ECEN 478: Senior Design

ECEN 478

Dr. Mahmoud Nabil Mahmoud
mnmahmoud@ncat.edu

North Carolina A & T State University

March 27, 2022

Outline

1. Introduction
2. State Diagrams
3. Flowchart
4. Data Flow Diagrams (DFD)
5. Entity Relationship Diagram
6. Unified Language Modeling

Motivation

- Functional Design Appropriate for function-oriented systems: inputs, outputs, and some transformation between them.
- There are various types of system behavior that designers need to be able to understand.
    - State behavior
    - Logic and flow
    - Data flow
    - Database relationships
    - Unified Modeling Language
    - ...

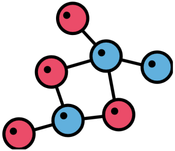
Learning Objectives

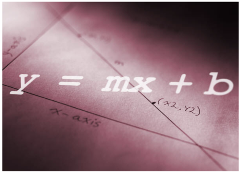
By the end of this chapter, you should:

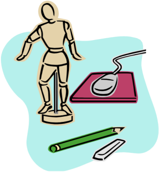
- Have a familiarity with the following modeling tools for describing ECE system behavior:
  - state diagrams,
  - flowcharts,
  - data flow diagrams,
  - entity relationship diagrams,
  - the Unified Modeling Language.
- Understand the intention and expressive power of the different models.
- Understand the domains in which the models apply.
- Be able to conduct analysis and design with the models.
- Understand what model types to choose for a given design problem.

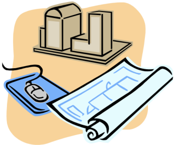
Models

Models - what do you think of?









Definations

Model

A Standardized representation of a system process, or object that capture the essential details without specifing the physical implementation.

Modeling Language

A collection of symbols graphical and/or numeric used for modeling within specific domain.

Intention (of model)

The class of behaviours the model is intended describe

Properties of A good model

A good model should be

- **Abstract** should be independent of final implementation

Properties of A good model

A good model should be

- **Abstract** should be independent of final implementation
- **Unambiguous** a single clear meaning to describr the behavior

Properties of A good model

A good model should be

- **Abstract** should be independent of final implementation
- **Unambiguous** a single clear meaning to describr the behavior
- **Allow for innovation** encourage exploration of alternative system implementation

Properties of A good model

A good model should be

- **Abstract** should be independent of final implementation
- **Unambiguous** a single clear meaning to describr the behavior
- **Allow for innovation** encourage exploration of alternative system implementation
- **Standardized** a common language that can be understood by designers.

Properties of A good model

A good model should be

- **Abstract** should be independent of final implementation
- **Unambiguous** a single clear meaning to describr the behavior
- **Allow for innovation** encourage exploration of alternative system implementation
- **Standardized** a common language that can be understood by designers.
- **Facilitate good communication** between team members

Properties of A good model

A good model should be

- **Abstract** should be independent of final implementation
- **Unambiguous** a single clear meaning to describr the behavior
- **Allow for innovation** encourage exploration of alternative system implementation
- **Standardized** a common language that can be understood by designers.
- **Facilitate good communication** between team members
- **Modifiable**

Properties of A good model

A good model should be

- **Abstract** should be independent of final implementation
- **Unambiguous** a single clear meaning to describr the behavior
- **Allow for innovation** encourage exploration of alternative system implementation
- **Standardized** a common language that can be understood by designers.
- **Facilitate good communication** between team members
- **Modifiable**
- **Remove unnecessary details & show important features**

Properties of A good model

A good model should be

- **Abstract** should be independent of final implementation
- **Unambiguous** a single clear meaning to describr the behavior
- **Allow for innovation** encourage exploration of alternative system implementation
- **Standardized** a common language that can be understood by designers.
- **Facilitate good communication** between team members
- **Modifiable**
- **Remove unnecessary details & show important features**
- **Break system into sub-problems.** Design hirearchy

