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
Architecture

Wireless Link

HEADQUARTERS

Airline Dataserver

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**
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

- **HHC Application**
- **Linux**
- **Networking Infrastructure**
- **Compute Mean and Variance**
- **List of numbers**
- **Mean, Variance**
- **Communication with HHC**
- **Communication with airline database**

**Computation of key statistics**
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

PARAMETERS

N numbers of Float type

INTERFACE

Computation of key statistics

RETURNS

Compute Mean and Variance

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
Implementation

Computation of key statistics

Module A

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

Implementation 1:

\[ \text{SUM} = \sum_{i=1}^{N} x_i \]

\[ \text{MEAN} = \frac{\text{SUM}}{N} \]

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

- 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**.
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..
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
Another Interface 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

Keypad

Card reader

Money output slot

Printer
Action: authentication

Parameters
Internal functionality
Returns
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
- Internal **functionality**

Returns
**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?
Protocol: cash_withdrawal

1. Authentication
   - Failure

2. Choose objective
   - Other objectives

3. Account
   - No accounts

4. Amount
   - Balance exceeded!
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 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
Data integrity

Retain the

- values
- order
- number

of bits in a package
Example 1

Bob sends a letter to Alice

Bob
Envelope
US Postal Service
Shipping Container

Alice
Envelope
UK Royal Mail
Shipping Container

ABC Airlines
Example 2

Web server -> Web page -> Web browser

Application
Web server

Operating system
File
Message

File system
Screen
HTML

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

- Seatback Application
  - Linux OS
  - TCP transport layer
    - WiFi Link Layer
      - WiFi Physical Layer
      - Networking Infrastructure
    - Packets
      - Radio Signals
      - Networking Infrastructure
  - message

- Passenger Information

- Airplane Server
  - Linux OS
  - TCP transport layer
    - WiFi Link Layer
      - WiFi Physical Layer
      - Networking Infrastructure
    - Packets
      - Networking Infrastructure
  - message
Computer & Comm. Industry Structure
Two ways to design a system

Decomposition from system requirements

Assembly from available components

Slide adapted from slides for *Understanding Networked Applications* by David G Messerschmitt. Copyright 2000. See copyright notice.
Components

Component: A subsystem purchased “as is” from an outside vendor

(Alternative – building your own subsystem)

A component implementation is encapsulated (although often configurable)

Slide adapted from slides for Understanding Networked Applications
By David G Messerschmitt. Copyright 2000. See copyright notice
The Palm OS we are buying “off the shelf” and integrating into our architecture. The Palm OS is a component.
Other Examples of components

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

Why is a component implementation encapsulated?
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: A subsystem design is contracted to an outside vendor.

Responsibility is delegated.