Lecture 3: Elaboration and System Architecture
Topics

• System design oriented software architecture concepts
  – component structures
• UML and the representation of architecture
  – components
• Java and architectural components
  – folders
Why Now?

- System Architecture is often considered design, not requirements/elaboration
- Personal past experience
  - Have a ready system metaphor we can use
- Industry past experience
  - Study other experiences to get some insight
- Examine technology to see what it will support
System Architecture

• Basic Pieces of the system
  – Components, Modules, Subsystems

• Structure of the System
  – Relationships between pieces

• Solution Strategies for System
  – Is this Architecture?
  – E.g. Analysis and Design Mechanisms
# Mechanisms

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Design</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Persistency</td>
<td>Relational DBMS</td>
<td>JDBC</td>
</tr>
<tr>
<td>Distribution</td>
<td>Remote Method Invocation</td>
<td>Java 2</td>
</tr>
</tbody>
</table>
Tiers

- Physical tiers
  - Separate machines, e.g. client/server

- Logical tiers
  - Logical components, e.g. GUI, Business Logic, Database

- Refinements
  - E.g.
    - Thin client: presentation logic only
    - Fat client: presentation logic plus business logic
Layers

- From OS – layers of progressively more abstract machines, each of which uses the services of lower levels
- Logical Tiers and Layers
  - GUI
  - Business/Application/Domain Logic
  - Data Base
Model-View Separation

• Layers model for GUI oriented application:
  View-GUI
  Model
• View layer displays information known from model layer
• Why?
  – Divide and conquer approach to development
  – Can make changes to GUI without touching the “guts” of the system
Model View Controller

• View = graphics presentation
• Model = Business Logic
• Controller = respond to input from View by calling on Model, updating view with information from model
• Java, VB, etc.- view classes have associated event responders. Controller merged into View
MV Separation and Callbacks

• View can call on model to get information to display
• How does lower level Model alert View about something that needs to be displayed?
• View calls a model method with an identifier for a method to be called when the condition of interest happens. Passed method is called a call back function
Observer/Observable - 1

• Based on a design pattern
• Java Classes
• Can be used to support callbacks
• Observers – register with observable
• Observable – when instructed, will call the observers’ update() method
Observer/Observable -2

- Observer is an interface
  - Implementer class must implement `update(Observable obs, Object arg)`
- Observable is a class
  - Includes
    - `addObserver(Observer obs)`
    - `notifyObservers(Object arg)`
Sample Callback for DatingSystem

• Requirement: we want the domain/business logic layer of the system to flash a warning on the screen if the logged on member is not paid up, and then continue normally with the logic for logging on

• Use of observer/observable
  – GUI implements Observer => must implement an update() method. Have this method flash the warning
  – DL extends Observable => it will have an addObserver(Observer obs) and notifyObserverMethod()
Use of the CallBack

• Assume at start up where the constructor for GUI is passed a reference to an instance of the DL
• GUI constructor calls DL.addObserver(this) to register itself as an observer of DL
• In the logOn logic in DL: if the special condition is recognized, notifyObservers() is called which causes the registered observers update() methods to be called which causes the warning to flash
UML and Components

- Components
  - Pieces of the system architecture
  - Layers, modules, logical tiers
  - May be nested subcomponents

- UML packages
  - Collections of classes and packages
  - May be used for modeling components
Subsystems

• Package plus one or more interfaces that define the services supported by the subsystem
• Layers can be modeled as subsystems
• Interfaces: set of method specifications, e.g. Java interfaces
Subsystems and Facades

• Façade
  – an interface object that is the single point of entry for the services of a subsystem
  - a common unified interface for a disparate set of implementations or interfaces
• Subsystem interface can be a façade which calls methods in an implemented subsystem
• Could also be a proxy which simulates a subsystem or which communicates with a remote implementation
• These patterns (façade and proxy) will be discussed later
Java Subsystems

• Possible units of organization:
  – Files
  – Folders
  – Classes
  – Packages
Files and Subsystems

• Files: units of compilation
• Compilation produces class files for each class in a file
• Too small a unit for organizing subsystems
• Class in one file has limited visibility (i.e. ability to create an instance of) to classes in other files.
Inner Classes and Subsystems

• Inner Class
  – Defined inside another class
  – Has access to variables of enclosing class

• Subsystems consist of a principal class (e.g. subsystem proxy controller), and its inner classes?

• Problems e.g.
  – Cannot have static members in inner classes
  – Cannot access inner classes from the outside
Folders and Subsystems

- Associate subsystems with folders
- When class file is compiled, compiled class files will go into the same folder
Packages

• Java Package is a collection of classes
• Package is associated with a folder
• Can use import statement in a file to identify source of classes used in file
• Use package statement to identify a file as belonging to a package
• Use import to identify sources of referenced classes
Packages and Subsystems

• Subsystem is a package
• Package has interface classes for the subsystem (e.g. a façade class)
• Visibility
  – Normal: can see the public class in another file
  – Package: can see non-private (public and “friendly” undesignated) entities in other files
Additional Packages

- **Globals**
  - E.g. MemberData in DS
- **Utility routines**
  - E.g. Classes used for tracing and debugging
- **Subsystem interface definitions?**
Dating System Architecture
Start Up Logic

start : Start

create(dB)
create(dL)

dB : DataBase

dL : DomainLogic
gUI : GUI

Copyright W. Howden
Distributed Systems Architecture

• Stand Alone versus Client Server
• Basic Model: Communicating State Machines
  – Components: client(s) and server
  – Component specification: state machine
State Machines

- Nodes correspond to states a system/component/object is in
- Transitions from one state to another caused by external events that are recognized/recorded by the component
- Transitions may also have
  - conditions that must be satisfied for transition to occur
  - actions that are carried out when a transition is followed
Some state machine notation

• Solid black dot shows the starting state(s)
• Circular transitions are allowed
• “[ cond ]” on a transition indicates a transition condition
• “/ action” on a transition indicates a transition action e.g. send or receive message from other component
• further discussion will be done in lecture on state charts
Client Server Model