TIM 50 - Business Information Systems

Lecture 10

Instructor: John Musacchio
UC Santa Cruz

May 1, 2014

Class Announcements
- Database Assignment 1 due 5/4 5/8 (by start of class!!!)
- Midterm 5/6
  - Study guide posted

Architecture Example

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
- Protect critical data
- Decouple data administration and application administration

Book distribution centers

Financial institution

Acquirer bank

Fulfillment logic

Customer logic

Databases

Customers

Enterprise

Inter-enterprise

Customer logic

Book distribution centers

Financial institution

Acquirer bank

Fulfillment logic

Customer logic

Book distribution centers

Financial institution

Acquirer bank

Fulfillment logic

Customer logic

Book distribution centers

Financial institution

Acquirer bank

Fulfillment logic

Customer logic

Book distribution centers

Financial institution

Acquirer bank

Fulfillment logic

Customer logic

Book distribution centers

Financial institution

Acquirer bank

Fulfillment logic

Customer logic

Book distribution centers

Financial institution

Acquirer bank

Fulfillment logic

Customer logic

Book distribution centers

Financial institution

Acquirer bank

Fulfillment logic

Customer logic

Book distribution centers

Financial institution

Acquirer bank

Fulfillment logic

Customer logic

Book distribution centers

Financial institution

Acquirer bank

Fulfillment logic

Customer logic

Book distribution centers

Financial institution

Acquirer bank

Fulfillment logic

Customer logic

Book distribution centers

Financial institution

Acquirer bank

Fulfillment logic

Customer logic

Book distribution centers

Financial institution

Acquirer bank

Fulfillment logic

Customer logic

Book distribution centers

Financial institution

Acquirer bank

Fulfillment logic
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

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

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

Data server

- Standard Database "queries" (SQL) from HHC Server
- Standard Database "queries" (SQL) relayed to DBMS via OS and infrastructure

A simple interface: from within Architecture

- Interfaces
- 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.

Example data types

- Integer
  - "natural number between -32,767 and +32,768"
  - Could be represented (in many ways) by 16 bits
  - \(2^{16} = 65,536\)
- 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 \(16+\times8 = 24\) 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**

- **PARAMETERS**
  - N numbers of float type

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

- **INTERFACE**
  - HIDDEN From Module A!!

**Implementation**

- **Module A**
  - Compute Mean and Variance
    - Implementation 1:
      - Mean = $\frac{\sum x_i}{N}$
      - Variance = $\frac{\sum (x_i - \text{Mean})^2}{N}$

- **Module B**
  - Compute Mean and Variance
    - Implementation 2:
      - Mean = $\frac{\sum x_i}{N}$
      - Variance = $\frac{\sum (x_i - \text{Mean})^2}{N}$

**Implementation**

- **Module A**
  - Compute Mean and Variance
    - Implementation 2:
      - Mean = $\frac{\sum x_i}{N}$
      - Variance = $\frac{\sum (x_i - \text{Mean})^2}{N}$

- **Module B**
  - Compute Mean and Variance
    - Implementation 1:
      - Mean = $\frac{\sum x_i}{N}$
      - Variance = $\frac{\sum (x_i - \text{Mean})^2}{N}$

---

**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.

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.