## Icons / Metaphors

<table>
<thead>
<tr>
<th>Metaphor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>🧠</td>
<td>Information</td>
</tr>
<tr>
<td>🕯️</td>
<td>Common Realization</td>
</tr>
<tr>
<td>📚</td>
<td>Knowledge/Competency Pattern</td>
</tr>
<tr>
<td>🪷</td>
<td>Governance</td>
</tr>
<tr>
<td>🌟</td>
<td>Alignment</td>
</tr>
<tr>
<td>🧔</td>
<td>Solution Approach</td>
</tr>
</tbody>
</table>
Agenda

1. Legacy Page-Based Application Server Technology
2. Distributed Object Computing Concepts
3. Object Management Architecture
4. Conclusion

Application Servers for Enhanced HTML
(a.k.a., Page-Based Application Servers – Tag- vs. Script-Oriented)

- **Examples**
  - **Page-Based Tag-Based**
    - Adobe ColdFusion
      - Based on Macromedia Coldfusion 5.0
      - Coldfusion Markup Language (CFML) technology
        - Available in Coldfusion MX 6/7 Server and Coldfusion 8/9
  - **Page-Based Script-Oriented**
    - Microsoft IIS with ASP
  - **Hybrid Page-Based Tag & Script-Oriented**
    - WithEnterprise Pty Ltd Tango 2000/WiTango
      - [http://www.witango.com](http://www.witango.com)

- Typically used as a mid-range Solution from a cost and scalability standpoint
- Technology stays within the familiar HTML confines
  - Create pages with mixture of HTML and proprietary tags or script code using (third-party) IDE, HTML editor, or plain text editor
  - Application server evaluates the code upon user requests and creates HTML pages on the fly
Application Servers for Enhanced HTML
Target Operating Platform is Traditional Web Application Platform

Application Servers for Enhanced HTML
Target Build Approach is Rapid Web Application Development
### Tag- vs. Script-Oriented Select Product Differentiators

- ColdFusion Studio 4.5 & MX 2004
  - HTML coding
  - Basic database integration
  - UltraDev 4 or Fusion may be used as alternative IDE on Coldfusion Studio 4.5
  - Dreamweaver, Flash, Fireworks, FreeHand may be used as alternative IDEs with MX 2004
- Visual InterDev (ASP)
  - Management of site development process
  - Scripting
  - Macromedia’s Drumbeat or NetObjects’ Fusion may be used as alternative IDEs

### Measurable Benefit of Tagging vs. Scripting for Select Products

- ColdFusion
  - Easy tag-oriented dynamic pages for simple tasks
  - Scripts may be used when more complex coding is required
    - e.g., arrays, case & switch statements, and error handling
- Example
  - Simple phone directory application: 2 custom tags + 1 SQL statement
  - Same with pure scripting approach would take 100 lines of ASP code ...
Server Platforms Support

- ColdFusion
  - Windows
  - Solaris
  - Linux
- ASP
  - Windows
  - Use ChiliSoft for other servers

Tag-Based Page-Based Typical Technology

- IDE + Application Server
- IDE
  - Creates pages with mixture of HTML and proprietary tags or script code
  - Visual page creation
    - Textual creation possible as well
- Application Server
  - Evaluates the code upon user requests and creates HTML pages on the fly
Page-Based Tag-Based Sample Application Server:
The ColdFusion Development Platform

Page-Based Tag-Based Sample Application Server:
ColdFusion Web Applications
Page-Based Tag-Based Sample Application Server:
How ColdFusion Works

### How ColdFusion Works

1. **HTTP Request**
2. **CF Page**
3. **Database Server**
4. **File System**
5. **Email Server**
6. **Distributed Objects**
7. **Other Enterprise Systems**

#### Client
- Web Browser
- Internet

#### Network
- Web Server
- Web Page

#### Server
- ColdFusion Server
- Other Enterprise Systems
- File System
- Database Server

---

Page-Based Tag-Based Sample Application Server:
The ColdFusion Development Process

- Write some code
- Save it as a page (use .cfm extension)
- View it in a browser
- Write some more code
- Save the page again
- View it in a browser
- etc.
Page-Based Tag-Based Sample Application Server:
Sample ColdFusion Application

<HTML>
<HEAD>
<TITLE>My First Page</TITLE>
</HEAD>
<BODY>
<STRONG>ColdFusion</STRONG>
<CFSET ProductName = "ColdFusion">
</BODY>
</HTML>

