Announcements

- Read Messerschmitt Ch 7 for Thursday.
- Folio 2 due Tuesday (if you are doing a folio!)
Student Talks
Sun Case
(continued)
Sun N-tier case

What does Sun make?
- Workstations
- Servers
- Software
How Successful had Sun been up to 1998?

- Founded in 1982
- Open Standards Workstation
  - Unix Operating System (Solaris)
  - TCP/IP networking
- 1988 - Revenues $1 billion
- 1993 - Market value $3.0 billion
- 1997 - Jumped from 3rd to 1st in Unix Server Market.
How Successful had Sun been up to 1998?

- 1993 - “The network is the computer.”
- 1994 - Internet explodes in popularity
Microsoft mid to late 90s

- Dominated Desktop software
  - Users familiar with Windows, Office, etc.

- NT servers
  - Fine for small intranets, “not industrial strength”
Sun N-Tier Case

- **What is Java?**
  - Programming Language
  - Portable between computers with different operating systems
  - Easy to write programs in
  - Easier re-use
  - But, programs are slow
What problems did the micro era produce?

- Desktops are expensive to maintain
  - TCO for windows PC $9900!

- Every PC had a lot of software that had to be maintained
  - Office, Windows, etc...

- Small differences, like the order in which software is installed, could make different PCs behave differently!
Sun’s Vision

- Thin Client model.
- Application Servers with Applications written in Java.
- NCs could retrieve applications from application server as needed.
- Applications compatible with any NC hardware and OS.
- Applications could be fixed, added, updated at the server level, rather than maintaining each PC.
Microsoft Vision

- Keep “fat-client” model
- Add some features to Windows to reduce administration costs
SUN 3 - Tier

Exhibit 1  Three-tier Architecture

Tier One

Tier Two

Tier Three

Asia

United States

Europe

Applets

App Server

Applets

App Server

Applets

App Server

Client Applets

Client Hardware

Client Applets

Client Hardware

Client Applets

Client Hardware

Database

JDBC: Stands for Java Database Connectivity. It is a programming interface that lets Java applications access a database via the SQL language.

RMI: Stands for Remote Method Invocation. It is the method by which a remote Java object from one location can be invoked from other Java virtual machines.

HTTP: Stands for HyperText Transport Protocol. It is the communications protocol used to connect to servers on the World Wide Web.
Sun N-tier

Tier One
- Client
- Client
- Client
- Client

Tier Two
- Webtop Server
- 1st time, applet sent
- Tier One
- Webtop Server
- 1st time Servlet sent
- Tier Three
- High latency servlet talks back & forth
- Tier Four

Tier Three
- App Server
- app server talks to central database
- Tier Two
- Webtop Server
- 1st time Servlet sent
- Tier Four

Tier Four
- Database
- United States
- Europe
- If bug found, change code here. Next time, corrected applet is sent down
- Tier Three
- App Server
- Tier Two
- Webtop Server
- Tier One
- Client
- Client
- Client
- Client

Asia
- Tier Four
- Tier Three
- Tier Two
- Tier One

United States
- Tier Four
- Tier Three
- Tier Two
- Tier One

Europe
- Tier Four
- Tier Three
- Tier Two
- Tier One
Sun N-Tier

**Step 1:** The user logs into his client and calls down an application. This message is sent to the Application Server.

**Step 2:** An initial applet is sent to the client. At the same time a servlet is sent to the Webtop Server.

**Step 3:** The applet talks back and forth with the Webtop Server via the LAN.

**Step 4:** As new data is received (e.g., a new customer’s name) the App Server communicates with the database to update that information.

**Local:** The Webtop Server and client communicate via a LAN.

**Remote:** The database and App Server communicate with the Webtop Server via a WAN.

**Exhibit 3** How the N-tier Architecture Works
Today

- 3-tier model common.
- Sun’s version of 4-tier model not-common.
- N-tier model where Webserver and Application Server on separate equipment also common.
- Sun’s hardware business not strong.
  - Linux on cheap PCs most common servers
  - Microsoft desktops replacing Sun workstations
Today

- **Java**
  - Common in Server implementations
    - Example: Java Servlet implementing application logic in a banking application.
  - Often used to push simple applets onto client
  - Not common
    - For “big” desktop applications
    - Office Suite in Java not popular
  - Microsoft is still in business...
What could have Sun done?

