Embedded Computing

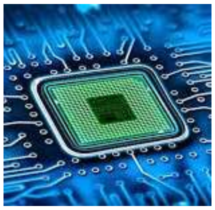
COEN 421/6341: Embedded Systems Design

Outline

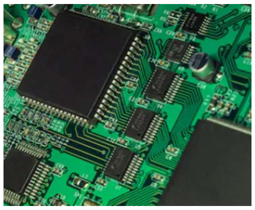
- Introduction
- Characteristics
- Challenges
- Performance
- Embedded system design process
- Design example: model train controller

Introduction

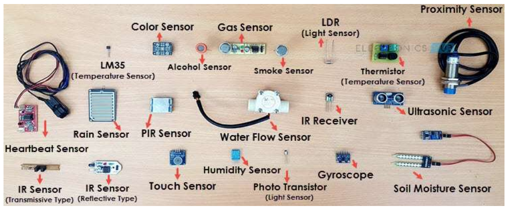
- Embedded systems
  - It is a **microcomputer system** embedded in a larger system and **designed for one or two dedicated services**
- Basic components



Microprocessor or microcontroller



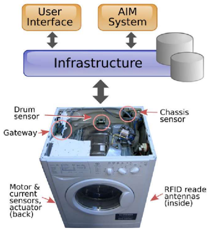
Program and Data Memory

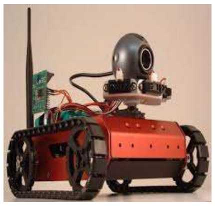


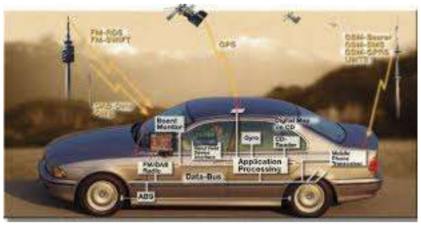
Input/output devices

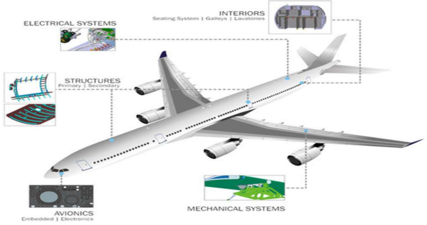
Introduction

Examples of Embedded Systems







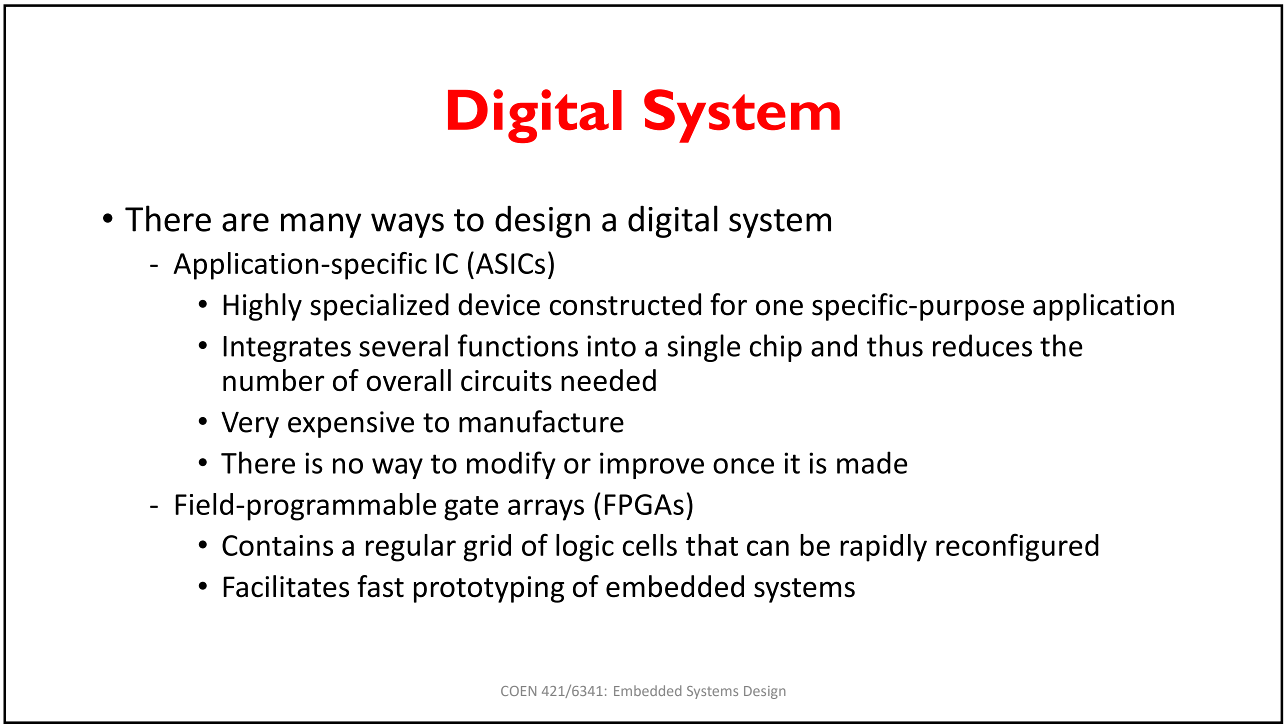


Introduction

- Embedding Computers
  - They can be tracked back to the earliest days of computing itself
    - e.g., Whirlwind, a computer designed at MIT in the late 1940s and early 1950s
      - Vacuum tube computer
      - Used to drive a flight simulator for training bomber crews
      - Real-time operation and control of an aircraft simulator
      - Would continually update a simulated instrument panel based on control inputs from the pilots

Introduction

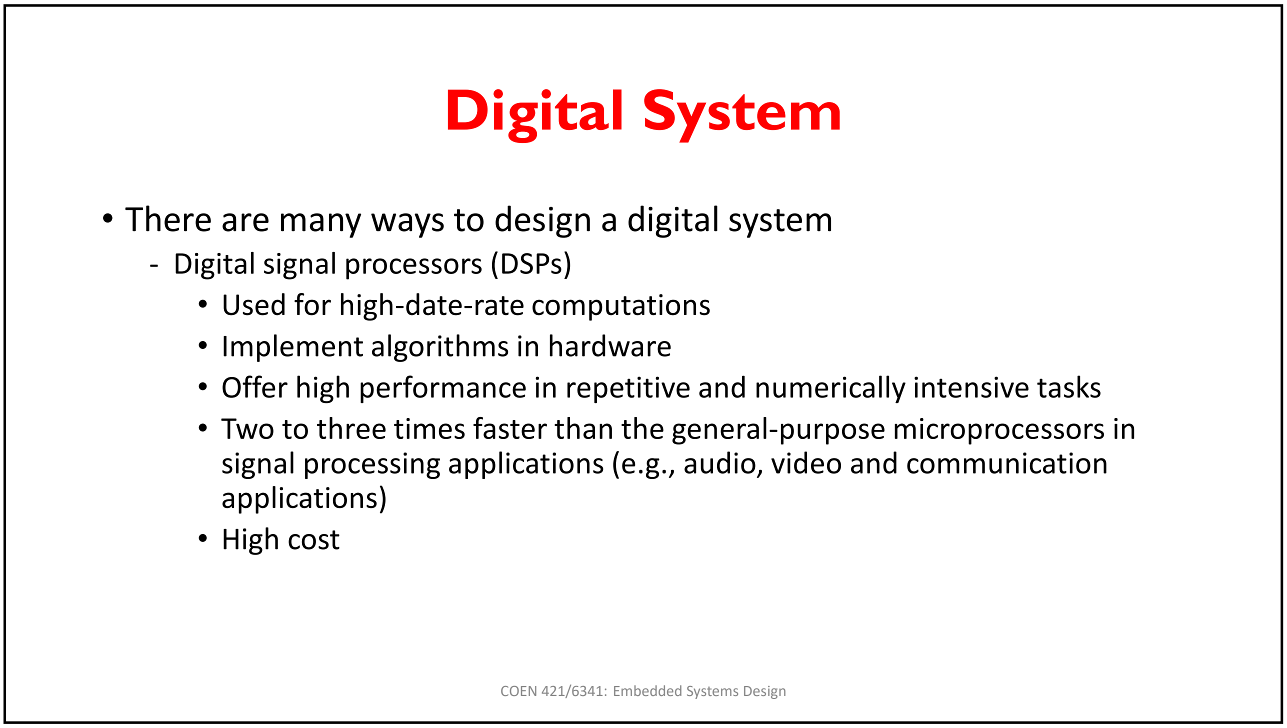
- Embedded systems
  - Used to replace mechanical or human controllers
    - Some control applications might need to use complex and sophisticated control algorithms to achieve the required performance
    - e.g., automobile industry
      - Needed for efficient solutions for low fuel consumption and less pollutant vehicles



Digital System

- There are many ways to design a digital system
  - Application-specific IC (ASICs)
    - Highly specialized device constructed for one specific-purpose application
    - Integrates several functions into a single chip and thus reduces the number of overall circuits needed
    - Very expensive to manufacture
    - There is no way to modify or improve once it is made
  - Field-programmable gate arrays (FPGAs)
    - Contains a regular grid of logic cells that can be rapidly reconfigured
    - Facilitates fast prototyping of embedded systems

7



Digital System

- There are many ways to design a digital system
  - Digital signal processors (DSPs)
    - Used for high-date-rate computations
    - Implement algorithms in hardware
    - Offer high performance in repetitive and numerically intensive tasks
    - Two to three times faster than the general-purpose microprocessors in signal processing applications (e.g., audio, video and communication applications)
    - High cost

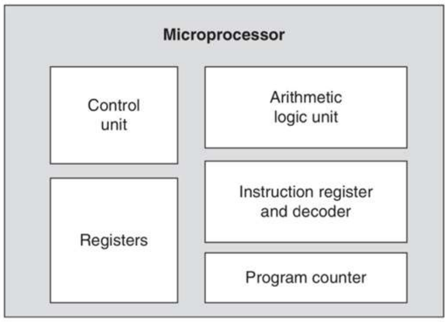
8

Digital System

- There are many ways to design a digital system
  - Application-specific instruction set processors (ASIPs)
    - Intermediary solution between ASICs and programmable processors
    - Consists of custom integrated circuitry that is integrated with an instruction set tailored to benefit a specific application
    - Provides a trade-off between the flexibility of a general-purpose processor and the performance of an ASIC

Digital System

- There are many ways to design a digital system
  - Microprocessors (general-purpose microprocessors)
    - Designed to perform arithmetic and logic operations that make use of a small storage (registers)
    - It has a control unit that is responsible for directing the processor to carry out stored program instructions
    - All the instructions that are fetched from memory are stored in the instruction register as binary values

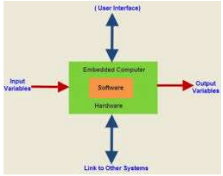


Why Use Microprocessors?

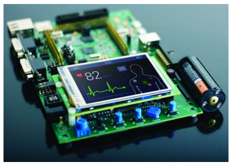
- To design our own custom logic for an ES project takes more time than using a predesigned instruction set processors
- Facilitates the design process
  - One team might be designing the board containing the microprocessor, I/O devices, memory ….
  - Another team can be writing programs at the same time
- Make easier to design families of products
  - New features can be added by adding code without changing the hardware
  - When the hardware must be redesigned for next-generation products, it may be possible to reuse the software, reducing development time and cost

The Big Picture

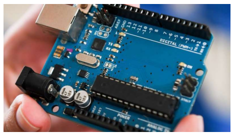
Design of ES



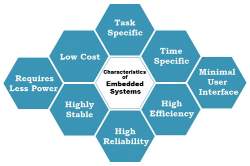
Embedded system



Why microprocessors / microcontrollers are used for control, user interface, signal processing ….



Unique characteristics



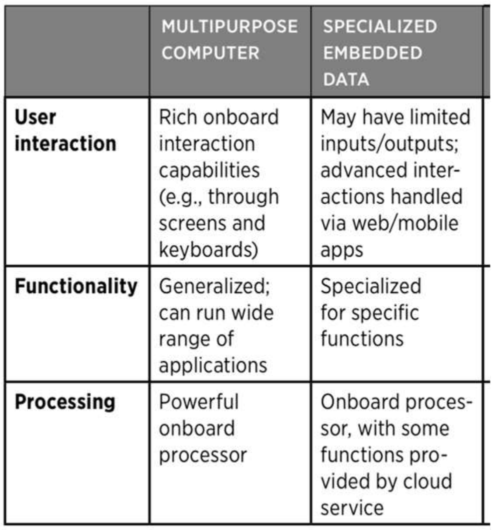
Unique challenges

Computing Platforms

- Why don't we use PCs for embedded computing?
  - Real-time performance requirements
  - Need of low power and low-cost architectures
  - PCs are designed to satisfy a broad mix of computing requirements and to be very flexible
    - This increases the complexity and price of the components
  - They cause the processor and other components to use more energy to perform a given function

Computing Platforms

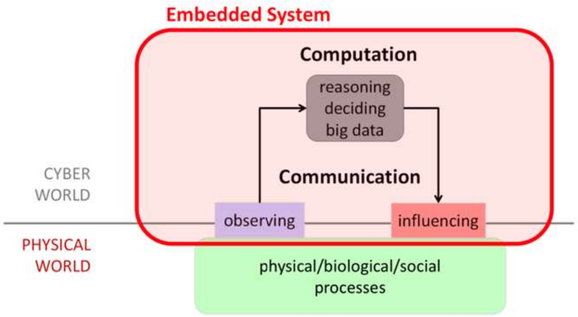
- Computer:
  - Stored program machine that fetches and executes instructions from a memory
- Different types of systems:
  - Different types of devices attached to the computer and load it with different types of software
- General-Purpose Computer
- Embedded Computer



	MULTIPURPOSE COMPUTER	SPECIALIZED EMBEDDED DATA
User interaction	Rich onboard interaction capabilities (e.g., through screens and keyboards)	May have limited inputs/outputs; advanced interactions handled via web/mobile apps
Functionality	Generalized; can run wide range of applications	Specialized for specific functions
Processing	Powerful onboard processor	Onboard processor, with some functions provided by cloud service

Cyber-Physical Systems

- Combines physical devices (plant) with computers that control the plant
- ES is the cyber part of the CPS



Embedded System

Computation

reasoning
deciding
big data

Communication

CYBER
WORLD

PHYSICAL
WORLD

observing

influencing

physical/biological/social
processes

Credits: Prof. Lothar Thiele

Use feedback to influence the dynamics of the physical world by taking smart decisions in the cyber world

15

ES Characteristics

- ES is different from GP systems
- Functionality is important in ES and GP systems
- However, ES must meet many other constrains:
- Sophisticated Functionalities
  - Complex algorithms
    - Operations/tasks to be performed might be very sophisticated (e.g., complicated filtering functions to optimize the performance of an automobile engine)
  - User interface
    - Complex user interfaces including multiple menus and many options might be used (e.g., moving maps in GPS navigation)

16

ES Characteristics

- Must often be performed to meet deadlines
  - Real-time
    - Tasks in ESs often have real-time requirements
  - Multi-rate
    - Many ESs have several real-time tasks going on at the same time (e.g., control operations that run at slow rates and others that run at high rates)
- Cost of various sorts
  - Manufacturing costs
    - Total cost of building the system, which is determined by many factors (e.g., microprocessor, amount of memory, types of I/O devices …)
  - Power and energy
    - Several ESs are energy-constrained and powered by batteries
    - When not battery-powered, energy consumption is yet critical

Safety and Security

- They are major concerns for ES designers
  - ESs are in safety-critical and other important systems that people use every day
    - e.g., cars, airplanes, medical equipment, …
  - Many ESs are connected to the Internet
    - Make them more vulnerable to attacks
  - Unsafe operation of ES makes security problems even more important than in information systems

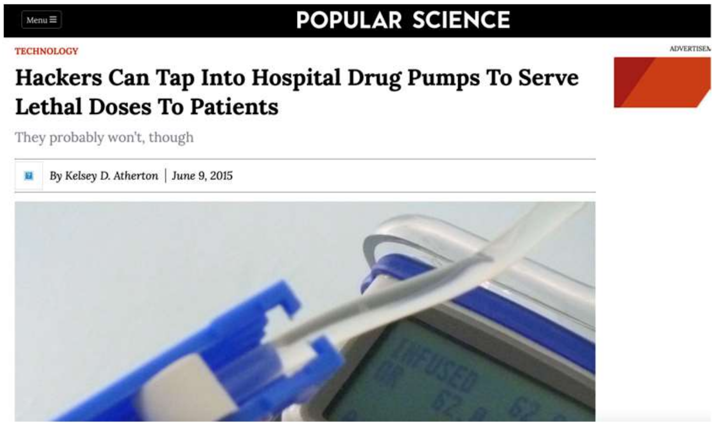
Safety and Security

- They are major concerns for ES designers
- Security
  - Relates to the system’s ability to prevent malicious attacks
- Integrity
  - Refers to data’s proper values
  - Attacker should not be able to change a value
- Privacy
  - Refers to unauthorized release of data
- Safety
  - Can be compromised from malicious external activity, poor system design, or improper use of the equipment

Safety and Security

- ESs are in safety-critical and other important systems that people use every day (e.g., cars, airplanes, industry, medical equipment, …)
- Many ESs are connected to the Internet
  - This makes them more vulnerable to attacks by malicious people
- Unsafe operation makes security problems even more important than in information systems
- Security and safety cannot be bolted on – they must be baked in