Properties of A good model

A good model should be

- **Abstract** should be independent of final implementation
- **Unambiguous** a single clear meaning to describr the behavior
- **Allow for innovation** encourage exploration of alternative system implementation
- **Standardized** a common language that can be understood by designers.
- **Facilitate good communication** between team members
- **Modifiable**
- **Remove unnecessary details & show important features**
- **Break system into sub-problems.** Design hirearchy
- **Substitute sequence of actions by a single action.**

Properties of A good model

A good model should be

- **Abstract** should be independent of final implementation
- **Unambiguous** a single clear meaning to describr the behavior
- **Allow for innovation** encourage exploration of alternative system implementation
- **Standardized** a common language that can be understood by designers.
- **Facilitate good communication** between team members
- **Modifiable**
- **Remove unnecessary details & show important features**
- **Break system into sub-problems.** Design hirearchy
- **Substitute sequence of actions by a single action.**
- **Assist in verification** Testing

Properties of A good model

A good model should be

- **Abstract** should be independent of final implementation
- **Unambiguous** a single clear meaning to describr the behavior
- **Allow for innovation** encourage exploration of alternative system implementation
- **Standardized** a common language that can be understood by designers.
- **Facilitate good communication** between team members
- **Modifiable**
- **Remove unnecessary details & show important features**
- **Break system into sub-problems.** Design hirearchy
- **Substitute sequence of actions by a single action.**
- **Assist in verification** Testing
- **Assist in validation** Customer Acceptance

Outline

1 Introduction

2 State Diagrams

3 Flowchart

4 Data Flow Diagrams (DFD)

5 Entity Relationship Diagram

6 Unified Language Modeling

State Diagram

State Diagram

State diagrams describe the behavior of systems with memory.

State Diagram

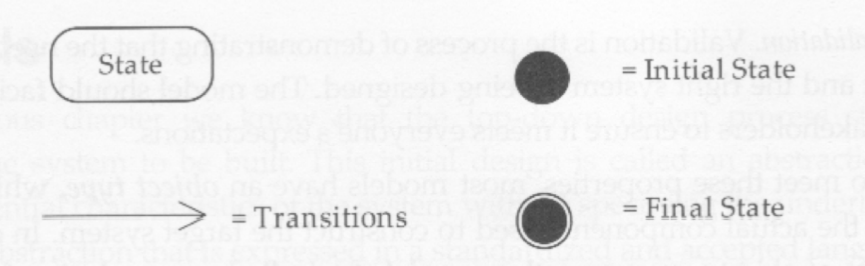
State Diagram

State diagrams describe the behavior of systems with memory.

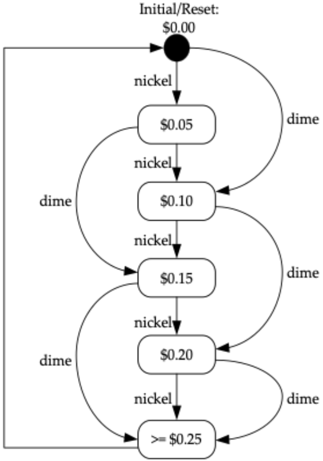
How to determine if a system has memory?

Can the same "set" of inputs" produce different outputs?"

