Build Distributed Object-Oriented Applications with .NET

Las Vegas
May 2005

Rockford Lhotka
Principal Technology Evangelist
Magenic Technologies

Prerequisites for presentation:
I assume you know: 1) VB.NET or C#; 2) OO concepts; 3) data concepts
Level: Intermediate to Advanced

Agenda
- Distributed Object-Oriented Architecture
- Business Objects
- Object-Oriented Design (for BOs)
- Object Persistence

Distributed Object-Oriented Architecture
- Distributed
  - Physical n-tier (client/server, web, etc)
- Object-Oriented
  - Objects defined by behavior, not data
- OLTP
- Best of both worlds
  - Benefits of distributed environments
  - Benefits of OOD/OOP

Application of Theory
- Windows Forms applications
- Web Forms applications
- Within services (SOA)
- Other

Business Objects
- Alias
  - Entity objects
  - Agents
  - Smart data containers
Data vs Objects

- **Good**
  - High functionality (filter/sort)
  - IDE integration, MS support

- **Bad**
  - Slow serialization (heavy)
  - Tie your future to ADO.NET
  - Business logic not encapsulated

DataSet/DataTable

- **Good**
  - High functionality (filter/sort)
  - IDE integration, MS support

- **Bad**
  - Slow serialization (heavy)
  - Tie your future to ADO.NET
  - Business logic not encapsulated

Data Transfer Objects

- **Good**
  - Avoid tie to ADO.NET
  - Fast serialization

- **Bad**
  - Poor functionality (no filter, databind)
  - Poor IDE support (today)
  - Business logic not encapsulated

Business Objects

- **Good**
  - Avoid tie to ADO.NET
  - Fast serialization
  - Business logic encapsulated

- **Bad**
  - Poor functionality (no filter, databind)
  - Poor IDE support (today)

What’s in an Object?

- **Business logic**
  - Validation
  - Calculation
  - Manipulation
  - Authorization

- **Data or state**
  - Direct state
  - “Borrowed” fields

Supporting the UI

- Data binding
- N-level undo
- Broken rule tracking
**Business Object Framework**

**Code Regions**
- Business properties and methods
- System.Object overrides

**Centralization of Business Rules**
- Data-centric worldview

**Managing Business Logic**
- BrokenRules
- Rules Manager

**Demo:** Creating a basic business object
Centralization of Business Rules

Properties
- Constructor
- Data retrieval
- Data storage

Code Regions
- Business rules

Demo:
Implementing business logic

Object-Oriented Design
- Objects are defined by behavior, not data
  - Simple root objects
  - Composite objects
    - Child objects, child collections
    - Root-level collections
  - Read-only objects
  - NameValue lists
  - Command objects

Root Objects
- Primary behavior: Editable object
- Simple or Collection
- Have behavior and state
- Support all concepts
  - N-level undo
  - Data binding
  - Object persistence
- Requirements
  - Must be serializable
  - Must inherit from CSLA base class
Child Objects

- Primary behavior: Editable object
- Objects contained by another object
- Have behavior and state
- Support all concepts except persistence
  - Persistence is controlled by parent object

Requirements
- Must be Serializable
- Must inherit from CSLA base class
- Must call MarkAsChild

Composite Objects

- Objects that contain other objects
- Only root object manages persistence
  - All other objects are child objects
  - This is a form of transactional relationship

Read-Only Objects

- Primary behavior: Provide read-only data
- Simple or Collection
- Have behavior and state
- Support persistence (read-only)
- Support data binding

Requirements
- Must be serializable
- Must inherit from ReadOnly CSLA base class

NameValue Lists

- Primary behavior: Provide read-only data
- Collection of name/value pairs
- Have state and fixed behavior
- Support persistence (read-only)

Requirements
- Must be serializable
- Must inherit from CSLA base class

Command Objects

- Primary behavior: Execute server-side code
- Have behavior and state
- Support persistence
  - May update or retrieve data on server
  - Typically private to a business object

Requirements
- Must be serializable
- Must implement DataPortal_Update method

Logical vs Physical Architecture
There's a huge difference between logical and physical n-tier architecture.

