CS2141 – Software Development using C/C++

UML
Introduction

• UML = Unified Modeling Language
• It is a standardized visual modeling language
  – Primarily intended for modeling software systems
  – Also used for business modeling
• UML evolved from earlier competing modeling languages
  – Based on the best parts of those earlier methods
  – Has continued to evolve since its creation
• UML is NOT a visual programming language
Architectural Views of UML

- UML is centered around a number of different types of diagrams, each modeling the system from a different perspective
  - *Use case diagrams* model the functionality of the system from the users' perspective
  - Structural diagrams model the static structure of a system
    - *Class diagrams* show the overall structure
    - *Object diagrams* show the structure at a particular time
Architectural View of UML cont.

- Interaction diagrams model the interaction of objects as they perform some operation
  - Sequence diagrams model the sequences of messages that are sent between objects to carry out some operation
  - Collaboration diagrams show the roles objects play in carrying out some operation

- Behavioral diagrams model the behavior of objects
  - A state diagram models the states an object can be in and the stimuli that cause it to change states
  - Activity diagrams show how the behaviors of objects involved in some operation depend on each other
Architectural Views of UML cont.

- Physical diagrams show how the parts of a system are organized in the real world.
  - A *component diagram* shows the organization of the parts of the system into packages.
  - *Deployment diagrams* display the physical locations of the components of the system.
Why use UML?

• Communicate information about a system
  – Diagrams can be understood by non-programmers
  – Models can serve as a blueprint for a system
  – Models can help document a system

• Even if the diagram itself is ultimately discarded, the act of creating it is useful since it helps you to understand whatever it is you're modeling
Use Case Diagrams

• A use case diagram models the users' view of the system
  – Describes what the system does, not how it does it
  – Shows how the user interacts with the system

• Useful for:
  – Determining features
  – Communicating with clients
  – Generating testcases
Use Case Diagrams cont.

- **Basic Vocabulary**
  - **Actor**: A person or thing involved in some task
  - **Use case**: Something the user does with the system
  - **Communication**: Lines linking actors and use cases
Example

- Use case diagram for a text editor:
Use Case Diagrams cont.

• More vocabulary:
  – Include - Like a procedure call
  – Extend - Like a procedure that is called sometimes depending on some condition
  – Generalizations - A specialization of some case
  – Boundary box - Group use cases together

• Examples on next slide...
Example 2

- Another use case diagram for a text editor:
Class Diagrams

• A class diagram models the classes in a system and how they are related

• Classes are modeled as boxes with compartments for:
  – The class name
  – Attributes - the data members of the class
  – Operations - the methods of the class
Class Diagrams cont.

- Compartments (except the name) can be omitted if not needed for the purpose of the diagram.
- Characters placed in front of class members indicate visibility:
  - + Public
  - # Protected
  - − Private
  - ~ Package
Class Diagrams cont.

- Other class modeling details:
  - The order of the compartments is always the same: class name, attributes, and operations
  - Members are listed in order of decreasing visibility, from public down to private
  - Functions for getting and setting attributes are often omitted from the diagram
  - Abstract classes are represented by having their class name in italics
  - Pure virtual functions also have their names in italics
Class Diagrams cont.

• Many different relationships:
  
  - *Associations* - Arrows indicate the direction of the relation. Class1 and Class2 know about each other, and Class2 knows about Class3, but Class3 is not aware of anyone else.
  
  - *Generalization* - Indicates inheritance - the Parent is a generalization of the Child1 and Child2.
Class Diagrams cont.

- **Composition** - A is composed of Bs, like a building is composed of rooms. Usually the lifetime of B is strongly tied to the lifetime of A.
  - **Aggregation** - Weaker form of composition. C has a collection of Ds, like a shopping list has a collection of items.
  - Don't worry too much about getting the diamonds right - if in doubt, don't include them.
Class Diagrams cont.

