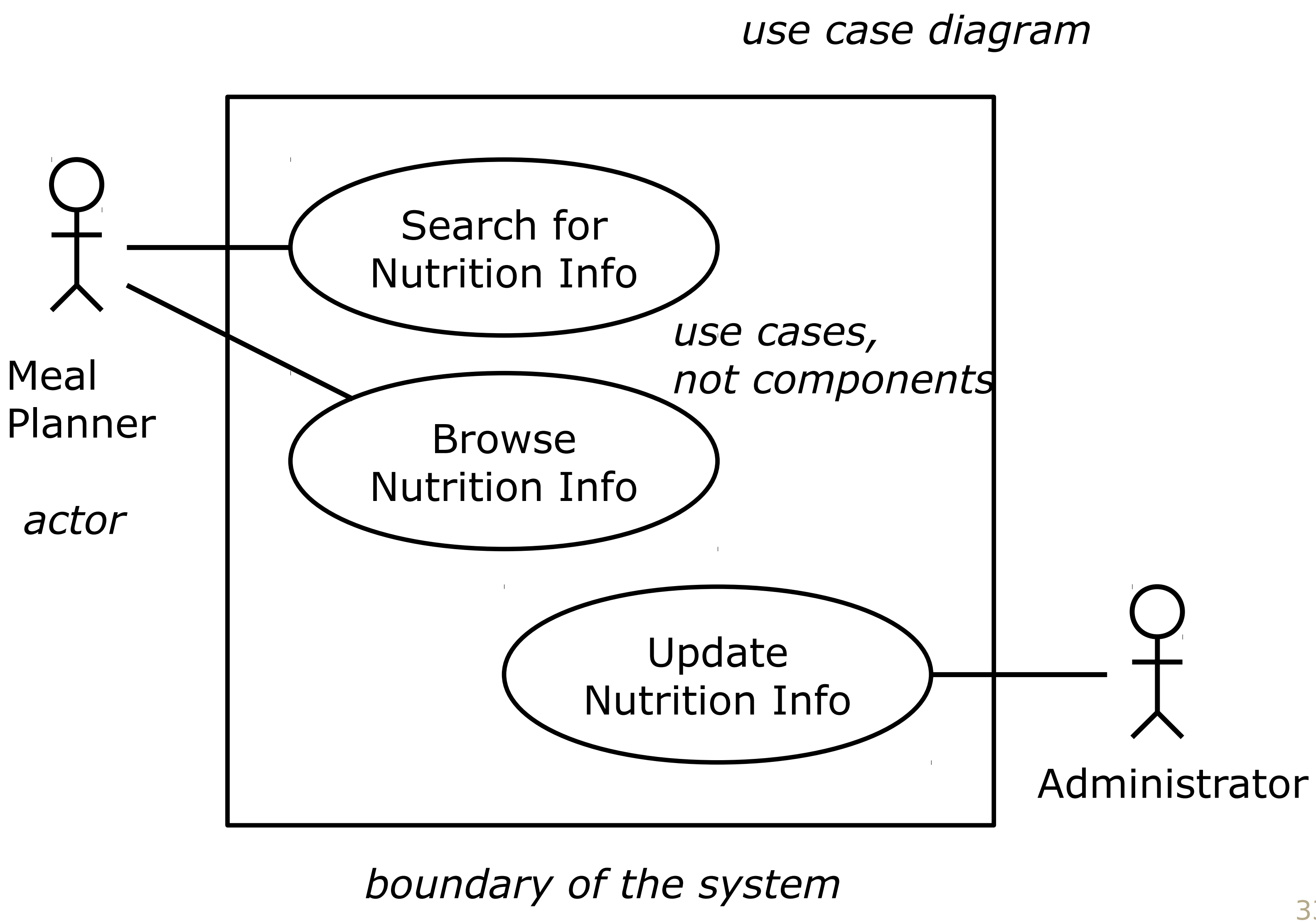
written from a “user” point-of-view

Defining Use Cases

Stages:

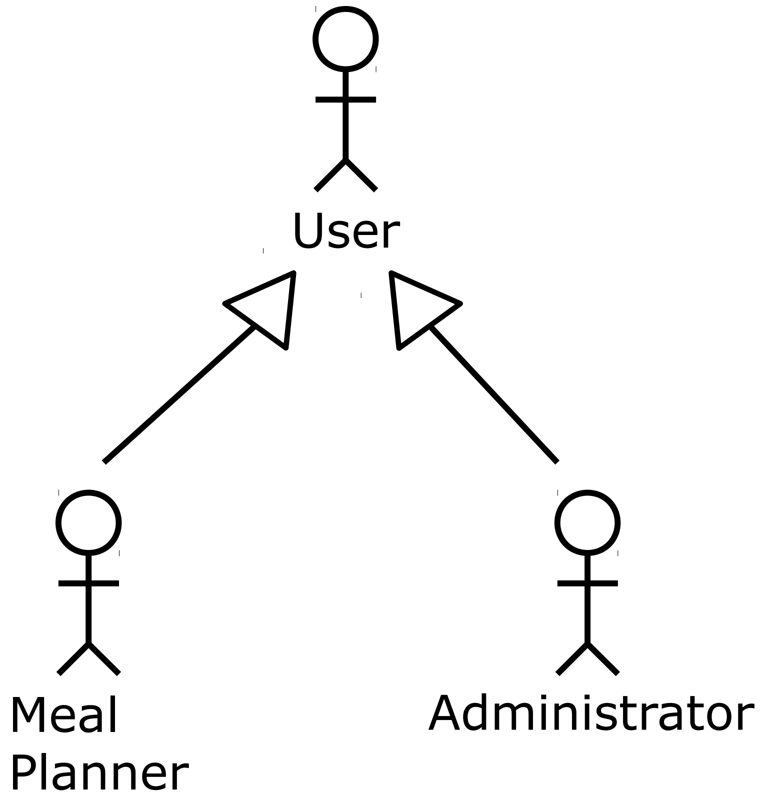
- identify the actors
  - consider different user roles and external systems
- define use cases
  - include all cases of use
- refine use cases
  - consider exceptional conditions and qualities
- relate use cases
  - consider inclusion and extension dependencies

