Event Driven Architecture with POJOs and ActiveMQ

Bruce Snyder
LogicBlaze
Who Is This Guy?

- 10+ years of software development
  - Service Oriented Architecture
  - Event Driven Architecture
  - J2EE
  - Middleware
  - Relational Databases
Open Source Leader

- Apache
- Geronimo
- ServiceMix
- ActiveMQ
- Castor
Book Author

Professional
Apache
Geronimo
Agenda

- Event Driven Architecture (EDA)
- Message Oriented Middleware (MOM)
- Java Message Service (JMS)
- Messaging Crash Course
- Apache ActiveMQ
- Message Driven Beans (MDBs)
- Message Driven POJOs (MDPs)
- Lingo
Event Driven Architecture

- Most systems use command-and-control scheme
  - One method calls another
- EDA is about handling events as they occur
  - Events are transmitted between loosely coupled systems
What Are Events?

- A significant change in state
- Examples
  - Keyless auto lock/unlock
  - Bar code scanned at grocery store
  - Motion sensor on a security system
  - Temperature changes beyond a threshold
Characteristics of EDA

- Events are broadcast
- Events are timely
- Communications are asynchronous
- Events are fine-grained
- Complex event processing
EDA Significance

- Caller is not aware of callee
- Statefulness is dramatically reduced or eliminated
- Events no longer take place one at a time
- Order of events is no longer important
Message Oriented Middleware

- Middleware to facilitate messaging in a distributed architecture
- Provides message handling or queuing
- Popular for integrating disparate systems
- Examples
  - IBM MQSeries (WebSphereMQ)
  - TIBCO
  - Apache ActiveMQ
Benefits of MOM

- Distributed
- Loosely coupled
- Platform and language independent
- Reliable
- Transactional
JMS - A MOM Standard

- A specification for messaging
  - Vendor neutral

- Features
  - Point-to-Point - One-to-one (Queues)
  - Publish/Subscribe - One-to-many (Topics)
  - Reliable - Guaranteed message delivery
  - Transactional - Think ACID
  - Message metadata is important
    - Headers and properties

Message Persistence
Messaging Crash Course

- The loosely coupled exchange of messages
  - Producers and consumers are not aware of one another
  - Communication is indirect via destinations
- Synchronous or asynchronous message delivery
- Optional features
  - Durability
  - Persistence
Messaging Domains

- Point-to-Point
  - One-to-one
- Publish-Subscribe
  - One-to-many
Point-to-Point Messaging

Domain

- Based on queues
- One consumer per message
- Messages sent to queues
  - Delivered once and only once
  - A message is delivered to a single consumer
- Queues retain messages
  - No timing dependency
  - Messages held until consumed or expired
Advantages of PTP Messaging

- Ideal for load balancing messages across consumers
  - Reliable load balancing with optional persistence
- Monitoring queues is relatively easy
  - Queue size and throughput
- Redelivery of messages
- Handle partial outages gracefully
  - Think system maintenance outages
Publish-Subscribe Messaging

Domain

- Based on topics
- Publishers and subscribers
  - Publisher had no knowledge of subscribers
- Messages are delivered to all subscribers
Publish-Subscribe Messaging Domain
Advantages of Pub-Sub Messaging

- Ideal for publishing business events
  - Only interested parties receive messages
  - Distributed observer pattern
- Facilitates Event Driven Architecture
  - Not the traditional command-control scheme of application development
Apache ActiveMQ

- JMS 1.1 compliant
- Integration with:
  - Geronimo, Spring, Tomcat, JBoss and any J2EE 1.4 container (e.g., WebLogic or WebSphere)
- Supported transports
  - TCP, UDP, multicast, SSL, HTTP, Jabber (XMPP), JXTA, etc.
- Additional capabilities
  - Pluggable persistence and security
Apache ActiveMQ

- Fast and highly scalable
  - Clustering, peer-to-peer, federated network support
- Many points of connectivity
  - Java, C/C++, .NET, Ruby, Perl, PHP, Python
- Easy to Use
  - Minimal configuration
- Open Source
  - Apache License 2.0
Apache ActiveMQ Architecture

- Pluggable broker and client
  - Transports
  - Wire formats
  - Persistence
  - Security
Transports

- Low Level Transports
  - VM
  - TCP
  - UDP
  - multicast
  - SSL
  - HTTP
  - Jabber (XMPP)
  - JXTA