Safety and Security



POPULAR SCIENCE

TECHNOLOGY

ADVERTISEM

Hackers Can Tap Into Hospital Drug Pumps To Serve Lethal Doses To Patients

They probably won't, though

By Kelsey D. Atherton | June 9, 2015

https://www.popsci.com/hacked-hospital-drug-pumps-could-kill-patients-their-beds/

- The LifecarePCA is a drug infusion machine for hospital patients, designed to correctly administer the right dosage of medicine straight into the arm of a person in need
- When it works as it should, mechanical precision thwarts human error and eases care
- If someone with malicious intent were to gain control of the system, however, they could instead inject all of a painkiller vial into a victim at once
- Recent work by security researcher Billy Rios shows that the LifecarePCA system, as well as five other models of drug delivery machines by Hospira, is vulnerable to hacks that can change the dosage of the drug delivered

Challenges for ES Design

- External constraints are one important source of difficulty in ES design
- Important problems that must be taken into consideration

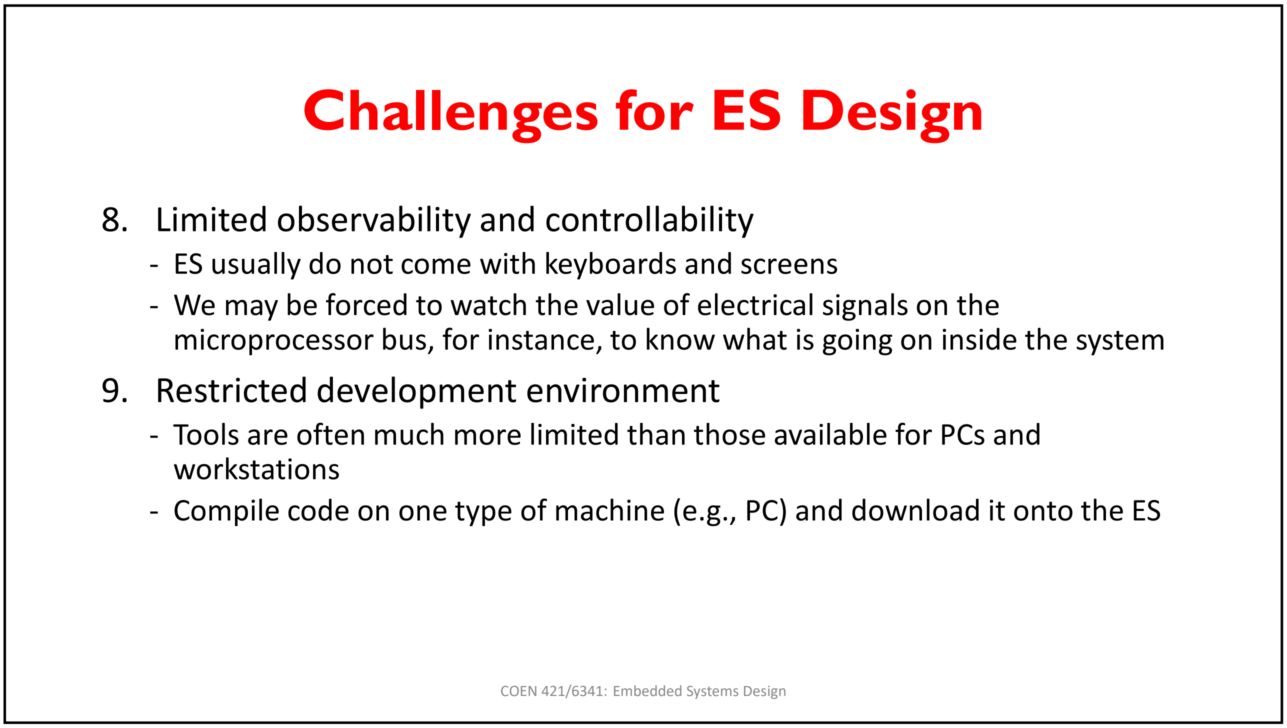
1. How much hardware do we need?
   - Hardware choice is important as it might impact ES performance and manufacturing cost
     - Microprocessor
     - Amount of memory
     - Peripheral devices
     - …

Challenges for ES Design

2. How do we meet deadlines?
   - “Brute force” approach: speed up the hardware so that the program runs faster
   - However, real-time != super fast
3. How do we minimize power consumption?
   - It is very important in battery-powered ES
   - In non-battery applications, excessive power consumption can increase heat dissipation
4. How we design for upgradeability?
   - Add new features by changing software

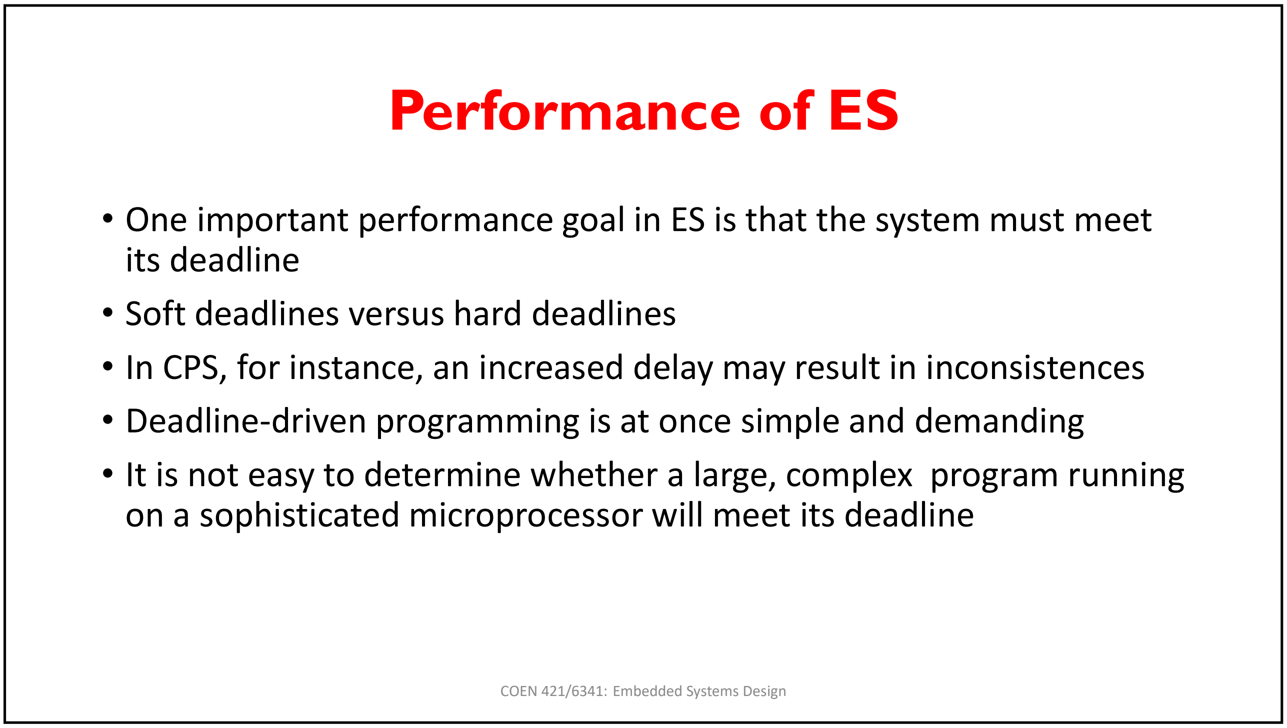
Challenges for ES Design

5. Does it really work?
   - Must guarantee that the system is reliable
   - Reliability must be considered since the design phase
   - If we wait until we have a running system and try to eliminate the bugs, we will be too late and it will be too expensive to fix them
6. Is it secure?
   - Attacks on embedded systems can not only gain access to personal data but also cause the physical systems controlled by those embedded computers to perform dangerous acts
7. Complex testing
   - Might have to run a real machine to generate the proper data
   - Timing of data is often important, meaning that we cannot separate the testing of an embedded computer from the machine in which it is embedded



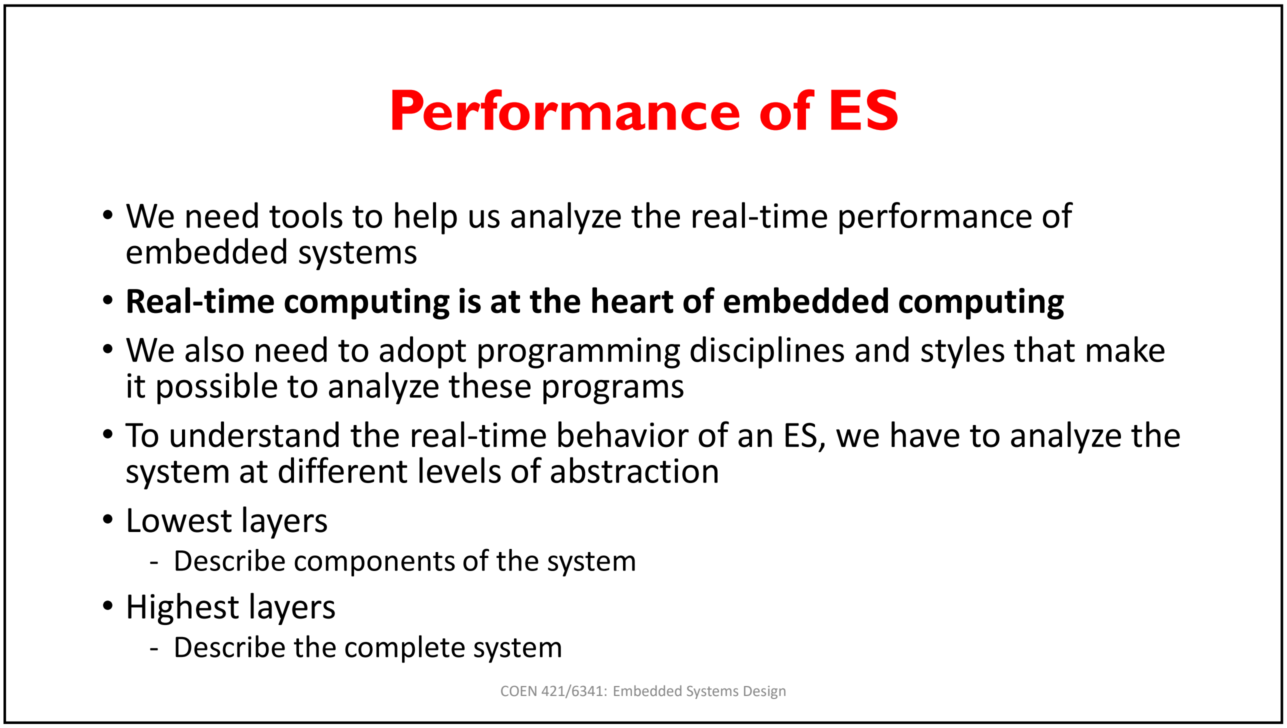
Challenges for ES Design

8. Limited observability and controllability
   - ES usually do not come with keyboards and screens
   - We may be forced to watch the value of electrical signals on the microprocessor bus, for instance, to know what is going on inside the system
9. Restricted development environment
   - Tools are often much more limited than those available for PCs and workstations
   - Compile code on one type of machine (e.g., PC) and download it onto the ES



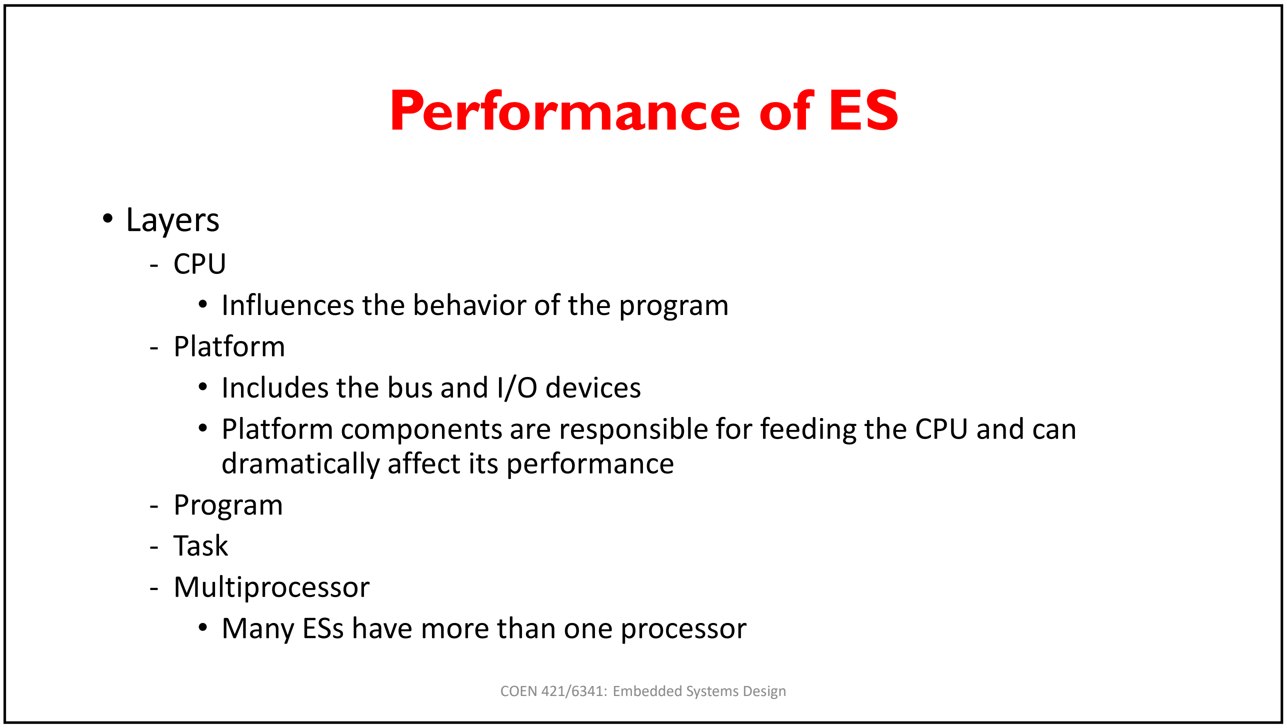
Performance of ES

- One important performance goal in ES is that the system must meet its deadline
- Soft deadlines versus hard deadlines
- In CPS, for instance, an increased delay may result in inconsistences
- Deadline-driven programming is at once simple and demanding
- It is not easy to determine whether a large, complex program running on a sophisticated microprocessor will meet its deadline



Performance of ES

- We need tools to help us analyze the real-time performance of embedded systems
- **Real-time computing is at the heart of embedded computing**
- We also need to adopt programming disciplines and styles that make it possible to analyze these programs
- To understand the real-time behavior of an ES, we have to analyze the system at different levels of abstraction
- Lowest layers
  - Describe components of the system
- Highest layers
  - Describe the complete system



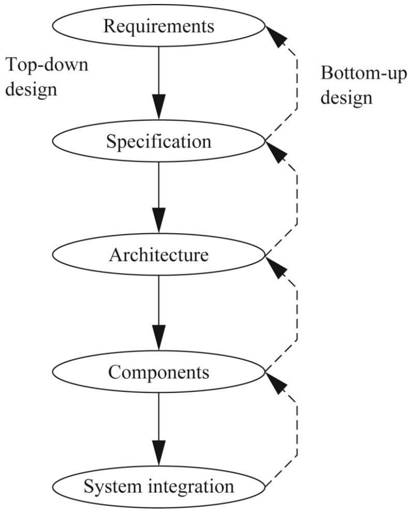
Performance of ES

- Layers
  - CPU
    - Influences the behavior of the program
  - Platform
    - Includes the bus and I/O devices
    - Platform components are responsible for feeding the CPU and can dramatically affect its performance
  - Program
  - Task
  - Multiprocessor
    - Many ESs have more than one processor

ES Design Process

- Design methodology:
  - Allow us to keep a scorecard on a design to ensure that we have done everything we need to do
  - Allow us to develop computer-aided design tools
    - Break the development process into manageable steps
  - Facilitate communication among members of a design team

ES Design Process



Top-down design:
start from most abstract description;
work to most detailed.
Bottom-up design:
work from small components to big system.
Real design uses both techniques.

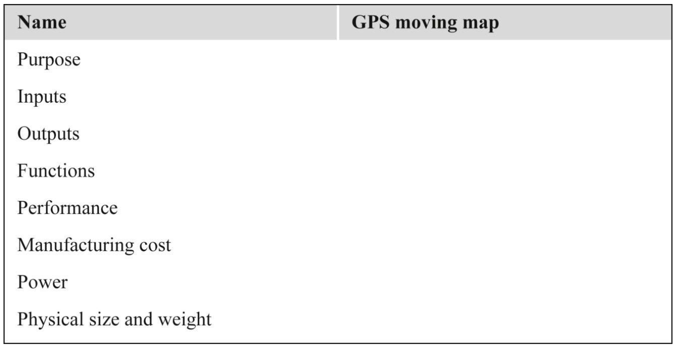
At each level of abstraction, we must:
analyze the design to determine characteristics of the current state of the design;
refine the design to add detail.