Page-Based Tag-Based Sample Application Server:
Outputting a Variable Value in ColdFusion

<HTML>
<HEAD>
<TITLE>My First Page</TITLE>
</HEAD>
<BODY>
<STRONG>ColdFusion</STRONG>
<CFSET ProductName = "ColdFusion">
<CFOUTPUT>
#ProductName#
</CFOUTPUT>
</BODY>
</HTML>
Page-Based Tag-Based Sample Application Server:
Querying a Data Source in ColdFusion

```html
<HTML>
<HEAD>
<TITLE>Course List</TITLE>
</HEAD>
<BODY>
<H1>Course List</H1>
<CFQUERY NAME="CourseList" DATASOURCE="cfsnippets">
SELECT CORNUMBER, CORNAME
FROM CourseList
</CFQUERY>
<CFOUTPUT QUERY="CourseList" >
#CORNUMBER# #CORNAME#<BR>
</CFOUTPUT>
</BODY>
</HTML>
```

Page-Based Tag-Based Sample Application Server:
ColdFusion Search Engine CFML Tags (1/2)

- **CFCOLLECTION**
  - Sets up collections that Verity uses to search indexes
  - `<CFCOLLECTION ACTION="CREATE" or "REPAIR" or "DELETE" or "OPTIMIZE" or "MAP" COLLECTION="collection_name" PATH="path_of_verity_directory" LANGUAGE="English" or "German" or "Finnish" or "French" or "Danish" or "Dutch" or "Italian" or "Norwegian" or "Portuguese" or "Spanish" or "Swedish">`

- **CFINDEX**
  - Indexes collections from documents or database records
  - `<CFINDEX COLLECTION="collection_name" ACTION="action" TYPE="type" TITLE="title" KEY="ID" BODY="body" CUSTOM1="custom_value" CUSTOM2="custom_value" URLPATH="URL" EXTENSIONS="file_extensions" QUERY="query_name" RECURSE="Yes" or "No" EXTERNAL="Yes" or "No" LANGUAGE="language">`

- **CFSEARCH**
  - Searches indexes for a match to the search criteria
  - `<CFSEARCH NAME="search_name" COLLECTION="collection_name" TYPE="criteria" CRITERIA="search_expression" MAXROWS="number" STARTROW="row_number" EXTERNAL="Yes" or "No" LANGUAGE="language">`
CFSEARCH Variables:
- What result variables are returned from CFSEARCH
  - #DocSearch.URL# (returns index’ URL)
  - #DocSearch.KEY# (unique identifier)
  - #DocSearch.TITLE# (based on document title)
  - #DocSearch.SCORE# (relevance of search term)
  - #DocSearch.Summary# (contains index’ 1st 500 chars)

CFSEARCH Return Variables:
- #DocSearch.recordCount#
- #DocSearch.recordsSearched#
- #DocSearch.columnList#
- Powerful and intuitive tag-based server scripting language
- Two-way visual programming and database tools
- Remote interactive debugging
- Web application wizards & tag-based component architecture
- Source control integration
- Secure file and database access via HTTP
Page-Based Tag-Based Sample Application Server:
ColdFusion Features - Scalable Deployment

- Multi-threaded service architecture
- Database connection pooling
- JIT page compilation and caching
- Dynamic load balancing
- Automatic server recovery and fail-over

Page-Based Tag-Based Sample Application Server:
ColdFusion Features - Open Integration

- Database connectivity (ODBC, OLE-DB, native database drivers)
- Embedded support for full text indexing and searching
- Standards-based integration (directory, mail, etc.)
- CORBA and COM+ connectivity
- Open extensibility with C/C++
### ColdFusion Features - Complete Security

- Integration with existing authentication systems (NT/Win 200x domains, LDAP directory servers)
- Advanced access control to files and data sources
- Support for existing database security
- Server sandbox security
- Support for Web server authentication, security, and encryption

### ColdFusion Studio

- Supports other languages than HTML
  - Handled Device Markup Language
  - Synchronized Multimedia Integration Language
- Visual Tool Markup Language
  - Support the inclusion of tag editing dialogs
  - Support the addition of XML capabilities
- CSS integration is clumsy (separate editor)
- Link management utility limited to page by page (no site diagramming)
Page-Based Tag-Based Sample Application Server:
ColdFusion App Server

- Supports clustering
- Addresses performance and scalability issues at most levels
- Supports ODBC, OLE, and native drivers for Oracle and Sybase
- Also supports stored procedures
- Supports server load balancing (Bright Tiger Technologies’ ClusterATS) and failover