Identify the Actors



Identify the Actors

Actor generalization:



Define/Refine Use Cases

Example:

Use Case Name	**SearchForNutritionInfo**
Participating Actors	Meal Planner (primary)
Goal	Meal Planner finds nutrition information
Trigger	Meal Planner chooses the Search option
Precondition	Meal Planner knows food name and amount.
Postcondition	On success, nutrition information displayed.
	…

Define/Refine Use Cases

Example:

user point of view

Basic Flow 1	System prompts Meal Planner to enter keywords.
2	Meal Planner submits keywords.
3	System lists matching foods, prompting for a selection.
4	Meal Planner browses and selects a food.
5	System prompts for food weight in units of 100 g.
6	Meal Planner enters food units.
7	System presents nutrition data for the amount of food.
	… *avoid implementation specifics*

Define/Refine Use Cases

Example:

Exceptions 3	If there are no matching foods
3.1	System displays an error
3.2	System returns to step 1
7	If given food units is non-numeric, use 0 and proceed

Define/Refine Use Cases

Example:

Qualities	System responds in under 2 s for list of matching foods and for nutrition data on a specific food.
Constraints	Use USDA nutrition data.
Includes	
Extends	
Related Artifacts	
Notes	
Open Issues	

Essential Use Case

SearchForNutritionInfo	
User Intention (Meal Planner)	*System Responsibility*
Initiate search.	
	Request keywords.
Submit keywords.	
	List matching foods to select.
Select a food.	
	Request food units.
Enter food units.	
	Present nutrition data.

Exercise

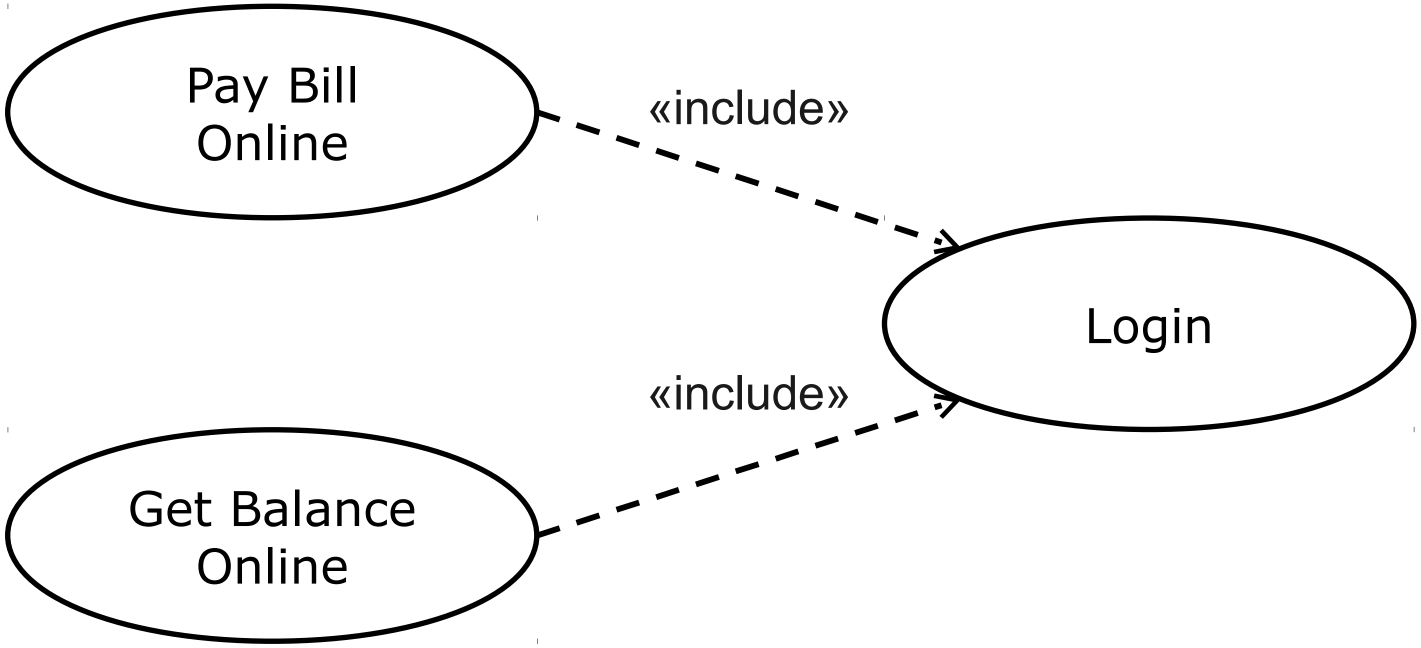
Question:

What is a basic flow for the task of withdrawing cash from a bank machine?

