Class Announcements

- Midterm Tuesday 2/10
- Study guide posted

Architecture Example

Time sharing

Two-tier client/server

Three-tier client/server
**System integration**

1. Architecture
2. Subsystem implementation
3. System integration
   - Bring together subsystems and make them achieve desired system functionality
   - Testing
   - Modifications often needed

**Emergence**

- Subsystems are specialized
- Have simple functionality

Higher-level system functionality arises from the interaction of subsystems

Called: **Emergence**

e.g. airplane flies, but subsystems can’t

**Why system decomposition?**

- Divide and conquer approach to containing complexity
- Reuse
- Consonant with industry structure (unless system is to be supplied by one company)
- Others?

**Networked computing infrastructure**

by

David G. Messerschmitt

**Layering**

<table>
<thead>
<tr>
<th>Elaboration or specialization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Services</td>
</tr>
<tr>
<td>Existing layers</td>
</tr>
</tbody>
</table>

**Example of Layering: networking**

```
Physical
  ↓
  Frames
    ↓
    Packets
      ↓
      Messages
        ↓
        Application
```

signals
Software Layering

- Application
- Middleware
- Operating System

Software Layering Diagram

Operating system functions
- Graphical user interface (client only)
- Hide details of equipment from the application
- Multitasking
- Resource management
  - Processing, memory, storage, etc
- etc

Middleware Functions
- Capabilities that can be shared by many applications, but that is not part of OS
  - Example: Database Management System (DBMS)
- Hide details of OS from application
  - Java Virtual Machine
- More purposes we’ll talk about later.

What’s a database?

Database
- File with specified structure
- Example: relational table

A Database

<table>
<thead>
<tr>
<th>Year</th>
<th>City</th>
<th>Accommodation</th>
<th>Rooms</th>
</tr>
</thead>
<tbody>
<tr>
<td>2002</td>
<td>Oakland</td>
<td>Resort</td>
<td>150</td>
</tr>
<tr>
<td>2002</td>
<td>Oakland</td>
<td>Bed &amp; Breakfast</td>
<td>120</td>
</tr>
<tr>
<td>2002</td>
<td>Oakland</td>
<td>Resort</td>
<td>150</td>
</tr>
<tr>
<td>2002</td>
<td>Berkeley</td>
<td>Camping</td>
<td>120000</td>
</tr>
<tr>
<td>2002</td>
<td>Berkeley</td>
<td>Bed &amp; Breakfast</td>
<td>540</td>
</tr>
<tr>
<td>2002</td>
<td>Berkeley</td>
<td>Resort</td>
<td>500000</td>
</tr>
<tr>
<td>2002</td>
<td>Albany</td>
<td>Camping</td>
<td>87000</td>
</tr>
<tr>
<td>2002</td>
<td>Albany</td>
<td>Bed &amp; Breakfast</td>
<td>3240</td>
</tr>
<tr>
<td>2002</td>
<td>Oakland</td>
<td>Bed &amp; Breakfast</td>
<td>95</td>
</tr>
<tr>
<td>2003</td>
<td>Oakland</td>
<td>Resort</td>
<td>350</td>
</tr>
<tr>
<td>2003</td>
<td>Oakland</td>
<td>Bed &amp; Breakfast</td>
<td>340</td>
</tr>
<tr>
<td>2003</td>
<td>Oakland</td>
<td>Resort</td>
<td>810</td>
</tr>
<tr>
<td>2003</td>
<td>Berkeley</td>
<td>Camping</td>
<td>110000</td>
</tr>
<tr>
<td>2003</td>
<td>Berkeley</td>
<td>Bed &amp; Breakfast</td>
<td>9900</td>
</tr>
<tr>
<td>2003</td>
<td>Berkeley</td>
<td>Resort</td>
<td>410000</td>
</tr>
<tr>
<td>2003</td>
<td>Albany</td>
<td>Camping</td>
<td>7650</td>
</tr>
<tr>
<td>2003</td>
<td>Albany</td>
<td>Bed &amp; Breakfast</td>
<td>87000</td>
</tr>
</tbody>
</table>

Storage Middleware example: DBMS
- Database Management System (DBMS)
  - Manage Multiple databases
  - Allow multiple applications to access common databases
  - Implement standard data “lookup” (query) functions.
Client - Server Computing

Client Server Example

Client: “I want to see www.google.com”

Server:

Client Server Example - Layers Revealed

3-Tier Client Server Architecture example

3-Tier Client Server Architecture example
3-Tier Client Server Architecture example

Application Server

Web Server

Shared data

Client

Database Management System (DBMS)

In some implementations Application Logic and Web Server can be put on Different machines.

What is Bob’s Balance?

DBMS Responsibilities

- Hide Changes in the Database hardware from the Application
- Standard operations on the data, including searches, such a search is called a *query*.
- Separate Database Management from Applications, so that many applications can access the same data.
- Security, Integrity, Backup, fault tolerance, etc..

