Design

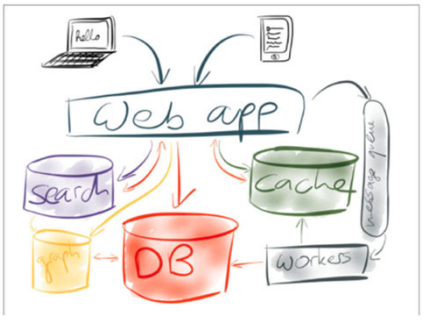
CSC 326

© NC State Software Engineering Faculty

1

Imaging you have an idea about software design, what will you do (to share with you teammate)?

You draw!



2

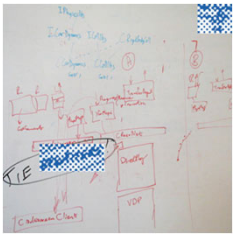
Why Study Design Notation?

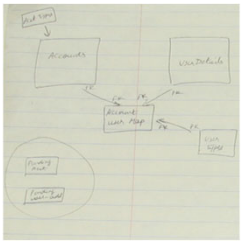
- Drawing and diagramming is essential task in software development

Understanding existing code
Design review
Explain to external members

Ad-hoc meetings (whiteboard)
Onboarding
Hallway art

Refactoring plans
Explain to secondary members
Documentation







"Let's go to the whiteboard: how and why software developers use drawings" **CHI'07** http://research.microsoft.com/pubs/74243/p557-cherubini.pdf

© NC State Software Engineering Faculty

3

Types of Diagrams

Interface Design
- Wireframes
- Flow Maps/Navigation Maps

Software Design
- Class Diagrams
- Sequence Diagrams
- State Diagrams

…

Many more specialized types



Specific diagram types [edit]

A
- Activity diagram used in UML 6/9 and SysML

B
- Bachman diagram
- Booch – used in software engineering
- Block diagram
- Block Definition Diagram (BDD) used in SysML
- Business & IT Diagram (B&IT) – used in business and IT modelling

C
- Carroll diagram
- Cartogram
- Category theory diagrams
- Cause-and-effect diagram
- Circuit diagram
- Class diagram – from UML 1/9
- Collaboration diagram – from UML 2.0
- Communication diagram – from UML 2.0
- Commutative diagram
- Component diagram – from UML 3/9
- Composite structure diagram – from UML 2.0
- Concept map
- Constellation diagram
- Context diagram
- Control flow diagram
- Contour diagram
- Cordier diagram
- Cross functional flowchart

D
- Data model diagram
- Data flow diagram
- Data structure diagram
- Dendrogram
- Dependency diagram
- Deployment diagram – from UML 9/9
- Dot and cross diagram
- Double bubble map – used in education
- Drakon-chart

E
- Entity-Relationship diagram (ERD)
- Event-driven process chain
- Euler diagram
- Eye diagram – a diagram of a received telecommunications signal
- Express-G
- Extended Functional Flow Block Diagram (EFFBD)

F
- Family tree
- Feynman diagram
- Flow chart
- Flow process chart
- Fusion diagram

L
- Ladder diagram
- Line of balance
- Link grammar diagram

M
- Martin ERD
- Message Sequence Chart
- Mind map – used for learning, brainstorming, memory, visual thinking and problem solving

N
- N2
- Nassi–Shneiderman diagram or structogram – a representation for structured programming
- Nomogram
- [Network connection]

O
- Object diagram – from UML 2/9

P
- Package diagram from UML 4/9 and SysML
- Parametric diagram from SysML
- PERT
- Petri net – shows the structure of a distributed system as a directed bipartite graph with annotations
- Phylogenetic tree - represents a phylogeny (evolutionary relationships among groups of organisms)
- Piping and instrumentation diagram (P&ID)
- Phase diagram used to present solid/liquid/gas information
- Plant Diagram
- Pressure volume diagram used to analyse engines
- Pourbaix diagram
- Process flow diagram or PFD – used in chemical engineering
- Program structure diagram

R
- Radar chart
- Radial Diagram
- Requirement Diagram Used in SysML
- Rich Picture
- R-diagram

S
- Sankey diagram – represents material, energy or cost flows with quantity proportional arrows in a process network.
- Sentence diagram – represents the grammatical structure of a natural language sentence.
- Sequence diagram from UML 8/9 and SysML
- SDL/GR diagram – Specification and Description Language. SDL is a formal language used in computer science.
- Smith chart
- Spider chart

© NC State Software Engineering Faculty

4

Unified Modeling Language

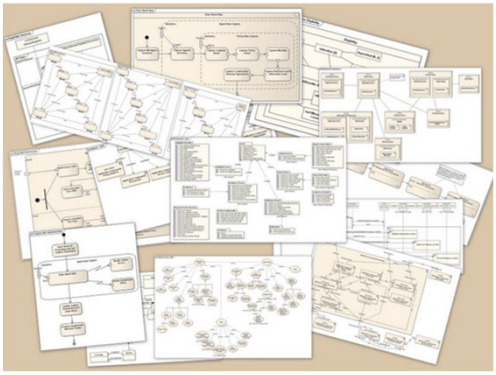
A general-purpose visual modeling language that is intended to provide a standard way to visualize the design of a system.

