Announcements

- Database Assignment 2 to be posted soon
- Assignment 4 on reading to be posted soon
- 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

Seat back devices

Wireless Link

servers

HEADQUARTERS

Airline Dataserver
HHC Architecture

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

- **Server Application**
- **LinuxOS**
- **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
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

- Computation of key statistics
  - List of numbers
  - Compute Mean and Variance
    - Mean, Variance

- HHC Application
- Linux
- Networking Infrastructure
- Communication with airline database
- Communication with HHC
- 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

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
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

Compute Mean and Variance

Module A

Computation of key statistics

Module B

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

Computation of key statistics

Module A

Module B

Compute Mean and Variance

Implementation 1:

- Should he use it?
  - NO!!!! Why??

- Either A should compute “SUM” himself, or sit down with B and redesign the interface.

\[
\begin{align*}
\text{SUM} &= \sum_{i=1}^{N} x_i \\
\text{MEAN} &= \frac{\text{SUM}}{N} \\
\text{VARIANCE} &= \sum_{i=1}^{N} (x_i - \text{MEAN})^2
\end{align*}
\]
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..
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
Layer above is a client of the layer below

Layer below as a server to the layer above

Each layer provides services to the layer above…

….by utilizing the services of the layer below and adding capability
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
Ignores structure and interpretation

Infrastructure
Deals with data
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

Screen

HTML

Application

File

Message

Operating system

File system

Network

Fragmentation

Collection of packets

Assembly
Example 3: Network Infrastructure Expanded

Seatback Application

message

Linux OS

TCP transport layer

WiFi Link Layer

WiFi Physical Layer

Networking Infrastructure

Passenger Information

Airplane Server

message

Linux OS

TCP transport layer

WiFi Link Layer

WiFi Physical Layer

Networking Infrastructure

Radio Signals

Packets

Packets
Computer & Comm. Industry Structure
Two ways to design a system

- Decomposition from system requirements
- Assembly from available components

Available components

Slide adapted from slides for *Understanding Networked Applications*
By David G Messerschmitt. Copyright 2000. See copyright notice
A component implementation is encapsulated (although often configurable)

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

(Alternative – building your own subsystem)

Slide adapted from slides for Understanding Networked Applications
By David G Messerschmitt. Copyright 2000. See copyright notice
The Linux OS we are buying “off the shelf” and integrating into our architecture. The Linux 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.
Suppose we choose to pay another firm to develop the user interface.

This is called **Outsourcing**.

Why would we do this?
System Integration

Suppose we
- Bring together all these subsystems
- and test them...

This is called **System Integration**
System integration

- Bring together subsystems;
- make them work together;
- to achieve a goal.

Requires

- Testing
- Making modifications to
  - architecture and/or
  - subsystem implementation

Slide adapted from slides for *Understanding Networked Applications* by David G Messerschmitt. Copyright 2000. See copyright notice.
Can System Integration be Outsourced?

- Of course!
Supplier Types

- Three types of suppliers:
  - Component Suppliers
  - Custom Subsystem Developers
  - System Integrators

- (Some suppliers are 2 or even 3 of above.)
Two ways to sell Software

Product
Customer installed and operated
Often (but not necessarily) sold or licensed at a fixed price

Service
Functionality provided over a wide-area network
Often (but not necessarily) sold by subscription

Slide adapted from slides for Understanding Networked Applications
By David G Messerschmitt. Copyright 2000. See copyright notice
Recall: Infrastructure and Applications

Infrastructure
- Equipment and/or software used by many applications

Applications
- Provide specific capabilities and features serving individual users.
Four possibilities

<table>
<thead>
<tr>
<th>Application</th>
<th>Product</th>
<th>Service</th>
<th>Infrastructure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft Office</td>
<td></td>
<td>Hotmail</td>
<td></td>
</tr>
<tr>
<td>Personal computer</td>
<td></td>
<td>Internet DNS</td>
<td></td>
</tr>
</tbody>
</table>

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

- Two types
  - Bundled
    - An infrastructure provider bundles applications with their infrastructure
      - Example: Comcast, telephony service providers
  - Unbundled
    - A provider of an application service without providing an infrastructure service
      - Examples?
Examples of unbundled ASP model

- Yahoo: Web-based calendar
- gmail: Web-based email
- Schwab: Web-based stock trading
Unbundled ASP model

Advantageous to user

- Proven way to reduce installation, integration, and maintenance costs
- Contractual obligation for availability and quality
- Location independence

Slide adapted from slides for Understanding Networked Applications
By David G Messerschmitt. Copyright 2000. See copyright notice
Advantages to supplier

- Ongoing revenue stream supporting upgrade and maintenance
- Usage-based revenue better aligned with user’s value proposition
- Opportunity for price discrimination, advertising revenue, etc.
Some pricing alternatives

Price discrimination?
Usage dependent?
Terms and conditions
- fixed, leasing, per-use, subscription
- warrantee, service level agreements

Bundles
- maintenance, support, releases, provisioning and operations

Who pays?
- sometimes not the end user
Infrastructure acquisition

Infrastructure

\[\begin{align*}
&\text{Build and operate} \\
&\text{Build but do not operate} \\
&\text{Do not build but operate} \\
&\text{Neither}
\end{align*}\]

Trend

Outsourced operations

System integrator

Service provider

---

Slide adapted from slides for *Understanding Networked Applications*
By David G Messerschmitt. Copyright 2000. See copyright notice
Application acquisition

Application

Develop internally

Buy as product

Contract development

Product w/ customization

Trend

Software supplier

Outsource developer

Supplier, consultants
Stovepipe vs. Integrated Infrastructure

**stovepipe architecture**

---or---

**Turnkey Solution**

- Single supplier provides all encompassing solution
- (complete with infrastructure)

**Integrated Infrastructure**

- Separate infrastructure that can support many applications

---

Slide adapted from slides for *Understanding Networked Applications* by David G Messerschmitt. Copyright 2000. See copyright notice.
From stovepipe to layering

Data  Voice  Video

Application-dependent infrastructure

Many applications

Integrated Infrastructure (Maybe broken into Additional layers.)