Relate Use Cases

Inclusion:

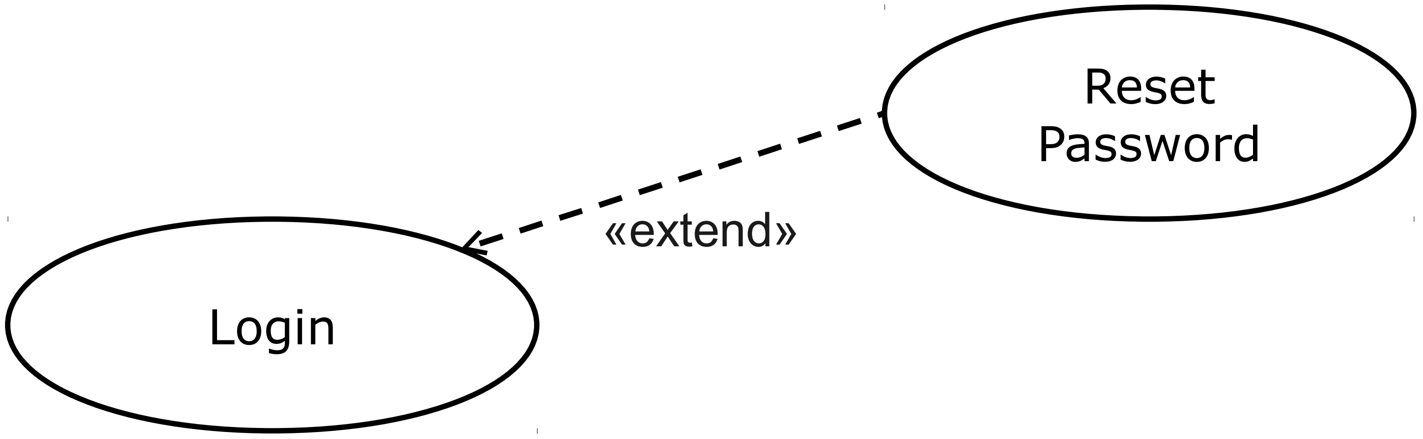
a use case may include another use case
(for necessary, shared behavior)



Relate Use Cases

Extension:

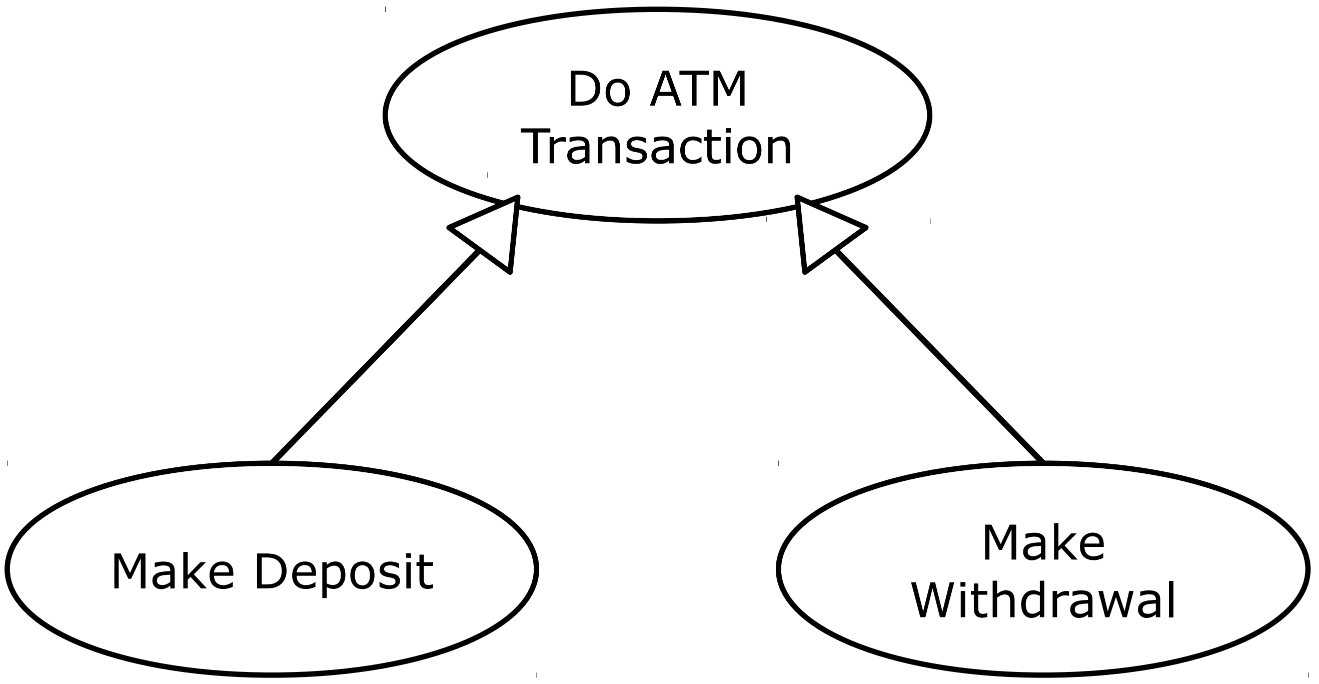
a use case may be extended by another use case (for optional or exceptional behavior)