© NC State Software Engineering Faculty

5

UML - Unified Modeling Language

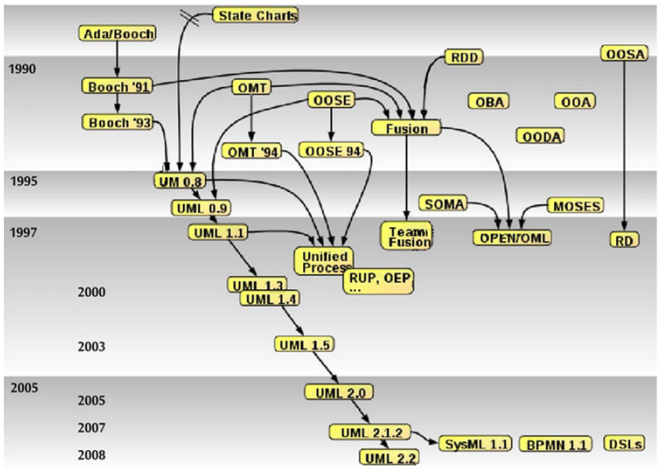
- UML 2.5 (the latest version)
  - A large graphical design language for modeling object-oriented software.
  - Resulted from the convergence of notations from three leading object-oriented methods:
    - Object-Modeling Technique (OMT)
    - Object-Oriented Software Engineering (OOSE)
    - Object-Oriented Software Development (Booch, authored by Grady Booch)



© NC State Software Engineering Faculty

6

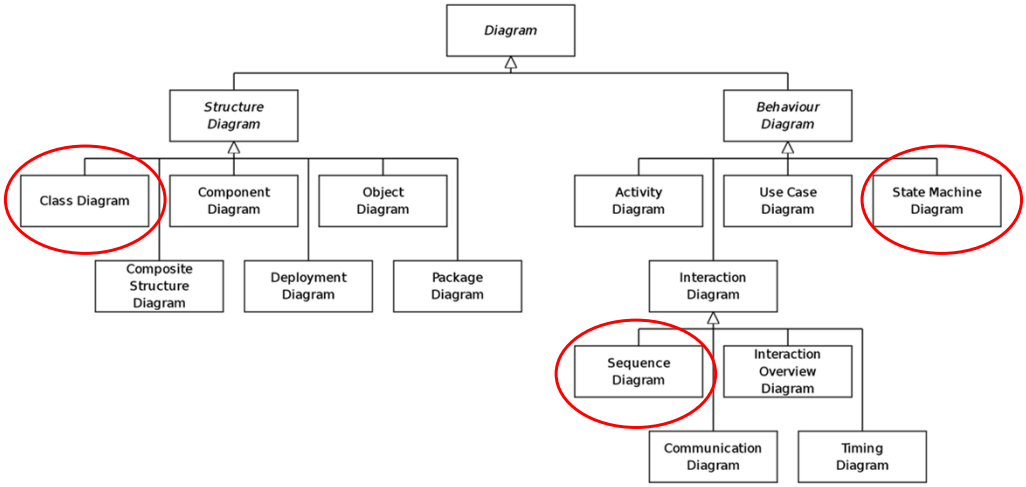
A brief history of UML



https://en.wikipedia.org/wiki/Unified_Modeling_Language#/media/File:OO_Modeling_languages_history.jpg

7

Types of Diagrams by UML



https://en.wikipedia.org/wiki/File:UML_diagrams_overview.svg

8

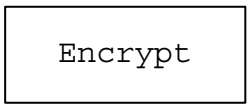
Class Diagrams

© NC State Software Engineering Faculty

9

What is a UML *Class* Diagram?

- It is a developer's view of the significant classes in a design
  - We can highlight what we consider important about a class
  - Other information about a class can be omitted from the diagram
- The simplest representation of a class is a box with the name of the class
  - Example: Class Encrypt

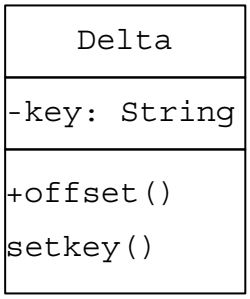


10

10

A Class with Attributes and Operations

- More generally, a class representation may show:
  - The class name is required
  - Attributes are like data fields
  - Operations or methods
- Only significant attributes and operations are shown



class name

attributes

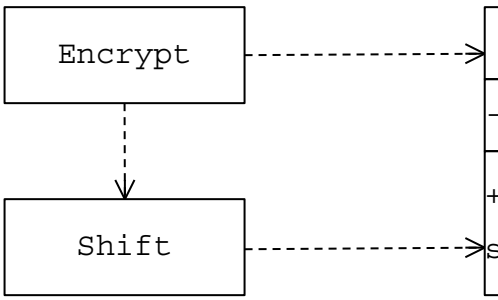
operations

11

11

A UML Class Diagram

The dashed arrows show dependencies; e.g., `Encrypt` depends on `Shift`



class name

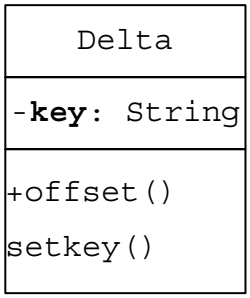
attributes

operations

12

12

UML is a Modeling Language

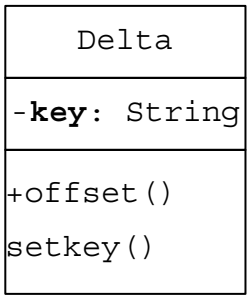


- If a class models a real-world entity
  - An attribute models a meaningful property of the entity
- Example: Class `Delta` is a **key** manager in an encryption system
  - Attribute `key` models an encryption key
- Attributes and operations can be made public or kept private
  - Class diagrams can therefore include private implementation details