ES Design Process: Requirements

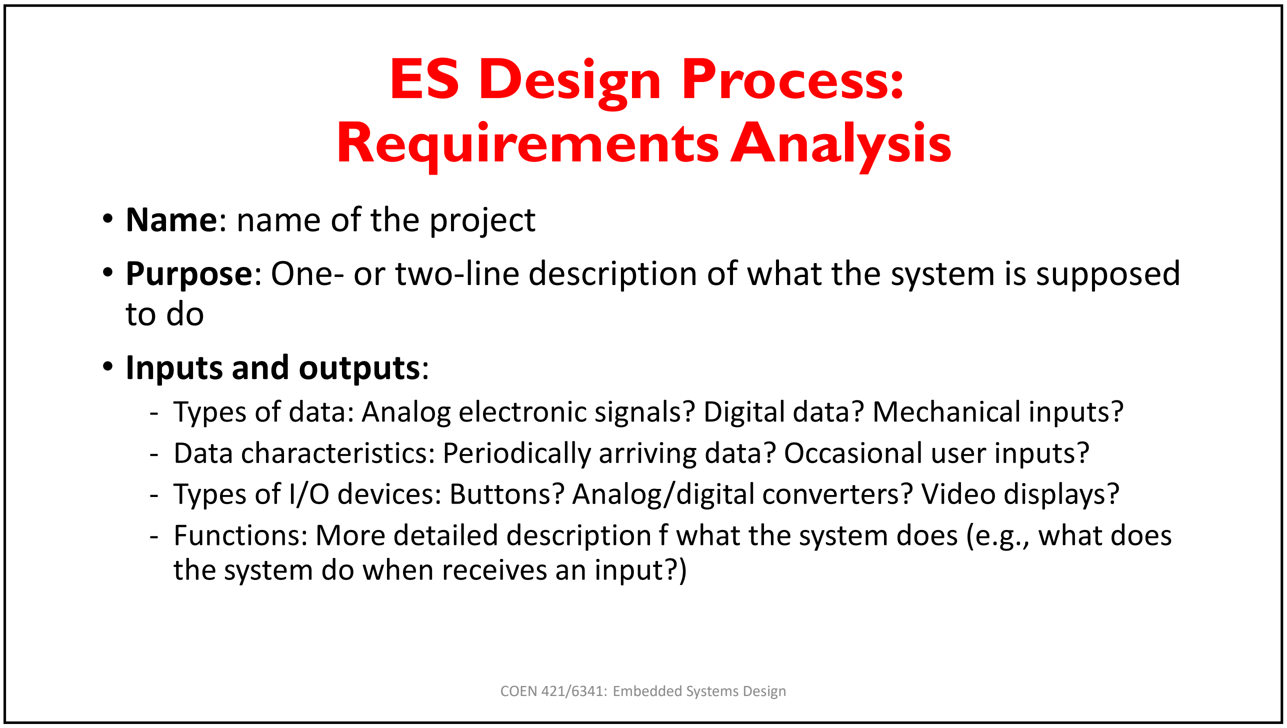
- Information regarding what is the system we are designing
- Generally done in two phases
  - 1. Gather informal description from the customers
  - 2. Refine the requirements into a specification that contains enough information to begin designing the system architecture
- Requirements
  - Functional
    - Capture the basic functions of the embedded system
  - Nonfunctional
    - Define some of the system attributes (performance, cost, physical size and weight, power consumption, security …)

ES Design Process: Requirements Analysis

- It can be complex and time consuming
- Capturing a relatively small amount of information in a clear, simple format is a good start toward understanding system requirements



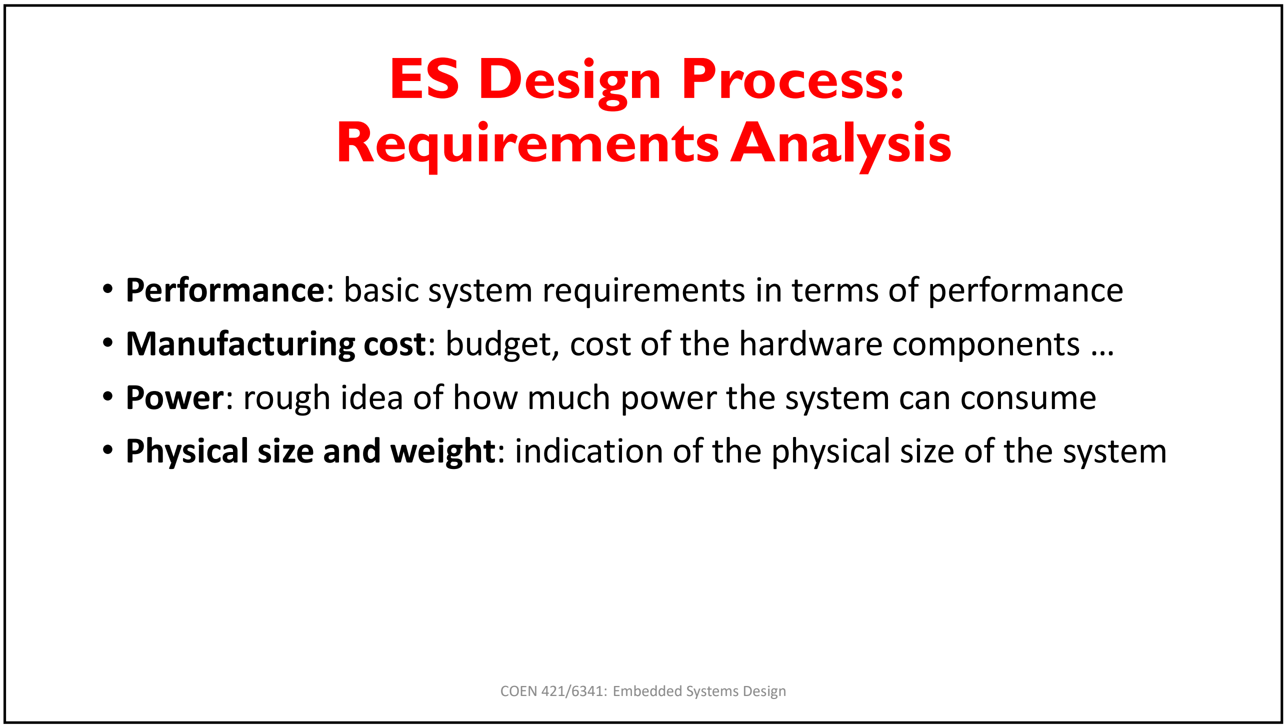
Name	**GPS moving map**
Purpose	
Inputs	
Outputs	
Functions	
Performance	
Manufacturing cost	
Power	
Physical size and weight	



ES Design Process: Requirements Analysis

- **Name**: name of the project
- **Purpose**: One- or two-line description of what the system is supposed to do
- **Inputs and outputs**:
  - Types of data: Analog electronic signals? Digital data? Mechanical inputs?
  - Data characteristics: Periodically arriving data? Occasional user inputs?
  - Types of I/O devices: Buttons? Analog/digital converters? Video displays?
  - Functions: More detailed description f what the system does (e.g., what does the system do when receives an input?)

33



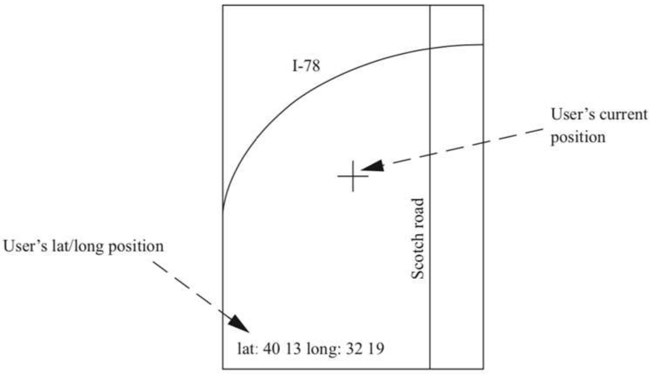
ES Design Process: Requirements Analysis

- **Performance**: basic system requirements in terms of performance
- **Manufacturing cost**: budget, cost of the hardware components …
- **Power**: rough idea of how much power the system can consume
- **Physical size and weight**: indication of the physical size of the system

34

ES Design Process: Requirements Analysis

- E.g.: Requirement analysis of a GPS Moving Map
  - Handheld device that displays for the user a map of the terrain around the user's current position
  - Display changes as the user changes position
  - Obtain position from the GPS



ES Design Process: Requirements Analysis

E.g.: Requirement analysis of a GPS Moving Map

- Functionality
  - Designed for highway driving and similar users
  - Should show major roads and other landmarks available in standard topographic databases
- User interface
  - Screen should have at least 400 x 600 pixel resolution
  - Device should be controlled by no more than three buttons
  - Menu system should pop up on the screen when buttons are pressed
- Performance
  - Map should scroll smoothly
  - Upon power-up, a display should take no more than 1s to appear
  - System should be able to verify its position and display the current map within 15s

ES Design Process: Requirements Analysis

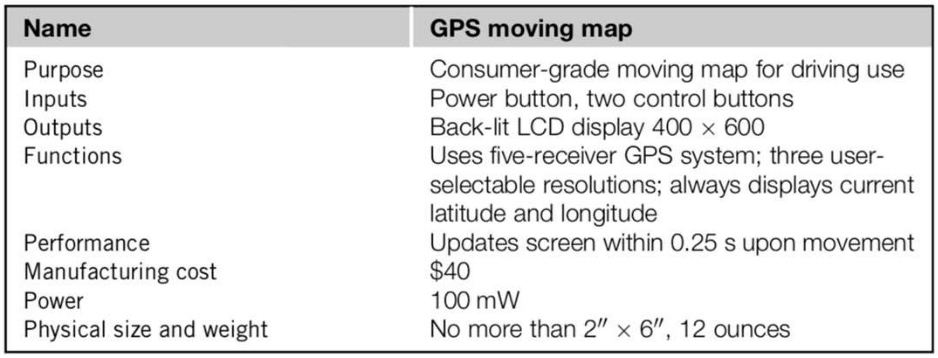
E.g.: Requirement analysis of a GPS Moving Map

- Cost
  - Selling cost price should be no more than $120
- Physical size and weight
  - Device should fit comfortably in the palm of the hand
- Power consumption
  - Should run for at least 8h on four AA batteries
  - At least 30min of those 8h comprising operation with screen on

37

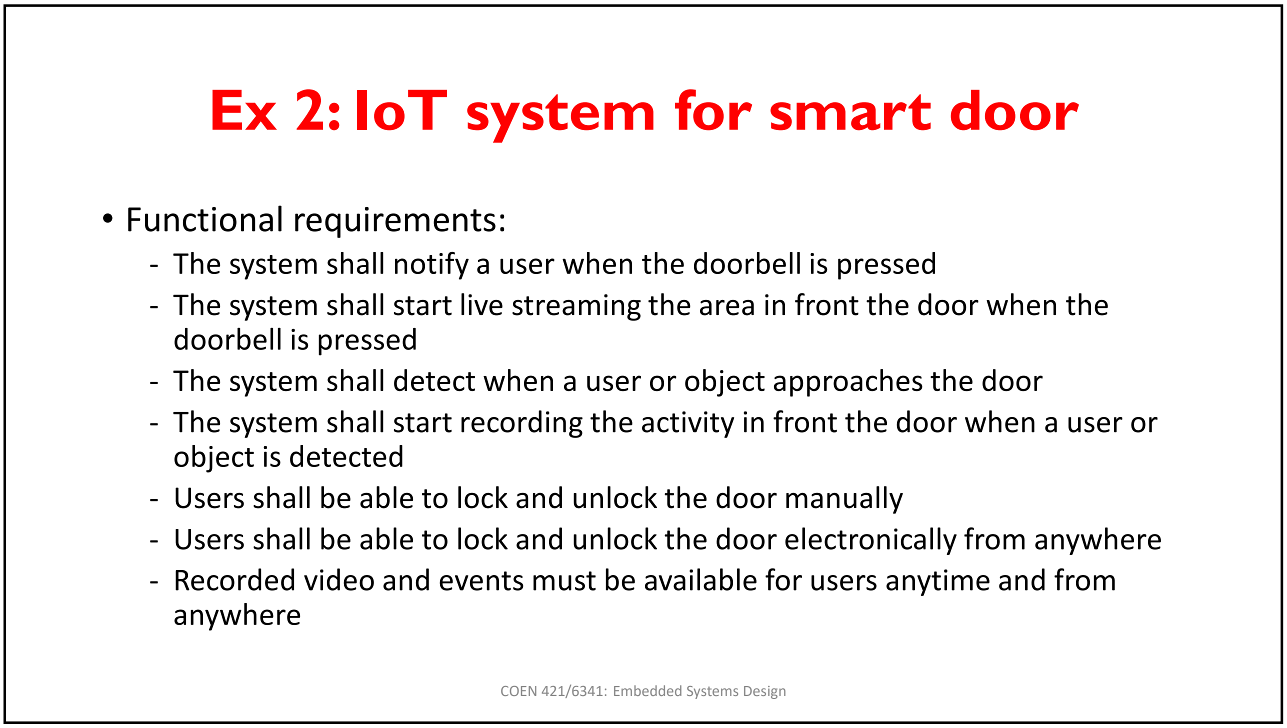
ES Design Process: Requirements Analysis

E.g.: Requirement analysis of a GPS Moving Map



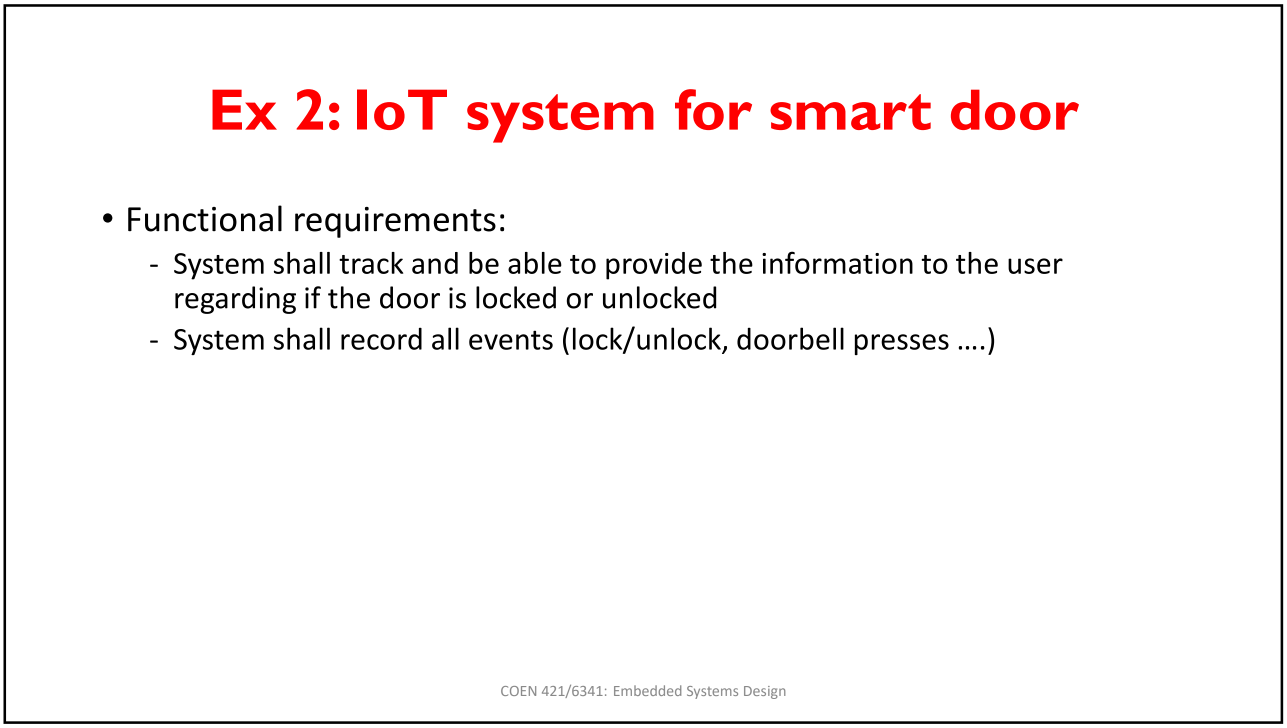
Name	GPS moving map
Purpose	Consumer-grade moving map for driving use
Inputs	Power button, two control buttons
Outputs	Back-lit LCD display 400 × 600
Functions	Uses five-receiver GPS system; three user-selectable resolutions; always displays current latitude and longitude
Performance	Updates screen within 0.25 s upon movement
Manufacturing cost	$40
Power	100 mW
Physical size and weight	No more than 2″ × 6″, 12 ounces

38



Ex 2: IoT system for smart door

- Functional requirements:
  - The system shall notify a user when the doorbell is pressed
  - The system shall start live streaming the area in front the door when the doorbell is pressed
  - The system shall detect when a user or object approaches the door
  - The system shall start recording the activity in front the door when a user or object is detected
  - Users shall be able to lock and unlock the door manually
  - Users shall be able to lock and unlock the door electronically from anywhere
  - Recorded video and events must be available for users anytime and from anywhere



Ex 2: IoT system for smart door

- Functional requirements:
  - System shall track and be able to provide the information to the user regarding if the door is locked or unlocked
  - System shall record all events (lock/unlock, doorbell presses ….)

Ex 2: IoT system for smart door

- Non-Functional requirements:
  - System shall not weight more than **X** grams (here you should have the value)
  - System shall not cost more than **Y** CAD
  - The dimensions of the system shall not be larger than **Z** cm x **A** cm

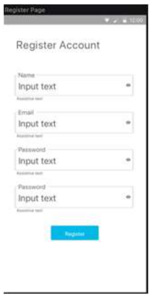
ES Design Process: Requirements Analysis

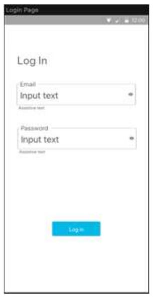
- After you write the requirements, you should check them for internal consistency
  - Did you forget to assign a function to an input or output?
  - Did you consider all the modes in which you want the system to operate?
  - Did you place an unrealistic number of features into a battery-powered, low-cost machine?