State Diagram Modeling Language

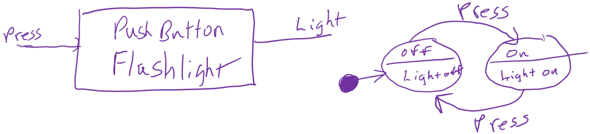


Example: Vending Machine

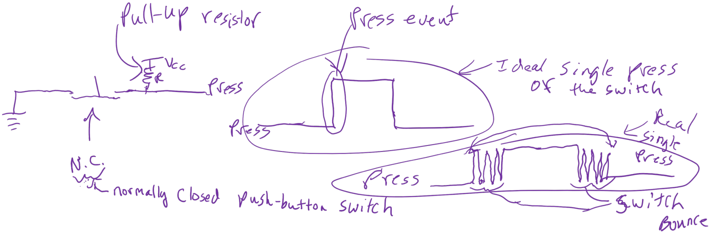


Example: Flashlight with Bouncing Push Button

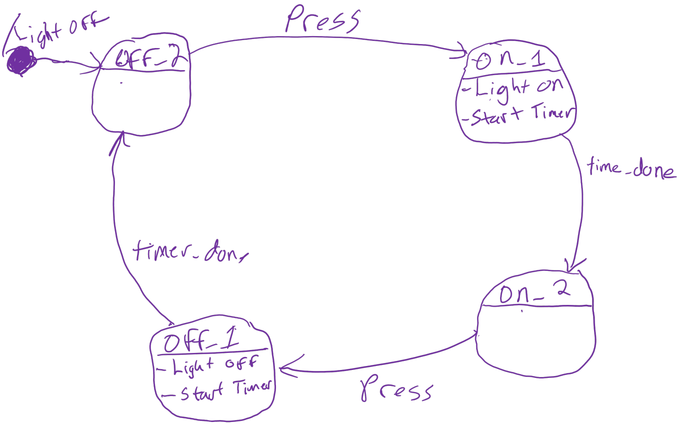
- Level – 0 Block Diagram



- Description (Bouncing Pushbutton Switch)



Example: State Diagram Flashlight with Bouncing Push Button



Outline

1 Introduction

2 State Diagrams

3 Flowchart

4 Data Flow Diagrams (DFD)

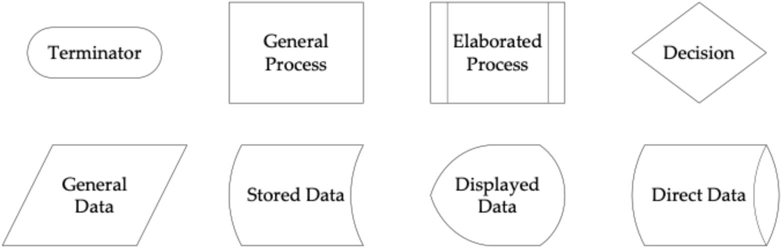
5 Entity Relationship Diagram

6 Unified Language Modeling

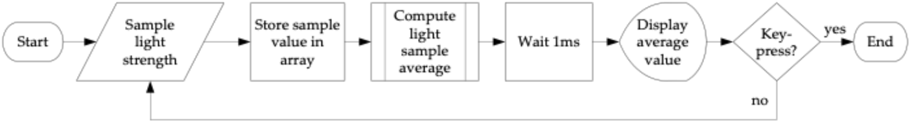
The “Lowly” Flowchart

Flowchart

The intention of a flowchart is to visually describe an algorithm, including its steps



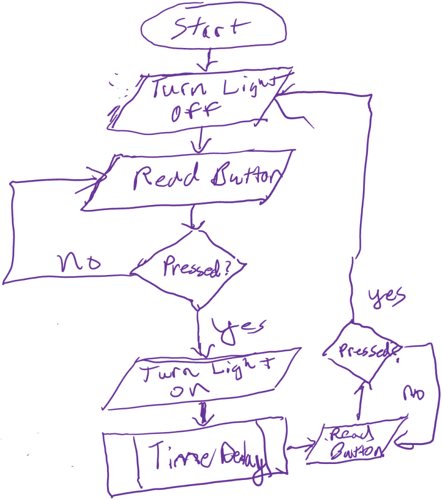
Example: Light Monitoring System



- Can you figure out the operation of the system by looking at this?
- That is why the flowchart is elegant and not so lowly.

Create a Flowchart for the Previous Flashlight

Create a Flowchart for the Previous Flashlight



Example: Security Robot

Requirements (loosely)

- Must roam randomly around facility
- Detect intruders by recognizing sound
- Set-off an alarm if detects noise, transmit position, and wait.
- Must regularly conduct a self-test to determine if it is working properly.

Design Details

- Has the three ultrasonic sensors and can measure distance to objects to the left, forward, and right.
- Has a microphone that it uses to monitor sounds.