13

13

Writing Class Properties as Attributes



- An attribute description has the following elements:
  - Visibility: + for public, - for private
  - Name: Only the name is required
  - Type
  - Multiplicity is about the number of objects represented by an attribute
  - Default value
- If specified, elements appear in the above order

14

14

Writing Attributes Within Class Diagrams

Only the name is required

- Write the name and the type
  - `center : Point`
- Add a default value
  - `center : Point = origin`
- Write the name and the visibility (public)
  - `+center`

15

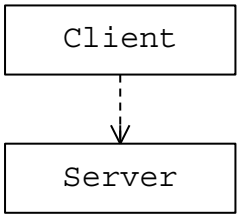
Multiplicity of Objects Represented by an Attribute

- A class may model at least one and possibly many objects
  - Example: Class `Student` models many students
  - Example: Class `Parser` models one syntax analyzer in a compiler
- Multiplicity specifies a range for the number of objects for an attribute
  - [*] one or more
  - [m..n] where m>=0 and n>=m
  - [n] abbreviation for 1..n

16

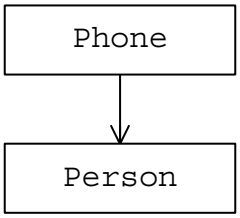
Relationships Between Classes

Note the different kinds of arrows and arrowheads



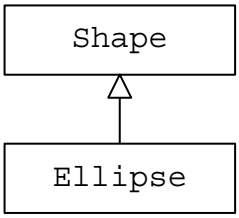
(a) Dependency

(*client depends on server. Server does not depend on client.*)



(b) Association

(each phone *belongs to / is associated with a person*)



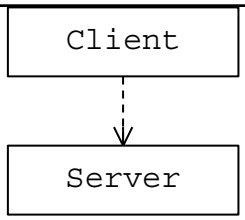
(c) Generalization

(An Ellipse *is a type of Shape. A Shape is a generalization of an Ellipse.*)

17

17

Class Dependencies



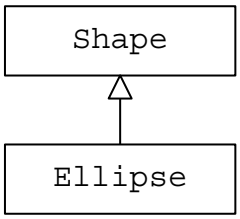
- A dependency is represented by a dashed arrow from the dependent class
  - The most common form of dependency is caller-callee or client-server
  - Dependencies can be through any form of coupling
- Example:
  - In a Publish-Subscribe architecture, the publisher raises events
  - The subscribers watch for events
  - The subscribers depend on the publisher, but they do not call it

18

18

Class Generalization (Subclass-Superclass)

- A generalization is represented by an unfilled triangular arrowhead
  - The arrow is from the subclass to the superclass



19

19

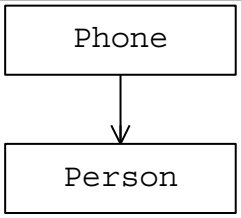
Class Properties as Associations

- UML provides two ways of modeling class properties
  - As attributes, written as lines of text within the box for the class
  - As associations, represented by an open arrow to the property
- How to choose between the two?
  - Use attributes when the property is relevant primarily to the given class
  - Use an association when the property has significance of its own

20

20

Showing Associations



- Associations can be bidirectional
  - Example: A student takes courses and courses have students
  - Bidirectional associations can be shown with arrows at both ends of a line
  - Alternatively, use a line without arrowheads at either end
- Associations can be named
  - Use verbs as names for readability
  - Example: The name "takes" for the association from student to course
  - Write the name next to the from class, or use a triangle to show direction

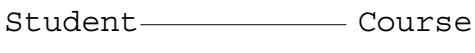
21

21

Ways of Writing Associations



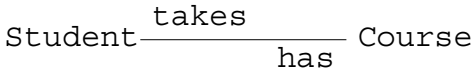
(a) Bidirectional association



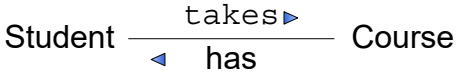
(b) Bidirectional association



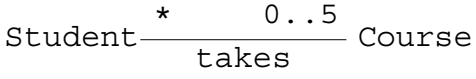
(c) Multiplicities at both ends



(d) Roles for both classes



(e) Verbs for names



(f) Role and multiplicities

22

22

Aggregation vs. Composition Whole-Part Relationship

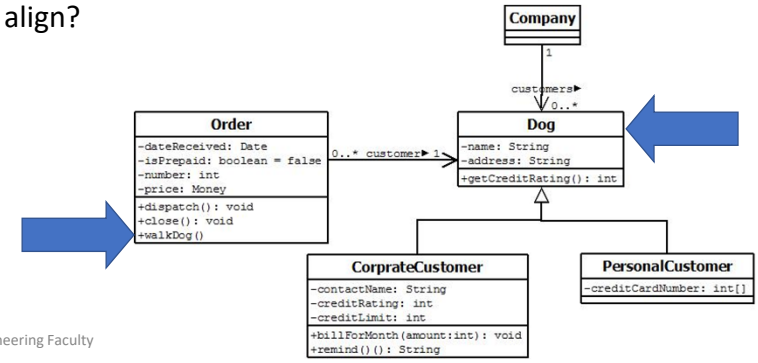
- Aggregation: stronger association (unidirectional)
  - Whole-Part relationship, where the part can exist separate from the whole
  - Ex. A dept contains a set of employees
  - Ex. A faculty contains a set of teachers
  - A white diamond at the end of the association next to the aggregate class
- Composition: stronger aggregation (unidirectional)
  - Whole-Part relationship, where the whole controls the lifecycle of the part
  - Ex. Invoice–InvoiceLine
  - A black diamond on the end of association next to the composite class

© NC State Software Engineering Faculty

23

Evaluating a Class/Class Diagram

1. Intention-revealing naming: Does the name of the object convey its abstractions? Does the abstraction have a natural meaning and use in the domain?
2. Single Responsibility: Do the name, main responsibility statement data and functions align?



