ISM 50 - Business Information Systems
Lecture 18

Instructor: John Musacchio
UC Santa Cruz
May 27, 2010

Class announcements

- Reading for Tues (6/1):
  - Messerschmitt Ch 10 (293-321)
  - Folio 3 due

Student Presentations

Student Presentation

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?

Domain Names

IP addresses are inconvenient for people
  - 32 bits hard to remember
  - 128 bits very hard to remember

Domain names
  - e.g. argus.eecs.berkeley.edu
  - Easier to remember than IP addresses
  - However, we need some way of mapping domain names to IP addresses.

Domain Name System (DNS)

Diagram showing the hierarchy of name servers starting from the root server and branching down to Berkeley, UCSC, EECS, and SoE name servers.
Internet Bottlenecks

- **First Mile (Server Capacity)** - 70% of website performance problems according to one study
- **Backbone** - Plentiful, but some shortage within metropolitan areas
- **Peering** - Exchange of traffic between NSPs
- **Last Mile to home**
  - 56 K modems are slow
  - Shared LAN limitations

Solutions

- **Expand Bandwidth**
  - Being done
- **Mirroring web cites**
  - Put exact copy of same web page to multiple servers
  - Tricky to duplicate content
- **Caching**
  - Problem: Stale Content
  - Problem: Hard to count "click throughs"
- **Content Distribution Networks**

Freeflow

- Deployed in 1999
- **Akamai Infrastructure**
  - 13000 servers in 954 networks by 2001
- **Customers** -
  - Large Commercial Websites
- **Revenue model** - $2000 per mbps served
  - (For comparison, normal Internet access cost 500 mbps at time)

2000 Financials

- **$196 Million Loss** (before special charges)
- **$90 million revenue**
- **520 gross margin, after deducting**
  - server depreciation
  - payments to network partners
  - Data center space
  - But, most expenses of shouldn’t grow at same rate as number of customers, so margin should improve
- **$201.5 million SG&A**
  - (selling general and administrative)
  - (largely sales force cost)
  - Again, this might not grow at same rate as the number of customers.
- **$40 million R&D**

Competition

- **Hosting firms (substitute)**
  - Exodus
- **Other CDNs**
  - Sandpiper, Adero, Mirror Image
- **Content Alliances**
  - Akamai’s competitors banded together to share networks
### 2001 Market Changes

**Bad**
- Dot-coms bust
- Customers leave
  - "churn rate goes to 22% per quarter"

**Good**
- Hosting firms go bust (exodus)
- Some CDN competitors go bust.
- Competing CDN alliances mired in problems

### EdgeSuite

- Assemble dynamic pages at edges rather than just serve heavy objects
- Value proposition
  - Performance improvement
  - Cost and complexity reduction
  - Scalability
  - Security
- Pricing - higher than old service
- Soon edge suite dominated revenue

### Technology

**Dynamic CDN technology: ESI (edge sides includes)**

**Develop as open standard why?**

Akamai not big and credible enough to force a de-facto standard on market

### Marketing

**Difference in selling old vs new products:**
- **Old product**
  - Geared toward speeding up websites
  - Revenues of their clients depended on speed
  - Easier to get sale
- **New Product**
  - Simplify company IT function
  - Cost vs. revenue center
  - Harder sell. More data driven...
  - Consequentially new product needs more professional sales force
- **Channels**
  - Distribution Partners (IBM) credibility
  - Direct Sales Force too

### Recent Performance
Chapter 9
Applications and the Organization

Build vs. Buy?

Purchase off the Shelf
- less time and cost
- benefits of using a “standard” solution
- support available
- must mold org to app
- no potential for competitive advantage

Outsource
- developers not as familiar with org as you
- more opportunity for customizing than off the shelf
- contractor may share knowledge with competitors
- contractor may have too much bargaining power

Make
- most customizable of 3
- easier iteration between conceptualization and development needed
- most risky
- org may lack competency to do it

Application Lifecycle

It is important to think beyond acquiring an application
- How do we come with the idea?
- How do we architect it?
- How do we implement?
- How do we extend and maintain it?

For this reason, the software engineering community came up with:
- Application Lifecycle Model

1) Conceptualization

What is the vision?
- What are the objectives?
- What is the business case?

- EXAMPLE: HHC to inform flight attendants which passengers are low and high value.
- Business Case:
  - Increase repeat business from high value customers.

1) Conceptualization -- Example:
2) Analysis

- Describe what the application will do.
- Enough info to allow "stakeholders" to review idea
- Don’t make highly detailed specifications
- Describe scenarios in which it is used
  - (Use Cases)

2) Analysis -- Example

- Example: Scenario:
  - REPORTING FUNCTION
    - FA wants to report that passenger in 13F is bad.
    - FA clicks "report pass." button followed by 13f
    - HHC finds from its data that Joe Schmoe is in 13f
    - When HHC is in radio range of WiFi AP, HHC tells server that Joe Schmoe is bad.

3) Architecture Design

- Decompose the application into subsystems
  - Hardware, software
  - Try use commercial off the shelf subsystems
  - Try to use standard infrastructure layers
    - Operating system, network, middleware, etc.

3) Architecture Continued

- Define the functionality, interaction and interfaces of subsystems
- While doing this, consider
  - Scalability
    - How easily can we increase the number of users and maintain performance?
  - Extensibility
    - How easily can we add new features in the future?
  - Administration
    - How much work will it take by humans to keep this running properly?
  - (Remember Sun thin vs fat client discussion)
4) Development Evolution

- Develop the details
  - Develop/program custom subsystems
  - Have contractor build outsourced pieces
  - Put together with off-the-shelf components

- Incremental
  - Start with simplest implementation and get it working
  - Later add more features.