Page-Based Tag-Based Sample Application Server:
ColdFusion Features Summary

- Advanced Editor
- Visual Database Tools
- Two-way Visual Programming
- Web Application Wizards
- Code Re-Use
- Interactive Debugging
- Dynamic Page Quality Assurance
- Tag Property Inspection
- Code Sweeper
- Extensible Tag Editors
- Custom Wizards
- Visual Tool Object Model
- Customizable Workspace
- Server-Side Source Control
- Shared Project Management
- One-Step Deployment
- Remote Team Development
Page-Based Tag-Based Sample Application Server:
ColdFusion 5.0 vs. MX 6.1/7.0

• MX Features:
  • Server Scripting
    ▪ CFML, XML, JSPs/custom tags
  • Integrated Application Services
    ▪ Flash, Web services
  • Flexible Application Deployment
  • High Performance Architecture
  • Advanced Development Capabilities
    ▪ ColdFusion Components (CFCs)
  • Enterprise Systems Integration
  • Advanced Server Management

Page-Based Tag-Based Sample Application Server:
Coldfusion - Using CFCs As A Facade

Idea is to keep code that invokes Java objects out of CFML

Hybrid Application
Page-Based Script-Oriented Sample Application Server:
PHP Technology

- Server-side, cross-platform HTML embedded scripting language
- PHP is an open source project of the Apache Software Foundation
- See http://www.php.net/index2.php

Example (hello.php):
<html><head><title>PHP Test</title></head>
<body>
<?php echo "Hello World<p"; ?>
</body></html>

Page-Based Script-Oriented Sample Application Server:
PHP Examples (1/3)

- Showing variables
  <?php echo $HTTP_USER_AGENT; ?>

- Getting a list of web server variables
  <?php phpinfo(); ?>

- Checking for Internet Explorer
  <?php
  if(strstr($HTTP_USER_AGENT,"MSIE")) {
    echo "You are using Internet Explorer<br"; 
  }
  ?>
Jumping in and out of PHP mode

```php
<?php
if(strstr($HTTP_USER_AGENT,"MSIE")) {
    ?><center><b>You are using Internet Explorer</b></center><?
} else {
    ?><center><b>You are not using Internet Explorer</b></center><?
}  
?>
```

Flexible HTML Forms Handling

- **Typical HTML form:**
  ```html
  <form action="action.php" method="post">
  Your name: <input type="text" name="name">
  You age: <input type="text" name="age">
  <input type="submit">
  </form>
  ```

- **Action.php is as follows:**
  ```php
  Hi <?php echo $name; ?>.
  You are <?php echo $age; ?> years old.
  ```
Page-Based Script-Oriented Sample Application Server:
PHP Software

- Source and binaries downloadable from:
- Includes
  - CGI binary plus server API versions for Apache, AOLserver, ISAPI and NSAPI
  - MySQL support built-in
  - Many other extensions

Extended Page-Based Tag-Based Approach:
Legacy XML Application Server Architecture (1/3)

HP Bluestone XML Server 1.0/Visual-XML – Overall Architecture
Extended Page-Based Tag-Based Approach: Legacy XML Application Server Architecture (2/3)

HP Bluestone XML Server 1.0/Visual-XML – Sample Request Processing

```
<?xml version="1.0"?>
<author-list>
  <author>
    <au_id>172-32-1176</au_id>
    <au_last_name>Johnson</au_last_name>
    <au_first_name>Michael</au_first_name>
    <phone>506-7223</phone>
    <address>3208 N. 45th, Wausau, WI</address>
    <city>Wausau</city>
    <state>WI</state>
    <country>USA</country>
    <zipcode>94015</zipcode>
  </author>
</author-list>
```

Extended Page-Based Tag-Based Approach: Legacy XML Application Server Architecture (3/3)

Binary Evolution Velocigen - IDE
Page-Based Application Server Technology Summary

- Page-Based Application Servers are either based on HTML tagging or scripting
- Page-Based Application Servers are typically used for supporting mid-range solutions from a cost and scalability standpoint
- ColdFusion is based on HTML extensions, and supports the development of tag-oriented dynamic pages for simple tasks
- PHP is a server-side cross-platform HTML embedded scripting language
- XML Application Servers are either used to support Message-Oriented Middleware (MOM) or Presentation-Oriented-Publishing (POP) and rely on server-side processing of XML documents

