Class Announcements

- Midterm Thursday 10/22
- How did it go?
  - Solutions discussed in class today
- Assignment 3
  - Out today, Tues, 10/27
  - Due back Tues, 11/3
- Pop quiz???

Architecture Example

Time sharing

- Point-to-point wire
  - (no network)
- ASCII terminal
  - (no graphics)
- Mainframe
  - (database and application server)

Two-tier client/server

- Local-area network
- Server/Mainframe

Three-tier client/server

- Client
- Application server
- Enterprise data 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**

- Elaboration or specialization  
- Services  
- Existing layers

**Example of Layering: networking**

- Application  
  - Messages  
- Transport  
  - Packets  
- Network  
  - Frames  
- Link  
  - Bits  
- Physical  
  - Signals
### Software Layering

![Software Layering Diagram](image)

### 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>Tourists</th>
</tr>
</thead>
<tbody>
<tr>
<td>2002</td>
<td>Oakley</td>
<td>Bed &amp; Breakfast</td>
<td>14</td>
</tr>
<tr>
<td>2002</td>
<td>Oakley</td>
<td>Resort</td>
<td>190</td>
</tr>
<tr>
<td>2002</td>
<td>Oakland</td>
<td>Bed &amp; Breakfast</td>
<td>54</td>
</tr>
<tr>
<td>2002</td>
<td>Oakland</td>
<td>Resort</td>
<td>230</td>
</tr>
<tr>
<td>2002</td>
<td>Berkeley</td>
<td>Bed &amp; Breakfast</td>
<td>12000</td>
</tr>
<tr>
<td>2002</td>
<td>Berkeley</td>
<td>Resort</td>
<td>300000</td>
</tr>
<tr>
<td>2002</td>
<td>Albany</td>
<td>Camping</td>
<td>92</td>
</tr>
<tr>
<td>2002</td>
<td>Albany</td>
<td>Bed &amp; Breakfast</td>
<td>5240</td>
</tr>
<tr>
<td>2002</td>
<td>Albany</td>
<td>Resort</td>
<td>186</td>
</tr>
<tr>
<td>2002</td>
<td>Humbolt</td>
<td>Bed &amp; Breakfast</td>
<td>296</td>
</tr>
<tr>
<td>2002</td>
<td>Humbolt</td>
<td>Resort</td>
<td>1</td>
</tr>
<tr>
<td>2002</td>
<td>Humbolt</td>
<td>Bed &amp; Breakfast</td>
<td>11000</td>
</tr>
<tr>
<td>2002</td>
<td>Humbolt</td>
<td>Resort</td>
<td>112000</td>
</tr>
<tr>
<td>2002</td>
<td>Humbolt</td>
<td>Bed &amp; Breakfast</td>
<td>2770</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 Server Example - Layers Revealed

3-Tier Client Server Architecture example

3-Tier Client Server Architecture example
**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.

**3-Tier Client Server Architecture in General**

- 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
- Decouple data administration and application administration
- Protect critical data
- Display responses of server
- Make requests of server
- Accept instructions from user
- Support multiple applications with common data
- Decouple data administration and application administration
- Protect critical data
- Display responses of server
- Make requests of server
- Accept instructions from user

**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>$1000000</td>
<td>Gold</td>
</tr>
</tbody>
</table>
Architecture

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

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

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?
In-plane Server

- Server Application
- LinuxOS
- Networking Infrastructure

Computation of key statistics

Communication with airline database

Again, we see layering and hierarchy.

Between each module we specify an interface.

Data server

- Standard Database "queries" (SQL) from HHC Server
- Database

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

List of numbers

Compute Mean and Variance

Mean, Variance

Computations of key statistics

Interfaces

N numbers of float type

Compute Mean and Variance

Mean, Variance

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 $2^{15} = 65,536$

- Float
  - "number of the form $m\times10^n/32768$, 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

- HIDDEN From Module A!!

