TIM 50 - Business Information Systems
Lecture 12
Instructor: John Musacchio
UC Santa Cruz
February 12, 2015

Goal:
- 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
- More legroom
- Offer to exchange seats

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

HHC Architecture
- 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 the In-plane Server setup, we see layering and hierarchy. Between each module, we specify an interface. Again, we use layering and hierarchy.

- Server Application
- Linux OS
- Networking Infrastructure
- Computation of key statistics
- Communication with airline database
- Communication with seat backs
- Standard Database "queries" (SQL) relayed to DBMS via OS and infrastructure

In the Data server setup:

- DBMS
- Database
- 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

- List of numbers
- Compute Mean and Variance
- Computation of key statistics
- Mean, Variance
- HHC Application
- Linux
- Networking Infrastructure
- DBMS
- Database
- Computation of key statistics
- Communication with HHC
- Communication with airline database

Interfaces

- N numbers of Float type
- Compute Mean and 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 $2^{16} = 65,536$
- 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

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.”

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**

- Making a module's implementation details inaccessible to other modules is called **encapsulation**.

**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

**Steps**

Define available actions

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

**Another Interface Example: Automatic teller machine (ATM)**

What is the interface between this machine and the customer?
### Interface building blocks

- Message on screen
- Keypad
- Card reader
- Money output slot
- Printer

### Action: authentication

**Parameters**

- Identity (card in slot)
- Institution (card in slot)
- PIN (typed on keypad)

**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**

- Account (touch screen from menu of choices)

**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?

![Diagram of cash_withdrawal protocol]

More on layering

by David G. Messerschmitt

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 as a server to the layer above

Layering

Elaboration or specialization

Existing layers

Layering builds capability incrementally by adding to what exists
Data and information

- **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

"Package" = file, message

- Infrastructure deals with a "package" of data (non-standard terminology)
  - collection of bits
  - specified number and ordering
- Infrastructure stores and/or communicates "packages" while maintaining data integrity
- "Package" = file, message

Data integrity

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

Example 1

- Bob sends a letter to Alice
- US Postal Service
- Envelope
- Shipping Container
- ABC Airlines

Example 2

- Web server
- Web page
- Screen
  - Web browser
  - HTML

- File system
- File
- Message

- Network
  - Fragmentation
  - Collection of packets
  - Assembly
Example 3: Network Infrastructure Expanded

<table>
<thead>
<tr>
<th>Seatback Application</th>
<th>Passenger Information</th>
<th>Airplane Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux OS</td>
<td>TCP transport layer</td>
<td>Linux OS</td>
</tr>
<tr>
<td>Packets</td>
<td>Packets</td>
<td>WiFi Link Layer</td>
</tr>
<tr>
<td>WiFi Physical Layer</td>
<td>WiFi Physical Layer</td>
<td>Networking Infrastructure</td>
</tr>
</tbody>
</table>

Computer & Comm. Industry Structure

Two ways to design a system

- System requirements
- Decomposition from system requirements
- Available components
- Requirements
- Assembly from available components

Components

- Component: A subsystem purchased "as is" from an outside vendor
- (Alternative – building your own subsystem)
- A component implementation is encapsulated (although often configurable)

Seatback Architecture

HHC Application
- Linux OS
- Networking Infrastructure

Other Examples of components

- Computer
- Disk drive
- Network
- Network router
- Operating system
- Integrated circuit
- Database management system

The Palm OS we are buying "off the shelf" and integrating into our architecture. The Palm OS is a component.
Interoperability

- Components are interoperable when they interact properly to achieve some desired functionality.

- Increasingly component interoperability cannot be dependent on end-user integration:
  - PC and peripherals
  - Enterprise, inter-enterprise, consumer applications
  - Role for standardization

Outsourcing

- Outsourcing: A subsystem design is contracted to an outside vendor.

Responsibility is delegated.