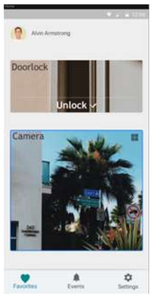
ES Design Process: Requirements Validation

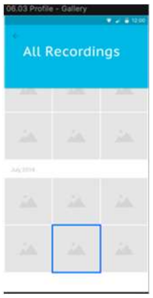
- Validating a set of requirements is a challenging task
- Requires understanding both what people want and how they communicate those needs
- One approach: build a mock-up
  - Sketch interfaces
  - Use canned data to simulate functionality in a restricted demonstration
  - Nonfunctional models of devices
  - At the end, the customer should be given a good idea of how the system will be used and how the user can react to it

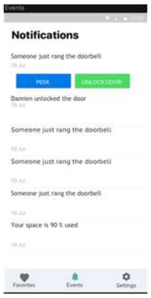
Ex 2: IoT system for smart door











Someone just rang the doorbell
19 Jul

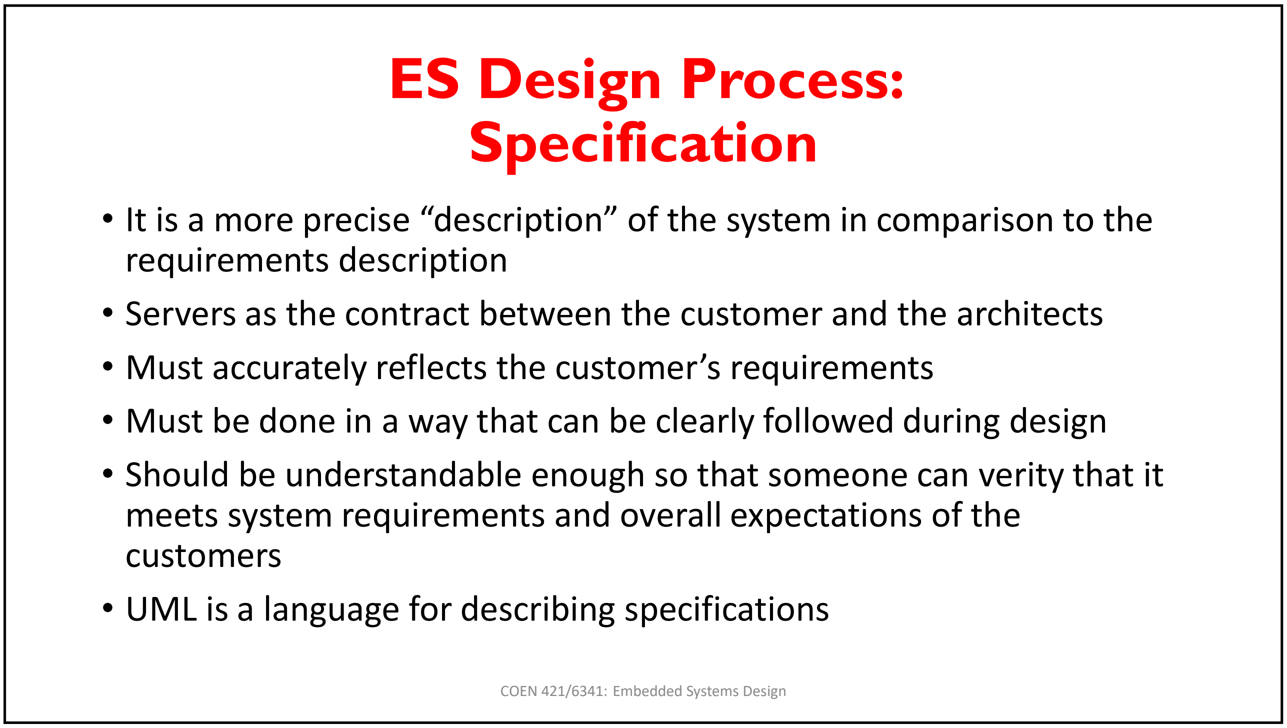
Damien unlocked the door
19 Jul

Someone just rang the doorbell
19 Jul

Someone just rang the doorbell
19 Jul

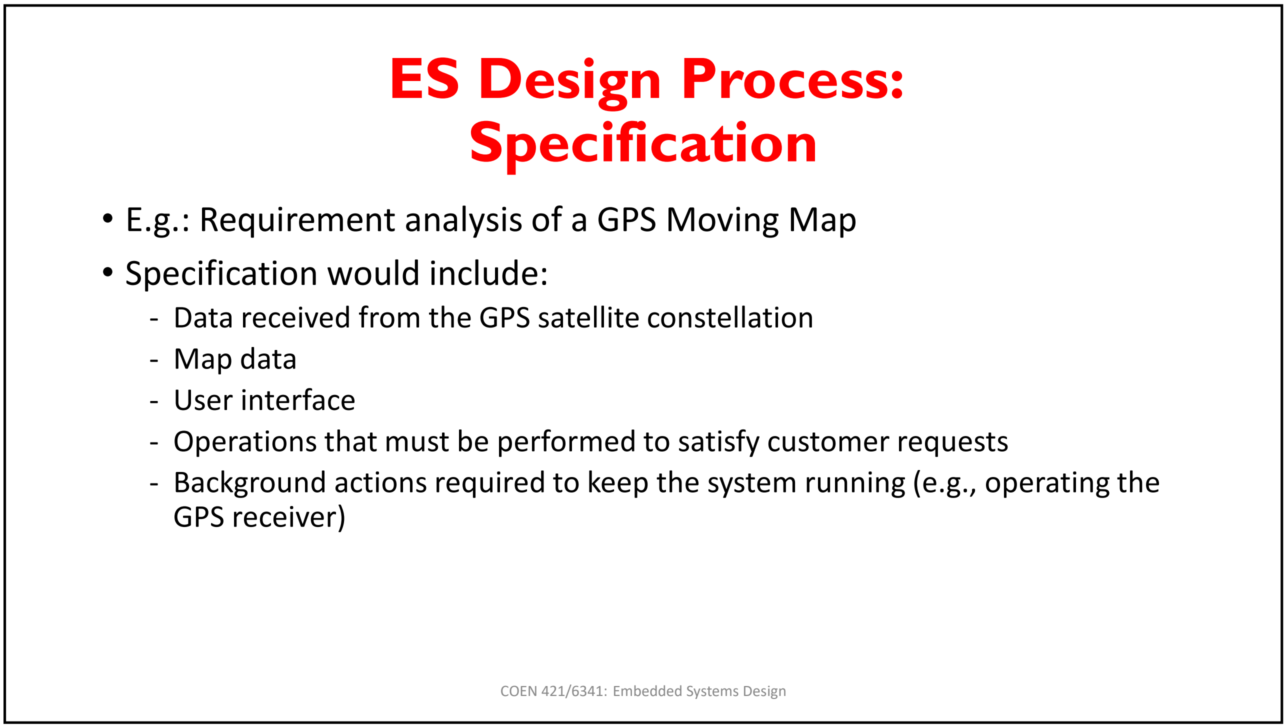
Someone just rang the doorbell
19 Jul

Your space is 90 % used
19 Jul



ES Design Process: Specification

- It is a more precise “description” of the system in comparison to the requirements description
- Servers as the contract between the customer and the architects
- Must accurately reflects the customer’s requirements
- Must be done in a way that can be clearly followed during design
- Should be understandable enough so that someone can verity that it meets system requirements and overall expectations of the customers
- UML is a language for describing specifications

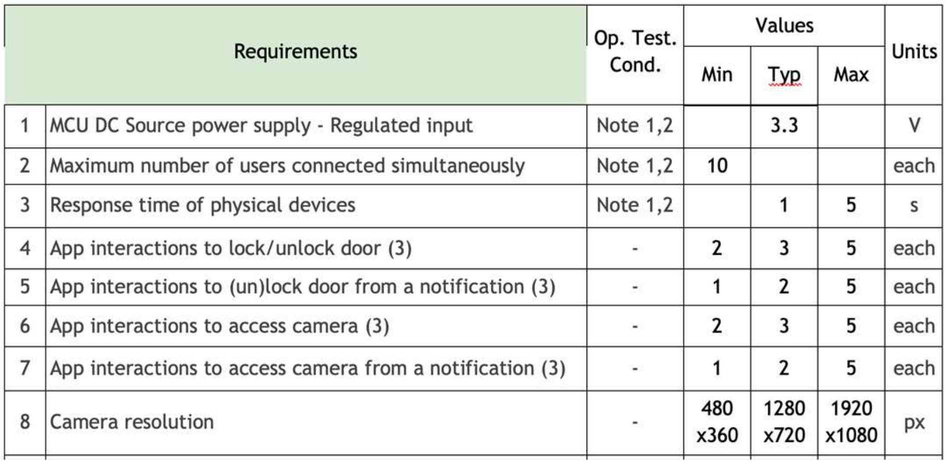


ES Design Process: Specification

- E.g.: Requirement analysis of a GPS Moving Map
- Specification would include:
  - Data received from the GPS satellite constellation
  - Map data
  - User interface
  - Operations that must be performed to satisfy customer requests
  - Background actions required to keep the system running (e.g., operating the GPS receiver)

Ex 2: IoT system for smart door

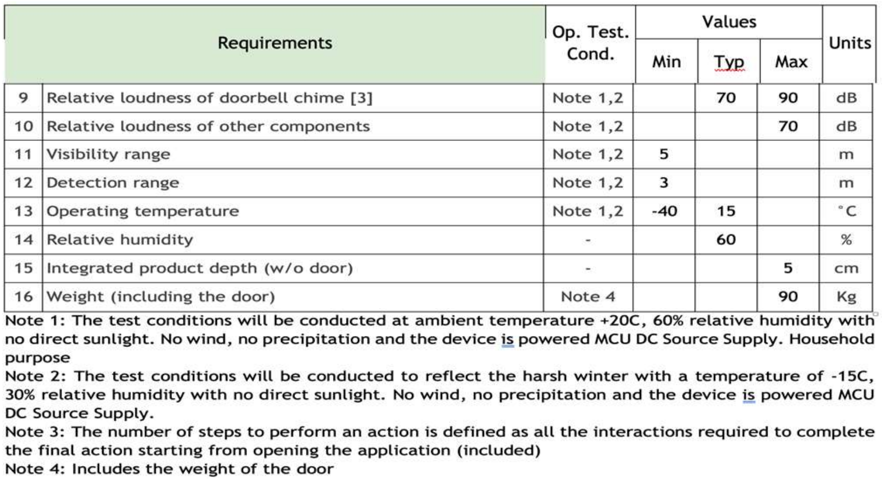
- Software and Hardware requirements



Requirements		Op. Test. Cond.	Values			Units
			Min	Typ	Max	
1	MCU DC Source power supply - Regulated input	Note 1,2		3.3		V
2	Maximum number of users connected simultaneously	Note 1,2	10			each
3	Response time of physical devices	Note 1,2		1	5	s
4	App interactions to lock/unlock door (3)	-	2	3	5	each
5	App interactions to (un)lock door from a notification (3)	-	1	2	5	each
6	App interactions to access camera (3)	-	2	3	5	each
7	App interactions to access camera from a notification (3)	-	1	2	5	each
8	Camera resolution	-	480 x360	1280 x720	1920 x1080	px

Ex 2: IoT system for smart door

- Software and Hardware requirements



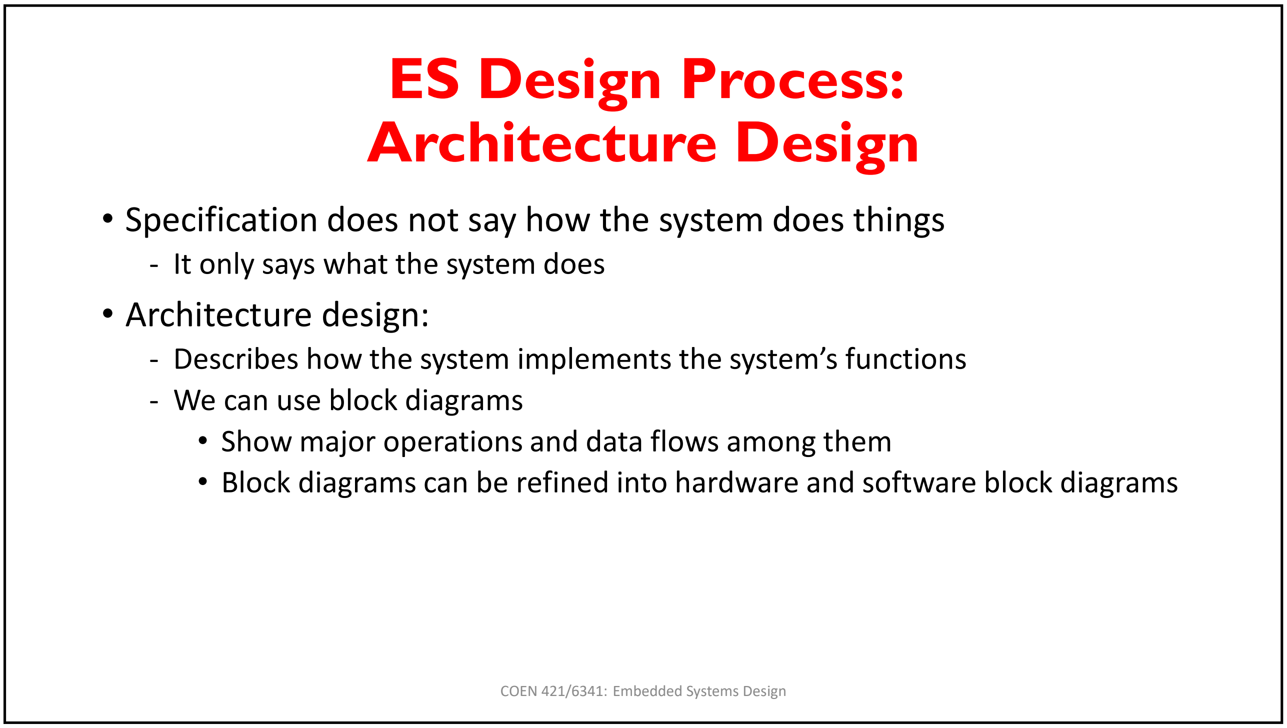
Requirements		Op. Test. Cond.	Values			Units
			Min	Typ	Max	
9	Relative loudness of doorbell chime [3]	Note 1,2		70	90	dB
10	Relative loudness of other components	Note 1,2			70	dB
11	Visibility range	Note 1,2	5			m
12	Detection range	Note 1,2	3			m
13	Operating temperature	Note 1,2	-40	15		°C
14	Relative humidity	-		60		%
15	Integrated product depth (w/o door)	-			5	cm
16	Weight (including the door)	Note 4			90	Kg

Note 1: The test conditions will be conducted at ambient temperature +20C, 60% relative humidity with no direct sunlight. No wind, no precipitation and the device is powered MCU DC Source Supply. Household purpose

Note 2: The test conditions will be conducted to reflect the harsh winter with a temperature of -15C, 30% relative humidity with no direct sunlight. No wind, no precipitation and the device is powered MCU DC Source Supply.

Note 3: The number of steps to perform an action is defined as all the interactions required to complete the final action starting from opening the application (included)

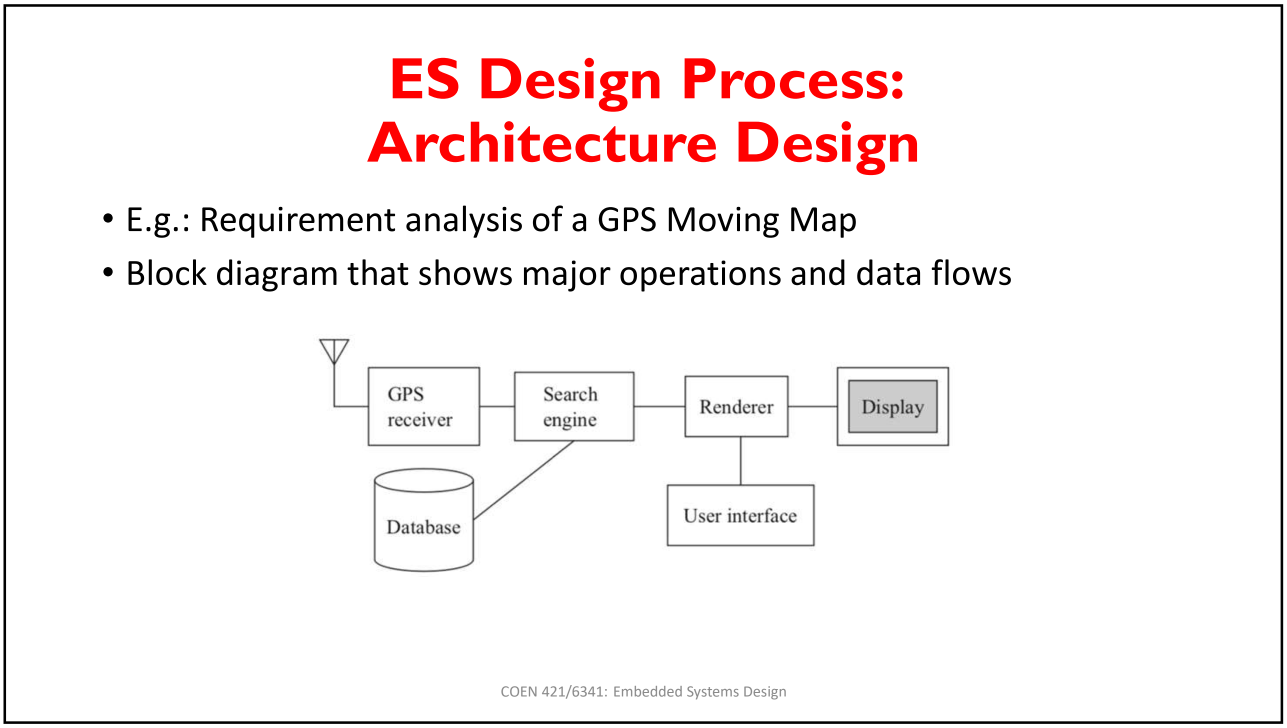
Note 4: Includes the weight of the door



ES Design Process: Architecture Design

- Specification does not say how the system does things
  - It only says what the system does
- Architecture design:
  - Describes how the system implements the system’s functions
  - We can use block diagrams
    - Show major operations and data flows among them
    - Block diagrams can be refined into hardware and software block diagrams

