TCSS 360: SOFTWARE DEVELOPMENT AND QUALITY ASSURANCE

Unified Modeling Language

Wes J. Lloyd
Institute of Technology
University of Washington - Tacoma

UML: ORIGINS

- **1980s:** object-oriented programming moved from research labs into the real world
  - Popular OO languages: Smalltalk and C++
  - Spurred interest in Object-oriented graphical design languages
- **1988-1992:** originators were Booch, Coad, Jacobson, Odell, Rumbaugh, Shlaer, Mellor, and Wirfs-Brock
- Basic OO concepts would reappear in very different notations, causing confusion
- Jim Rumbaugh left GE to join Grady Booch at Rational
  - Alliance formed, critical mass of market share followed
- **1997:** Rational released UML 1.0

OBJECTIVES

- From chapter 11.2: Just Enough UML
  - UML uses
  - UML diagrams
    - Use case
    - Class
    - State
    - Sequence

UML USES: SKETCH

- Most common use of UML
- Used to communicate some aspect of a system, to helps others better understand it
- Used for
  - forward engineering: build diagrams before coding
  - reverse engineering: build diagrams from existing code
- Strives to be informal and dynamic
- Emphasize only classes, attributes, operations, and relationships of interest
- More concerned with selective communication than complete specification

WHAT IS UML?

- UML: Unified Modeling Language
- Depicts object-oriented software systems ***visually***
- Open standard used most often with plan-and-document software development processes
- Descriptive language: rigid formal syntax
  - But not often rigidly applied
- Use in Agile:
  uses UML sparsely to illustrate key aspects of the system
- Use in Plan-and-document:
  highly formal processes with complete full set of use case, class, and sequence diagrams

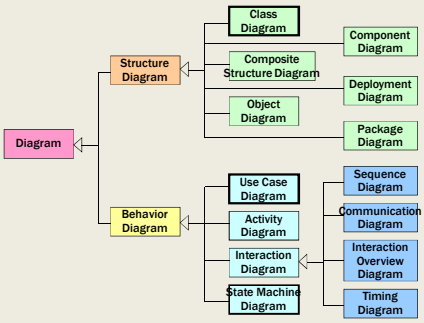
UML USES: BLUEPRINT

- Goal is completeness
- More definitive, while sketch is explorative
- Describes detailed design to follow in writing source code
- Complete so programmer can follow it
- Can be used to develop blueprint-level models that show interfaces of subsystems or classes
  - Developers then work out the implementation details
- Reversed engineered UML: diagrams convey detail about source code that is easy for developers to understand

UML USES: AS A PROGRAMMING LANGUAGE

- Specifies the complete system in UML so code can be automatically generated
- Use from the coding perspective, rather than a conceptual perspective
- Diagrams compiled directly into executable code → *The UML becomes the source code*
- Productivity challenge: strive to require less effort than coding in other programming language(s)
- Model specification challenge: how to formally model behavioral logic of UML – required for code generation
  - Done with interaction, state, and activity diagrams

CLASSIFICATION OF DIAGRAM TYPES



TYPES OF UML DIAGRAMS

Diagram Name	Purpose
Activity	Models procedural and parallel behavior
Class	Models classes, attributes, operations and relationships
Communication	Models interaction between objects
Component	Models structure and connection of components
Composite Structure	Models runtime decomposition of a class
Deployment	Models deployment of artifacts to nodes
Interaction overview	Mixes the sequence and activity diagram

USE CASE DIAGRAM

- Use cases serve as a technique for capturing functional requirements of a system
- Describes typical interactions between the users and the system itself; provides a narrative of how a system is used
- Use case consists of a set of one or more scenarios tied together by a common user goal
- Users are referred to as actors; an ***actor*** is a role that carries out a use case
- An ***actor*** need not always be a person; can also be an external automated or manual system