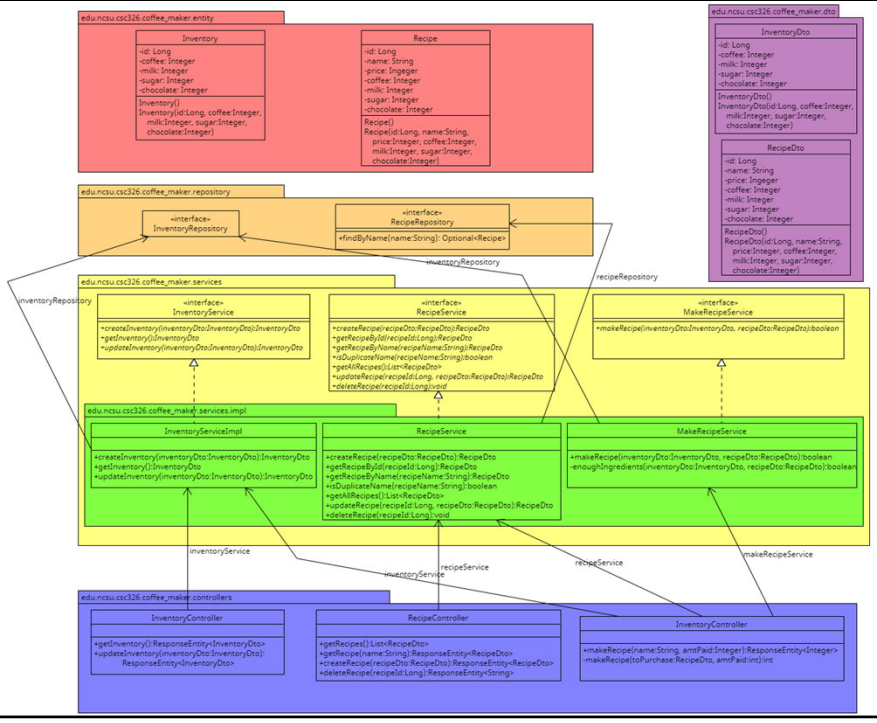
© NC State Software Engineering Faculty

24

Entity-Relationship Diagrams

25

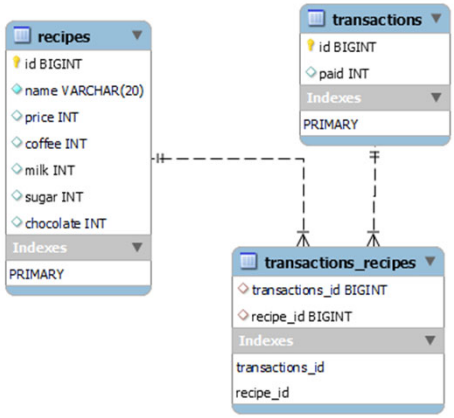
CoffeeMaker Backend Class Diagram (Initial Requirements)



26

Entity-Relationship (ER) Diagrams

- Relational databases can be modeled using Entity-Relationship (ER) Diagrams
  - Tables show attributes and other important information.
  - Connectors show relationships:
    - These connectors show a 1 to many relationship
- You can create these diagrams in MySQL Workbench (or reverse engineer them)
- Since we are using an ORM, the class diagram provides a description of the data in the database, but ER diagrams provide additional context, particularly around joins.



© NC State Software Engineering Faculty

27

Sequence Diagrams

© NC State Software Engineering Faculty

28

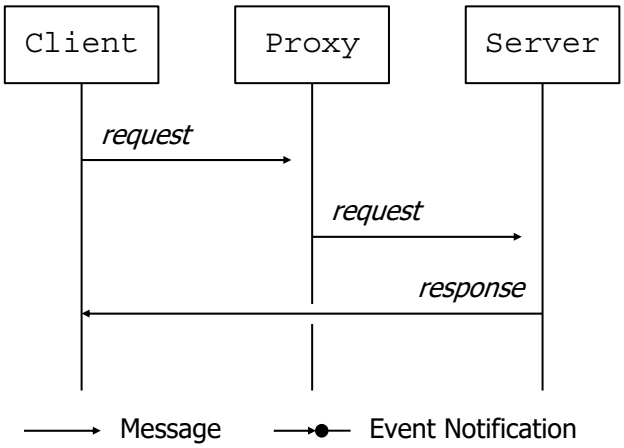
Sequence Diagrams

- A sequence diagram shows communication between run-time components
  - Objects and processes are examples of components
- Time flows top down
  - Each component is represented by a vertical line
  - The line is topped by a box with the name of the component
- A message to B is modeled by a horizontal arrow from A's line to B's line