Legacy Enhanced HTML vs. New Generation Page-Based Script-Oriented App. Servers

- Application Servers for Enhanced HTML (traditional)
  - a.k.a., Page-Based Application Servers
  - Tag-Oriented (e.g., Adobe ColdFusion)
  - Script Oriented (e.g., Microsoft IIS with ASP, PHP)
  - Mostly Used to Support Standalone Web Applications
  - Typically less expensive than standalone and IDE-based servers
  - HTML-based development
- New Generation Page-Based Script-Oriented App. Servers
  - First Generation Extensions (e.g., Microsoft IIS with COM+/ASP)
  - Servlet/JSP Environments
  - XSP Environment
  - Can now be used as front-end to enterprise applications
  - Hybrid development environments
### Agenda

<table>
<thead>
<tr>
<th></th>
<th>Legacy Page-Based Application Server Technology</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Distributed Object Computing Concepts</td>
</tr>
<tr>
<td>3</td>
<td>Object Management Architecture</td>
</tr>
<tr>
<td>4</td>
<td>Conclusion</td>
</tr>
</tbody>
</table>

### Architecture Transformation Categories (1/2)

**Refactoring Components vs. Architecture Using Patterns**

<table>
<thead>
<tr>
<th>Transformation Type</th>
<th>Component</th>
<th>Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Added functionality, rules, and/or constraints</td>
<td>Convert Quality Requirement to Functionality</td>
<td>Impose Architectural Pattern</td>
</tr>
<tr>
<td>Restructuring</td>
<td>Apply Design Pattern</td>
<td>Impose Architectural Style</td>
</tr>
</tbody>
</table>

*In principle, repeat each step for each quality attribute*

*Note: Prime Citizens in Object Management Architecture (OMA) are Components*
Architecture Transformation Categories (2/2)
Sample Architectural Patterns

- From MUD to Structure…
  - Layers, Pipe and Filters, Blackboard
- Distributed Systems…
  - Broker, Pipe and Filters, Microkernel
- Interactive Systems…
  - MVC, PAC
- Adaptable Systems…
  - Microkernel, Reflection…

Traditional Distributed Computing Context

- Traditional distributed computing
  - Meant to support programs written in a procedural programming language
  - Provides the ability to invoke procedures/functions hosted by various programs distributed over the network via:
    - Remote Procedure Calls (RPC)
    - External Data Representation (XDR)
  - Subsumes the “client-server” or “n-tier” architectural patterns
Layers and tiers typically represent interchangeable terms

Common tier topologies
- Two Tier - also known as “Client-Server”
- Three Tier
- N Tier

A two tier system typically partitions responsibility between an application and a database server
Traditional Distributed Computing
Two Tier Architectural Pattern - Advantages

- Simple!
- Ideal for rapidly prototyping an application with persistent storage requirements
- Suitable for script based platforms (e.g. Python, Perl, JavaScript) where additional architectural layers are unnecessary
- Provides greater levels of performance (less “time on the wire” as business logic is located on the client)

Traditional Distributed Computing
Two Tier Architectural Pattern - Disadvantages

- Complex and expensive to maintain and redeploy
- Poor scalability - how do you handle an increasing user base?
Traditional Distributed Computing
Three Tier Architectural Pattern - Overview

- Partitions responsibility into three classic layers
  - User Interface
  - Business Logic
  - Persistence

Traditional Distributed Computing
Three Tier Architectural Pattern - Advantages

- Many of the key advantages of layering
  - Reuse
    - Maintainability: each layer can be understood and modified without needing to understand higher or lower levels
  - Effective use of technology paradigms
    - A business logic layer can represent data as objects
    - A RDMS layer can represent data as tables
    - No direct impedance mismatch by "shoehorning" one into the other!
  - Specialization - staff can localize their skills around the maintenance of specific layers
### Traditional Distributed Computing

#### Three Tier Architectural Pattern - Disadvantages

- As per classic layering...
  - Performance
  - Redundancy
  - Complexity
    - With implications for maintainability!

