Architectural Patterns/Styles

Charlie Garrod  Michael Hilton
Administrivia

- Homework 6 checkpoint – Monday Dec 4th
- Final Exam Review: Dec 13th, 2-4pm Wean 5409
- Final Exam: Dec 15th, 5:30-8:30pm Wean 7500
Last Time:

- Design Patterns
ARCHITECTURAL PATTERNS/STYLES
Design Patterns
Architectural Styles
Architectural Styles
Architectural Styles vs Design Patterns
Monolithic Application

+ Simple to start
+ Simple to deploy
+ Fast time to first feature
- Difficult for new developers to come up to speed
- Continuous deployment is difficult
- Scaling can be difficult
- Can devolve into “big ball of mud”
Layers

Layer 1: Transmits bits: velocity, bit-code, connection, etc.
Layer 2: Detects and corrects errors in bit sequences
Layer 3: Selects a route from sender to receiver
Layer 4: Breaks messages into packets and guarantees delivery
Layer 5: Provides dialog control and synchronization facilities
Layer 6: Structures information and attaches semantics
Layer 7: Provides miscellaneous protocols for common activities

Presentation Layer: Component, Component, Component
Business Layer: Component, Component, Component
Persistence Layer: Component, Component, Component
Database Layer: Component, Component, Component, Component
Layers

• Context:
  – A large system that requires decomposition

• Problem:
  – Low separation of concerns.
  – Parts of system are not interchangeable
  – Lack of grouped components hurts understandability and maintainability
  – Lack of boundaries makes tasking difficult

• Solution:
  – Define layers of abstraction
  – Specify services between boundaries

• Beware:
  – Antipattern: Sinkhole
  – Antipattern: Lasagna
Pipe and filter
Pipe and filter

• Context:
  – Processing data stream

• Problem:
  – Need to process or transform a stream of data
  – Non-adjacent steps don’t share information
  – Need to reuse certain steps in the process

• Solution:
  – Each filter transforms the data, then moves it on to the next step

• Beware:
  – Error Handling
  – Data transformation overhead
Blackboard
Blackboard

• Context:
  – An immature domain where no closed approach is known to be feasible

• Problem:
  – A complete search of solution space is not feasible
  – Multiple algorithms possible for different subtasks
  – Some algorithms work on the output of others
  – Uncertain data and aprox solutions are involved

• Solution:
  – Independent programs working cooperatively on common data
  – Inspect and update data

• Beware:
  – Difficult to test
  – Difficult establishing a good control strategy
Model-View-Controller
Model-View-Controller

• Context:
  – Interactive applications with a flexible Human-Computer interface

• Problem:
  – How to develop an application not dependent on interface
  – Need ability for application to support different interfaces
  – Allow simultaneous development

• Solution:
  – Model – View – Controller division

• Beware:
  – Code navigability
  – Increased complexity
Broker
Broker

- **Context:**
  - Decoupled components interact through remote service invocations

- **Problem:**
  - Scaling for large scale systems
  - Components should be decoupled and distributed

- **Solution:**
  - Brokers mediate between clients and servers

- **Beware:**
  - Less efficient
  - Lower fault tolerance
Microkernel

Monolithic Kernel based Operating System

- Application
  - System Call
  - VFS
  - IPC, File System
  - Scheduler, Virtual Memory
  - Device Drivers, Dispatcher, ...

Hardware

Microkernel based Operating System

- Application
- UNIX Server
- Device Driver
- File Server
- Basic IPC, Virtual Memory, Scheduling

Hardware

user mode

kernel mode
Microkernel

- **Context:**
  - The development of several applications that use similar interfaces on same core

- **Problem:**
  - Should cope with continuous hardware and software evolution
  - Platform should be portable, extensible and adaptable

- **Solution:**
  - Encapsulate fundamental services of your application platform in a microkernel
  - Other functionality provided by internal servers

- **Beware:**
  - Complexity of design and implementation
Event-driven architecture
Event-driven architecture

• **Context:**
  – Building a loosely coupled, more responsive system

• **Problem:**
  – Build a system that reacts to events in the world around it
  – Only have to decide what to do, not when to do it

• **Solution:**
  – Event creators, managers, and consumers

• **Beware:**
  – Security risks
  – Increased complexity
Peer-to-peer
Peer-to-peer

• Context:
  – A system where each node has the same capabilities and responsibilities
• Problem:
  – A situation where it is not feasible to know ahead of time which nodes will be servers
  – Large amounts of data need to be sent transmitted
• Solution:
  – Decentralized computing
  – Highly robust in the face of node failure
  – Highly scalable
• Beware:
  – No server to manage data
  – No always used for legal purposes
Service-oriented architecture
Service-oriented architecture

• Context:
  – Services are provided to other components over a network

• Problem:
  – Building a distributed system
  – Expose a service no objects

• Solution:
  – Each service should:
    • Represent a business activity with a specific outcome
    • Be self-contained
    • A black-box for its consumers
    • May consist of underlying services

• Beware:
  – High investment cost
Exercise:

- **Styles:**
  - Monolith
  - Layers
  - Pipe and Filter
  - Blackboard
  - MVC
  - Broker
  - Peer-to-peer
  - Microkernel
  - Event-driven
  - Service-oriented

- **Application**
  - Online banking application
  - API for third party tools to get banking information
  - Compiler
  - Optical Character recognition
  - VR content delivery system
  - VR game
  - Insurance claim processing system