29

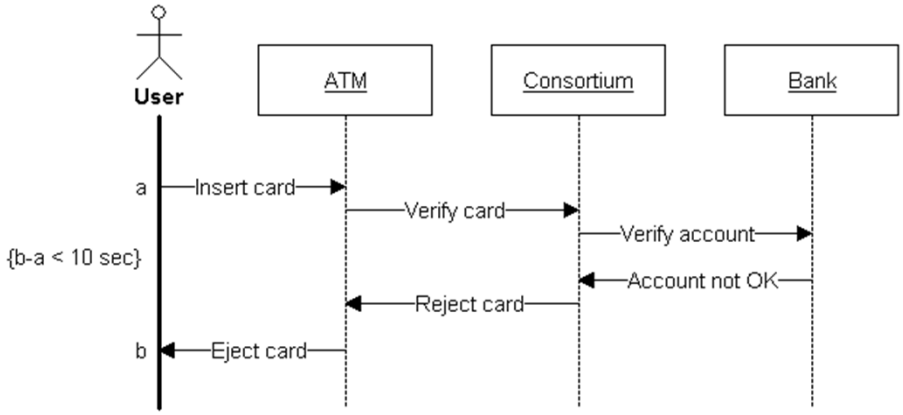
A Sequence Diagram

The client's request is forwarded by the proxy



30

Scenario - withdraw cash

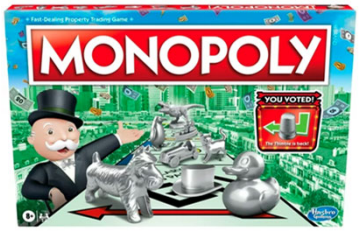


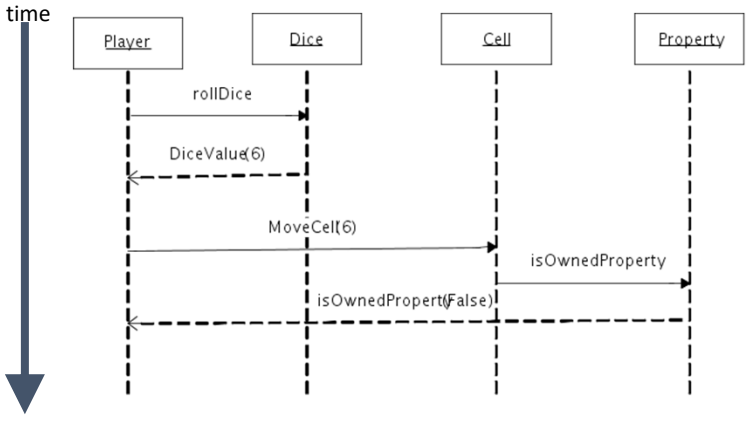
© NC State Software Engineering Faculty

31

Scenario - Monopoly

A player rolls the dice and gets a 6. The player moves 6 cells. The player lands on a cell that is an un-owned property. The player's turn is over.



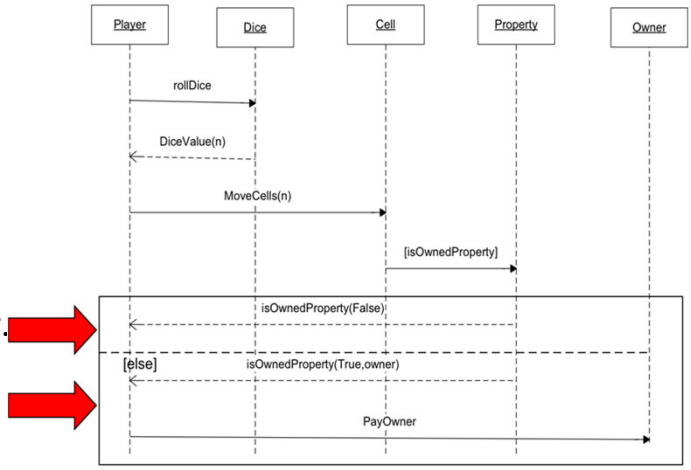


© NC State Software Engineering Faculty

32

Conditional Logic

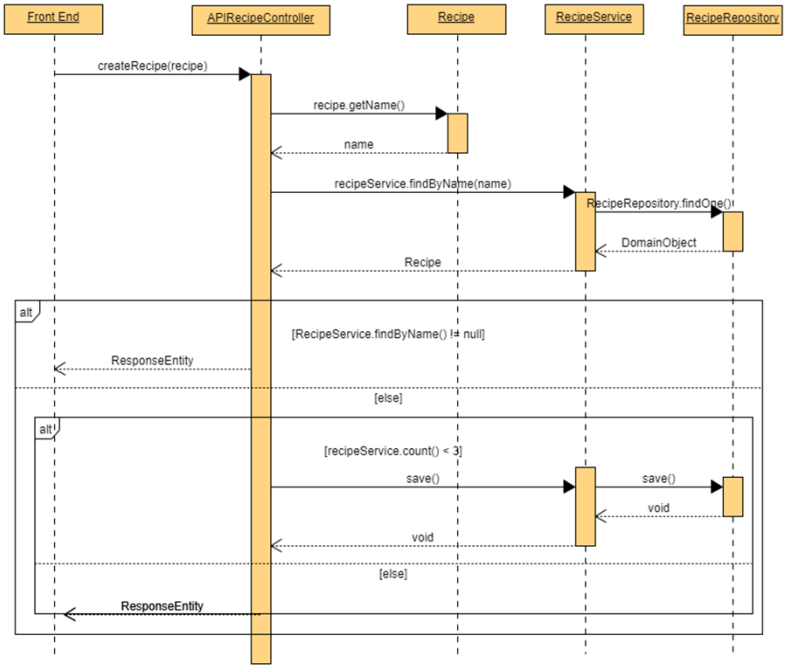
- If the player lands on a cell that is unowned property, the player's turn is over
- If the player lands on a cell that is owned, the player must pay rent to the owner of the property
- Then the player's turn is over.