Relate Use Cases

Use case generalization:

a use case may be a specialization of a more general use case



User Stories

Specifying Needs

Write down all the requirements. ➡ **Users only get what *was written*.**

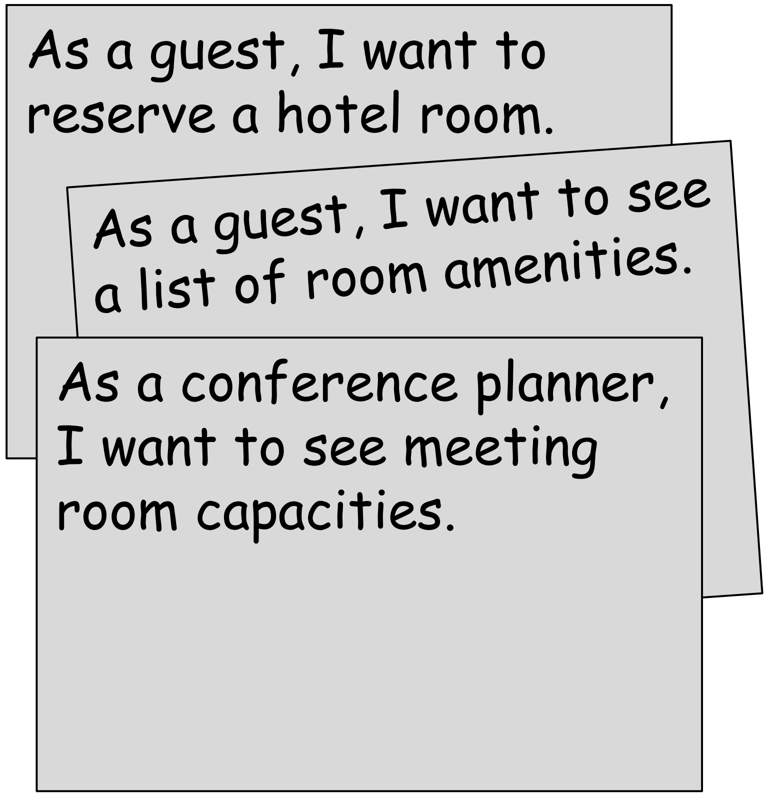
Agile method: less writing, more talking

Users get what they want.

Specifying Needs

User story:

written description of what a user wants to achieve with the system


As a guest, I want to reserve a hotel room.

As a guest, I want to see a list of room amenities.

As a conference planner, I want to see meeting room capacities.

on index cards

Typical forms:

As a «user role»,
I want «goal».

As a «user role»,
I want «goal»,
so that «reason».

Defining User Stories

Tips:

describe *what* not *how*

- avoid technical details or choices of technologies,
  unless it is a development constraint

avoid epics for near-term needs

- better to split up huge stories into more, smaller stories (but not too small)

Defining User Stories

Tips:

prioritize user stories

- discuss with the user what they find of most value, and stage development on that first

can attach an effort estimate to complete

- normally sized to take days, not many weeks

use stories to plan development tasks

- create work items in the iteration plan

“SMART Work Items”

S	Specific
M	Measurable
A	Achievable
R	Relevant
T	Time Boxed

Link:
http://xp123.com/articles/invest-in-good-stories-and-smart-tasks/

“INVEST in Good Stories”

I	Independent
N	Negotiable
V	Valuable
E	Estimable
S	Small
T	Testable

Link:
http://xp123.com/articles/invest-in-good-stories-and-smart-tasks/

Testable User Stories

Front of the card:

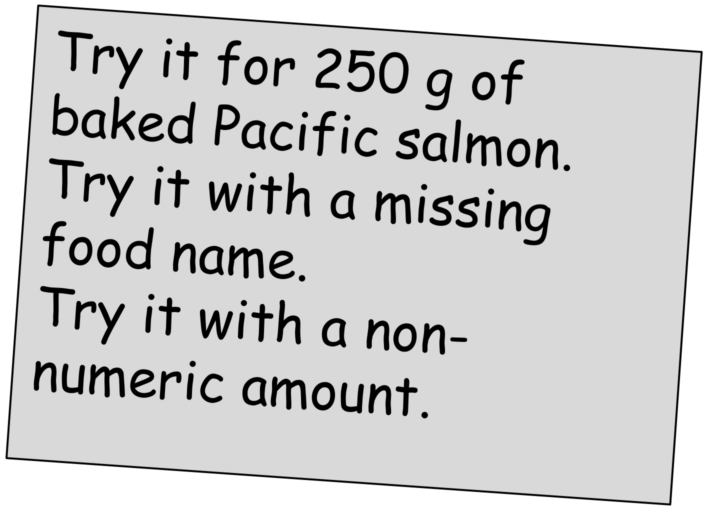
As a meal planner, I want to see nutrition information for a given amount of a given food.

Testable User Stories

Back of the card:

Link:
http://xprogramming.com/articles/expcardconversationconfirmation/

acceptance tests

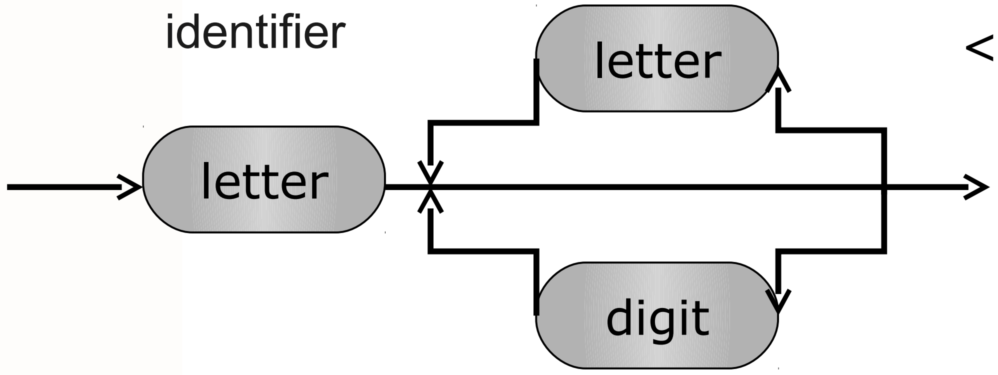


Try it for 250 g of baked Pacific salmon.
Try it with a missing food name.
Try it with a non-numeric amount.