- Compete on price with cheap PC servers running Linux?
- Sell a fat-client workstation that runs Windows and is price competitive with Dell, HP PCs, etc...
- Sell workstations at a price premium over PCs, focus on software reliability, run some Microsoft application, build brand cachet.
- Focus on Java based software and IT services for enterprises, withdraw from low-end hardware...
- Something else?
Architecture Example
Architecture

- Conceptualization
  - What is it you are trying to do?

- Example Concept:
  - Small HHC for flight attendants.
  - HHC tells flight attendants which passengers are higher priority.
    - Who paid the highest fares
    - Who has been a more valuable customer in past
  - Flight attendant discriminates based on this
    - Free drinks, meals, and pillows to valuable customers
    - Ignore less valuable customers
Example Concept:
Architecture

- How do you begin to architect a solution for a problem like this?

- Break it into modules!
Architecture

HEADQUARTERS

Airline Dataserver

HHC Server

Airline Intranet

Wireless Link

HHC
When a module is composed of sub-modules, the architecture is **hierarchical**.
We are using a **layered architecture** as well.

- Allows reuse of previously built infrastructure.
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?
HHC Server

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

- HHC Server Application
  - Windows OS
  - Networking Infrastructure

- Computation of key statistics
- Communication with HHC
- 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 our HHC Server Architecture

- Compute Mean and Variance
  - List of numbers
  - Mean, Variance

- Computation of key statistics

- HHC Application
- Palm OS
- Networking Infrastructure

- 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 $2^{16} = 65,536$

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 $7 \times n$ 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

Module A

Computation of key statistics

Module B

Compute Mean and Variance

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

- One module should not be concerned with other module’s implementation
  - \textit{“Separation of concerns.”}
- One module should see the other only through its interface - implementation details hidden.
  - \textit{Abstraction}

Implementation 1:

- One module should not be concerned with other module’s implementation
  - \textit{“Separation of concerns.”}
- One module should see the other only through its interface - implementation details hidden.
  - \textit{Abstraction}
Throughout 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

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

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

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

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

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

Hello: I’m the HHC of Airplane#1234

Hello: I’m the gate 32 server

These were the unruly passengers on last flight

“Passengers noted”

Tell me about the passengers of my next flight

Return Passenger Data

Tell me about the weather at my next destination.

Return Weather Data

(Might be passed As an array of a compound data type “passenger,” which in turn is composed of standard types like integer, and string)
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 or printed**
- Menu of actions or returns from an action
- Touch selection of action

**Keypad**
- Input parameters to an action

**Card reader**
- Authentication, input parameters

**Money output slot**
- Returns money
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

authentication ➔ failure

choose objective ➔ other objectives

account ➔ no accounts

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

Each layer provides services to the layer above.

....by utilizing the services of the layer below and adding capability

Layer below as as a server to the layer above
Layering builds capability incrementally by adding to what exists.
Three types of software

- **Application**

  • Components and frameworks:
    
    What is in common among applications

- **Infrastructure:**

  Basic services (communication, storage, concurrency, presentation, etc.)
Part of Microsoft vs. DOJ dispute

Microsoft position

DOJ position

Application

Components and frameworks

Infrastructure
Major layers

Network

Operating system

Middleware

Application frameworks and components

Applications
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

In the simplest case, the infrastructure deals with a package of data (non-standard terminology)

- collection of bits
- specified number and ordering

The objective of the infrastructure is to store and communicate packages while maintaining data integrity

File for storage, message for communication
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

ABC Airlines

Alice

Envelope

UK Royal Mail

Shipping Container
Example 2

- Web server
- Web browser
- Screen
- HTML
- File
- Message
- File system
- Network
- Fragmentation
  - Collection of packets
  - Assembly
Example 3

HHC Server Application -> Passenger Information -> HHC Client Application

Windows OS

Networking Infrastructure (Contains: TCP/IP, WiFi)

Collection of Packets

Palm OS

Networking Infrastructure (Contains: TCP/IP, WiFi)
Example 3: Network Infrastructure Expanded
Example 4

HHC Server Application → "Send me today’s flight information" → DBMS

- Windows OS
- Unix OS

Networking Infrastructure Layers within TCP/IP, WiFi → Collection of Packets → Networking Infrastructure Layers within: TCP/IP, WiFi
Information in the infrastructure

Sometimes it is appropriate for the infrastructure to assume structure and interpretation for data

- to add capabilities widely useful to applications
- to help applications deal with heterogeneous platforms, where representations differ

At most, data types
Data and information

Application
Deals with information

Assumes structure and interpretation

Assumes standard data types

Infrastructure
Deals with data types