Implementation

- Computation of key statistics
  - Mean, Variance
- Implementation 1:
  - 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 2:
  - 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

- **PARAMETERS**
  - Module A
  - Module B

  *N numbers of Float type* 
  *Computation of key statistics*

  Interface:
  - **Compute Mean and Variance**
  - **2 Numbers of Float type that signify Mean, Variance**

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

Possible software interface

- **Menu of actions**
  - action-1
  - action-2
  - action-3
  - ...

- **Example:**
  - Action 1: Compute mean
  - Action 2: Compute variance
  - Action 3: Compute mode
  - Etc.

Protocol

- In addition to atomic actions, an interface may define **protocols**
  - **Protocol** == finite sequence of actions required to achieve a higher level function
  - One action can be shared by multiple protocols
  - Multiple modules may participate in a protocol

Protocol Example

**Automatic teller machine (ATM)**

What is the interface between this machine and the customer?
Steps

Define available actions
Define, for each higher level function, a protocol
  - Single action or a finite sequence of actions

Interface building blocks

Message on screen or printed
  - Menu of actions or returns from an action
  - Touch selection of action

Keypad
  - Input parameters to an action

Card reader
  - Authentication, input parameters

Money output slot
  - Returns money

Action: authentication

Parameters

Internal functionality

Returns

Internally, it contacts institution and matches against its database, institution noted for all subsequent actions (example of state)

Returns

Screen message (“Invalid PIN” or menu of available actions)

Action: specify_account

Parameters

Internal functionality

Returns

Internally, choice noted for all subsequent actions (another example of state)

Returns

None
**Action: amount**

Parameters
- Dollars_and_cents (typed on keypad)

Internally, amount noted (another example of state)

Returns
- Success or failure (state dependent, for example for a withdraw failure when dollars_and_cents exceeds balance)

**Protocol: cash_withdrawal**

What is the sequence of actions?

**Protocol: cash_withdrawal**

1. **authentication** → failure
2. **choose objective** → other objectives
3. **account** → no accounts
4. **amount** → balance exceeded!

**Goals**

Understand better
- how layering is used in the infrastructure
- how it contains complexity
- how it coordinates suppliers
- how it allows new capabilities to be added incrementally

**Interaction of layers**

Layer above is a client of the layer below

Each layer provides services to the layer above...

...by utilizing the services of the layer below and adding capability

Layer below acts as a server to the layer above

More on layering

by David G. Messerschmitt
Layering builds capability incrementally by adding to what exists.

Three types of software

- Application
- Components and frameworks:
  - What is in common among applications
- Infrastructure:
  - Basic services (communication, storage, concurrency, presentation, etc.)

Part of Microsoft vs. DOJ dispute

- Microsoft position
- DOJ position

Three types of software

- Application frameworks and components
- Middleware
- Operating system
- Network

Major layers

- Application
  - Deals with information
  - Assumes structure and interpretation
- Infrastructure
  - Deals with data
  - Ignores structure and interpretation
Data and information in layers

- The infrastructure should deal with data, or at most minimal structure and interpretation.
- The application adds additional structure and interpretation.
- This yields a separation of concerns.

<table>
<thead>
<tr>
<th>Package = file, message</th>
</tr>
</thead>
<tbody>
<tr>
<td>In the simplest case, the infrastructure deals with a package of data (non-standard terminology):</td>
</tr>
<tr>
<td>- collection of bits</td>
</tr>
<tr>
<td>- specified number and ordering</td>
</tr>
<tr>
<td>The objective of the infrastructure is to store and communicate packages while maintaining data integrity.</td>
</tr>
<tr>
<td>File for storage, message for communication</td>
</tr>
</tbody>
</table>

Data integrity

Retain the:
- values
- order
- number of bits in a package

Example 1

Bob sends a letter to Alice

Example 2

Web server -> Web page -> Screen
Application
Operating system
Network
Fragmentation
Collection of packets
Assembly

Example 3

HHC Server Application -> Passenger Information
HHC Client Application
Windows OS
Palm OS
Networking Infrastructure
Collection of Packets
Networking Infrastructure
Collection of Packets
**Information in the infrastructure**

Sometimes it is appropriate for the infrastructure to assume structure and interpretation for data:

- to add capabilities widely useful to applications
- to help applications deal with heterogeneous platforms, where representations differ

At most, data types