© NC State Software Engineering Faculty

33

Example from old version of CoffeeMaker

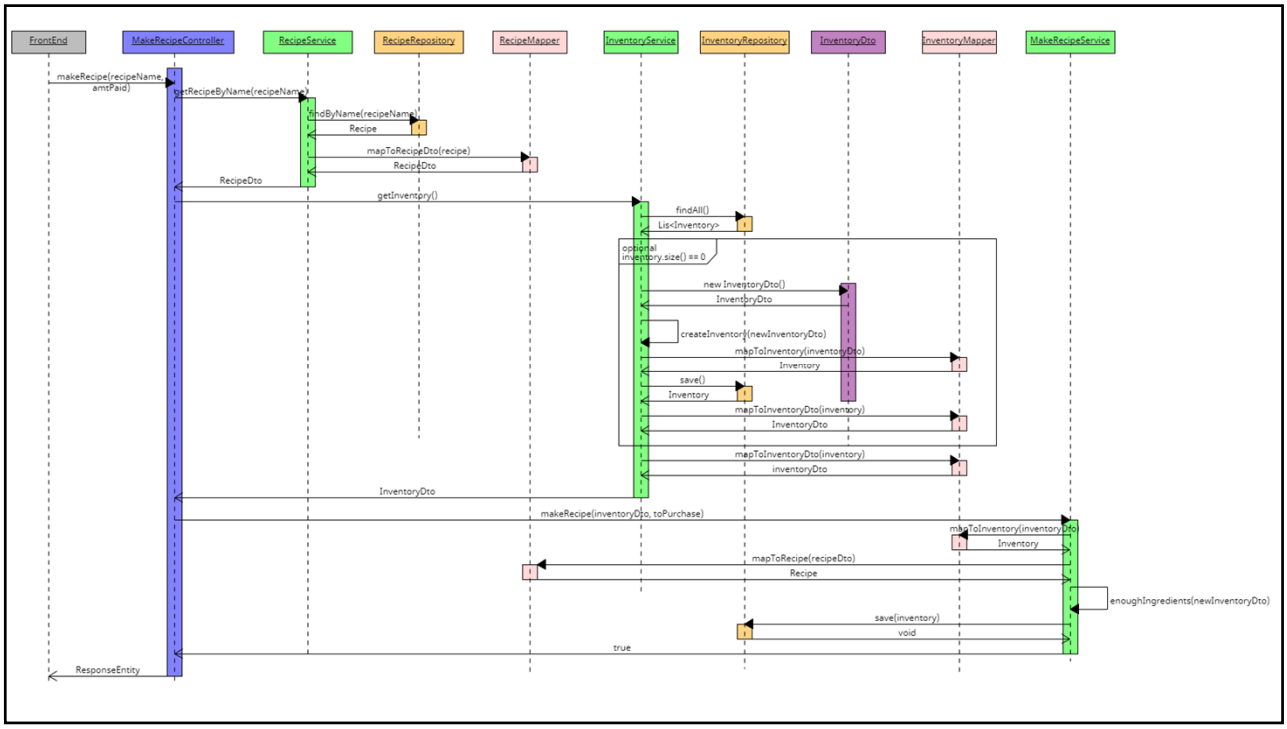


34

Uses

- (obvious, formal) Designing flow of information through a system
- (less obvious, less formal use) Debugging a complex system
- (pragmatic) Figuring out what to test
  - Hint: be sure to cover each arrow! These are integration points!

35



36

State Diagrams

© NC State Software Engineering Faculty

37

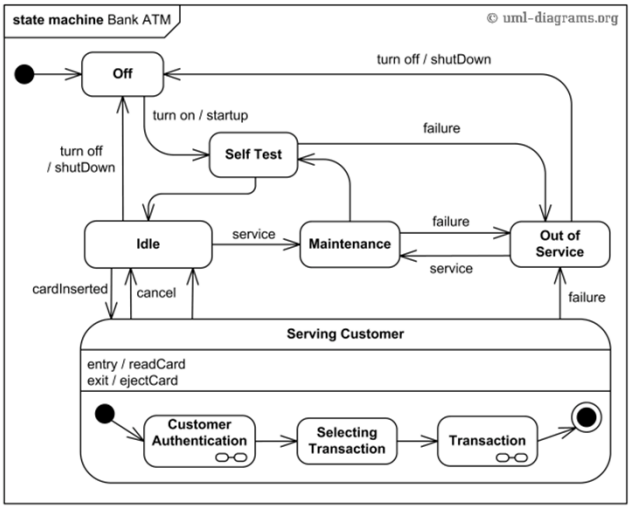
State Diagrams

- A state diagram (also called a state machine diagram) depicts the various states that an object may be in and the transitions between those states.
- Appropriate to be developed for complex objects.

