Lecture 09:
Architecture Styles (4)

Emad Shihab
Adapted from Ahmed E. Hassan and Spiros Mancoridis
Recap of Last Class

■ Automated Stock Trading System
■ Two architectures
  – Pipe-and-Filter + Repository + (EB) Implicit Invocation
  – Object Oriented + Repository + (PS) Implicit Invocation
Layered Style

Virtual Machine

Client-Server

Adapted from Taylor et al. 2010
Layered Style

- Architecture is separated into ordered layers
  - A program in one layer may obtain services from a layer below it
Layered Variants

■ **Virtual Machine**
  – An ordered sequence of layers
  – Each layer services the layer above it

■ **Client-Server**
  – Clients send service request to server
  – Server replies as needed with requested information
Layered Style

■ Components
  – VM: Layers (comprised of one or more programs)
  – CS: Client and Server

■ Connectors
  – VM: Procedure calls
  – CS: Remote procedure calls
Layered Style

- **Topology**

  - VM: Linear; cross layer in special cases
  - CS: Two-level; client-to-client communication prohibited
Layered Style Advantages

- **VM**
  - Clear dependence structure
  - Upper levels immune to changes at lower levels
  - Lower levels are independent of upper levels

- **CS**
  - Centralization of computation and data at server
  - Single powerful server can serve many clients
Layered Style Disadvantages

- **VM**
  - Having too many layers can be inefficient (may need to cross layers)
  - Not easy to divide software systems into layers

- **CS**
  - Heavy dependence on communication network
Layered Style Examples

■ VM
  – Operating systems
  – Network protocol stack

■ CS
  – Business applications
QA evaluation of Layered Style

- **Performance**
  - VM: In some cases need to cross layers
  - CS: May be restricted by network capacity

- **Availability**
  - VM: lower layers vs. higher layers?
  - CS: Failure at server affects all clients

- **Modifiability**
  - Change to a layer will affect, at most, 2 layers

- **Portability?**
Architectural Styles Wrap Up
## Repository

<table>
<thead>
<tr>
<th>Summary</th>
<th>Use it when…</th>
<th>Avoid it when…</th>
</tr>
</thead>
<tbody>
<tr>
<td>Independent programs, access and communicate exclusively through a global repository</td>
<td>order of processing dynamically determined and data-driven</td>
<td>interactions between the independent programs require complex regulation</td>
</tr>
</tbody>
</table>

Adapted from Taylor et al. 2010
# Pipe-and-Filter

<table>
<thead>
<tr>
<th>Summary</th>
<th>Use it when...</th>
<th>Avoid it when...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Separate programs, aka filters, executed, potentially concurrently. Pipes route data streams between filters</td>
<td>problem easily formulated as a set of sequential, severable steps</td>
<td>interaction between components required</td>
</tr>
</tbody>
</table>

Taylor et al. 2010
# Object Oriented

<table>
<thead>
<tr>
<th>Summary</th>
<th>Use it when…</th>
<th>Avoid it when…</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objects encapsulate state and accessing functions</td>
<td>many complex and interrelated data structures</td>
<td>strong independence between components necessary.</td>
</tr>
</tbody>
</table>

Adapted from Taylor et al. 2010
## Implicit Invocation

<table>
<thead>
<tr>
<th>Summary</th>
<th>Use it when…</th>
<th>Avoid it when…</th>
</tr>
</thead>
<tbody>
<tr>
<td>(PB) Publishers broadcast messages to subscribers</td>
<td>subscription data is small and efficiently transported.</td>
<td>middleware to support high-volume data is unavailable.</td>
</tr>
<tr>
<td>(EB) Independent components asynchronously emit and receive events communicated over event buses</td>
<td>components are concurrent and independent</td>
<td>guarantees on real-time processing of events is required</td>
</tr>
</tbody>
</table>

Adapted from Taylor et al. 2010
## Layered

<table>
<thead>
<tr>
<th>Summary</th>
<th>Use it when…</th>
<th>Avoid it when…</th>
</tr>
</thead>
<tbody>
<tr>
<td>(VM) Virtual machine, or a layer, offers services to layers above it.</td>
<td>many applications can be based upon a single, common layer of services</td>
<td>1. many levels are required (causes inefficiency)</td>
</tr>
<tr>
<td>(CS) Clients request service from a server.</td>
<td>centralization of computation and data at a single location (the server) promotes manageability and scalability</td>
<td>network bandwidth and reliability are limited</td>
</tr>
</tbody>
</table>