<table>
<thead>
<tr>
<th>Traditional Distributed Computing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>When to Use Three Tier Architectural Pattern (Gartner Group)</strong></td>
</tr>
<tr>
<td><strong>Use a three-tier architecture if any of the following conditions apply:</strong></td>
</tr>
<tr>
<td>- Systems with many application programs (more than 20)</td>
</tr>
<tr>
<td>- Mix of application programs of different languages or origins</td>
</tr>
<tr>
<td>- Two or more heterogeneous data sources</td>
</tr>
<tr>
<td>- e.g., two RDBMS or a DBMS and a file system</td>
</tr>
<tr>
<td>- Expected life of application beyond three years</td>
</tr>
<tr>
<td>- Many modifications and additions anticipated</td>
</tr>
<tr>
<td>- High-volume workload</td>
</tr>
<tr>
<td>- e.g., more than 10,000 transactions per day</td>
</tr>
<tr>
<td>- Significant inter-application communication, including inter-Enterprise communication</td>
</tr>
<tr>
<td>- Upsizing</td>
</tr>
<tr>
<td>- i.e., the application may grow over time so one or more previous conditions apply</td>
</tr>
</tbody>
</table>
Understanding Distributed Object Computing (DOC) Components

- **Component**
  - A component is an encapsulated part of a software system
  - A component has an interface
  - Components serve as the building blocks for the structure of a system
  - At a programming-language level, components may be represented as modules, classes, objects or as a set of related functions

Programming Level Components
Object-Oriented Programming (OOP)

- **OOP** is a method of implementation in which programs are organized as cooperative collections of objects
  - Objects are instances of some class
  - Classes are members of a class hierarchy
  - Classes are united via inheritance relationships in which child classes can share members of parent classes and have similar structure and behavior

- **Class**
  - Language type with structure and behavior
  - Named or anonymous
  - Contain member variables and methods

- **Object**
  - Instance of a class with own state, behavior, and identity
- **OOP Infrastructure**
  - Provides Language Compiler and Tools
  - Facilitates OOAD process

- **OOP Conceptual Framework**
  - Object Model (OM)
    - Abstraction, encapsulation, modularity, inheritance hierarchy, polymorphic typing
  - OM notions can be applied to component programming
    - e.g., modularity in OOP via “componentization”!
  - OOP and Component Programming are difficult to separate
    - e.g., abstraction, encapsulation, etc.

---

- **Polymorphism**
  - **Subtype polymorphism**
    - Abstract base classes with inheritance and abstract methods (C#)
      ```csharp
      abstract class A
      {
        void f();
      }
      class B extends A
      {
        void f() { Console.WriteLine("B class");}
      }
      ```
  - **Parametric polymorphism**
    - Uses template classes or functions (C++)
  - **Ad-hoc polymorphism**
    - Achieved via operator and function overloading (+ -> concatenate)
DOC Component (1/2)

- Cannot exist without a component infrastructure
  - Compilers use different name calling conventions
  - Must introduce a component infrastructure that hides the underlying complexity of different tools and systems used when creating and deploying components

- Neither an object nor a class nor a combination
  - May include many classes
  - Component exports interfaces rather than methods
  - Interfaces do not always have 1-1 correspondence with a class

- Not a module (closed static unit that encapsulates embedded abstractions)
  - Component can only contain binary units (no source code)

- A component interface is the unit of component behavior in which some logically grouped methods are exposed to clients

DOC Components (2/2)

- Component properties
  - Contain compiled program files
  - Package compiled programs in a file that has a format known to the component infrastructure
    - e.g., .DLL, .CAB, .JAR, .EXE
  - Language neutral
    - JEE EJB example:
      - May use software bridge to use JavaBeans from VC++
      - e.g., ActiveX bridge (component infrastructure)
  - Registers with Component infrastructure via a unique name
  - Development tools may use introspection to explore components at runtime
**DOC Component Infrastructures & Models**

- **Component Infrastructure**
  - Set of system services based on a component standard that ensure the properties of the component are immutable and enables rules of component composition and interaction

- **Component Model**
  - Defines component interaction and composition standards

- **Components subsume standards, component infrastructures, tools, and OOAD**

- **Enterprise assurance**
  - **Performance**
    - e.g., Concurrency (stateful/stateless components), caching of data or component, distributed transaction management (many components may modify databases)
  - Reusability (plug and play)
  - Scalability (resource mgmt)
  - etc.

