Lecture 3: Elaboration and System Architecture
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 1

• Observer is an interface
  – Implementer class must implement update
    (Observable obs, Object arg)

• Observable is a class
  – Includes
    • addObserver(Observer obs)
    • notifyObservers(Observer arg)
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 Proxies

• Proxy – stands in for something, takes its place, simulates or interfaces with some entity

• Subsystem interface can be a proxy which may either simulate a subsystem, or call methods in an implemented subsystem

• Proxy is another example of a pattern
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
• Use different class path designators so that compiler and java run time can find referenced class files
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
• Visibility
  – Normal: can see the public class in another file
  – Package: can see non-private (public and “friendly” undesignated) entities in other files
Note on security and package visibility

• Suppose you have a system library class “Window” in “java.awt” that contains a friendly (not public or private) variable “String warningString”.

• Create a file, for example, called SetWarning with a SetWarning class, and include “package java.awt” in the file.

• Put the class in a subdirectory java/awt somewhere on the class path and you can use it to alter this variable.

• Security manager will stop you if you try this in an applet (i.e. try adding to system packages) that is loaded from the Internet.
Additional Packages

• Globals
  – E.g. MemberData in DS

• Utility routines
  – E.g. Classes used for tracing and debugging

• Subsystem interface definitions?
Dating System Architecture

Dating System

«subsystem»
GUI

«subsystem»
Business Logic

«subsystem»
DataBase

Gobals

Utilities

Start
Start Up Logic

start : Start

crate(dB)

crate(dL)

dB : DataBase

dL : DomainLogic

gUI : GUI
Distributed Systems Architecture

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