**Logical Architecture**

- Presentation
- UI
- Business Logic
- Data Access
- Data Storage and Management

**Logical Layers**

- Breaks application into logical layers
  - Layers may all run on a single machine
  - Layers may run on separate machines
- Provides
  - Maintainability
  - Readability
  - Flexibility
  - Reuse
  - Lower development/maintenance costs

**Physical Tiers**

- Deploy logical layers to physical tiers
- Like creating a multi-threaded app, where the threads are on separate machines
- Trade off between
  - Performance
  - Scalability
  - Fault tolerance
  - Security

**Web App**

- Presentation
- UI
- Business Logic
- Data Access
- Data Storage and Management

**Web App (secure)**

- Presentation
- UI
- Business Logic
- Data Access
- Data Storage and Management
Object Persistence

- Three models
  - UI in charge
  - Object in charge
  - Class in charge

- Distributed environments require serialization

Presentation/Data Storage

- Presentation
- UI
- Business Logic
- Data Access
- Data Storage and Management

Data-Centric (Fat UI)

- UI
- Business Logic
- Data Access

Data-Centric ("thin" UI)

- UI
- Business Logic
- Data Access

Data-Centric (MTS/COM+)

- UI
- Business Logic
- Data Access

Distributed Objects (COM+)

- UI
- Business Logic
- Data Access
Distributed OO (CSLA)

- UI
- Business Logic
- Data Access

Dealing with Serialization

- Two types
  - XmlSerializer
    - Can't create total clones of objects
    - Can only deal with a subset of .NET types
    - Has high flexibility to deal with schema changes
  - BinaryFormatter/SoapFormatter
    - Can create total clones of objects
    - Can deal with any <Serializable()> .NET type
    - Vulnerable to schema changes

Binary/Soap Formatter Issues

- Difficulty adapting to new assembly version

```csharp
Dim t As New System.IO.FileStream(FileName, FileMode.Create, FileAccess.Write, FileShare.Read)
Try
    f.Serialize(t, theObject)
Finally
    t.Close()
End Try
```

- No notification of serialization/deserialization
- One option – create an interface
  - Our objects must implement the interface
  - Any code that causes serialization must call the interface

```csharp
Public Interface ISerializationNotification
    Sub Serializing()
    Sub Serialized()
    Sub Deserialized()
End Interface
```

- Schema changes break deserialization
- DO NOT use this type of serialization for persistence
- Use this type of serialization for transient data transfers
  - Moving an object across the network
  - Storing an object in ASP.NET Session

Network Transport Options

<table>
<thead>
<tr>
<th></th>
<th>Web Svc</th>
<th>DCOM</th>
<th>Remote</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data-centric</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Object-oriented</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>XML/SOAP</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Binary</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Pass thru firewall</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>IDE support</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Code controlled</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Peer to peer</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Events/callbacks</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Stateful objects</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>
Where do Web Services Fit?

Provider | Consumer
--- | ---
Web Service | Presentation
Business Logic | Business Logic
Data Access | Data Access
Data Storage | Data Storage

Where does DCOM fit?

Application
Presentation
Business Logic
Data Services
Data Management

Where does Remoting fit?

Application
Presentation
Business Logic
Remoting
Data Services
Data Management

Basic Models

- "UI in charge"
  - UI controls entire process
- "Object in charge"
  - Object controls process at UI request
- "Class in charge"
  - Class methods mediate between UI and object

"UI in Charge"

"UI in Charge"

- Advantages
  - Directly use .NET serialization
  - Objects are "truly" distributed
  - Business objects are simpler
- Disadvantages
  - More complex UI code
  - Less code reuse

Dim obj As TheClass
obj = myServer.Load(key)

Data-centric Service
**Object in Charge**

```
Dim obj As TheClass
obj.Load(key)
```