Relational Database

<table>
<thead>
<tr>
<th>Customer</th>
<th>Balance</th>
<th>Customer Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>$527</td>
<td>Silver</td>
</tr>
<tr>
<td>Bob</td>
<td>$0.50</td>
<td>Bronze</td>
</tr>
<tr>
<td>Charles</td>
<td>$100000</td>
<td>Gold</td>
</tr>
</tbody>
</table>

3-Tier Client Server Architecture in General

- Accept instructions from user
- Make requests of server
- Display responses of server
- Takes inputs from client
- Decides what to be done next
- Decides what shared data to access and manipulates it
- Processes shared data
- Support multiple applications with common data
- Protect data administration and application administration

3-Tier Client Server Architecture in General

Financial institution

Book distribution centers

Fulfillment logic

Customer logic

Acquirer bank

Clients

Orders

Merchandise

Book distributors

Customers

Enterprise

Inter-enterprise

Slide adapted from slides for Understanding Networked Applications
By David G Messerschmitt. Copyright 2000. See copyright notice.
Architecture

- How do you begin to architect a solution for a problem like this?
- Break it into modules!

Goal:
New Seat Back system
- New in-flight seatback system
- Sell upgrades and seat swaps
  - (People who want to get away from sick people...)
  - More legroom
  - Offer to exchange seats

HHC Architecture

- When a module is composed of sub-modules, the architecture is hierarchical.

Granularity tradeoff.

- How big should we make the modules
  - Many simple small ones
  - Or a few complicated big ones...

- This aspect of modularity is called granularity.

- Which is better?

• We also make use of layers
In-plane Server

- Server Application
- LinuxOS
- Networking Infrastructure

Between each module we specify an interface.

- Again, we see layering and hierarchy.
- Between each module we specify an interface.

Communication with airline database

Data server

- Standard Database
  - "queries" (SQL) from HHC Server
  - Our architecture makes use of the Existing interface of the airline database, so we don’t need to redesign it!

A simple interface: from within Architecture

- Compute Mean and Variance
- Computation of key statistics
- Communication with airline database

Interfaces

- N numbers of Float type
- 2 Numbers of float type that signify: Mean, Variance

Interface specifications are often made precise by using data types.

- Example type: float
  - A number with a decimal place
  - Has a certain allowable range, and precision.

More on Data types

- Data passing an interface is often specified in terms of a limited number of standard data types.

- Data type = range of values and allowable manipulation.

- Data type does not presume a specific representation, to allow heterogeneous platforms.

  - Representation must be known when data passes a specific module interface.

Example data types

- Integer
  - “natural number between -32,767 and +32,768”
  - Could be represented (in many ways) by 16 bits
    - since $2^{16} = 65,536$

- Float
  - “number of the form $m \times 10^n / 32768$, where $m$ is in the range -32,767 to +32,768 and $n$ is in the range -255 to +256”
  - Could be represented by $16 + 8 = 24$ bits
More data types

**Character**
- "values assuming a-z and A-Z plus space and punctuation marks"
  - could be represented by 7 or 8 bits

**Character string**
- "collection of n characters, where n is customizable"
  - could be represented by 7*n bits

Compound data types

Programmer-defined composition of basic data types

Example:

```java
Employee {
    String name;
    String address;
    Integer year_of_birth;
    etc;
}
```

Interfaces

- Computation of key statistics
  - N numbers of float type
  - 2 Numbers of float type that signify: Mean, Variance

- One module should not be concerned with other module’s implementation
  - "Separation of concerns."
- One module should see the other only through its interface - implementation details hidden.
  - Abstraction

Implementation

- Module A
  - Computation of key statistics
  - Implementation 1:
    - $\text{MEAN} = \frac{\sum_{i=1}^{N} x_i}{N}$
    - $\text{VARIANCE} = \frac{\sum_{i=1}^{N} (x_i - \text{MEAN})^2}{N}$

- Module B
  - Computation of key statistics
  - Implementation 2:
    - $\text{MEAN} = \frac{\sum_{i=1}^{N} x_i}{N}$
    - $\text{VARIANCE} = \frac{\sum_{i=1}^{N} (x_i - \text{MEAN})^2}{N}$

- Though different, this implementation is ok too.
  - We can choose the implementation details however we want, as long as we comply with the agreed interface.

- Should he use it?
  - NO!!!! Why??
- Either A should compute “SUM” himself, or sit down with B and redesign the
Encapsulation

- The designer of B might take measures to hide “SUM” from A so that A is not able to violate the agreed interface.
  - Example: B does not declare “SUM” as a global variable.
- Making a module’s implementation details inaccessible to other modules is called **encapsulation**.

Interfaces

- This simple interface example allows for only one action of module B.
  - Action is “Compute mean and variance.”
- Other examples are possible.