Distributed System Architectures

- Architectures for distributed systems
  - Part 1: Architectural styles
  - Part 2: Client-server architectures
  - Part 3: Decentralized, peer-to-peer, and other architectures

Part 1: Architectural Styles

- Important styles of architecture for distributed systems
  - Layered architectures
  - Object-based architectures
  - Data-centered architectures
  - Event-based architectures
  - Resource-based architectures
Layered Design

- Each layer uses previous layer to implement new functionality that is exported to the layer above
- Example: Multi-tier web apps

Object-based Architecture

- Each object corresponds to a component
- Components interact via remote procedure calls
  - Popular in client-server systems
Event-based architecture

- Communicate via a common repository
  - Use a publish-subscribe paradigm
  - Consumers subscribe to types of events
  - Events are delivered once published by any publisher

Shared data-space

- “Bulletin-board” architecture
  - Decoupled in space and time
  - Post items to shared space; consumers pick up at a later time
Resource-oriented Architecture

- Example of ROA: Representational State Transfer (REST)
  - Basis for RESTful web services
  - Resources identified through a single naming scheme
  - All services offer same interface (e.g., 4 HTTP operations)
  - Messages are fully described
  - No state of the caller is kept (stateless execution)
  - Example: use HTTP for API
    - http://bucketname.s3.aws.com/objName
    - Get / Put / Delete / Post HTTP operations
  - Return JSON objects
    ```json
    {
      "name": "test.com",
      "messages": ["msg 1", "msg 2", "msg 3"],
      "age": 100
    }
    ```
  - Discuss: Service-oriented (SOA) vs. Resource-oriented (ROA)

### OOA vs. ROA vs. SOA

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Object-oriented</th>
<th>Resource-oriented</th>
<th>Service-oriented</th>
</tr>
</thead>
<tbody>
<tr>
<td>Granularity</td>
<td>Object instances</td>
<td>Resource instances</td>
<td>Service instances</td>
</tr>
<tr>
<td>Main Focus</td>
<td>Marshalling parameter values</td>
<td>Request addressing (usually URLs)</td>
<td>Creation of request payloads</td>
</tr>
<tr>
<td>Addressing / Request routing</td>
<td>Routed to unique object instance</td>
<td>Unique address per resource</td>
<td>One endpoint address per service</td>
</tr>
<tr>
<td>Are replies cacheable?</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Application interface</td>
<td>Specific to this object / class – description is middleware specific (e.g. IDL)</td>
<td>Generic to the request mechanism (e.g. HTTP verbs)</td>
<td>Specific to this service – description is protocol specific (e.g. WSDL)</td>
</tr>
<tr>
<td>Payload / data format description</td>
<td>Yes – usually middleware specific (e.g. IDL)</td>
<td>No – nothing directly linked to address / URL</td>
<td>Yes – part of service description (e.g. XML Schema in WSDL)</td>
</tr>
</tbody>
</table>

End of Part 1

- Reminder: No laptop or phone use during class. Masks mandatory.

- Career Fair on Feb 24th

Part 2: Client-Server Architectures

- Most common style: client-server architecture
- Application layering
  - User-interface level
  - Processing level
  - Data level
• Search engine architecture with 3 layers

Multitiered Architectures

• The simplest organization is to have only two types of machines:
• A client machine containing only the programs implementing (part of) the user-interface level
• A server machine containing the rest,
  – the programs implementing the processing and data level
A Spectrum of Choices

• From browser-based to phone-based to desktop apps

Three-tier Web Applications

• Server itself uses a “client-server” architecture
• 3 tiers: HTTP, J2EE and database
  – Very common in most web-based applications
Edge-Server Systems

- Edge servers: from client-server to client-proxy-server
- Content distribution networks: proxies cache web content near the edge
- Evolved into edge computing model

Part 3: Decentralized Architectures

- Peer-to-peer systems
  - Removes distinction between a client and a server
  - Overlay network of nodes
- Chord: structured peer-to-peer system
  - Use a distributed hash table to locate objects
    - Data item with key $k \rightarrow$ smallest node with id $\geq k$
Content Addressable Network (CAN)

- CAN: d-dimensional coordinate system
  - Partitioned among all nodes in the system
  - Example: \([0,1] \times [0,1]\) space across 6 nodes
    - Every data item maps to a point
    - Join: pick a random point, split with node for that point
    - Leave: harder, since a merge may not give symmetric partitions

Unstructured P2P Systems

- Topology based on randomized algorithms
  - Each node pick a random set of nodes and becomes their neighbors
    - Gnutella
  - Choice of degree impacts network dynamics
SuperPeers

- Some nodes become “distinguished”
  - Take on more responsibilities (need to have or be willing to donate more resources)
  - Example: Skype super-peer in early Skype

Collaborative Distributed Systems

- BitTorrent: Collaborative P2P downloads
  - Download chunks of a file from multiple peers
    - Reassemble file after downloading
  - Use a global directory (web-site) and download a .torrent
  - .torrent contains info about the file
    - Tracker: server that maintains active nodes that have requested chunks
    - Force altruism:
      » If \( P \) sees \( Q \) downloads more than uploads, reduce rate of sending to \( Q \)
Autonomic Distributed Systems

• System is adaptive - self-managing systems
  – Monitors itself and takes action autonomously when needed
    • Autonomic computing, self-managing systems
• Self-*: self-managing, self-healing
• Example: automatic capacity provisioning
  – Vary capacity of a web server based on demand

![Diagram of Feedback Control Model]

Feedback Control Model

• Use feedback and control theory to design a self-managing controller
  – Can also use machine learning or reinforcement learning