Problem: Develop a Flow Chart of its operation

Outline

1 Introduction

2 State Diagrams

3 Flowchart

4 Data Flow Diagrams (DFD)

5 Entity Relationship Diagram

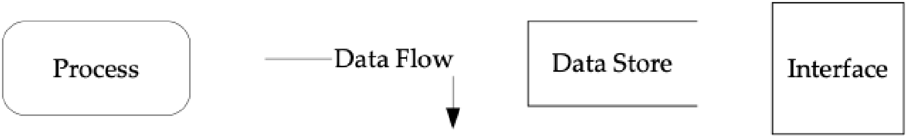
6 Unified Language Modeling

Data Flow Diagrams

Data Flow Diagram

The main use is to model processing and flow of data within the system

- DFDs can have levels, just like functional block diagrams



Example: Video Browsing System

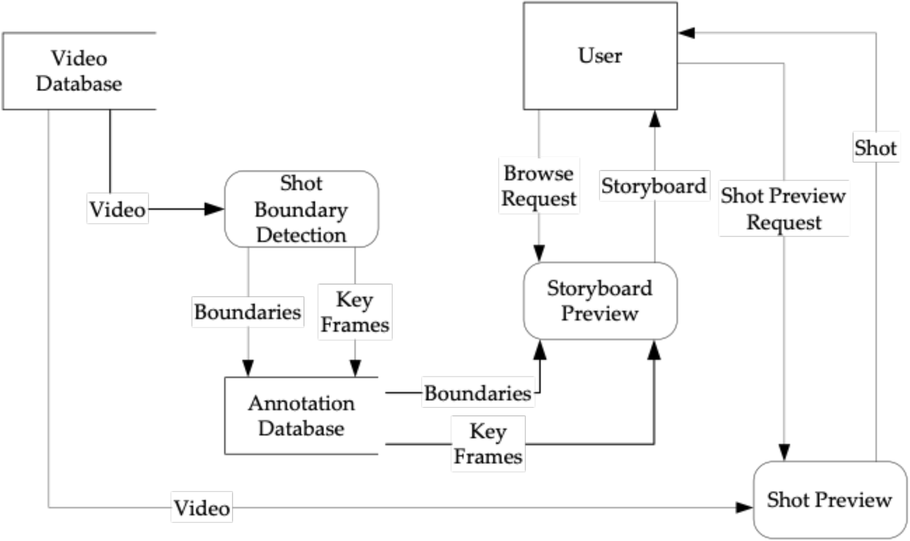
Inputs

- Video: Extemal video to the system that is entered into the video database.
- Browse Request: User request to browse a particular video.
- Shot Preview Request User request to preview a particular short from a video.

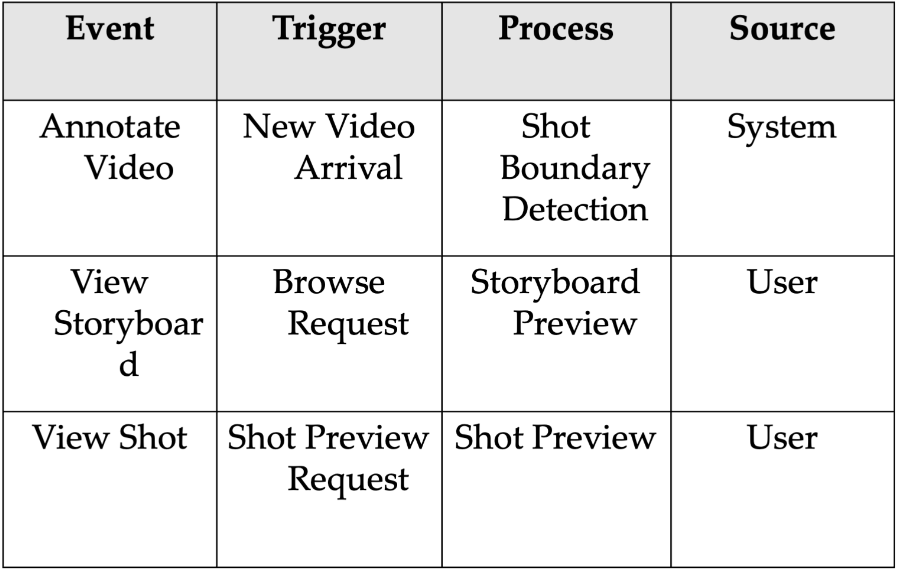
Outputs

- Storyboard: A sequence of frames Summarizing the entire Video.
- Shot: The complete video corresponding to the still image in the stor rhoard.

