Architectural Patterns

CS 718 lecture series

Prof. Rushikesh Joshi
IIT Bombay
provide a single layer of abstraction on top of many related functions
Wrapper – an example

- the semop system call in unix-- it wraps around many functions related to semaphores
  
  int semop(int semid, struct sembuf *sops, unsigned nsops);
A large system is decomposed into layers of abstractions
Layers: examples

- operating systems – system calls, inner kernel, hardware abstractions, hardware
- networking layers
- APIs-platform independent Implementation-
  Platform dependent implementation
- 3-tiered architecture – UI, BL, Data
Pipelines

Processing steps in a pipeline are sometimes referred to as filters, and the connectors as pipes.

Stream of data passes through pipes and filters.
pipelines-- examples

- language processing – lexical analysis, syntax analysis, semantic analysis, code generation, optimization
- unix pipes and filters
- instruction pipelines
The difference between layers and pipes
client does not provide an interface, the only communication from the server to client is through return results or exception/error values.
client-server examples

- Lan services on a unix machine/windows machine
  - ldap, portmapper, nslookup, directories
- RMI, RPC, Web-servers and web clients
Blackboard

Collaborative problem solving through knowledge sharing

- Blackboard controller monitors and activates knowledge sources and task solvers
- Blackboard operates on/use/share

Diagram:
- Blackboard
- Blackboard controller
- Knowledge sources and task solvers
blackboard-examples

- Linda tuple space
- shared memory based parallel/distributed problem solvers
Master-Slave

Master

Task distribution

Slave

Slave

Slave
**master-slave example**

- used in parallel computing on clusters
  - master process splits a big task into subtasks, distributes and coordinates slave processes, collects and collates results
coordinating communication (requests, replies, exceptions) in a distributed remote service invocation scenario
broker--examples

- direct communication broker: connect and then let client communicate directly
- trader broker (select one of many servers)
- middleware brokers (e.g., orb in corba)
  - locating servers, supporting interoperability
Example: Activation Broker -1

server machine

messaging broker

client machine

client process

service repository
Example: Activation Broker -2

server machine

messaging broker

specific req

client machine

client process

service repository
Example: Activation Broker -3

Server machine

Activation server

Messaging broker

Client machine

Client process

Service repository
**Example: Activation Broker -4**

- **Server Machine**
  - Activation Server
  - Messaging Broker
  - USE

- **Client Machine**
  - Client Process

- **Service Repository**
Example: Activation Broker -5

- server machine
- activation server
- messaging broker
- register
- USE
- service repository
- client machine
- client process
**Example: Activation Broker -6**

- **Server Machine**
  - Activation Server
  - Register Broker

- **USE**
- **Service Repository**

- **Client Machine**
  - Client Process

- **Specific Req**

- **Messaging**

Example: Activation Broker -7

server machine

activation server

messaging broker

USE

forward specific req

specific req

client machine

client process

service repository

USE
Example: Activation Broker -8

server machine

service repository

LOAD

USE

specific req

server

forward specific req

register

activation server

messaging broker

client process

client machine
Example: Activation Broker -9

server machine

activation server

server start event

messaging broker

USE

specific req

server

LOAD

USE

client machine

specific req

client process

service repository
**Example: Activation Broker -10**

server machine

activation server

messaging broker

specific req

client machine

specific req

client process

service repository
Activation Policies

- Per client's request
- Per client
- Per service request
- Per server
Who does the registration into repository

- A separate application that creates server implementations
- Server implementations are registered in implementation repositories
- Server interfaces can be registered with interface repositories
Model - View – Controller -1

view

controller

Model

<table>
<thead>
<tr>
<th>name</th>
<th>value1</th>
<th>value2</th>
<th>value3</th>
</tr>
</thead>
<tbody>
<tr>
<td>pradeep</td>
<td>20</td>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td>vinay</td>
<td>10</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>namita</td>
<td>15</td>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>yogita</td>
<td>12</td>
<td>12</td>
<td>1</td>
</tr>
</tbody>
</table>
Model - View – Controller -2

View

Controller

Model

<table>
<thead>
<tr>
<th>Name</th>
<th>User</th>
<th>Control</th>
<th>Inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>pradeep</td>
<td>20</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>vinay</td>
<td>30</td>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>namita</td>
<td>30</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>yogita</td>
<td>30</td>
<td>20</td>
<td>1</td>
</tr>
</tbody>
</table>
Model - View – Controller -5

User/Control inputs

Controllers

Model

Views

Pull Updates

Change Intimation

Update Model

<table>
<thead>
<tr>
<th>Name</th>
<th>Change 1</th>
<th>Change 2</th>
<th>Change 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pradeep</td>
<td>20</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>Vinay</td>
<td>30</td>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>Namita</td>
<td>30</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>Yogita</td>
<td>30</td>
<td>12</td>
<td>1</td>
</tr>
</tbody>
</table>
Model - View – Controller -6

- Model contains the state information.
- User inputs event sent to controller.
- Model is displayed in view.
Properties of MVC

• controller not aware how the state is displayed
• model not aware of how the state is displayed
• controller does not directly inform the view about updates
• model sends update intimations to view
• the view can pull in the updates
mvc -- examples

• smalltalk MVC – different look and feel standards
• Java pet-store example
• observer based systems
Pull Type Architecture

- Event publisher
- Event subscriber

Function call

Event is pulled by subscriber from publisher

Action
Push type Architecture

event publisher  event subscriber

event is pushed by publisher to subscriber

action
Combining them.. Pull Push Architecture

- **Pull Event**: Event is pulled by the channel from the publisher.
- **Push Event**: The channel pushes the event to the subscriber.
Combining them.. Push Pull Architecture

- Publisher pushes to the channel
- Subscriber pulls from the channel
Combining them.. Push Push Push Architecture

publisher pushes to channel
channel pushes it to subscriber
Combining them.. Pull Pull Architecture

channel pulls from publisher
subscriber pulls from the channel
Design issues

- Buffer spaces
  - at source
  - at intermediate channel
- Connectivity
  - dynamic
  - disconnections
- Service orientation, payment model
- Failure Handling
- Performance
Design of interfaces

interface pushConsumer {
    notifyEvent(Event e);
    disconnectingPushPublisher();
}

interface pushPublisher {
    subscribe (pushConsumer c)
    unsubscribe (pushConsumer c)
}

interface pullPublisher {
    boolean subscribe (pullConsumer c);
    unsubscribe (pullConsumer c);
    Event pullevent ();
}

interface pullConsumer {
    disconnectingPullPublisher();
}