49



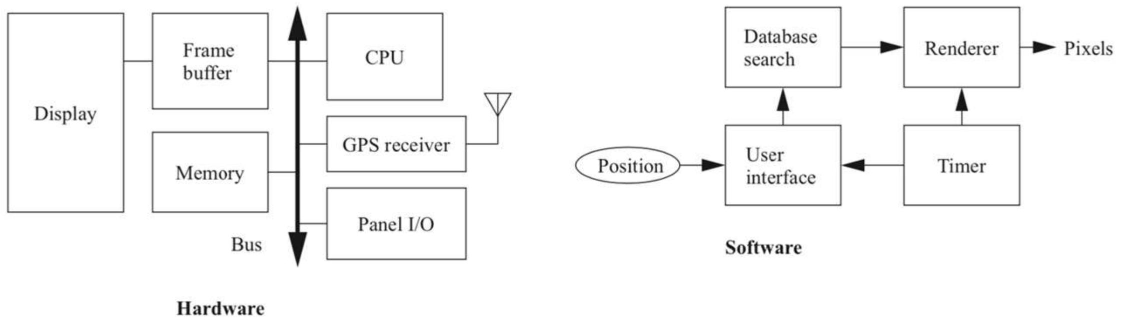
ES Design Process: Architecture Design

- E.g.: Requirement analysis of a GPS Moving Map
- Block diagram that shows major operations and data flows

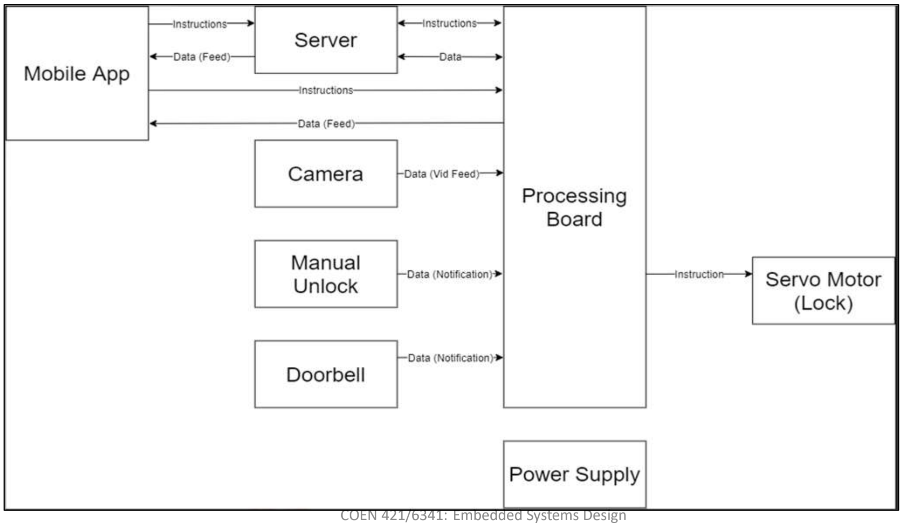
50

ES Design Process: Architecture Design

- E.g.: Requirement analysis of a GPS Moving Map

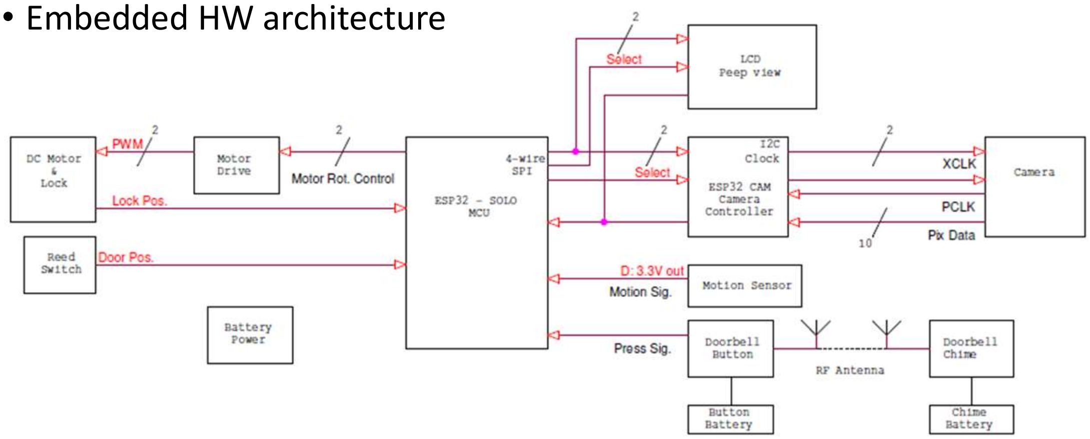


Ex 2: IoT system for smart door



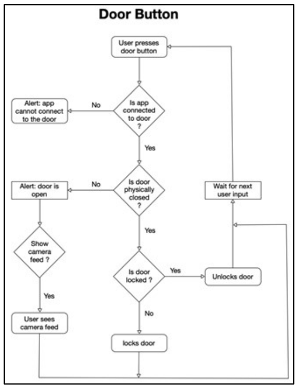
Ex 2: IoT system for smart door

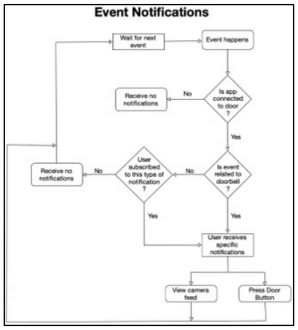
- Embedded HW architecture

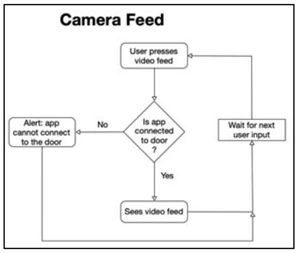


Ex 2: IoT system for smart door

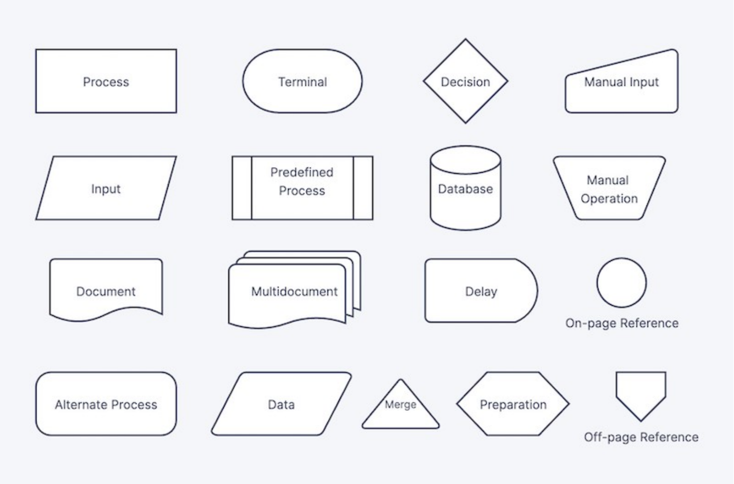
- SW Flow Charts





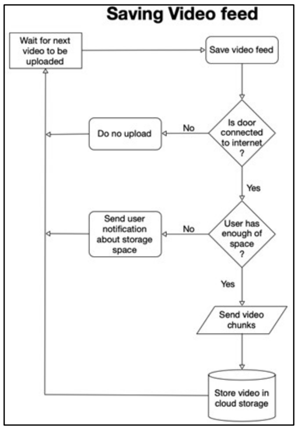


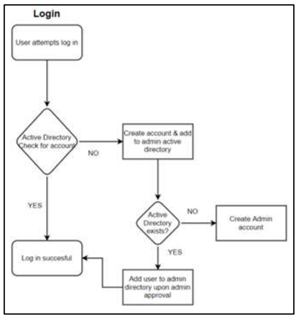
Flow Chart Common Symbols

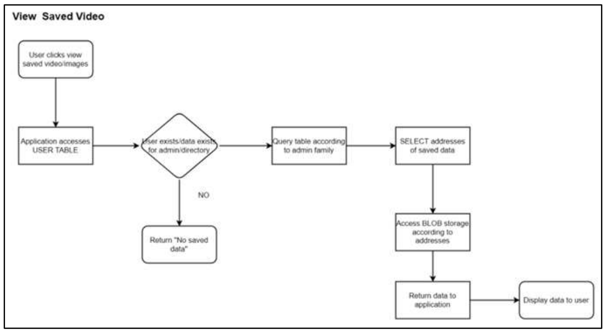


Ex 2: IoT system for smart door

- SW Flow Charts

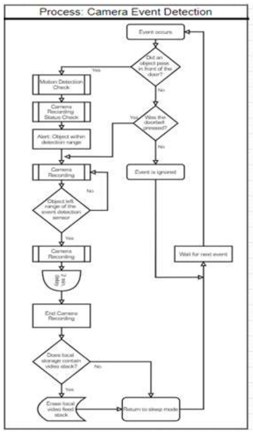

Saving Video feed
Wait for next video to be uploaded
Save video feed
Is door connected to internet ?
No
Do no upload
Yes
User has enough of space ?
No
Send user notification about storage space
Yes
Send video chunks
Store video in cloud storage

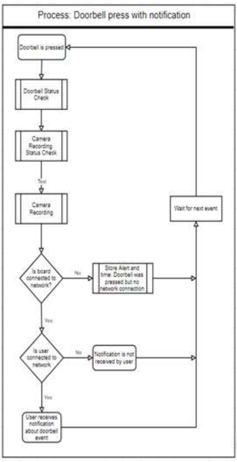

Login
User attempts log in
Active Directory Check for account
NO
Create account & add to admin active directory
YES
Active Directory exists?
NO
Create Admin account
YES
Add user to admin directory upon admin approval
Log in succesful


View Saved Video
User clicks view saved video/images
Application accesses USER TABLE
User exists/data exists for admin/directory
NO
Return "No saved data"
Query table according to admin family
SELECT addresses of saved data
Access BLOB storage according to addresses
Return data to application
Display data to user

Ex 2: IoT system for smart door

- HW Flow Charts


Process: Camera Event Detection
Event occurs
Did an object pass in front of the door?
Yes
No
Motion Detection Check
Camera Recording Status Check
Alert: Object within detection range
Yes
Was the doorbell pressed?
No
Event is ignored
Camera Recording
No
Object left range of the event detection sensor
Yes
Camera Recording
2 sec delay
Wait for next event
End Camera Recording
Does local storage contain video stack?
No
Yes
Erase local video feed stack
Return to sleep mode

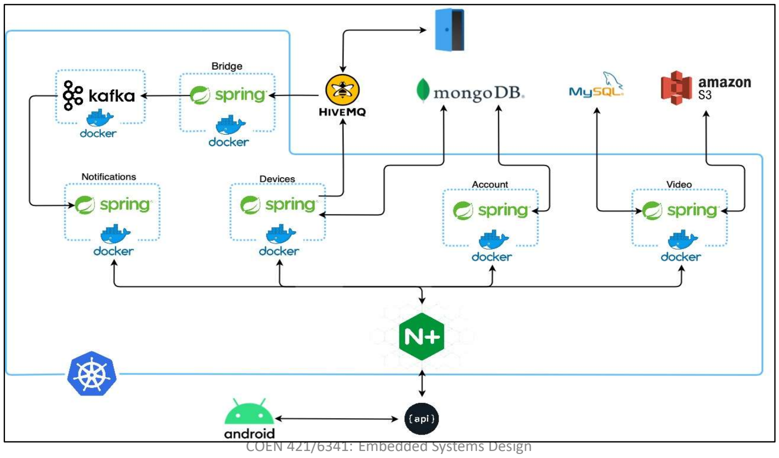

Process: Doorbell press with notification
Doorbell is pressed
Doorbell Status Check
Camera Recording Status Check
True
Camera Recording
Wait for next event
Is board connected to network?
No
Store Alert and time: Doorbell was pressed but no network connection
Yes
Is user connected to network
No
Notification is not received by user
Yes
User receives notification about doorbell event

ES Design Process: Components

- Architectural design tell us what components we need
- Component design tell us how to build those components in conformance to the architecture and specification
- It includes both hardware (FPGAs, boards …) and software modules
- Some of the components will be ready-made
- Other components you will have to design yourself
  - Printed circuit board
  - Custom programming

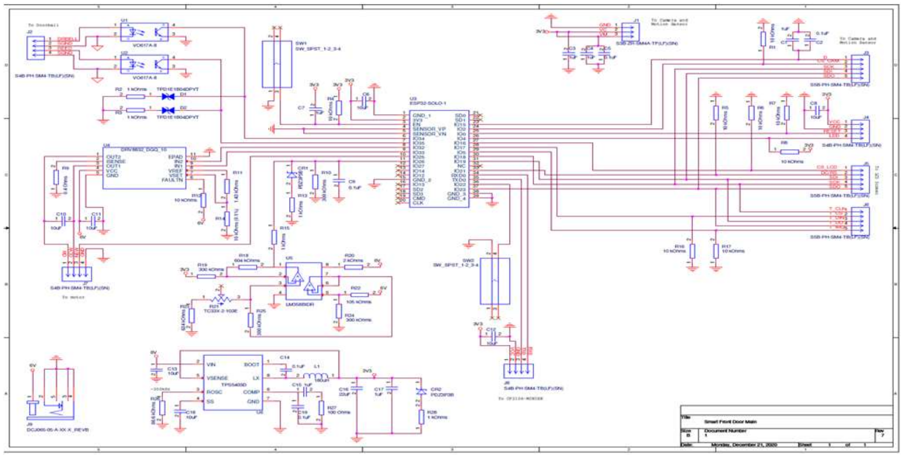
Ex 2: IoT system for smart door

- Cloud SW architecture



Ex 2: IoT system for smart door

- Embedded HW Schematic Diagram

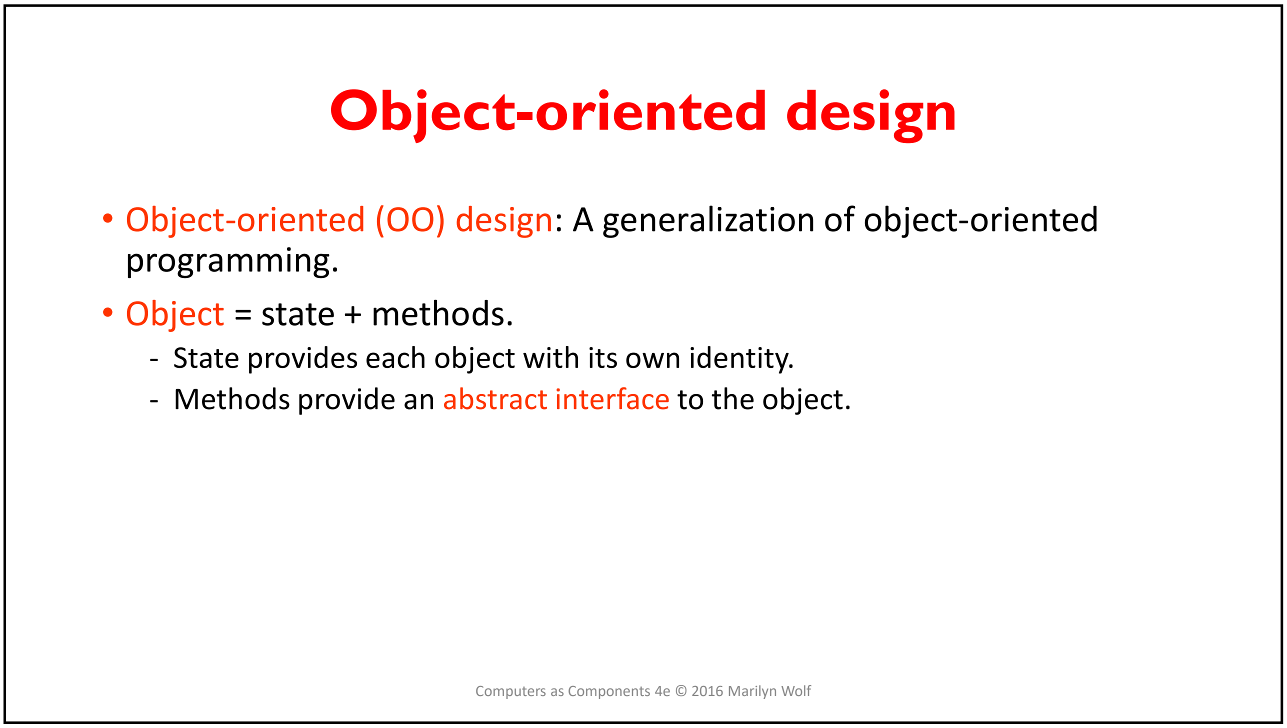


ES Design Process: System Integration

- Components must be put together after they are built
- System integration
  - Consists of a lot more than just plugging everything together and standing back
  - Components might not work
  - Bugs might appear during system integration
    - Good planning can help us find bugs quickly
      - Build up the system in phases
      - Run properly chosen tests
      - If we debug only a few modules at a time, we are more likely to uncover the simple bugs and be able to recognize them
  - It is difficult because it usually uncovers problems