Example: Video Browsing System



The DFD Event Table



Event	Trigger	Process	Source
Annotate Video	New Video Arrival	Shot Boundary Detection	System
View Storyboard	Browse Request	Storyboard Preview	User
View Shot	Shot Preview Request	Shot Preview	User

Outline

1 Introduction

2 State Diagrams

3 Flowchart

4 Data Flow Diagrams (DFD)

5 Entity Relationship Diagram

6 Unified Language Modeling

Entity Relationship Diagram

ERD

The intention of an ERD is to catalog a set of related objects (entities), their attributes, and the relationships between them.

- **Entities** They are generally in the form of tangible objects, roles played, organizational units, devices, and locations.

Entity Relationship Diagram

ERD

The intention of an ERD is to catalog a set of related objects (entities), their attributes, and the relationships between them.

- **Entities** They are generally in the form of tangible objects, roles played, organizational units, devices, and locations.
- **Attributes** features used to differentiate between instances of entities.

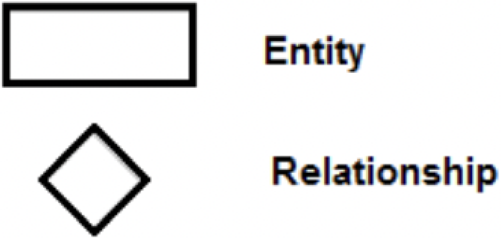
Entity Relationship Diagram

ERD

The intention of an ERD is to catalog a set of related objects (entities), their attributes, and the relationships between them.

- **Entities** They are generally in the form of tangible objects, roles played, organizational units, devices, and locations.
- **Attributes** features used to differentiate between instances of entities.
- **Relationship** How the entities are related to one another

ERD Symbols



Values on arcs represent the cardinality of the relashionship

Example

Assume the college want to store the data of three entities

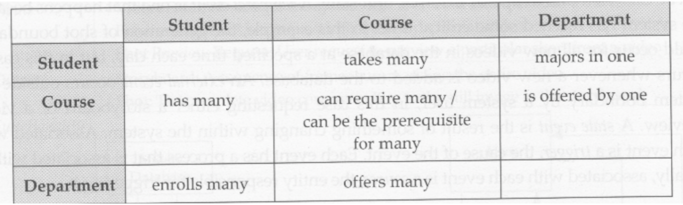
- Student
- Course
- Departments

Example

Assume the college want to store the data of three entities

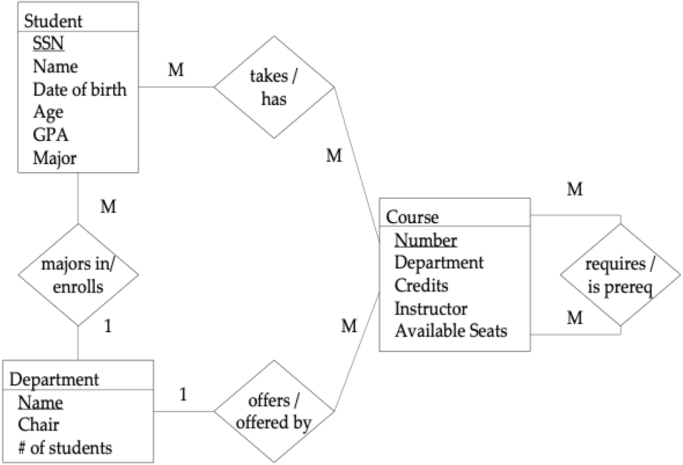
- Student
- Course
- Departments

First we build entity relationship matrix



	Student	Course	Department
Student		takes many	majors in one
Course	has many	can require many / can be the prerequisite for many	is offered by one
Department	enrolls many	offers many	

Example


Student
SSN
Name
Date of birth
Age
GPA
Major
M
takes /
has
M
M
majors in/
enrolls
1
Course
Number
Department
Credits
Instructor
Available Seats
M
requires /
is prereq
M
M
Department
Name
Chair
# of students
1
offers /
offered by

Outline

1. Introduction
2. State Diagrams
3. Flowchart
4. Data Flow Diagrams (DFD)
5. Entity Relationship Diagram
6. Unified Language Modeling

Unified Modeling Language

- For object-oriented software design. Value in applying it to ECE Systems.
- Has 6 (at least) different views of systems (unified!).
- Only an overview is provided here

Example

- Pretty example – web ordering of groceries followed by home delivery.