© NC State Software Engineering Faculty

38

ATM Example



39

Transitions

- Transitions are the result of the invocation of a method that causes an important change in state
- Each transition has a label that comes in three parts. All the parts are optional.

trigger-signature [guard] / activity

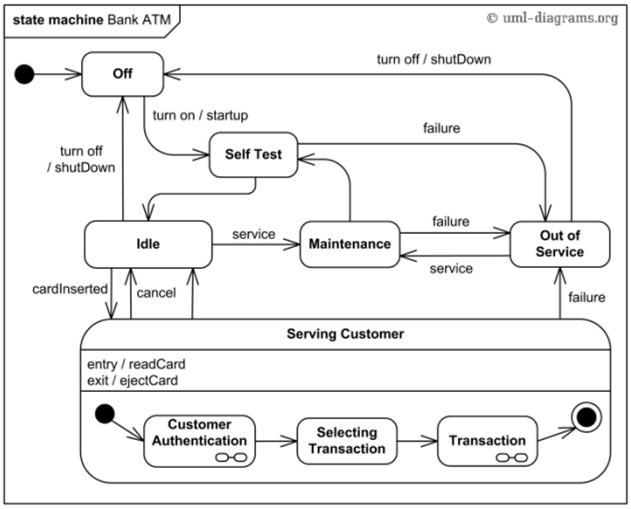
candle removed [door closed] / reveal lock

- The *trigger-signature* is usually a single event that triggers a potential change of state.
  - Missing trigger-signature [rare] – you take the transition immediately.
- The *guard*, if present, is a Boolean condition that must be true for the transition to be taken.
  - Missing guard – always take the transition.
- The *activity* is some behavior that's "executed" during the transition.
  - Missing activity – don't do anything during the transition.