System modeling

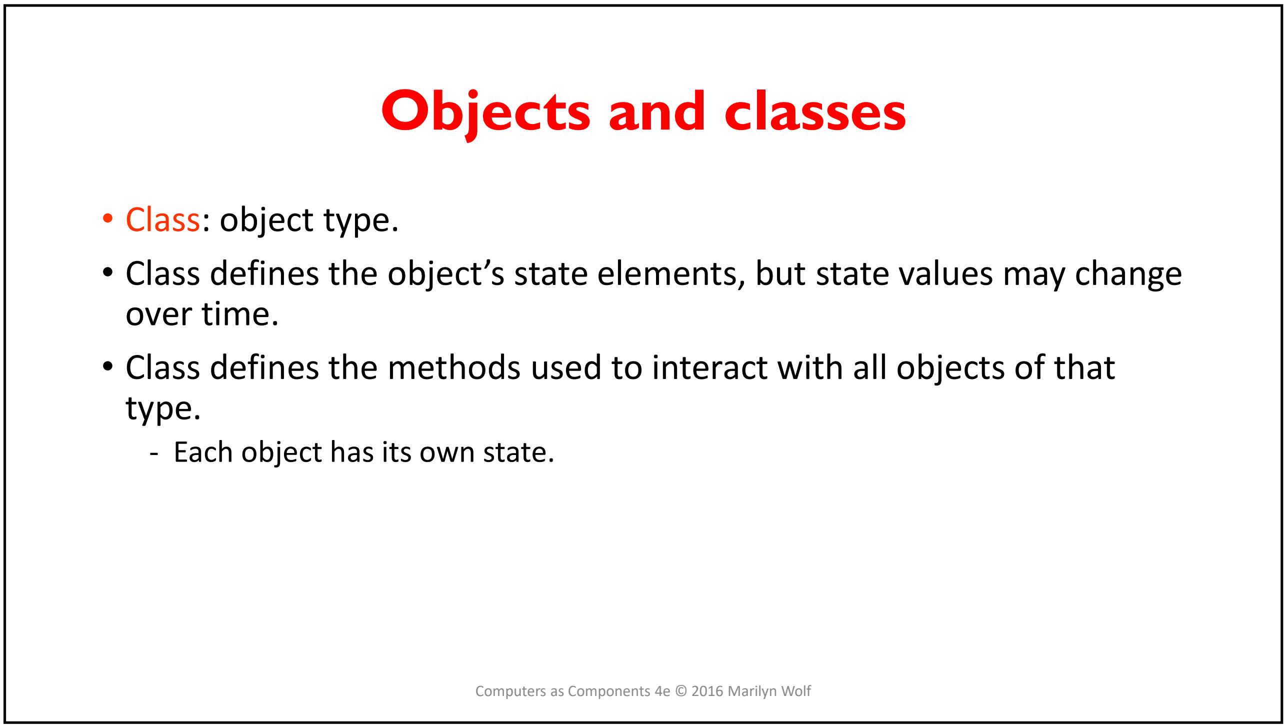
- Need languages to describe systems:
  - useful across several levels of abstraction;
  - understandable within and between organizations.
- Block diagrams are a start, but don’t cover everything.



Object-oriented design

- Object-oriented (OO) design: A generalization of object-oriented programming.
- Object = state + methods.
  - State provides each object with its own identity.
  - Methods provide an abstract interface to the object.

63



Objects and classes

- Class: object type.
- Class defines the object’s state elements, but state values may change over time.
- Class defines the methods used to interact with all objects of that type.
  - Each object has its own state.

64

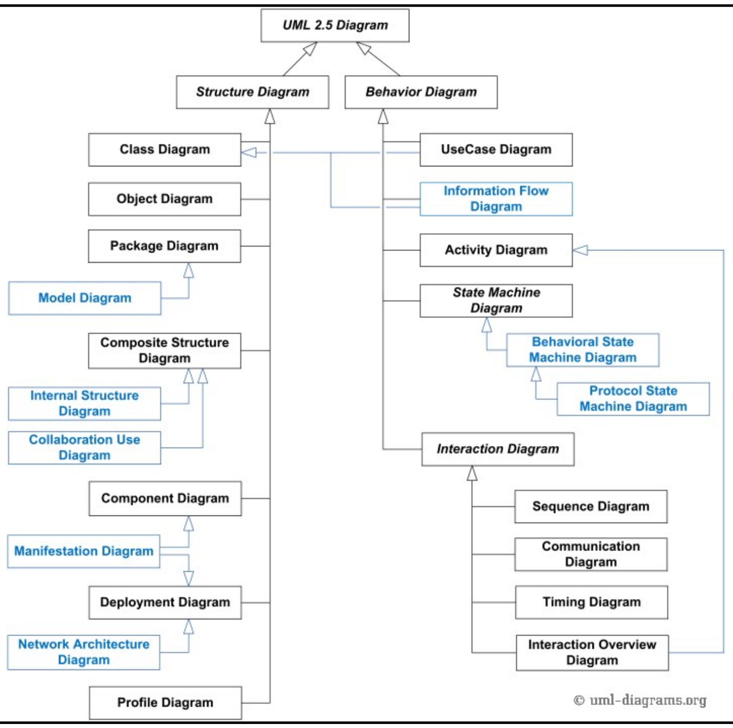
OO design principles

- Some objects will closely correspond to real-world objects.
  - Some objects may be useful only for description or implementation.
- Objects provide interfaces to read/write state, hiding the object’s implementation from the rest of the system.

UML

- Goals:
  - object-oriented;
  - visual;
  - useful at many levels of abstraction;
  - usable for all aspects of design.
- Structural description of the system
  - Describes the basic components
- Behavioral description
  - Specify the behavior of the system as well as its structure

UML



UML 2.5 Diagram

Structure Diagram

Behavior Diagram

Class Diagram

Object Diagram

Package Diagram

Model Diagram

Composite Structure Diagram

Internal Structure Diagram

Collaboration Use Diagram

Component Diagram

Manifestation Diagram

Deployment Diagram

Network Architecture Diagram

Profile Diagram

UseCase Diagram

Information Flow Diagram

Activity Diagram

State Machine Diagram

Behavioral State Machine Diagram

Protocol State Machine Diagram

Interaction Diagram

Sequence Diagram

Communication Diagram

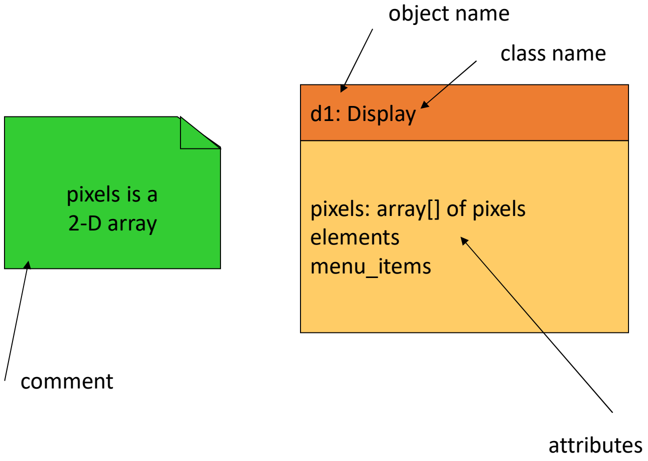
Timing Diagram

Interaction Overview Diagram

© uml-diagrams.org

67

UML object



object name

class name

d1: Display

pixels: array[] of pixels
elements
menu_items

pixels is a 2-D array

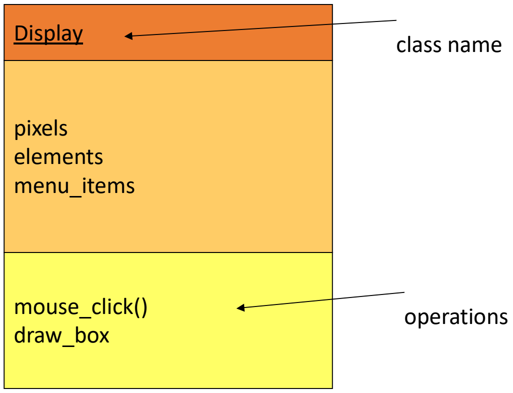
comment

attributes

68

UML Class

Class diagram is UML structure diagram which shows structure of the designed system at the level of classes and interfaces, shows their features, constraints and relationships - associations, generalizations, dependencies, etc.



Display

pixels
elements
menu_items

mouse_click()
draw_box

class name

operations

Computers as Components 4e © 2016 Marilyn Wolf

69

Links and associations

- Link: describes relationships between objects.
- Association: describes relationship between classes.

Computers as Components 4e © 2016 Marilyn Wolf

70

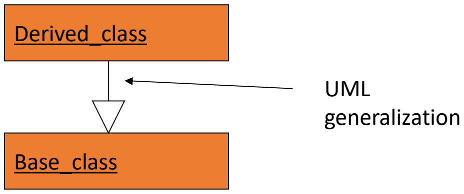
Relationships Between Objects and Classes

- Association: objects communicate but one does not own the other.
- Aggregation: a complex object is made of several smaller objects.
- Composition: aggregation in which owner does not allow access to its components.
- Generalization: define one class in terms of another.
- Dependency: there is a semantic connection between dependent and independent model elements. If changes to the definition of one element (the server or target) may cause changes to the other (the client or source)

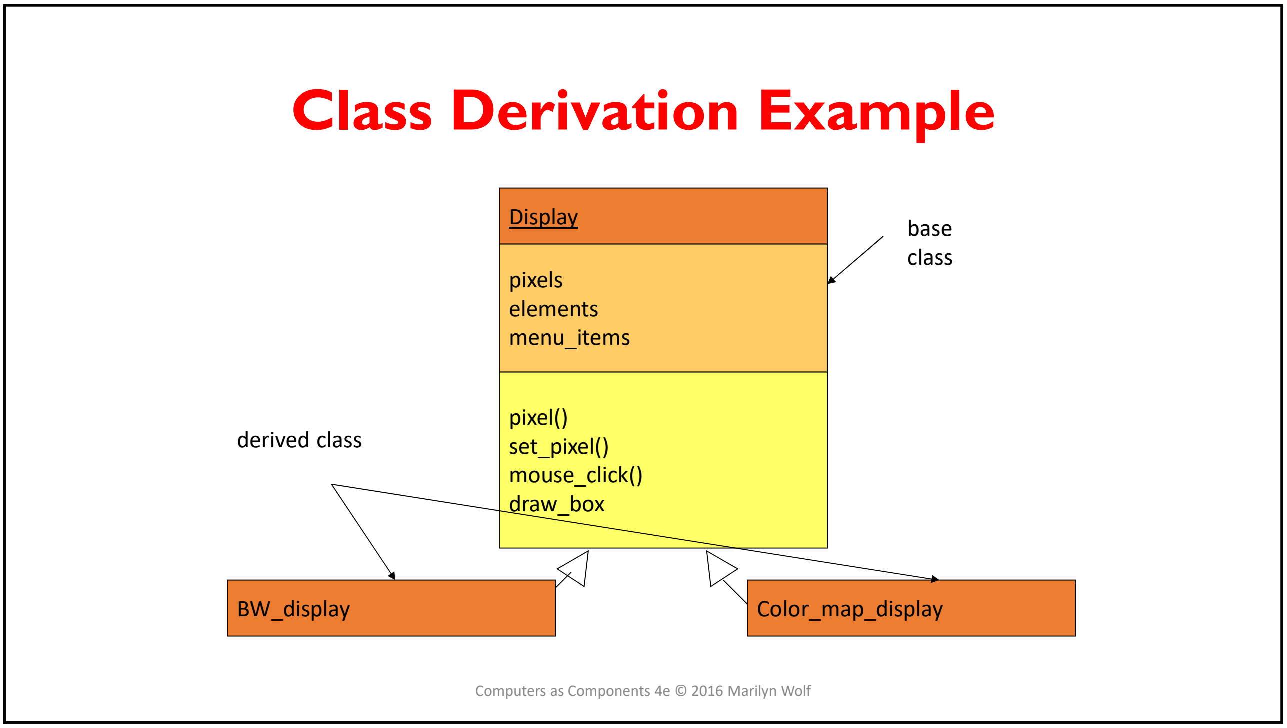
71

Class Derivation

- May want to define one class in terms of another.
  - Derived class inherits attributes, operations of base class.



72



Class Derivation Example

Display

pixels
elements
menu_items

pixel()
set_pixel()
mouse_click()
draw_box

base
class

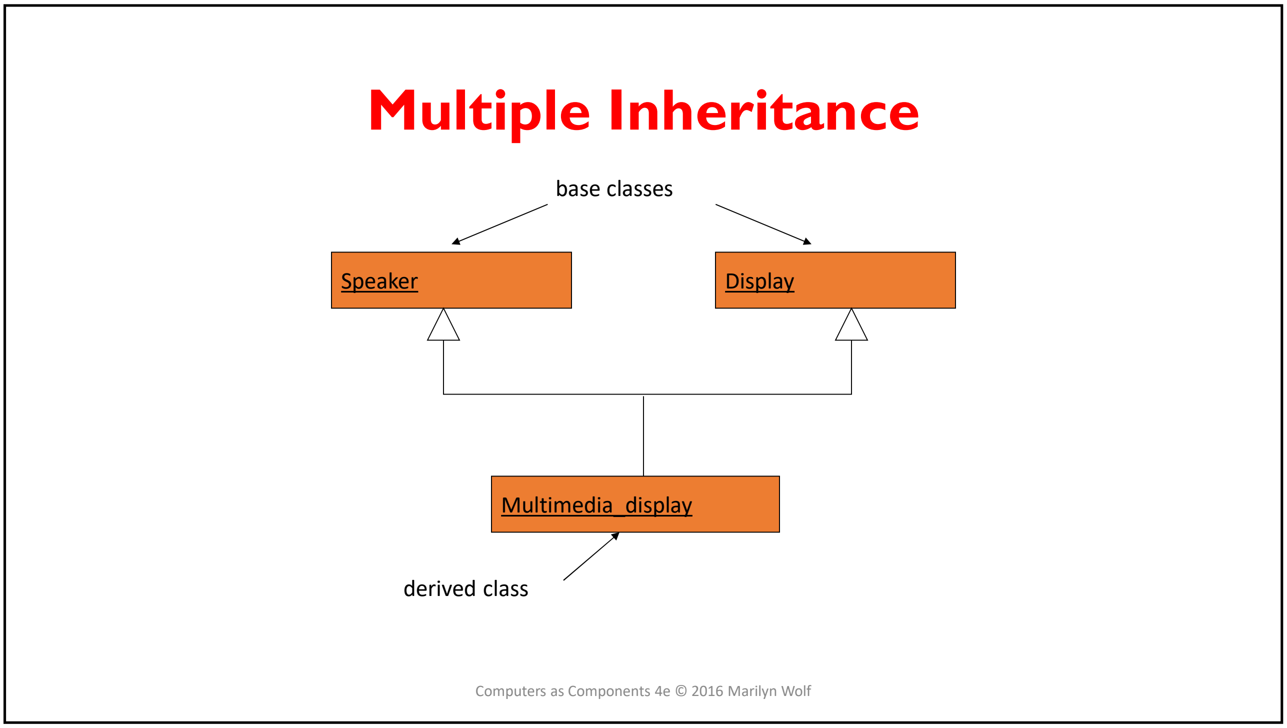
derived class

BW_display

Color_map_display

Computers as Components 4e © 2016 Marilyn Wolf

73



Multiple Inheritance

base classes

Speaker

Display

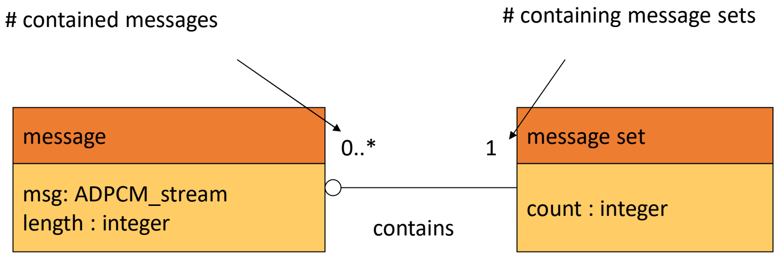
Multimedia_display

derived class

Computers as Components 4e © 2016 Marilyn Wolf

74

Association Example



# contained messages

containing message sets

message

msg: ADPCM_stream
length : integer

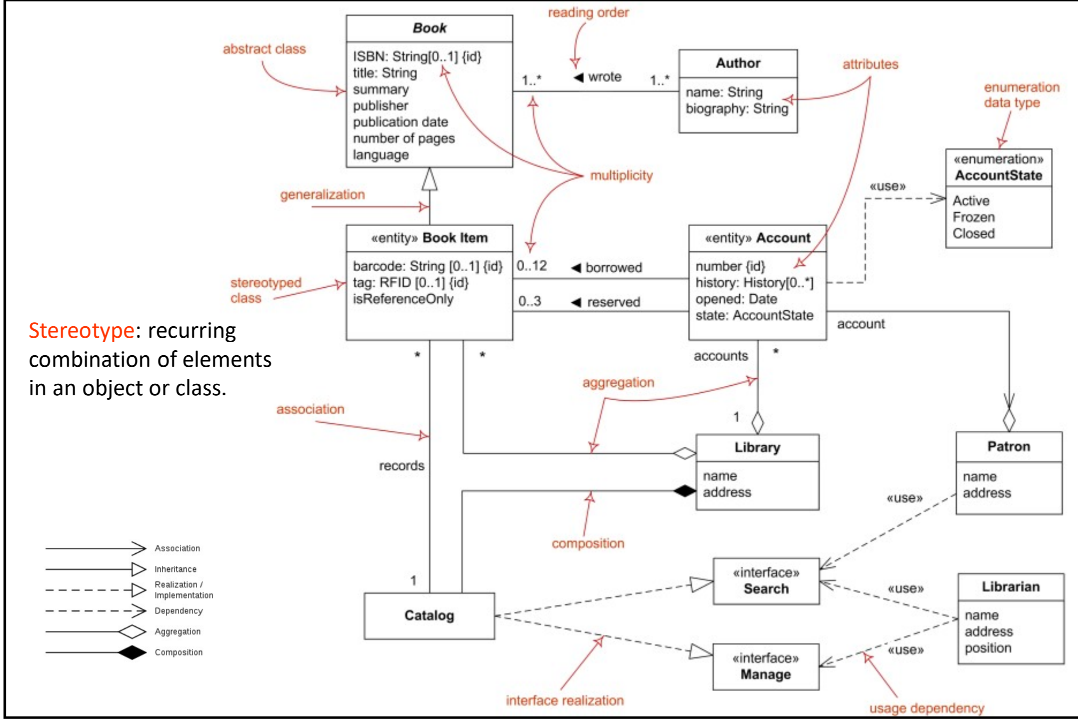
0..*

1

message set

count : integer

contains



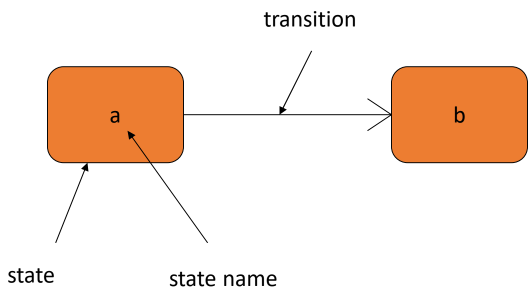
Stereotype: recurring combination of elements in an object or class.