Example

- Pretty example – web ordering of groceries followed by home delivery.
- The “virtual-Grocer” system.

Example

- Pretty example – web ordering of groceries followed by home delivery.
- The “virtual-Grocer” system.
- User has a barcode scanner connected to home computer.

Example

- Pretty example – web ordering of groceries followed by home delivery.
- The “virtual-Grocer” system.
- User has a barcode scanner connected to home computer.
- They can scan a used item and automatically order it from the grocery store.

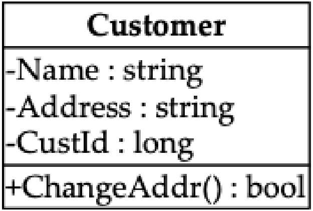
Example

- Pretty example – web ordering of groceries followed by home delivery.
- The “virtual-Grocer” system.
- User has a barcode scanner connected to home computer.
- They can scan a used item and automatically order it from the grocery store.
- Place the order and groceries delivered at scheduled time.

1- Static View

- Object view of software.
- **Classes** represent
    - Data Methods (functions) that operate on the data

  Objects are

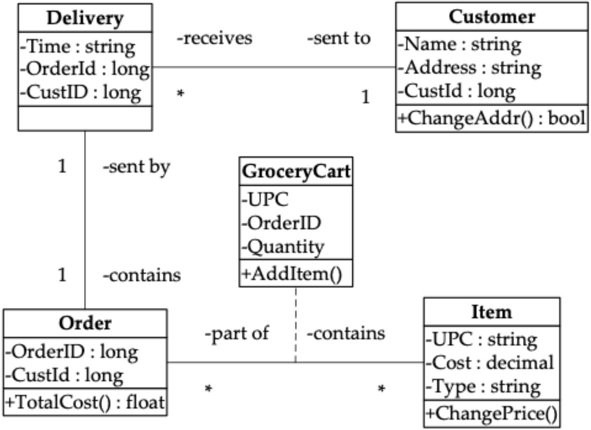


Customer

-Name : string
-Address : string
-CustId : long

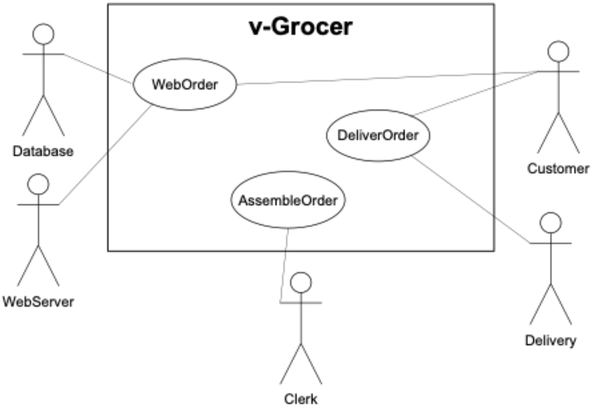
+ChangeAddr() : bool

1- Static View

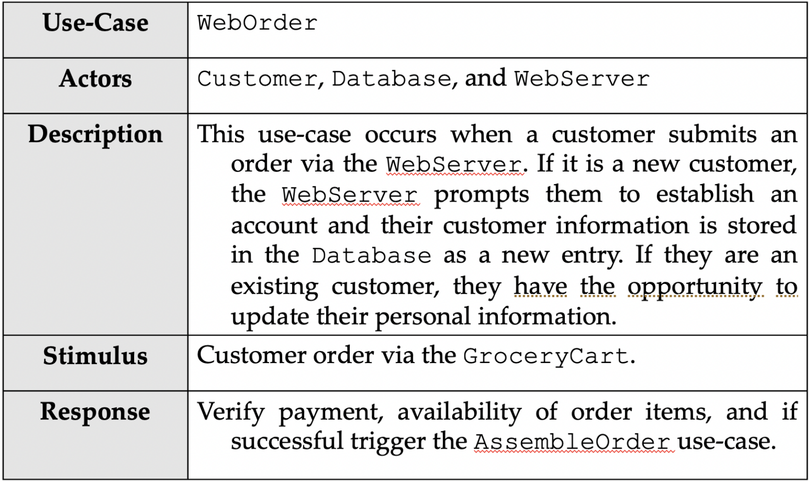


2- Use-Case View

- Used to capture the overall behavior from the user point of view
- Characterized by a Use-Case Diagram