---

**Distributed Object Computing – Applicable Patterns**

**Broker Architectural Pattern – High-Level Description**

- Used to decouple distributed applications by routing remote service invocations through a mediating broker
- Broker component is responsible for coordinating communications, forwarding requests, returning replies and propagating exceptions
- Also known as “Hub and Spoke” architectural pattern
- Monolithic applications are significantly less flexible and scalable than applications partitioned into distributed components
- With distribution, a component becomes coupled to communication mechanisms and server component identities
- Services for adding, removing, exchanging and starting components are also required
Broker Architectural Pattern – Solution Description

- Introduce a broker to arbitrate communication
- Servers register themselves with the broker, and expose method interfaces
- Can be updated, added or moved from the broker registry
- Clients only have a single dependence - on the broker

Broker Architectural Pattern – Architectural Components

- Client - a component that communicates with server(s) via the broker
- Server - a component that services requests from clients via the broker
- Broker - the central point for message routing
- Client-Side Proxy - an object used for client-to-broker communication, used to hide implementation details of the protocol and transmission medium
- Server-Side Proxy - an object used for server-to-broker communication, used to hide protocol / communication details
Distributed Object Computing – Applicable Patterns
Broker Architectural Pattern – Structure 1/5

- **Client-side Proxy**
  - pack_data
  - unpack_data
  - send_request
  - return

- **Broker**
  - main_event_loop
  - update_repository
  - register_service
  - acknowledgment
  - find_server
  - find_client
  - forward_request
  - forward_response

- **Server-side Proxy**
  - pack_data
  - unpack_data
  - call_service
  - send_response

- **Client**
  - call_server
  - start_task
  - use_Broker_API

- **Bridge**
  - pack_data
  - unpack_data
  - forward_message
  - transmit_message

- **Server**
  - initialize
  - enter_main_loop
  - run_service
  - use_Broker_API

Distributed Object Computing – Applicable Patterns
Broker Architectural Pattern – Structure 2/5

- **Class**
  - Broker

  - **Responsibility**
    - (Un-)registers servers.
    - Offers APIs.
    - Transfers messages.
    - Error recovery.
    - Interoperates with other brokers through bridges.
    - Locates servers.

  - **Collaborators**
    - Client
    - Server
    - Client-side Proxy
    - Server-side Proxy
    - Bridge

- **Class**
  - Client-side Proxy

  - **Responsibility**
    - Encapsulates system-specific functionality.
    - Mediates between the client and the broker.

  - **Collaborators**
    - Client
    - Broker

- **Class**
  - Server-side Proxy

  - **Responsibility**
    - Calls services within the server.
    - Encapsulates system-specific functionality.
    - Mediates between the server and the broker.

  - **Collaborators**
    - Server
    - Broker

- **Class**
  - Bridge

  - **Responsibility**
    - Encapsulates network-specific functionality.
    - Mediates between the local broker and the bridge of a remote broker.

  - **Collaborators**
    - Broker
    - Bridge
Distributed Object Computing – Applicable Patterns
Broker Architectural Pattern – Structure 5/5

- Direct Communication Broker System
  - Direct link to server
- Message Passing Broker System
  - Focus on transmission of data
  - Type of message determines the behavior of the broker
- Trader System
  - Service identifiers are used to access server functionality
  - Request can be forwarded to more than one server
- Callback broker system
  - Event-driven
**Distributed Object Computing – Applicable Patterns**

**Broker Architectural Pattern - Known Uses**

- CORBA
- IBM SOM/DSOM
- Microsoft OLE 2.x
- WWW
- ATM-P: Message passing broker
  - Telecommunication switching system based on ATM

**Broker Architectural Pattern - Advantages**

- Location transparency
  - Broker isolates clients from specific knowledge of server components
- Programming language independence
  - Broker supports client/server components written in different programming languages
- Flexibility
  - Clients and servers can be changed/extended with minimal impact to other components
- Portability (via layering)
  - Broker isolates components from other component dependencies such as communication protocols, message formats and operating systems
- Interoperability between brokers (bridge)
- Reusability (of services)
Broker Architectural Pattern - Disadvantages

- Testing and debugging
  - Testing a client fully requires distributed functionality to be in place
  - Requires testing of broker component interoperability
  - Tedious and time-consuming

- Efficiency
  - Delay introduced by indirection layer
  - Delayed binding time and communication overheads reduce performance

- Lower fault tolerance due to replication of components
  - Fault a broker or a server…

Broker Architectural Pattern - Steps to Implement Broker

1. Design an object model
2. Define the component interoperability model
   - e.g., binary, IDL, SOAP over HTTP
3. Specify the broker API for communicating with clients and servers
4. Build client and server proxies
   - Client proxy to hide details of interaction with broker
   - Server proxy to receive requests from broker and pass back results and exceptions
5. Design the broker component
   - Specify the “on-the-wire” protocol
   - Implement a directory service for associating local server identifiers with physical machines
   - Provide marshalling and un-marshalling capabilities for message transformation (e.g. from XML to objects)
Agenda

