Class Announcements

- Database Assignment 1 due 5/1 - 5/8 (by start of class!!!)

- Midterm 5/6
  - Study guide posted
Architecture Example
3-Tier Client Server Architecture in General

Client

- Accept instructions from user
- Make requests of server
- Display responses of server

Application Server

- Takes inputs from client
- Decides what to be done next
- Decides what shared data to access and manipulates it
- Processes shared data

Shared data

- Support multiple applications with common data
- Protect critical data
- Decouple data administration and application administration
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!
New in-flight seatback system

- Sell upgrades and seat swaps
  - (People who want to get away from sick people ...)
- Offer to exchange seats
Architecture

- HEADQUARTERS
  - Airline Dataserver

- Wireless Link
- Seat back devices
  - Wireless Link
  - servers
When a module is composed of sub-modules, the architecture is **hierarchical**.
HHC Architecture

- Seatback Application
  - Linux OS
  - Networking Infrastructure
- Coordination With On Plane Server
- User Interface
  - Data Management

- We also make use of layers
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

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

**Server Application**

**LinuxOS**

**Networking Infrastructure**

**Computation of key statistics**

**Communication With seat backs**

**Communication with airline database**

Standard Database “queries” (SQL) relayed to DBMS via OS and infrastructure
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**
  - Mean, Variance
  - List of numbers

**Computation of key statistics**

- **HHC Application**
- **Linux**
- **Networking Infrastructure**

**Communication with HHC**

**Communication with airline database**
Interfaces

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 $2^n = 65,536$

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

PARAMETERS

N numbers of Float type

INTERFACE

Compute Mean and Variance

RETURNS

2 Numbers of float type that signify: Mean, Variance
**Implementation**

- 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

**Module A**

- Computation of key statistics

**Module B**

- Compute Mean and Variance

Implementation 1:

- $X_i, i=1..N$
- **MEAN, VARIANCE**

**HIDDEN From Module A!!**

- $\text{MEAN} = \frac{1}{N} \sum_{i=1}^{N} x_i$
- $\text{VARIANCE} = \sum_{i=1}^{N} (x_i - \text{MEAN})^2$
Implementation

Module A
Computation of key statistics

Module B
Compute Mean and Variance

Implementation 2:

\[
\begin{align*}
  \text{SUM} &= \sum_{i=1}^{N} x_i \\
  \text{MEAN} &= \frac{\text{SUM}}{N} \\
  \text{VARIANCE} &= \sum_{i=1}^{N} (x_i - \text{MEAN})^2
\end{align*}
\]

- 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.
**Implementation**

Module A

Computation of key statistics

Module B

Compute Mean and Variance

\[
\begin{align*}
\text{SUM} &= \sum_{i=1}^{N} x_i \\
\text{MEAN} &= \frac{\text{SUM}}{N} \\
\text{VARIANCE} &= \frac{1}{N} \sum_{i=1}^{N} (x_i - \text{MEAN})^2
\end{align*}
\]

“\text{I need to get the sum, I’ll just take it from B}”

- 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 modules implementation details inaccessible to other modules is called encapsulation.
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

Example:

*Action 1: Compute mean*
*Action 2: Compute variance*
*Action 3: Compute mode*

Etc..