Behavioral description

- Several ways to describe behavior:
  - internal view;
  - external view.

77

State machines



78

Event-driven state machines

- Behavioral descriptions are written as event-driven state machines.
  - Machine changes state when receiving an input.
- An event may come from inside or outside of the system.

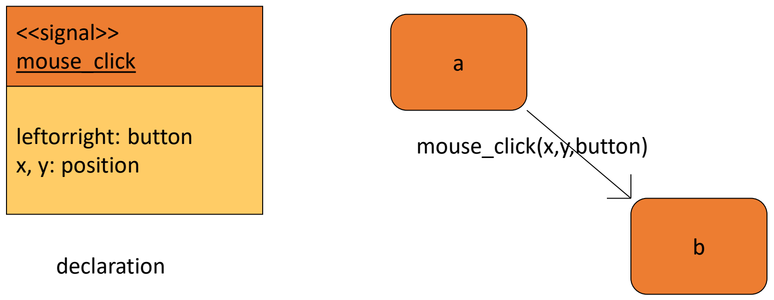
79

Types of events

- Signal: asynchronous event.
- Call: synchronized communication.
- Timer: activated by time.

80

Signal event

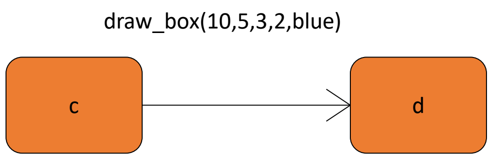


declaration

event description

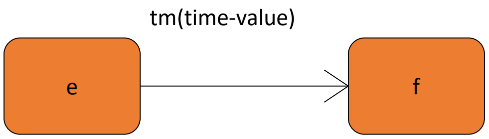
81

Call event



82

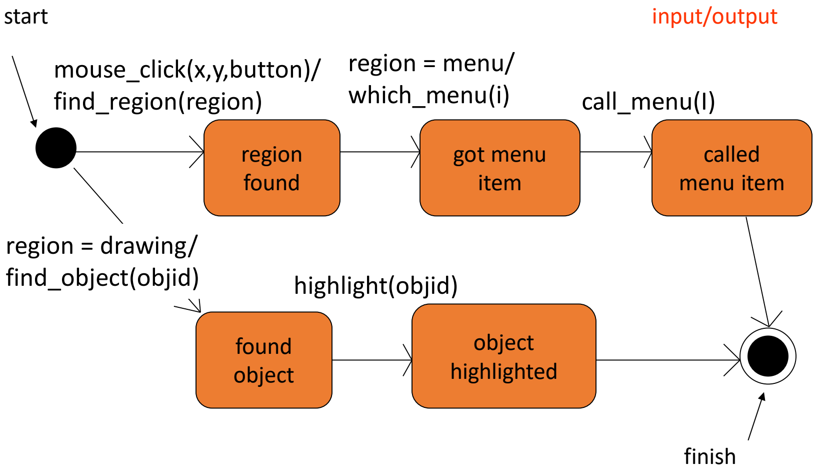
Timer event



Computers as Components 4e © 2016 Marilyn Wolf

83

Example state machine



Computers as Components 4e © 2016 Marilyn Wolf

84

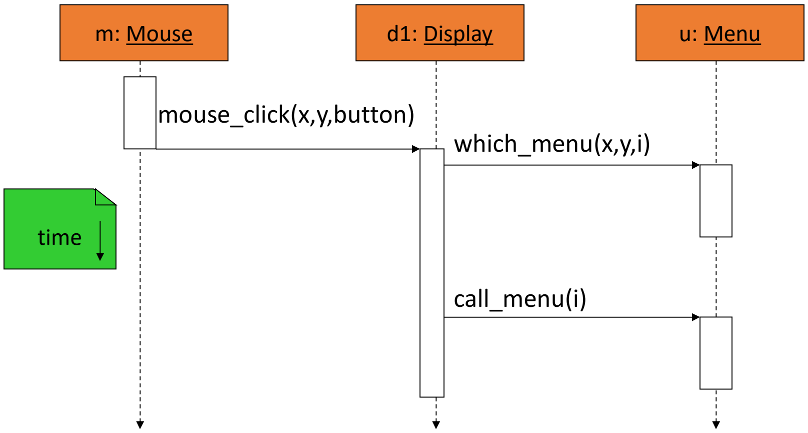
Sequence diagram

- Shows sequence of operations over time.
- Relates behaviors of multiple objects.

Computers as Components 4e © 2016 Marilyn Wolf

85

Sequence diagram example



Computers as Components 4e © 2016 Marilyn Wolf

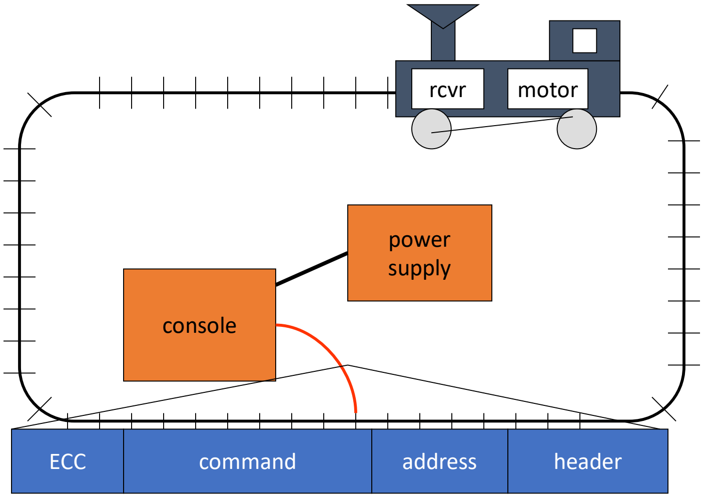
86

Purposes of example

- Follow a design through several levels of abstraction.
- Gain experience with UML.

87

Model train setup



88

Requirements

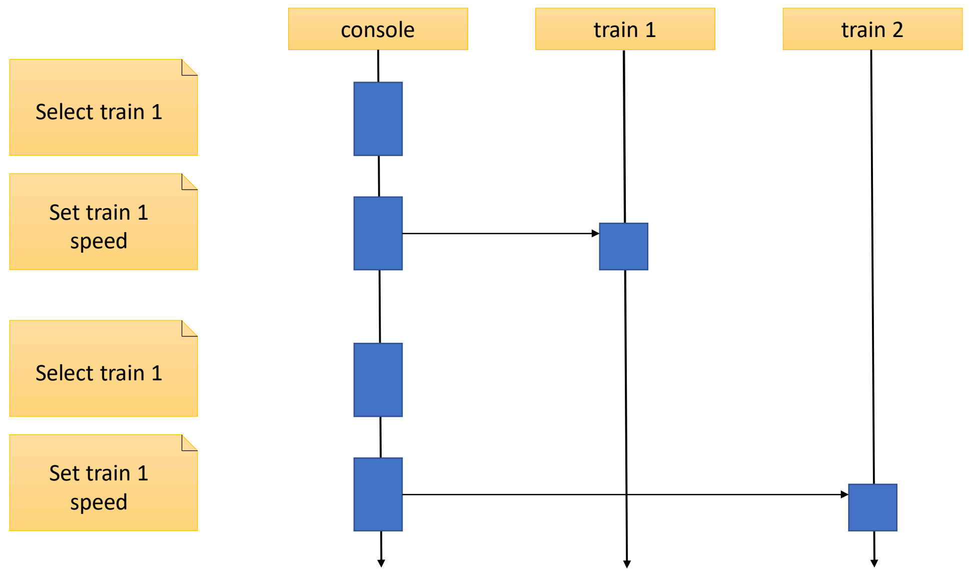
- Console can control 8 trains on 1 track.
- Throttle has at least 63 levels.
- Inertia control adjusts responsiveness with at least 8 levels.
- Emergency stop button.
- Error detection scheme on messages.

89

Requirements form

name	model train controller
purpose	control speed of <= 8 model trains
inputs	throttle, inertia, emergency stop, train #
outputs	train control signals
functions	set engine speed w. inertia; emergency stop
performance	can update train speed at least 10 times/sec
manufacturing cost	$50
power	wall powered
physical size/weight	console comfortable for 2 hands; < 2 lbs.

90



console

train 1

train 2

Select train 1

Set train 1 speed

Select train 1

Set train 1 speed

91

Digital Command Control

- DCC created by model railroad hobbyists, picked up by industry.
- Defines way in which model trains, controllers communicate.
  - Leaves many system design aspects open, allowing competition.
- This is a simple example of a big trend:
  - Cell phones, digital TV rely on standards.

92

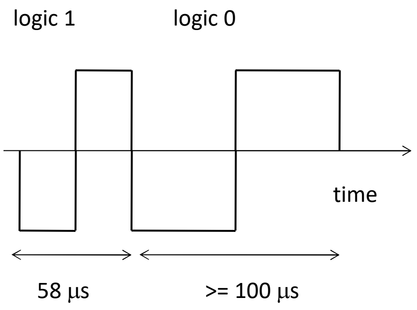
DCC documents

- Standard S-9.1, DCC Electrical Standard.
  - Defines how bits are encoded on the rails.
- Standard S-9.2, DCC Communication Standard.
  - Defines packet format and semantics.

93

DCC electrical standard

- Voltage moves around the power supply voltage; adds no DC component.
- 1 is 58 μs, 0 is at least 100 μs.



94

DCC communication standard

- Basic packet format: PSA(sD)+E.
- P: preamble = 1111111111.
- S: packet start bit = 0.
- A: address data byte.
- s: data byte start bit.
- D: data byte (data payload).
- E: packet end bit = 1.

95

DCC packet types

- Baseline packet: minimum packet that must be accepted by all DCC implementations.
  - Address data byte gives receiver address.
  - Instruction data byte gives basic instruction.
  - Error correction data byte gives ECC.

96

Conceptual specification

- Before we create a detailed specification, we will make an initial, simplified specification.
  - Gives us practice in specification and UML.
  - Good idea in general to identify potential problems before investing too much effort in detail.

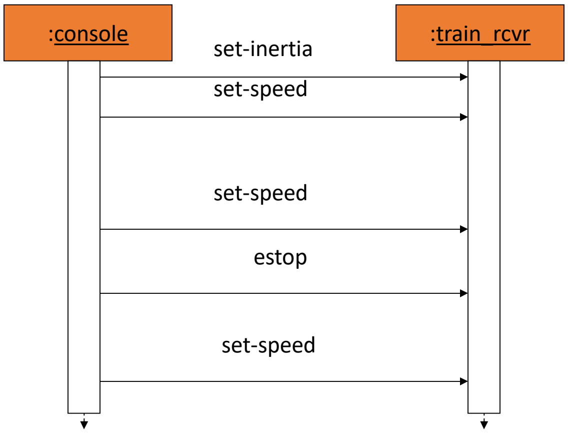
97

Basic system commands

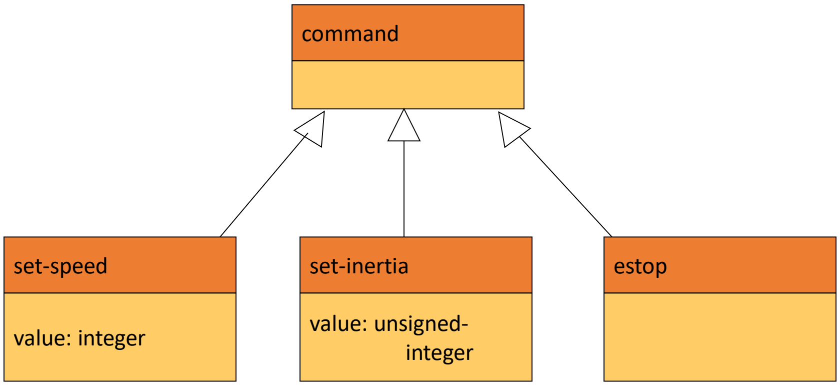
command name	parameters
set-speed	speed (positive/negative)
set-inertia	inertia-value (non-negative)
estop	none

98

Typical control sequence



Message classes

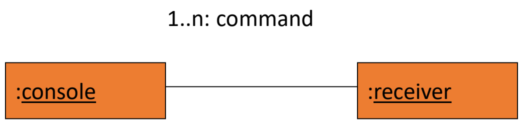


Roles of message classes

- Implemented message classes derived from message class.
  - Attributes and operations will be filled in for detailed specification.
- Implemented message classes specify message type by their class.
  - May have to add type as parameter to data structure in implementation.

Subsystem collaboration diagram

Shows relationship between console and receiver (ignores role of track):



System structure modeling

- Some classes define non-computer components.
  - Denote by *name.
- Choose important systems at this point to show basic relationships.

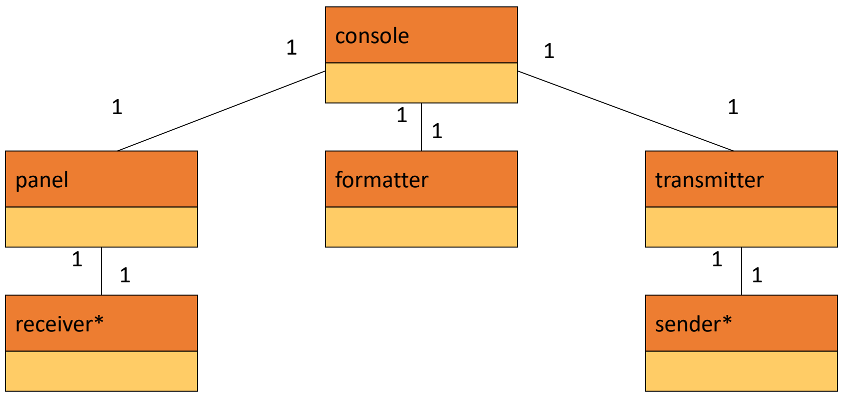
103

Major subsystem roles

- Console:
  - read state of front panel;
  - format messages;
  - transmit messages.
- Train:
  - receive message;
  - interpret message;
  - control the train.

104

Console system classes



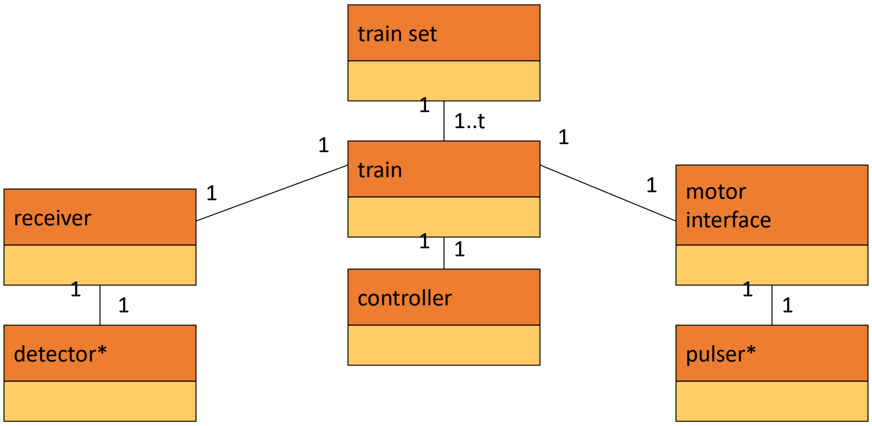
105

Console class roles

- panel: describes analog knobs and interface hardware.
- formatter: turns knob settings into bit streams.
- transmitter: sends data on track.