1. Legacy Page-Based Application Server Technology
2. Distributed Object Computing Concepts
3. Object Management Architecture
4. Conclusion

Understanding Enterprise Computing Layers/Facilities

- Distributed Object Computing
- Application Servers
- Client Interfaces (MOM & POP Applications)
- Web Application Modeling (MOF/UML/XML)
- Adaptiveness & Model-Driven Application Generation
DOC Platform Standards

- OMA/CORBA
  - Borland VisiBroker Environment
  - Progress’ Orbacus
- RMI and RMI-IIOP
- DCOM
- DOC Platform Interoperability
- Web-Enabled DOC Applications

DOC Component/Container Models

- CORBA 3 Component Models
  - CCM
- J2EE Component Models
  - JSP, Servlet, EJB, MDB, CMP, etc.
- COM+
  - MTS
- etc.
**Common Object Request Broker Architecture (CORBA)**

- Open distributed computing infrastructure/platform (OMA)
- Programming language independent DOC environment
- Based on OMG IDL, and IIOP/GIOP
- Standardized by the OMG
- Automates common networking programming tasks
  - Object registration, location, activation
  - Request demultiplexing
  - Framing and error handling
  - Parameter marshalling and demarshalling
  - Operation dispatching

**CORBA 3 Features**

- Component Model (CCM)
- Quality-of-service control
- Messaging invocation model
- Tightened integration with the Internet
- POA (Portable Object Adapter)
- EJB and Java support
- OMG specifications for analysis and design, and application interoperability
  - UML, MOF, XMI, Common Warehouse Model
Object Management Architecture (OMA)
(http://www.omg.org/oma/)

- OMG’s vision of a component-based software environment
  - Plug and play software environment
  - Leverage component standardization

Original OMG Reference Model Architecture
Domain-independent interfaces used by many distributed object programs

- Naming Service: allows clients to find objects based on names
- Trading Service: allows clients to find objects based on their properties
- Other services: persistence, lifecycle management, security, transactions, event notification, etc.

Horizontally-oriented interfaces

- Oriented towards end-user applications
- E.g.: Distributed Document Component Facility (DDCF)
  - Compound document facility based on OpenDoc
  - Allows for presentation and interchange of objects based on a document model (e.g., linking of spreadsheet object into a report document)
- Printing, Secure Time, Internationalization, and Mobile Agent Facilities
Role similar to Object Services and Common Facilities
Oriented towards specific application domains
E.g., Product Data Management (PDM) Enablers for the manufacturing domain
Other possibilities in the telecommunications, medical, and financial domains

Interfaces developed for a given application
Not standardized
Might become candidates for future OMG standardization
CORBA Object Implementation

- Defines operations that implement a CORBA IDL interface
- Object implementations can be written in a variety of languages
  - C, C++, Java, Smalltalk, Ada, etc.
Program that invokes an operation on an object implementation
- Accessing the services of a remote object is transparent to the caller
  - As simple as calling a method on an object
  - e.g., obj -> op(args)

**CORBA Client**

- Provides mechanism for transparently communicating client requests to target object implementations
- Simplifies distributed programming by decoupling the client from the details of the method invocations
- Client requests appear to be local procedure calls
- ORB is responsible for finding the object implementation, activating it, delivering the request to the object, and returning a response to the caller
ORB Interface

- Abstract interface for an ORB
- ORB may be implemented in various ways
  - One or more processes, set of libraries, etc.
- Interface provides various helper functions
  - Converting object references to strings
  - Creating argument lists for requests made through
    the dynamic invocation interface

CORBA IDL Stubs and Skeletons

- Serve as the “glue” between the client and server
  applications, and the ORB
- CORBA compiler automates the transformation
  between CORBA IDL definitions and the target
  programming language
- Use of compiler reduces the potential for
  inconsistencies between client stubs and server
  skeletons
- Use of compiler facilitates automated optimizations
**CORBA Dynamic Invocation Interface (DII)**

- Allows a client to directly access the underlying request mechanisms provided by an ORB
- Applications use the DII to dynamically issue requests to objects without requiring IDL interface-specific stubs to be linked in
- DII allows clients to make non-blocking deferred synchronous (separate send and receive operations) and oneway (send-only) calls

