Is SOA Just N-Tier in Other Clothing?

San Francisco
February 2005

Rockford Lhotka
Principal Technology Evangelist
Magenic Technologies
Level: Intermediate

Rockford Lhotka
Magenic Technologies
Principal Technology Evangelist

Service-Oriented Design

- 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

What Gets Transferred?

- Parameters?
  - SO as distributed procedures
  - SO as distributed components
- Object state?
  - SO as distributed agents or mobile objects
- Idempotent messages?
  - SO as a message-based model
  - Can serve any of the models above

Active vs Passive Data

- Active data
  - Data travels with semantic meaning and context
  - Fits with distributed agent/mobile object model
- Passive data
  - Data travels with no semantic meaning
  - At best, only syntactic metadata is included
  - Fits with procedural and component models

Host and Client

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

Client
- Presentation
- Business Logic
- Data Access
- Data Storage

Nobody's more serious about Microsoft.

Microsoft
Software Legend

VS Live! Nobody’s more serious about Microsoft.
**DataSet/DataTable**

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

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

**Data Transfer Objects**

- **Good**
  - Platform independent
  - 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**
  - Not platform independent
  - Poor functionality (no filter, databind)
  - Poor IDE support (today)

**SO as Distributed Procedures**

**or**

**SO as Distributed Components**

In short, what if services are just RPC with angle brackets?

**Services as RPC**

- Implies basically synchronous activity
- XML adds interop to the RPC story
- Apply lessons from pure RPC
  - Chunky communication
  - API design
- Apply lessons from MTS/COM+/J2EE
  - Chunky communication
  - Component (API) design
  - Transaction management

**SO as Mobile Objects**

Can distributed agents play in an SO world?
SO and Mobile Objects

- Mobile objects require a transport
  - XML is as good as anything (if bulky and slow)
  - Web services, queues, DCOM, RMI are all fine
- State transformations bad
  - SO allows state transformations
- “Messages” are objects, not data
  - Typically not idempotent
  - Typically not intended for bus architecture

SO as a Message-Based Model

What does this mean anyway?

SO and Message-Based Models

- Messages are response/request data
  - Response/request data is idempotent
  - Different from resource, activity or reference data
- Synchronous or asynchronous delivery OK
- Message could be packed parameter data
  - Enables procedural or component models
  - Asynchronous aspect adds complexity
- Message could be packed object data
  - Enables mobile object model
  - Data transformation/routing adds complexity

The Prevalent Model

- Distributed components
- Communication is via RPC with angle brackets
- Design is component-oriented
  - Very comparable to MTS/COM+/J2EE
- This is the default Indigo model

The Runner-Up

- Distributed procedures
- Communication is via messages
- Design is procedural and asynchronous
  - Distributed parallel computing
  - Atomic, stateless procedures act on messages
- Possible with Indigo, but not the default

Tiers, Layers and Services, Oh My!
**Tiers, Layers and Services**

- N-Tier systems have layers and tiers
  - Layers are logical separation of responsibility
  - Tiers are physical deployment of layers
- What about SO?
  - Service = Layer: services are logical separations
  - Services can have internal layers
  - Internal layers can be deployed into tiers

**Logical Layers**

- Breaks application/service into logical layers
  - Layers may all run on 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

**N-Layer Logical Architecture**

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

**Service Host**

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

**Service Host (secure)**

- **Presentation**
- **UI**
- **Business Logic**
- **Data Access**
- **Data Storage and Management**
Imagine There’s No N-Tier

- A tier is a service
  - A service can have layers
  - A service can *not* have tiers *(by definition, since a tier is another service)*
- Any network boundary is a service boundary

**Downsides**
- Business logic duplication
- Potential network overhead
Traditional N-Tier App Sans Tiers

Windows GUI
- Presentation
- UI
- Business Logic
- Service Facade (Data Access)
- Business Engine
- UI
- Business Logic
- Service Facade (Data Access)
- Persistence Engine
- UI
- Business Logic
- Data Access
- Data Storage and Management

The Point Being...

- N-Tier and SO are \textit{not the same}
  - They are similar
  - But not identical
- N-Tier
  - Designed for optimized behavior \textit{within} an app
  - Minimize redundant code, maximize performance
- SO
  - Designed for reuse of high-level functionality
  - Minimize redundant functionality, maximize interop

For more information

- 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