Augmenting Requirements

Can add other descriptions, for example:

- use cases to user stories
- data schemas
- sample input and output
- user interface mockups and storyboards
- grammars (language syntactic/lexical structure)



```
<identifier> ::=
     <letter> { <letter> | <digit> }
```

State Models

Modeling behavior:

used in formally modeling the behavior of a specific object in response to external events

UML State Diagram

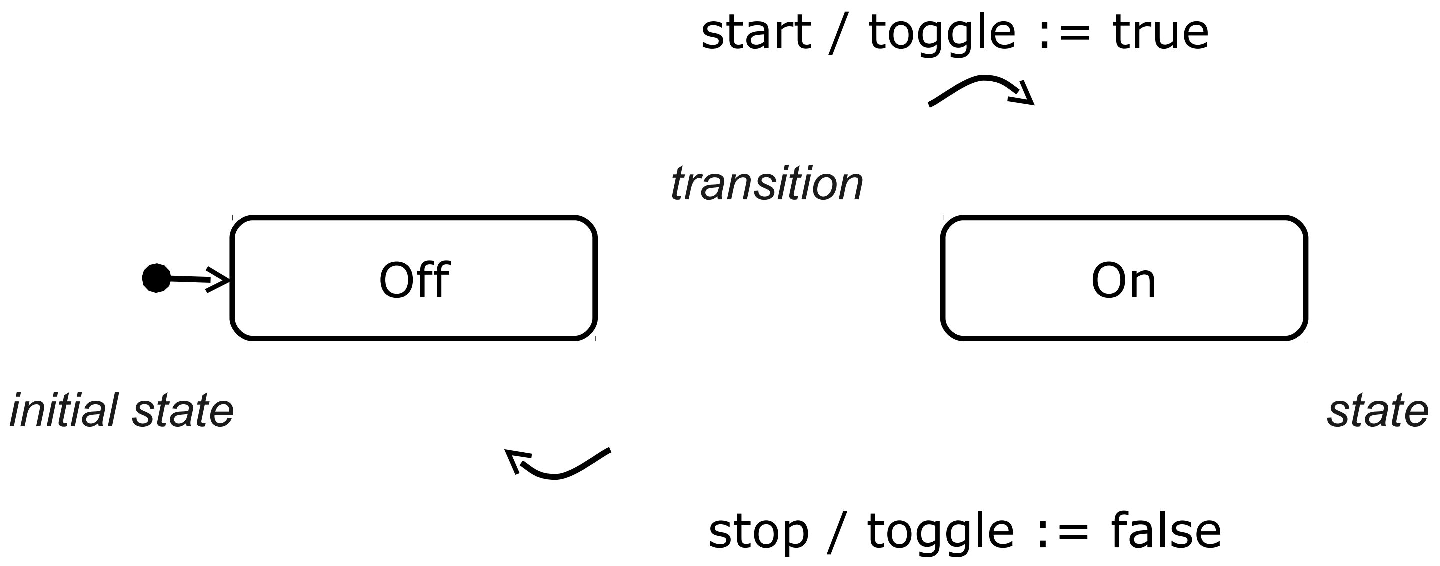
Modeling behavior:

states in which something can be in

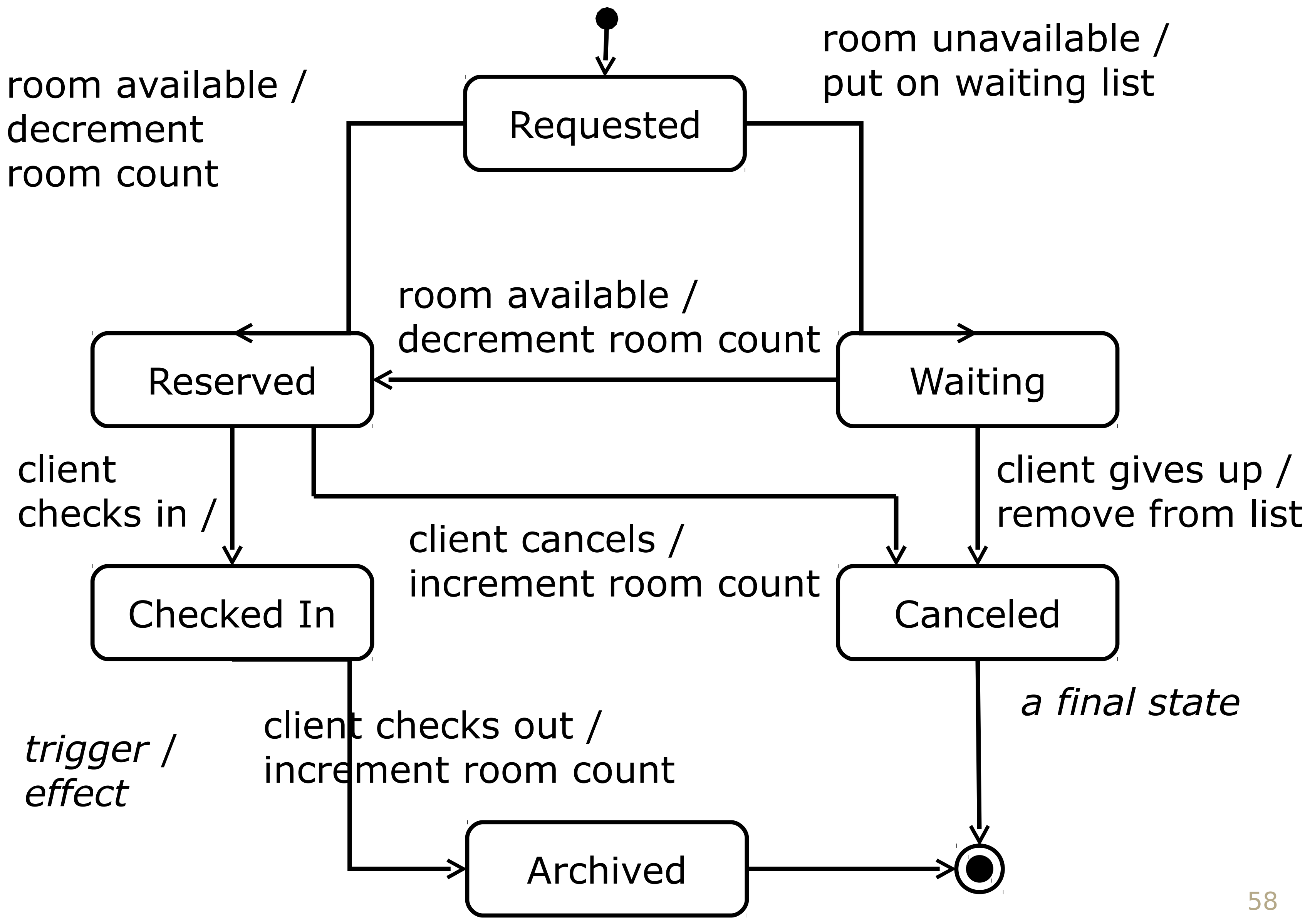
- a situation represented by attribute values