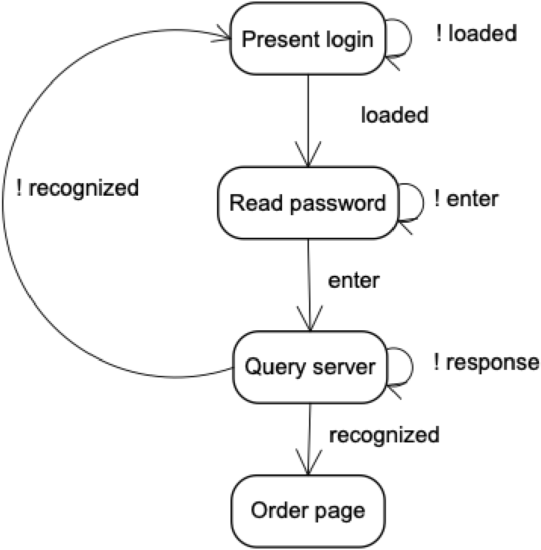


2- Use-Case View



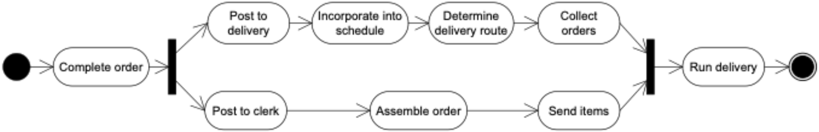
Use-Case	`WebOrder`
Actors	`Customer`, `Database`, and `WebServer`
Description	This use-case occurs when a customer submits an order via the `WebServer`. If it is a new customer, the `WebServer` prompts them to establish an account and their customer information is stored in the `Database` as a new entry. If they are an existing customer, they have the opportunity to update their personal information.
Stimulus	Customer order via the `GroceryCart`.
Response	Verify payment, availability of order items, and if successful trigger the `AssembleOrder` use-case.

3- State Machine View



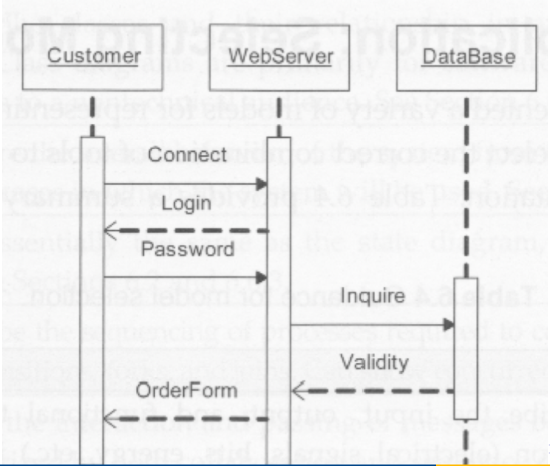
4- Activity View

- Intention = describe a sequence of activities needed to complete a task.



5- Interaction View

- Intention = to show interaction between objects (when they must cooperate to do something useful)
- Example is sequence diagram



6- Physical View

- Show the physical components that constitute the system.
- Can think of this much more generally than presentation in UML.

Summary

- Models are an abstraction of system.
- Models can be thought of as a design specification.
- Models have different intentions for describing behavior.
- Models should encourage innovation and provide for clear documentation.

Questions