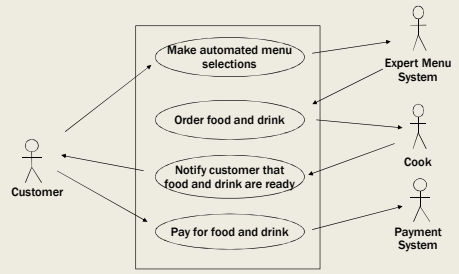
TYPES OF UML DIAGRAMS - 2

Diagram Name	Purpose
Object	Models example configurations of instances
Package	Models compile-time hierarchical structure
Sequence	Models sequence interaction between objects
State Machine	Models how events change an object over its life
Timing	Models timing interaction between objects
Use Case	Models how users interact with a system

USE CASE DIAGRAM - 2

- A use case diagram is like a graphical "table of contents" of the use cases for a system
  - It shows the use cases, the actors, and the relationships between them
- Use cases represent external view of the system
- No correlation to the classes in the system
  - They can serve as a starting point for writing software validation test cases

USE CASE DIAGRAM EXAMPLE



CLASS DIAGRAMS

- Describes types of objects in the system and the various kinds of static relationships that exist among them
- Shows properties and operations of a class and constraints that apply to object connections
- Class box has three parts:
  - Name of the class
  - Attributes of the class
  - Operations of the class

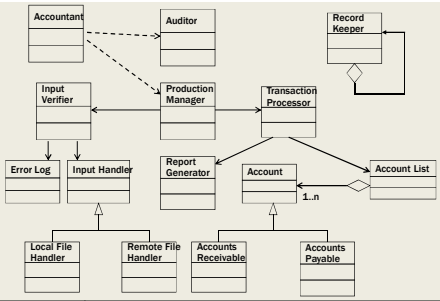
Name
Attributes
Operations

- Properties represent structural features of a class and consist of attributes and associations

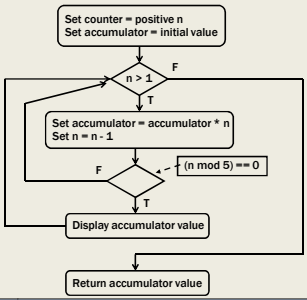
ACTIVITY DIAGRAM

- Serves as a technique to describe procedural logic, business process logic, and work flow
- Is similar to a flowchart except that it can also show parallel behavior
- States the essential sequencing rules to follow, thereby allowing concurrent algorithms to be used
  - Consequently, an activity diagram allows whoever is doing the process to choose the order in which to do certain things
- Can be used to describe the actions in a use case

EXAMPLE CLASS DIAGRAM



ACTIVITY DIAGRAM EXAMPLE



CLASS BOX

Name of class
Attributes: object properties
Operations: object methods

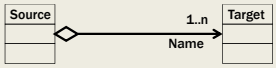
CLASS DIAGRAM: ATTRIBUTE

`visibility name: type multiplicity = default {property-string}`

- Example
  `+ criticalMsg: String [1] = "Error message" {readonly}`
- Syntax
  - Visibility marker: public (+) or private (-)
  - Name: name of the attribute in the programming language
  - Type: Type of the attribute in the programming language
  - Multiplicity: how many objects fill the property
  - Default: Default value of the attribute at instantiation
  - {property-string}: additional properties of the attribute
- Describes a property as a line of text within the class box
- Used for representing value types

CLASS DIAGRAM: AGGREGATION AND COMPOSITION

- Aggregation and composition are sometimes viewed as special types of associations and have their own UML symbol of a diamond at the source end of a line
- Aggregation is a part-of relationship
- Composition is more restrictive than aggregation
  - Diamond is filled in (i.e. shaded)
  - Part pointed to doesn't exist without the whole



CLASS DIAGRAM: OPERATION

`visibility name (parameter-list) : return-type {property-string}`

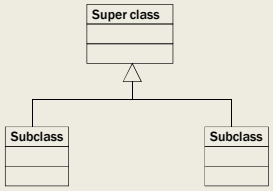
- Example:

`+ computeTotal (account: Account) : float`

- Syntax
  - Visibility marker: public (+) or private (-)
  - Name: name of the operation in the programming language
  - Parameter-list: list of parameters passed to the operation
    - Syntax: `direction name : type = default-value`
    - Direction is (in), (out), or (inout); default is (in)
  - Return-type: Type of the return value if there is one
  - {property-string}: additional properties of the operation
- Portrays actions that a class carries out (e.g. methods)
- Include: queries or modifiers; modifiers change object state

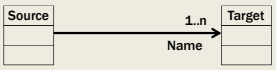
CLASS DIAGRAM: GENERALIZATION

- Portrays inheritance between a super class and a subclass
- Represented by line with triangle at the target end



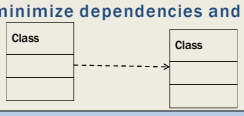
CLASS DIAGRAM: ASSOCIATION

- Represented by a solid line between two classes directed from source to target class
- Points to dependent (coupled) classes
- Name of the association goes at the target end
- Target end links the class that is the type of the property
- Multiplicities are shown at both ends but usually at the target end
- Arrows may be bidirectional



CLASS DIAGRAM: DEPENDENCY (COUPLING)

- **Dependency:** between two elements if changes to the definition of one element (i.e., the source or supplier) may cause changes to the other element (i.e., the client)
- Examples
  - Class sends a message to another class
  - Class mentions another as a parameter to an operation
- If class changes its interface, then messages sent to that class may no longer be valid
- General rule: minimize dependencies and note cycles



CLASS DIAGRAMS: WHEN TO USE

- Class diagrams are the backbone of UML and are the most used diagrams
- Normally use only a subset of the notations available: class box, attributes, operations, association, aggregation, and generalization
- Class diagrams only model software structure; it is easy to get too focused on class diagrams and ignore behavior
  - State diagrams support modeling class behavior
  - Sequence diagrams model interactions among objects of various classes

TUTORIAL 4

- Java data persistence w/ postgreSQL and heroku…
- http://faculty.washington.edu/wlloyd/courses/tcss360/tutorials/TCSS360_w2017_Tutorial_4.pdf

UML: FOR REQUIREMENTS ANALYSIS

- **Use case diagrams** help describe how people interact with the system
- **Activity diagrams** show context for use cases and details of how complicated use cases work
- **Class diagram** drawn from conceptual perspectives, helps build a rigorous vocabulary of the domain
  - Shows attributes and operations of interest in domain classes and relationships among the classes
- **State diagrams** show various states of a domain class and events that change that state

QUESTIONS

UML: FOR SOFTWARE DESIGN

- **Class diagrams**: Show design classes, attributes and operations, and their relationships with domain classes
- **Sequence diagrams:** help combine use cases to see what happens in the software
- **Package diagrams:** shows large-scale organization of the software (packages of classes)
- **State diagrams:** shows various states of objects (classes), and events that change their state
- **Deployment diagram** shows the physical layout of the software

THE CONTENTS OF THIS SLIDE SET ARE BASED ON THE FOLLOWING REFERENCES:

- Armando Fox, David Patterson, Engineering Software As A Service: An Agile Approach Using Cloud Computing, 1st edition (v1.2.1), Strawberry Canyon LLC., 2016. ISBN-13: 978-0984881246. [Chapter 11]
- Martin Fowler, *UML Distilled, 3rd edition*. Addison-Wesley, 2004. [Chapters 3, 5, 9, 11]

UML: ORIGINS - 2

- Family of graphical notations to help in describing and designing software systems
- Focuses particularly object-oriented software designs
- Coordinated by the Object Management Group
  - International, open membership, not-for-profit technology standards open consortium of companies
- From the unification of many OO graphical modeling languages that thrived in the 1980s and early 1990s