- **Multiplicity** indicates the number of instances that can be on either end of a relationship.
  - 0..1 Zero or one instance
  - 0..* Any number
  - 1 Exactly one instance
  - 1..* At least one
  - n..m General form
Example

• Class diagram for a text editor:

- **SpellChecker**
  - `dictionary: vector<string>`
  + `checkSpelling(text:string): void`
  + `loadDictionary(filename:string): void`
  - `addToDictionary(word:string): void`
  + `saveDictionary(filename:string): void`

- **EditorGui**
  + `load: Button`
  + `save: Button`
  + `spellCheck: Button`
  + `quit: Button`
  - `text: TextArea`
  + `loadFile()`
  + `saveFile()`

- **TextArea**
  + `beforeCursor: string`
  + `afterCursor: string`
  + `keyPressed(key:char)`
  + `getText(): string`

- **SpellCheckGui**
  - `ignore: Button`
  - `replace: Button`
  - `add: Button`
  - `curWord: string`
  + `getResponse(string): int`

- **FileChooserGui**
  - `currentDir: string`
  + `getFilename(): string`
Example

Diagram showing a class diagram with classes such as `Anchor`, `Object`, `Stick`, `String`, `Spring`, and `Spacer`. The classes have attributes like `mass`, `theta`, `force`, `velocity`, `position`, and `lastPosition` for `Object`, and `a`, `b`, and `length` for `Anchor`, `Stick`, `String`, `Spring`, and `Spacer`. The diagram illustrates the relationships and associations between these classes.
Object Diagrams

- An object diagram shows instances of classes and their relationships at a particular point in time
- Useful for explaining complex relationships
- Consider this small class diagram:
Class Diagrams cont.

- An object diagram could show how instances of those classes are used to represent a house:
Sequence Diagrams

• A sequence diagram details how an operation is carried out
  – Shows what messages are from one object to another and when they are sent
  – Organized vertically by time - time flows down
  – Horizontal axis shows classes or class roles
  – Usually an individual diagram shows the sequence of events for some particular feature rather than for the whole program
Sequence Diagrams cont.

- **Class Identification** - a box with underlined name in form of \textit{InstanceName : ClassName}.

- **Class Lifeline** - a dotted line indicating the object exists.

- **Termination** - An X at the end of the lifeline indicating the object was destroyed.
Sequence Diagrams cont.

- **Activation** - A box over the lifeline indicates that class or object has control.
  - *Simple message* - A line with a line arrow indicates a message or function call.
  - *Synchronous message* - Indicated by a line with a filled arrow. A dashed line with an arrow in opposite direction indicates a return.
**Sequence Diagrams cont.**

- *Asynchronous message* - A line with a half arrow indicates a message that does not stop processing in the sender
  - *Call to self* - An object calling itself is indicated by a message and a sub-activation box
  - Usually messages are labeled
Example

- Sequence diagram for text editor spell checking:

![Sequence diagram](image)
Collaboration Diagrams

- A collaboration diagram models the flow of messages between objects
- Vocabulary is similar to sequence diagrams
  - Classes are represented by boxes with names in the form of \textit{instance/role name : class name}. Instance names are underlined
  - Message types are the same as in sequence diagrams
  - Messages have a sequence number
  - Time is indicated by sequence numbers rather than the arrangement of the diagram
Example

• Collaboration diagram for text editor spell checking:

[Diagram showing a sequence diagram with arrows and method calls such as `getText()`, `checkSpelling(text)`, `getResponse(word)`]
Statechart Diagrams

- A statechart diagram shows the states an object can be in and the transitions between states.

![Statechart Diagram]

- Initial State
- Final State
- Intermediate States
- Transitions

- 'Quit' pressed
- 'Check Spelling' pressed
- 'Save File' pressed
- 'Load File' pressed
- File Saved
- File Loaded
- Spellcheck Complete
Activity Diagrams

• An activity diagram is like a flowchart
• Shows the logic of some operation
  – States are actions
  – Can have multiple objects. The diagram is divided into swimlanes, one lane for each object
  – Can have branches like a flowchart
    • Drawn as diamonds
    • Need guard expressions to label the transitions out
  – Can have forks and joins
Example

• Activity diagram for a vending machine:
Component and Deployment Diagrams

- A component diagram shows the relationships between the major parts of a system

```
Component

Name : Type

Package

Name

Class or Object

Name

Dependency
```
Component and Deployment Diagrams cont.

• A deployment diagram shows where the components of a system are physically located

• In addition to the vocabulary from component diagrams, a deployment diagram uses *nodes* and *communication relationships*:

![Diagram showing deployment of components with nodes and communication relationships]