ISM 50 - Business Information Systems
Lecture 17

Instructor: Ram Akella
UC Santa Cruz
Mar 9, 2010

Class announcements

- Reading for Thursday:
  - Messerschmitt Ch 10 (293-321)
  - Ch1 of networking mini-book (now on web)
- Folio 3 due Today
- Business Paper and Assignment 4 due on Thursday

Student Presentations

Student Presentation

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)

Hierarchy in Addresses vs. Names

Addresses hierarchical in topology
- Maximize "wild cards" and distribute address administration

Names hierarchical in administration
- Single administered organizations often distributed topologically (e.g. ibm.com)
Transport Protocols

- The Internet is unreliable
  - It will make a "best effort" to get your packet to its destination
- Packets can be lost because of
  - Congestion
  - Link errors
  - Routing problems

OSI Layers

- Application
- Presentation
- Session
- Transport
- Network
- Link
- Physical

Small/Medium Business

- Internet Explorer, Outlook Email, Real Player,...
- TCP, UDP
- Internet Protocol (IP), ...
- Ethernet, Wi-Fi, SONET, ...
- Modulation Schemes: QAM, OFDM, etc...

ISP Topology

Network Service Provider
Large E-Business

Web Caching
- Speed up web page loading by storing previously seen components locally

Akamai Case

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 some 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
  - 13,000 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 ($230 million in 1999)
- $90 million revenue
- 720 gross margin, after deducting
  - server depreciation
  - payments to network partners
  - Data center space
- But, most expenses of shouldn't grow at some 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 some 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

<table>
<thead>
<tr>
<th>Year</th>
<th>2004</th>
<th>2005</th>
<th>2006</th>
<th>2007</th>
<th>2008</th>
</tr>
</thead>
<tbody>
<tr>
<td>Revenue</td>
<td>$203,050</td>
<td>$101,230</td>
<td>$144,020</td>
<td>$162,240</td>
<td>$87,798</td>
</tr>
<tr>
<td>Total cost of operations</td>
<td>$167,240</td>
<td>$173,370</td>
<td>$177,580</td>
<td>$1,073,040</td>
<td>$87,040</td>
</tr>
<tr>
<td>Net income (loss)</td>
<td>$35,810</td>
<td>$27,860</td>
<td>$66,440</td>
<td>$306,160</td>
<td>$87,798</td>
</tr>
</tbody>
</table>

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 app-to-app
  - no potential for competitive advantage
- Outsource
  - developers not as familiar with acq 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
  - must risky
  - acq 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
Application Lifecycle

Stages:
1. Conceptualization
2. Analysis
3. Architecture Design
4. Development Evolution
5. Testing and Evaluation
6. Deployment
7. Operations, Maintenance, and Upgrade

1) Conceptualization

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

- EXAMPLE: HHIC 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
  - PA wants to report that passenger in 13F is bad.
  - PA clicks “report pass” button followed by 13f
  - HHIC finds from its data that Joe Schmoe is in 13f
  - When HHIC is in radio range of WiFi AP, HHIC 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.
Design a hierarchical architecture.

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 Sunthin 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 FAs

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
- 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

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
- 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
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)

<table>
<thead>
<tr>
<th>ATM Object</th>
<th>invoke: check_bal()</th>
</tr>
</thead>
<tbody>
<tr>
<td>return: $5000</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Schmoe Account</th>
<th>invoke: withdraw()</th>
</tr>
</thead>
<tbody>
<tr>
<td>return: “successful”</td>
<td></td>
</tr>
</tbody>
</table>

- 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