directed *transitions* between states

- triggered by events, input, time, messages, etc.



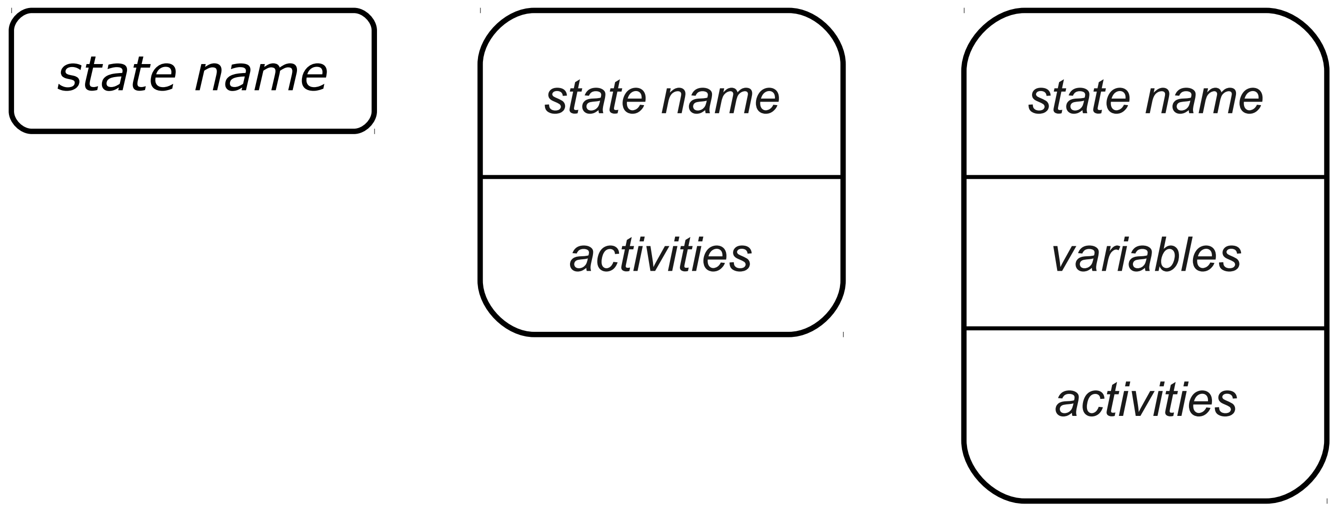
UML State Diagram



Requested
room available / decrement room count
room unavailable / put on waiting list
room available / decrement room count
Reserved
Waiting
client checks in /
client cancels / increment room count
client gives up / remove from list
Checked In
Canceled
a final state
trigger / effect
client checks out / increment room count
Archived

UML State Diagram

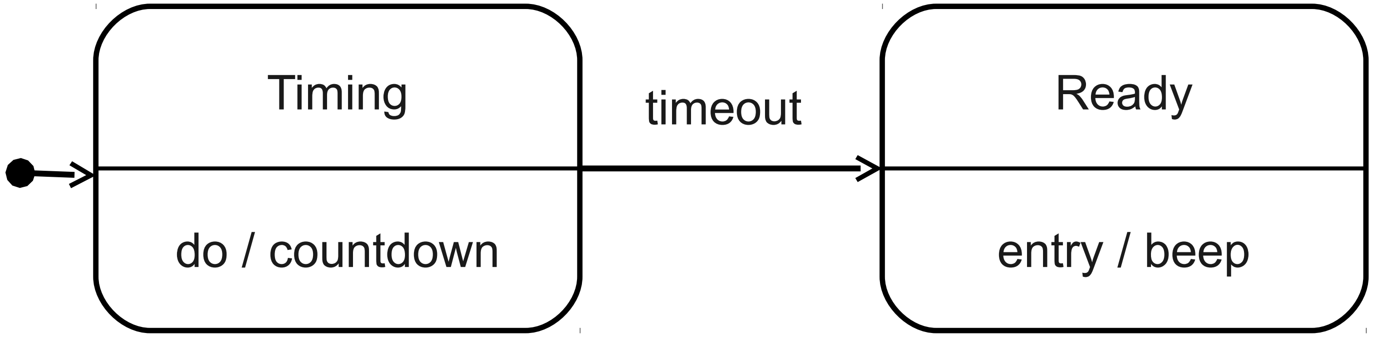
States:



UML State Diagram

Activities in states:

entry / *action*	perform *action* when entering state
do / *action*	perform *action* while in state
exit / *action*	perform *action* when exiting state

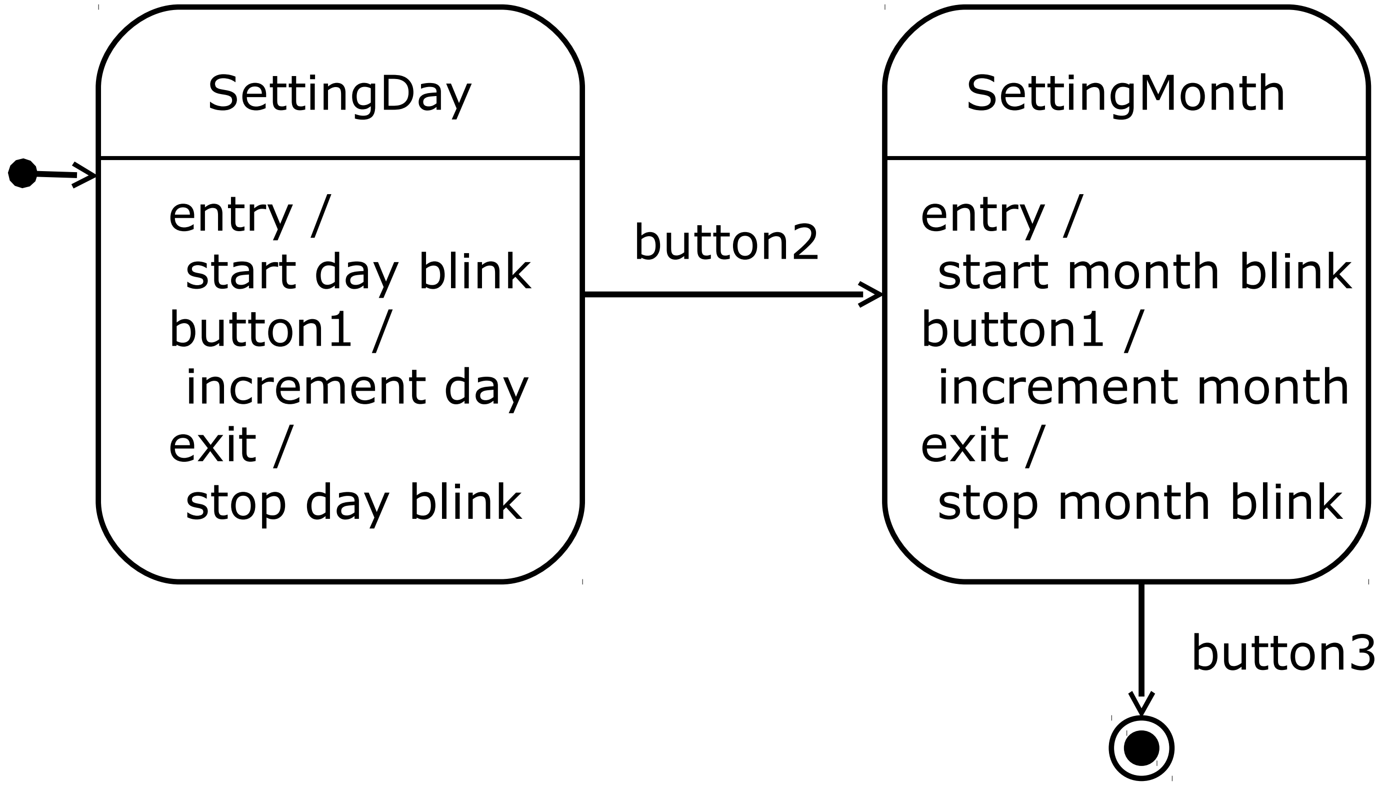


UML State Diagram

Activities in states:

also called an internal transition

trigger / action	when *trigger* occurs, perform *action*



UML State Diagram

Transitions:

general form of transition label

trigger [*guard*] / *effect*

if in a current state,
and *trigger* occurs,
and *guard* constraint (if any) is true …

then perform state exit actions (if any),
perform corresponding transition *effect* (if any),
perform new state entry actions (if any);

otherwise, stay at current state

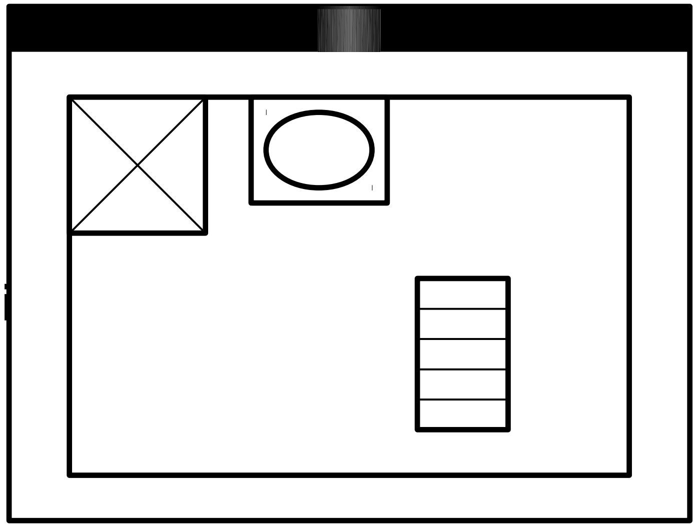
Room Planner

Canvas:
    to place and move items

Mouse events:
    click

    press/drag/release

Item type menu:
    choices of fixtures and furni



Room Planner

Placing an item:
user clicks on canvas outside any item

- system shows the item type menu

user chooses an item type from the menu

- system hides the item type menu

user clicks on the canvas

- system draws item of the chosen type at the mouse location

Room Planner

Moving an item:

- user presses inside an item
    - system highlights item
- user drags item
    - system shows moving item
- user releases mouse
    - system puts item at new location
    - system removes item highlighting

Room Planner

States:

waiting

- nothing happening

moving

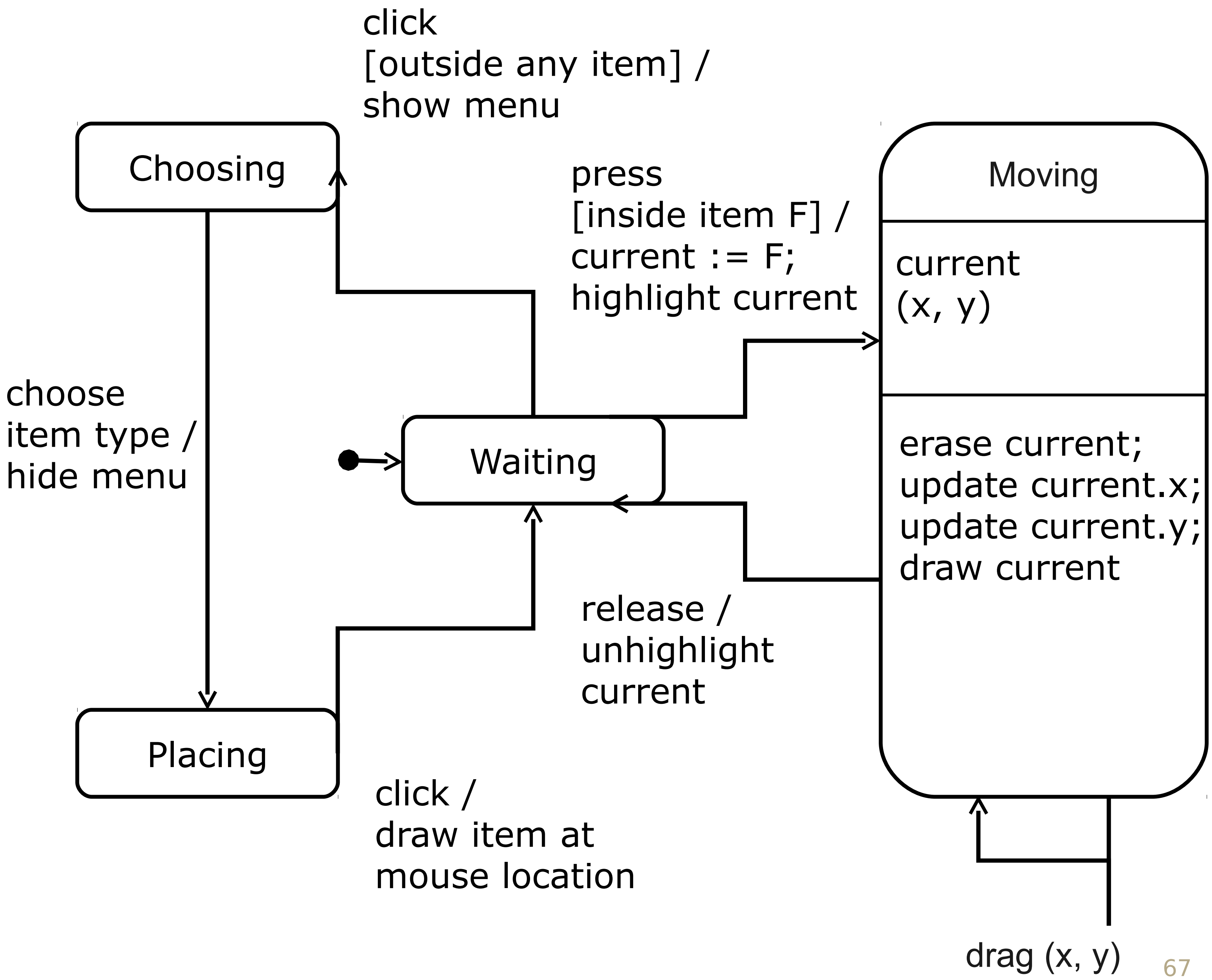
- moving item to new position

choosing

- choosing item type from menu

placing

- placing chosen shape


click
[outside any item] /
show menu
Choosing
press
[inside item F] /
current := F;
highlight current
Moving
current
(x, y)
choose
item type /
hide menu
Waiting
erase current;
update current.x;
update current.y;
draw current
release /
unhighlight
current
Placing
click /
draw item at
mouse location
drag (x, y)

UML State Diagram

Tips:

check for completeness

- states reachable?
- missing transitions?
- events not considered?
- unforeseen situations?

check for dangerous situations

- e.g., exiting without having saved edits

UML State Diagram

Tips:

check for consistency

- similar interactions have similar effects?
- effects are visible and give good feedback?

aid the user

- is undo appropriate, in a given state?
- is cancel or escape appropriate?
- is invoking help appropriate?