Software Engineering with the UML

Dr. Clemente Izurieta
Computer Science Department
Montana State University
Agenda

• What is UML and why use it
• UML Diagrams
  – Use case diagram
  – Class diagram
  – Sequence diagram
  – State diagram
• Q/A
UML

- The Unified Modeling language
  - Standard
  - World-wide adoption and a necessity in order to communicate:
    - Application designs
    - Architecture
    - Behavior
    - Structure
    - Deployment
UML

• Maintained by the OMG (Object Management Group), a non profit group for industry specifications
• Current version of UML is 2.2, released in February of 2009.
  – http://www.omg.org/spec/UML/2.2
Use Case Diagrams

• A view of the system that focuses on behavior as seen from outside users
• Describe interactions between users and the system
• The system is the subject under development
• Use cases are used to capture functional requirements
• Describe end to end functionality. Produces a result of value to at least one user and leaves the system in a stable state
Use Case Diagrams Elements

• **Actors**
  – A role played by an external entity (human or otherwise) that interacts with the system

• **System Boundary**
  – The “line” that separates the system from the actors

• **Use Case**
  – A sequence of actions that a system performs
System

- **Actor1**
  - Process A
    - extension points
  - Process B
    - extension points

- **Actor2**

- **External Actors** do not have to be human, they could be other computer systems (servers, databases, etc.)

A use case captures a requirement

Generated by UModel

www.altova.com
Use Case Relationships

• Association
  – Actors communicate with processes

• Specialization/Generalization
  – A specialized use case inherits behavior (sequence of actions) from the parent
  – Can add and override behavior
  – Can be used anywhere a parent can
Use Case Relationships

• Inclusion
  – A relationship from the base use case to the included use case. The base case contains the included case and depends on it

• Extension
  – A relationship from the extended use case to the base case. The extension augments the behavior of the base case. The base case does not depend on the extension use case
Class Diagrams

• Used to represent requirements class models
  – Model domain concepts
  – Classes do not represent software classes, only environment/domain entities

• Used to represent structure of designs
  – Most common use
  – Not all semantics can be captured. Sometimes class diagrams are decorated with OCL
Class Diagrams

- A concept can be represented by a single UML class or a set of UML classes.
- Concepts share the same properties expressed as attributes and relationships between classes.
- A design pattern is a concept.
Class Diagrams
Class Diagrams

- Different kinds of relationships exist
  - **Associations** are structural relationships between objects
    - Can be uni or bi directional
    - **Aggregations** represents a whole-and-part relationship
    - **Composition** is a stronger form of aggregation
  - **Use Dependencies** are temporal (parameters, local variables, etc.)
  - **Generalizations** represent inheritance
  - **Realizations** represent implementation of interfaces
Class Diagrams
Class Multiplicity

Multiplicity:
- * zero or many
- 1..* one or more
- 1..20 one to twenty
- 8 exactly 8
- 3, 8, 9 exactly 3, 8 or 9
Class Reflexive Associations
Class Associations
Class Associations

Class diagram with relationships:
- Store has a relationship with Person:
  - Employee: Person[^]\n  - Contractor: Person[^]

Person has relationships:
- Child: Person[^]
- Parent

XOR relationship between Store and Person.
Aggregation
Composition
Generalization
Realization

![UML Diagram]

- **Shape**
  - draw()
  - CalcArea()

- **Rectangle**
  - getHypotenuse()

- **Circle**
  - getRadius()

- **Triangle**
  - getType()

- **Square**

Generated by UModel | www.altova.com
Use Dependency
A Simple Design Pattern
Behavioral Diagrams

• Class diagrams focus on the static structure of a design
• What about behavior?
  – Sequence diagrams
  – State diagrams
  – Behavior deals with the instantiation of classes (objects) and the interactions at runtime
Sequence Diagrams

- A sequence diagram is an interaction diagram that emphasizes the time ordering of messages.
- A collaboration diagram (not used as much) is semantically equivalent to the sequence diagram, but it emphasizes the structural organization of the objects.

1. The UML User Guide. Booch, Jacobson, Rumbaugh
Sequence Diagrams

• Most messages sent in object oriented programming are synchronous. They block until the receiver returns
Sequence Diagrams

Object1:ClassA

1: Found Message(...)

A message whose caller is not important or known

Generated by UModel

www.altova.com
Sequence Diagrams

- Asynchronous messages do not block
Sequence Diagrams

- Recursion
Sequence Diagrams

1: Create
2: Reply
3: Message
4: Recurse(...) 
5: Reply
6: Destruct

Destruction can be done synchronously or asynchronously
Sequence Diagrams

- Conditionals
Sequence Diagrams

Object1: ClassA

1: Create1

2: Reply1

3: value = Array[i]

4: computeSquare(value)

5: Return square

This is a collection. The loop applies to this collection as long as the condition is true

Generated by UModel www.altova.com
Sequence Diagrams

- Shortcut to repetition

The message is sent to every element of the collection for as long as the condition is true.
State Diagrams

- Behavioral
  - In response to events, an object can be in any one of a number of states

- Events:
  - Represent how objects respond to external messages

- Events include:
  - Instantiation or destruction of an object
  - Sending a message to an object

- State:
  - Is the aggregation of the values of the attributes (or a subset of the attributes) of an object

---

1 Introduction to SE using UML, Colorado State University, Dr. R. France
State Diagrams

- A cell phone object with 2 states

[State Diagram Image]
State Diagrams
State Diagrams

- Guards provide conditional execution of transitions
State Diagrams

• Events can be classified:
  – Signal events
    • A message from one object to another
  – Change events
    • When the guard on a condition becomes true
      the event triggers the transition
  – Time events
    • After a given time period (the event) a transition
      is triggered
State Diagrams

• Object Lifecycle
  - A typical object goes through the following general model of behavior:
State Diagrams

- Reflecting object lifecycle in a state diagram.

Diagram showing state transitions for a light switch.
State Diagrams

- Objects and Threads
  - A passive object (left) waits for events
  - An active object (right) performs actions on its own

Introduction to SE using UML, Colorado State University, Dr. R. France
State Diagrams

• Objects and Threads
  – Two concurrent method invocations

```
Initialize Object
↓
Wait for a message
↓
Handle Request
↓
Terminate Object
```
State Diagrams

- A state symbol can have one or more compartments (all optional)
  - Name compartment
  - Internal transition compartment
    - Contains internal actions or activities
    - `event(args) [condition]/action`
    - `entry/action` (invoked implicitly)
    - `exit/action` (invoked implicitly)
    - `do/machine name` (invokes a nested state machine)
State Diagrams

- The sequence of operations as a result of an OFF event is:
  - printf ("exiting");
  - printf ("to off");
  - lamp.off();
State Diagrams

- Conditional branching
State Diagrams

• Composite states: all orthogonal regions detect the same events and respond simultaneously
UML

- Many resources and books are available
- Trust all OMG sources
- Use the latest version of UML
- Best tool I have found: Altova UModel
  - http://www.altova.com
Thanks!
Q/A