106

Train system classes



train set

train

receiver

motor interface

controller

detector*

pulser*

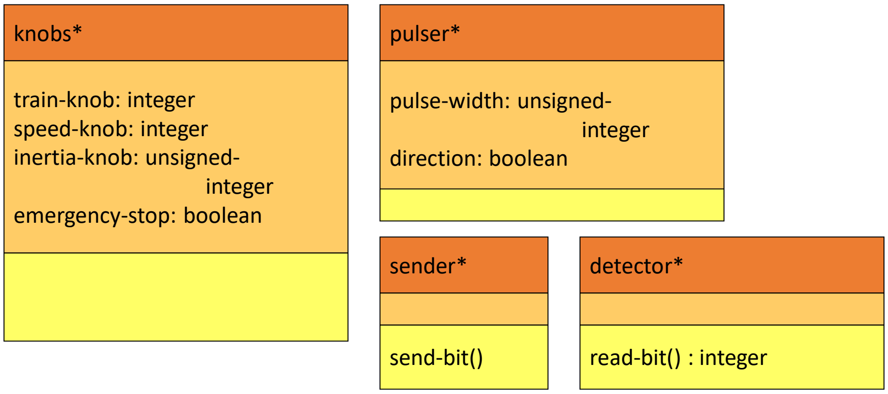
Train class roles

- receiver: digitizes signal from track.
- controller: interprets received commands and makes control decisions.
- motor interface: generates signals required by motor.

Detailed specification

- We can now fill in the details of the conceptual specification:
  - more classes;
  - behaviors.
- Sketching out the spec first helps us understand the basic relationships in the system.

Console physical object classes



knobs*

train-knob: integer
speed-knob: integer
inertia-knob: unsigned-integer
emergency-stop: boolean

pulser*

pulse-width: unsigned-integer
direction: boolean

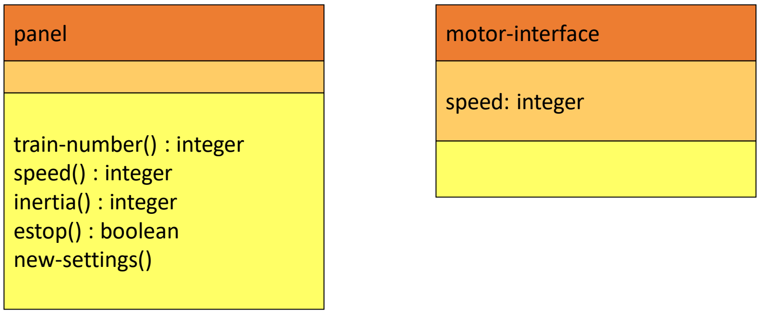
sender*

send-bit()

detector*

read-bit() : integer

Panel and motor interface classes


panel

train-number() : integer
speed() : integer
inertia() : integer
estop() : boolean
new-settings()

motor-interface

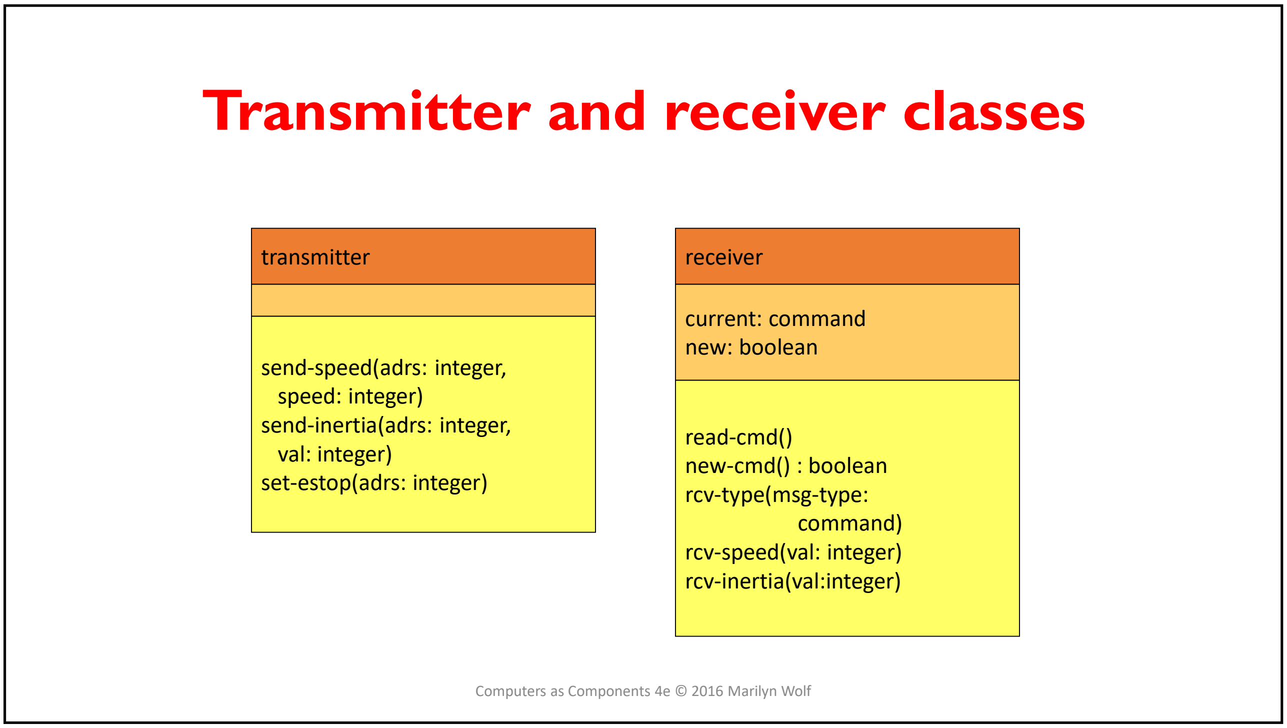
speed: integer

111

Class descriptions

- panel class defines the controls.
  - new-settings() behavior reads the controls.
- motor-interface class defines the motor speed held as state.

112



Transmitter and receiver classes

transmitter

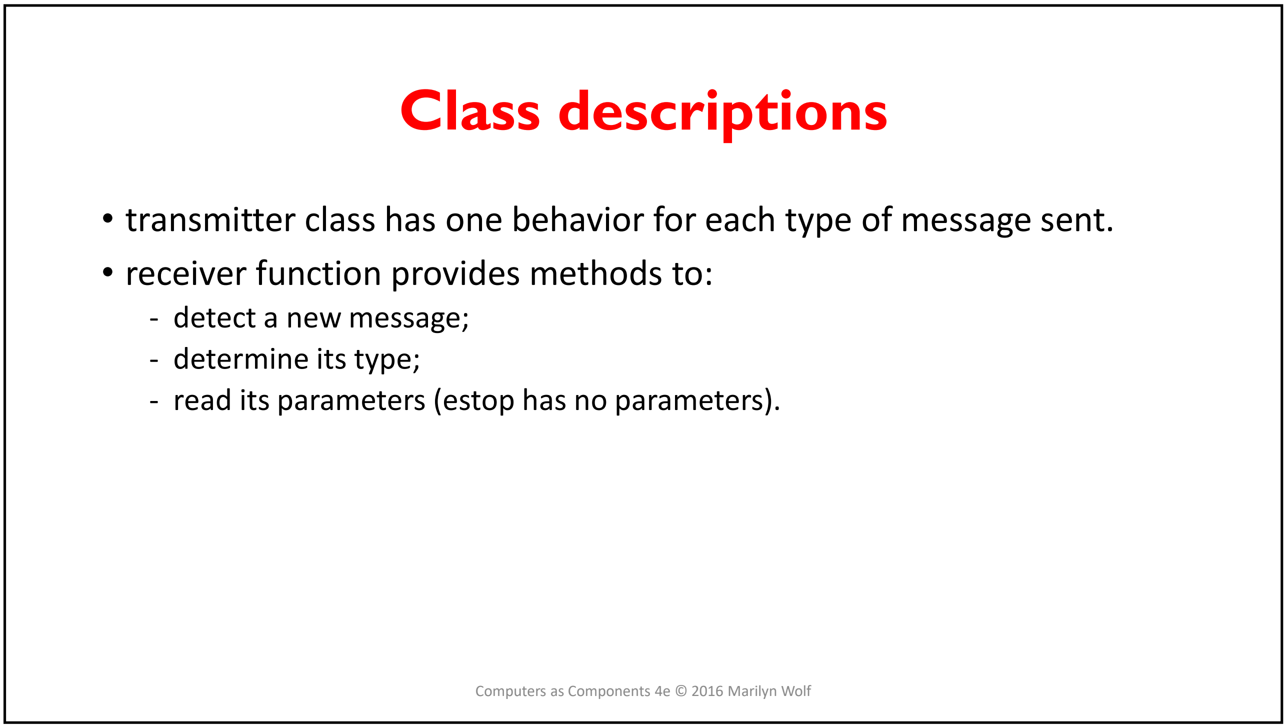
send-speed(adrs: integer,
speed: integer)
send-inertia(adrs: integer,
val: integer)
set-estop(adrs: integer)

receiver

current: command
new: boolean

read-cmd()
new-cmd() : boolean
rcv-type(msg-type:
command)
rcv-speed(val: integer)
rcv-inertia(val:integer)

113

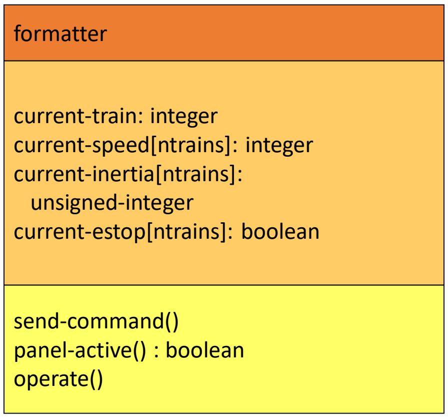


Class descriptions

- transmitter class has one behavior for each type of message sent.
- receiver function provides methods to:
  - detect a new message;
  - determine its type;
  - read its parameters (estop has no parameters).

114

Formatter class



formatter

current-train: integer
current-speed[ntrains]: integer
current-inertia[ntrains]:
unsigned-integer
current-estop[ntrains]: boolean

send-command()
panel-active() : boolean
operate()

115

Formatter class description

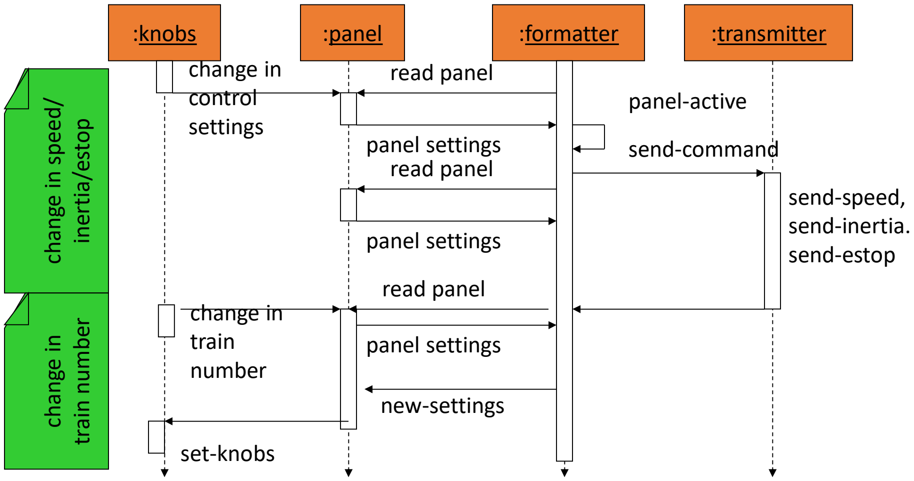
- Formatter class holds state for each train, setting for current train.
- The operate() operation performs the basic formatting task.

116

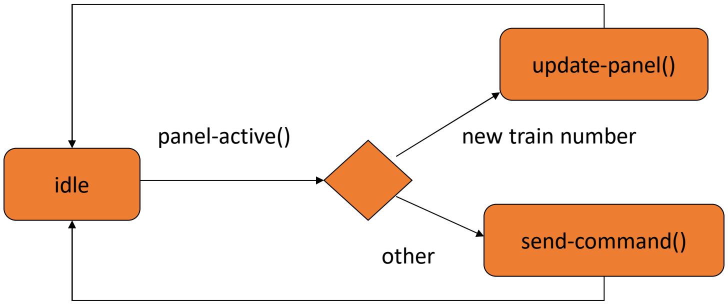
Control input cases

- Use a soft panel to show current panel settings for each train.
- Changing train number:
  - must change soft panel settings to reflect current train's speed, etc.
- Controlling throttle/inertia/estop:
  - read panel, check for changes, perform command.

Control input sequence diagram



Formatter operate behavior



idle

panel-active()

new train number

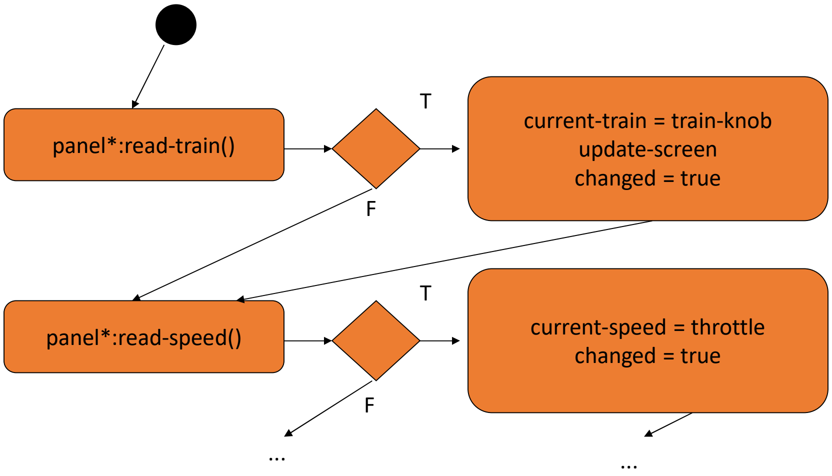
update-panel()

other

send-command()

119

Panel-active behavior



panel*:read-train()

T

current-train = train-knob
update-screen
changed = true

F

panel*:read-speed()

T

current-speed = throttle
changed = true

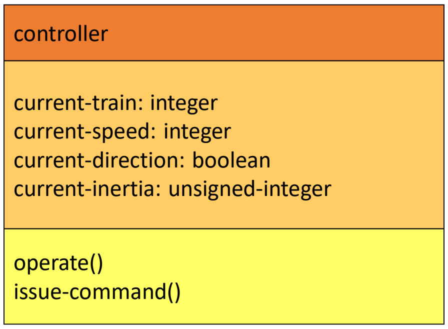
F

...

...

120

Controller class



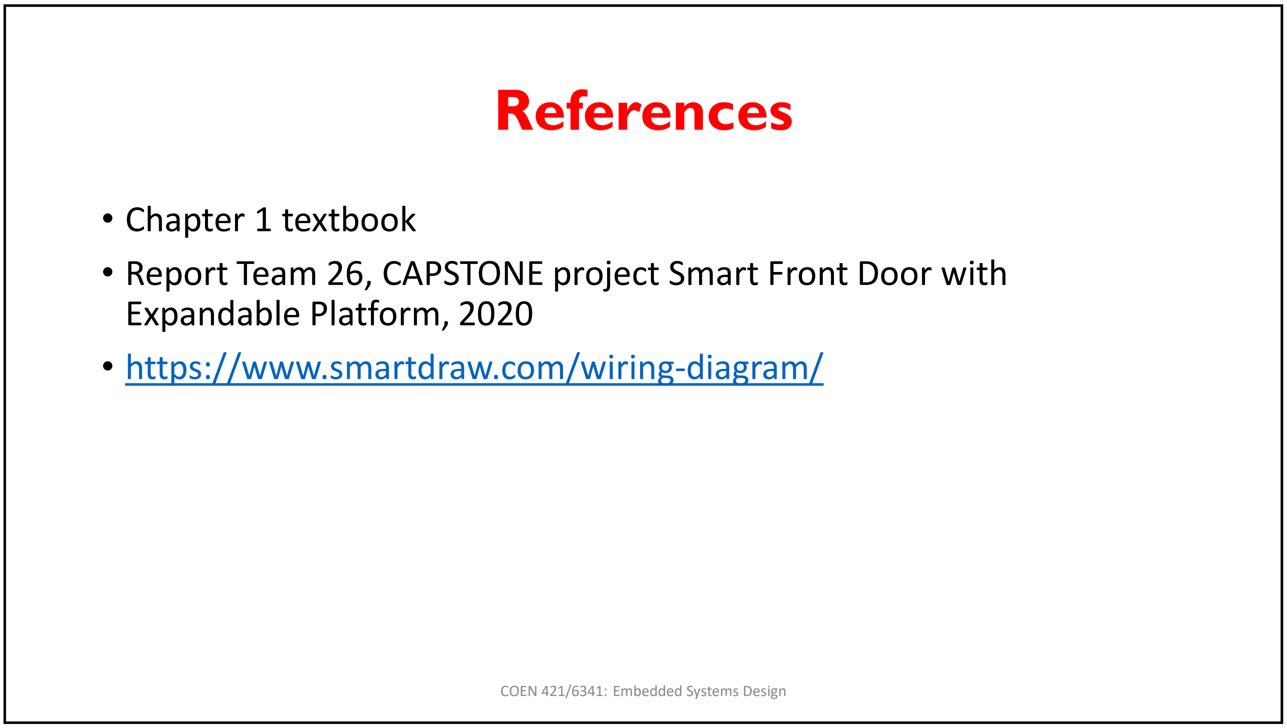
controller

current-train: integer
current-speed: integer
current-direction: boolean
current-inertia: unsigned-integer

operate()
issue-command()

Setting the speed

- Don’t want to change speed instantaneously.
- Controller should change speed gradually by sending several commands.



References

- Chapter 1 textbook
- Report Team 26, CAPSTONE project Smart Front Door with Expandable Platform, 2020
- https://www.smartdraw.com/wiring-diagram/