5) Testing

- A must!
  - If architected well, we can test subsystems independently.
  - Alpha test - offline test of prototype
  - Beta test - test in intended environment with cooperative users
    - Example - give HHC to initial group of FA’s

6) Deployment

- Convert from previous processes if necessary
  - Example: CISCO ERP (all at once)
  - Or, you could do incrementally

- Train users
  - Example: Frito-Lay HHC

- Data importation
  - (if necessary)

7) Operations, Maintenance, Upgrade

- Maintain Security
- Repair Problems
- Correct performance short comings (Cisco ERP)
- Add features

Application Lifecycle Model

concluding remarks

- ALM rarely followed precisely
- Many times projects loop between stages
- ALM followed more closely in larger companies
- Alternative:
  - Rapid Iterative Prototyping
    - (Cisco did some of this in the ERP case.)

Chapter 10 – Application Architecture
Application Architecture

- **Decomposition** - Divide the architecture into interacting modules.
- **Assembly** - Find subsystems available for purchase
- Most architecture design is a mixture of decomposition and assembly.

Decomposition Example

- Example: manage bank accounts
  - Decompose into software modules for
    - transaction processing,
    - statement generation
  - Further decompose transaction processing module into deposit and withdraw modules...

Assembly Example

- Example: ecommerce platform
  - Acquire
    - Linux pc (application server)
    - IBM Mainframe (data server)
    - Oracle DBMS
    - Apache Web Server Software
  - Assemble all pieces together.
  - Mix with custom developed application logic module.

Object-Oriented Architectures

- Object-Oriented Programming (OOP) Languages
  - C++
  - Java
  - Smalltalk
- The basic unit of modularity in OOP is an object.

Objects

- Example: Bank account
  - has a balance of $5000
  - belongs to Joe Schmoe
  - is a checking account
  - can have money deposited to it
  - can have money withdrawn from it

- An attribute is a numerical value or data that is externally visible, and may be changeable.
  - Ex: The bank account's balance is $5000
- A method is an action available at the object interface
  - Other objects invoke method, pass parameters and get returned data or other objects.
  - We can invoke the "check_balance" method and get returned the number $5000
Object Classes and Instances

- Some objects share types of attributes and methods.
  - They have the same **class**

**Example**
- Class: Bank_Account
- Instances:
  - Schmoe_Account balance: $5000
  - Smith_Account balance: $10000
- Each instance is a separate object with its own data

Declaring Classes

When we program, we define or "declare" each class we plan to use.
- Example: We plan to use a class called "bank_account"
  - It will have the attributes: balance, owner, etc...
  - It will have the methods: check_bal, withdraw, deposit, ...
  - Later on we fill in the details of what each method does.
- Once we declare a class, we can create instances of it.
  - Schmoe_account, smith_account, etc...

Method Invocation

- Objects communicate with each other by invoking each other's methods
  - (method invocation)

```
ATM Object

<table>
<thead>
<tr>
<th>Schmoe_Account</th>
</tr>
</thead>
<tbody>
<tr>
<td>check_bal()</td>
</tr>
<tr>
<td>withdraw( $500)</td>
</tr>
<tr>
<td>balance: $5000</td>
</tr>
<tr>
<td>withdraw()</td>
</tr>
<tr>
<td>check_bal()</td>
</tr>
<tr>
<td>balance: $5000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Smith_Account</th>
</tr>
</thead>
<tbody>
<tr>
<td>check_bal()</td>
</tr>
<tr>
<td>withdraw( $1000)</td>
</tr>
<tr>
<td>balance: $10000</td>
</tr>
<tr>
<td>withdraw()</td>
</tr>
<tr>
<td>check_bal()</td>
</tr>
<tr>
<td>balance: $10000</td>
</tr>
</tbody>
</table>

Invoke: check_bal() return: $5000
Invoke: withdraw ( $500) return: "successful"
```

**Terminology:**
- Client object -- object invoking the method is the
- Server object -- object whose methods are being invoked

Software Objects

- In OOP an object can
  - Represent a real world entity
    - Bank account
  - Be a proxy of a real world entity
    - Proxy of a customer
    - Other software talks to proxy using method invocations
  - Model a real-world entity
    - For purposes of simulation
    - Motion of a train

ORDBMS

- Earlier in the class we talked about relational DBMS
  - The most common database management system that organizes data into tables.
- ORDBMS (Object Relational DBMS)
  - Retrieve and store object instance data in a relational database

Remote Method Invocation

- Sometimes we want to allow an object to invoke methods on an object located on another machine.
- This is called Remote Method Invocation (RMI)
- Doing this requires middleware called
  - Distributed Object Management (DOM)
Software Reuse

- Size and complexity of applications growing dramatically
- In order to contain costs, we need to be able to reuse pieces of software
- Reuse is difficult. Why?
- OOP was developed in part to promote re-use, but has had limited success in that regard.

Software components

- Software components are reusable modules that can be bought from outside vendors.
- How is a component different from an object?
  - More importance on
    - Encapsulation
    - Well defined and documented interfaces

Component Assembly Tools

- Visual or integrated development environment (IDE)
  - MS Visual Studio
  - IBM Visual Age
  - Symantec Visual Café
- Scripting Assembly - Text based
  - TCL
  - Perl
  - JavaScript

Software Frameworks

- A preexisting architecture and library of components from a common vendor to help developers
- Enables reuse, and ensures component interoperability.
- Examples:
  - Sun J2EE/Java Beans
  - Microsoft .Net