Application-independent
Stovepipe vs. Integrated Infrastructure

- What are some examples of each?

- What are the advantages of each approach?
Vertical Integration vs. Diversification

- A company is **vertically integrated** when it makes rather than buys the subsystems in its products.

- A **diversified** company produces products across different industry segments.
Vertical Integration vs. Diversification

Why do customers favor less vertical integration?

- Prefer competition amongst component suppliers
- Mix and match components
- Reduced lock in

Disadvantages??

- Customer needs to integrate components from different suppliers.
Vertical Integration vs. Diversification

Why do customers favor diversification?

- Reduce coordination costs by having to deal with fewer suppliers.
General Trend

- Less Vertical Integration
- More Diversification

- Of course there are exceptions...
Today’s supplier structure

Applications

Frameworks and components

Middleware

Infrastructure (network, OS) software

Equipment (network, computers)

Semiconductors, components
Standardization
Purpose of a standard?

- Allow products or services from different suppliers or providers to be interoperable
Scope of a standard

Included:

- interfaces (physical, electrical, information)
- architecture (reference model)
- formats and protocols (FAP)
- compliance tests (or process)

Excluded:

- implementation
- (possibly) extensions
Reference model

Decide decomposition of system
- where interfaces fall

Defines the boundaries of competition and ultimately industrial organization
- competition on the same side of an interface
- complementary suppliers on different sides
- hierarchical decomposition at the option of suppliers
- (possibly) optional extensions at option of suppliers
Some issues

Once a standard is set

- becomes possible source of industry lock-in; overcoming that standard requires a major (~10x?) advance
- may lock out some innovation

In recognition, some standards evolve

- IETF, CCITT (modems), MPEG
- backward compatibility
Types of standards

*de jure*
- Sanctioned and actively promoted by some organization with jurisdiction, or by government

*de facto*
- Dominant solution arising out of the market
- Voluntary industry standards body

Industry consortium
Common or best practice

Examples?
Examples

de jure
- GSM, ISDN Telephone interface

de facto
- Microsoft Windows API (Application Programming Interface)
- Intel Pentium instruction set,

Voluntary industry standards body
- IEEE (Institute of Electrical and Electronic Engineers) 
- IETF (Internet Engineering Task Force)

Industry consortium
- bluray

Best practice
- Windowed GUI
The changing process

- As technology and industry move more quickly, the global consensus standards activity has proven too unwieldy
  - e.g. ISO

- “New age” standards activities are more informal, less consensus driven, a little less political, more strategic, smaller groups
  - e.g. IETF

Programmable/extensible approaches for flexibility
  - e.g. XML, Java
Reasons for change

- From government sanction/ownership to market forces
  - Increasing fragmentation
  - Importance of time to market

Greater complexity

- Less physical/performance constraint for either hardware or software
**Lock-in**

(Particularly open) standards reduce consumer lock-in

- Consumers can mix and match complementary products

Increase supplier lock-in

- Innovation limited by backward compatibility
- e.g. IP/TCP, x86, Hayes command set
Aside: Network Effects

- The value of owning some products goes up if lots of other people have it too.
  - Examples?

- This phenomenon is called “network effects”

- How do standards influence network effects?
Network effects

Standards can harness network effects to the industry advantage

- Revenue = (market size) x (market share)

Increases value to customer

Increases competition

- Only within confines of the standard
- But forces customer integration or services of a system integrator

Slide adapted from slides for Understanding Networked Applications By David G Messerschmitt. Copyright 2000. See copyright notice
Why standards?

de jure are customer driven to reduce confusion and cost
de facto standards are sometimes the result of positive feedback in network effects

Customers and suppliers like them because they

- increase value
- reduce lockin

Governments like them because they

- promote competition in some circumstances
- May believe they can be used to national advantage
Approaches

Consensus
- ISO

Collaborative design
- MPEG

Competitive “bake off”
- IETF
Open vs. Proprietary Standards

- **Open standard** - a standard that is well documented, unencumbered by intellectual property rights and restrictions, and available to any vendor.

- What are the advantages?

- What are the disadvantages?
Why companies participate

Pool expertise in collaborative design
  - e.g. MPEG

Have influence on the standard

Get technology into the standard
  - Proprietary, with expectation of royalties
  - Non-proprietary

Reduced time to market
Standards applied to Business Processes?

Can you standardize business processes?

Yes!

- ISO 9000
  - A set of standardized business processes for Quality Management.
  - Supports TQM (Total Quality Management)

- RosettaNet
  - A set of standardized business processes, and accompanying standardized data interfaces/formats for conducting e-business.