**Advantages**
- Simpler UI code
- Easier reuse of objects
- More abstract model

**Disadvantages**
- Can’t directly use .NET serialization
- Objects include data access code

---

**Class in Charge**

```
Dim obj As TheClass
obj = TheClass.Load(key)
```

**Advantages**
- Directly use .NET serialization
- Simpler UI code
- Objects are ‘truly’ distributed
- Easier reuse of objects
- More abstract model

**Disadvantages**
- Objects include data access code

---

**DataPortal Concept**

- Supports “Class in Charge” model
- RESTful programming
- Object serialization
  - Anchored vs unanchored objects
- Enables Distributed OO

---

**DataPortal Framework**

[Diagram showing DataPortal Framework]
If the DataPortal is designed right you don’t need to worry about any of these details.

- All data access code goes in the object
  - DataPortal_Fetch
  - DataPortal_Update
  - DataPortal_Delete
  - DataPortal_Create

**Code Regions**

- Factory methods
- Constructors
- Criteria
- Data access

**Demo:**

Implementing basic data access

**DataPortal Design**


Server.ServicedDataPortal calls back into the object

**Enterprise Services (COM+) Contexts**
Making this work with COM+

- AppDomain
- DataPortal
- Business DLL

Demo: Using Enterprise Services

ORM

- Relational vs Object models
- Multi-database
- Inheritance

OO-Relational Mapping

Handling Multiple Data Sources

Mapping to Complex Objects
**Concurrency**

- Pessimistic
- Optimistic
  - Last write wins
  - First write wins
- First write wins
  - Timestamp
  - CRC
  - Field comparison

**Command Objects**

- Used to execute server-side code
- Should fit naturally into the object model
  - Typically private objects in a business object
  - Business object exposes shared or instance method
- Requirements
  - Must be Serializable
  - Must implement DataPortal_Update

**Multi-Client Coordination**

- Concurrency makes bad user experience
  - Users want to know they are wasting time
- Notification service
  - Smart client only (no async notification on web)
  - Notify one client when another does action
  - Publish/subscribe model
Change Notification

- How does one client know when another client changed its data?

![Diagram showing client, client, client, client, notification server, application server, and server connections]

Demo: Notification server

- Reports vs Lists
- Reporting tools for Reports
- ObjectAdapter for Lists

Demo: Reporting

Batch Processing

- Objects designed around behavior, not data
- Solutions
  - Use stored procedures
  - FlyWeight design pattern

Web Services

- Web services
  - Architectural considerations
  - Versioning
  - SOA vs API vs components-on-the-web
Where do Web services fit?

- Application 1
  - Presentation
  - Business Logic
  - Data Services
  - Data Management

- XML Web Services

Application 2
- Presentation
- Business Logic
- Data Services
- Data Management

Web Services and SOA

- Separation of implementation/interface
- Implementation
  - Our business objects
- Interface
  - XML schema
    - May be defined by VB or C# classes or XSD
    - Ultimately exposed via WSDL

Web Services and OO Apps

- Two viewpoints
  - Web Service creator
  - Web Service consumer
- Creator or Host
  - Application that exposes web services for others
- Consumer or Client
  - Application that calls web services

Where do Web Services Fit?

Provider
- Web Service
- Business Logic
- Data Access
- Data Storage

Consumer
- Presentation
- Business Logic
- Data Access
- Data Storage

Service Host

- Presentation
- UI
- Business Logic
- Data Access
- Data Storage and Management

Service Client

- Presentation
- UI
- Business Logic
- Service Façade (Data Access)
- Service Host (Data Management)
For more information

- www.lhotka.net/cslanet
- www.lhotka.net/go/vslive05.aspx
- Design Patterns (Gamma, et al)
- Patterns of Enterprise Application Architecture (Fowler)
- Metropolis (Pat Helland's SOA presentation)

Thank you!

Rockford Lhotka
rocky@lhotka.net
http://www.lhotka.net

Nobody's more serious about Microsoft.