© NC State Software Engineering Faculty

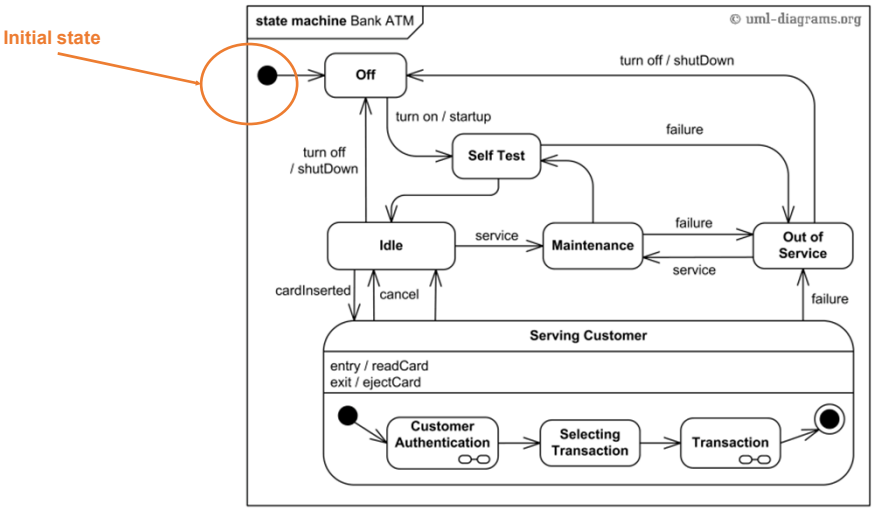
40

ATM Example



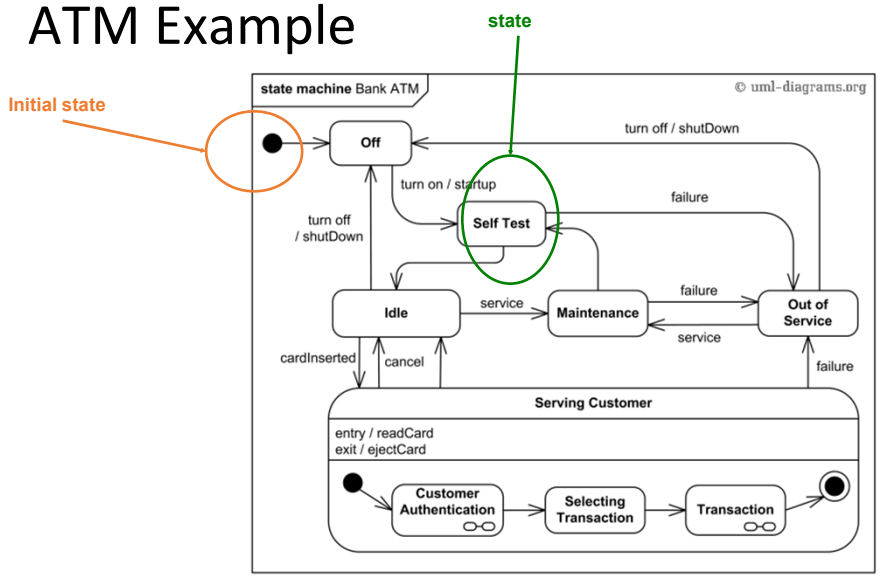
41

ATM Example



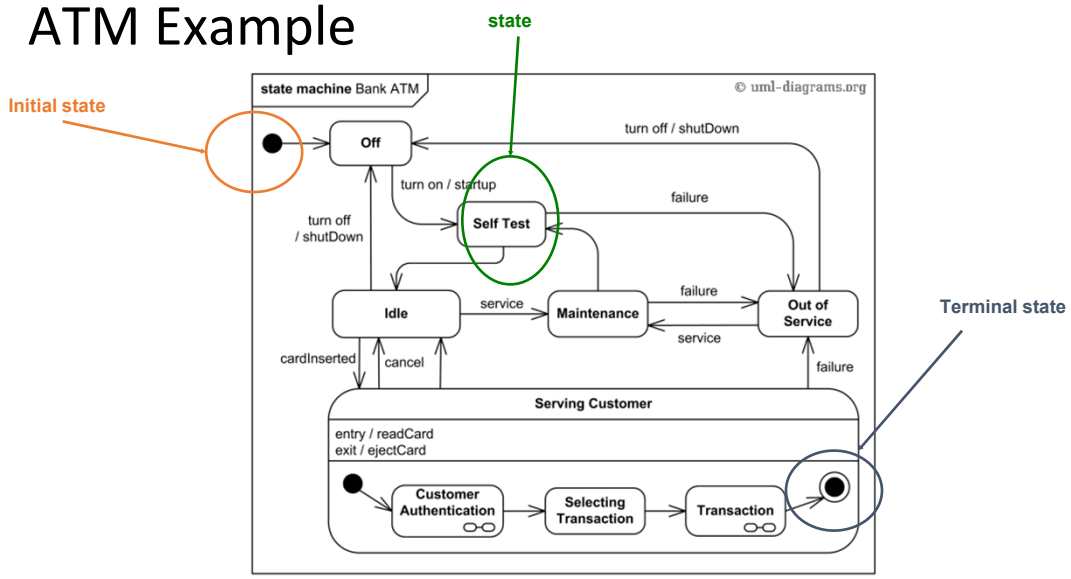
42

ATM Example



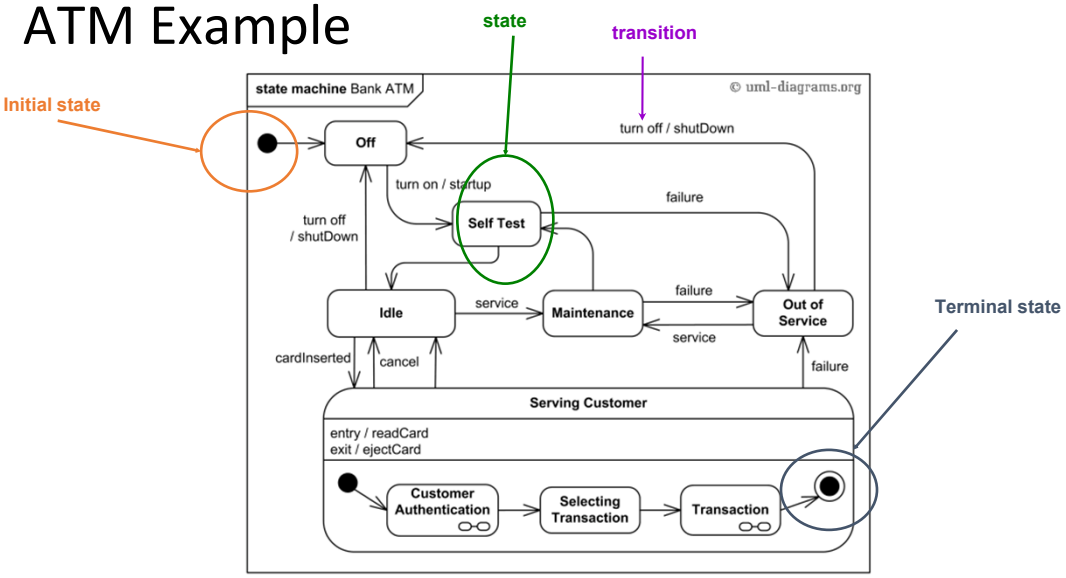
43

ATM Example



44

ATM Example



45

Rules for State Diagrams

- There is one initial state (can be multiple final states).
- Every state can be reached from the initial state.
- From each state, there must be a path to a final state.
- Every transition between states must be labeled with an event that will cause that transition.
- When an event occurs, you can take only one transition. If you have multiple transitions with the same event, the guards must be mutually exclusive.
- Transitions that are not shown are illegal OR show transitions that cause errors.

© NC State Software Engineering Faculty

46

Uses

- Understand all paths through a system
- Testing all paths through a system (up to length n, when there are loops)
- Ensuring all error states recover elegantly

47

Onboarding Project – Backend Design

48

Requirements Update!

- CoffeeMaker is currently limited to only 4 ingredients. Our update will allow arbitrary ingredients!
  - UC2: Add Recipe – updated
  - UC4: Edit Recipe – updated
  - UC5: Update Inventory – updated
  - UC7: Add Ingredients – new
- Team Portion of OBP: UC5 & UC7
- Individual Portion of OBP: UC2 & UC4

49

Backend Design Expectations

- Static Design – Class Diagram
  - Team – update backend design for UC5 (Add Inventory) and UC7 (Add Ingredients)
  - Individual – update backend design for UC2 (Add Recipe) and UC4 (Edit Recipe)
- Dynamic Design – Sequence Diagrams
  - Team – sequence diagram for ***one*** scenario related to UC5 and/or UC7, scenario description, and justification for successful implementation
  - Individual – sequence diagram for ***one*** scenario related to UC2 and/or UC4, scenario description, and justification for successful implementation

50