**CORBA Dynamic Skeleton Interface (DSI)**

- Server side analogue to the client side DII
- Allows an ORB to deliver requests to an object implementation that does not have compile-time knowledge of the type of the object it is implementing
- Client making the request has no idea whether the implementation is using type-specific IDL skeletons or dynamic skeletons
- Assists the ORB with delivering requests to the object
- Assists the ORB with activating the object
- Associates object implementation with the ORB
- Can be specialized to provide support for certain object implementation styles (e.g., OODB object adapters for persistence)

**CORBA/ORB Products**

- Oracle BEA WLE (formerly BEA M3)
- Expertsoft CORBAplus
- IBM WebSphere Product Family (Component Broker)
- Microfocus VisiBroker Middleware Products
- IONA OrbixWeb
- Merant CORBA Technology for COBOL
- PeerLogic DAIS
- ObjectSpace Voyager
- etc.

CORBA Architecture

Sample Java CORBA Application
CORBA Development Process
(pre-CORBA3)

Coding CORBA Server with BOA ImplBase
(pre-CORBA 2.3)
Coding CORBA Server with BOA TIE
(pre-CORBA 2.3)

Coding CORBA Server with POA Inheritance
(post-CORBA 2.3)
Coding CORBA Server with POA Delegation (post-CORBA 2.3)

- A valuetype can support an interface
- The client can receive a local copy of the valuetype or a remote interface
Example:
module Test {
    valuetype OBV {
        private long amount;
        long getAmount();
        init(in long newAmount);
    };
    interface ByValue {
        OBV getOBV();
    };
};
Example:

```idl
module SimpleStocks {
    interface StockMarket {
        float get_price in string symbol;
    }
};
```

Example:

```java
package SimpleStocks;
import java.rmi.*;
import java.util.*;

public interface StockMarket extends java.rmi.Remote {
    float get_price( String symbol ) throws RemoteException;
}
```
Example:

```idl
[ uuid(7371a240-2e51-11d0-b4c1-444553540000),
  version(1.0)
] library SimpleStocks
{ importlib("stdole32.tlb");
  [ uuid(BC4C0AB0-5A45-11d2-99C5-00A02414C655),
    dual
  ]
  interface IStockMarket : IDispatch
  { HRESULT get_price([in] BSTR p1, [out, retval] float * rtn);
  }
```

Example:

```idl
[ uuid(BC4C0AB3-5A45-11d2-99C5-00A02414C655),
] coclass StockMarket
{ interface IStockMarket;
};
```
Implementation of an IIOP Bridge

ORB Interoperability (1/2)
ORB Interoperability (2/2)

CORBA Interoperability

- Stubs/Skeleton
- GIOP
- IIOP
- TCP
- FP
- SNA

ORB Web Communication

- HTTP/1.1:
  - http://www.w3.org/Protocols/rfc2616/rfc2616.html
- IIOP tunneling
  - IIOP over HTTP
  - Sample Implementation
    - VisiBroker’s Gatekeeper
Summary – Key Application Server Objectives

- CORBA, RMI/JRMP, and DCOM were historically developed as separate Distributed Object Computing Platforms
- CORBA and RMI-IIOP were made compatible at the protocol layer
- Sun JavaSoft implemented JavaIDL to provide compatibility between CORBA and RMI-IIOP at the application level
- Programming applications on top of the various platforms requires different knowledge and skills
- The various platforms can interoperate using protocol bridges
- All DOC platforms are web-enabled
Class Project

- Project Description
  > The project focus is two-fold:
  - Based on a framework-based enterprise application of your choice, you will implement common facilities and application-level services on top of various types of application server technologies to support the various aspects of your chosen application.
  - As you transition from one platform to another you will study and develop model-based migration and interoperability tools that leverage off of the cutting-edge concepts subsumed by modern Model Driven Architectures (MDAs).

Assignments & Readings

- Readings
  - Handouts posted on the course Web site
  - Explore the Microfocus VisiBroker, Progress Orbacus, Oracle/Sun RMI, Oracle/Sun RMI-IIOP, and Microsoft DCOM Environments
  - Read white papers/documentation at:
    - Read Microfocus VisiBroker, Progress Orbacus, Oracle/Sun RMI, Oracle/Sun RMI-IIOP, and Microsoft DCOM related whitepapers on the vendor sites
Assignment:

1. Investigate distributed object computing platforms’ development environments for the technologies covered in this session. Write a short report that documents your findings and recommendations with respect to selection criteria in support of development environments for application server technologies covered in this session.

2. See homework #2 specification

Next Session: J2EE and .Net OMA Implementations