- Composite Transport
  - failover:
  - discovery:
  - peer:
Transports Used By

- Client to broker communication
  - The `<transportConnector>` element
- Broker to broker communication
  - The `<networkConnector>` element
Wire Formats

- OpenWire
  - Binary
  - Native clients provided for C/C++, Java, .NET
  - Small and fast

- STOMP
  - Text
  - Clients for Perl, PHP, Python, Ruby, Pike
  - Support for connection, security, send, receive, subscribe, unsubscribe, message ack, transactions, JMS headers
Persistence

- High performance journal
  - Used for short-term persistence
  - Checkpoints to long-term storage (JDBC)

- JDBC provider
  - Defaults to embedded Apache Derby
  - Capable of using any RDBMS with JDBC drivers
    - E.g., MySQL, PostgreSQL, Oracle, DB2, Sybase, etc.

- Kaha provider
  - Complete persistence without JDBC
Security

- Authentication
  - JAAS based provider
- Authorization
  - Operations associated with roles
  - Supports wildcards
- Message level authorization
  - Based on pluggable policies
Network of Brokers

- Federated network of brokers
  - Many brokers acting as a single, logical broker
  - Use network connectors between each other
  - Store and forward strategy
  - Brokers use static or discovery based routing
  - Clients can also use static or discovery based routing
Master/Slave

- Fully replicated
- Shared filesystem (SAN)
- Shared database
Fully Replicated Master/Slave HA

- A logical broker is created from a pair of physical brokers
- All messages replicated to both physical brokers
  - Allows for automatic failover from master to slave
- Avoids loss of messages in the event of catastrophic hardware failure
Shared Filesystem Master/Slave

- Utilizes a directory on a shared filesystem
- No restriction on number of brokers
- Simple configuration
- One master selected at random
Shared Database Master/Slave

- Recommended when using a shared database
- No restriction on the number of brokers
- Simple configuration
- Clustered database negates single point of failure
- One master selected at random
ActiveMQ Advanced Features

- Wildcards on subscriptions
- Selectors
- Virtual destinations
- Exclusive consumers
- Message groups
- Total ordering
- Consumer priority
- Retroactive consumer
Configuration

- Java based configuration for JMS client/broker
- URI configuration mechanism
  - vm://localhost?broker.persistent=false
  - tcp://localhost:61616?jms.useAsyncSend=true
- JNDI provider
  - Dynamic lookup of pre-configured destinations
- Pure Spring based XML config
- POJO based config
- Extended Spring based XML config using
Advantages of ActiveMQ

- Enterprise ready
- Commercially supported
  - http://logicblaze.com/
- Open source
  - Apache License 2.0
- Widely deployed
What Are MDBs?

- Java objects for handling messaging
- Part of EJB 2.0 spec
- Provides asynchronous message processing
- Stateless
- Server-side components
- Transaction-aware
- Concurrent message handling
- No interfaces
- Not accessible via RMI
Disadvantages of MDBs

- *Require EJB container*
- EJB APIs are very static
- EJB lifecycle is very heavy
- Each MDB listens to only a single destination
- Not otherwise useful
Message Driven POJOs (MDPs)

- Plain Old Java Objects
- No knowledge of JMS
- Universally usable
- Developer need not worry about JMS
What Is Lingo?

- Spring remoting with async calls
- Spring remoting for POJOs via JMS
  - Spring remoting uses dynamic proxies
    - J2SE via `java.lang.reflect.Proxy`
    - Or via CGLib ([http://cglib.sf.net/](http://cglib.sf.net/))
Message Exchange Patterns

- Lingo supports various message exchange patterns (MEPs)
  - Synchronous request-response
  - Asynchronous request-response
  - One-way messaging
  - Asynchronous consumption
Request-Response

- Uses the JMS semantics under the hood
  - JMSReplyTo property
  - Temporary queues for PTP messaging
  - JMSCorrelationID property
- Avoids creating/destroying consumer per request
- Ability to share the same consumer and producer across many threads
One Way Invocations

- Methods defined as async, one way
  - Via the MetadataStrategy class
  - Default strategy
    - Void methods that do not throw checked exceptions
Asynchronous Consumption

- Similar to a JMS subscribe
  - Register interest and wait for messages
Remote Callbacks

- Register remote callbacks from the client side to be invoked asynchronously

```java
ServerService serverService =
    (ServerService) ctx.getBean("serverService");

FooCallbackImpl callback = ctx.getBean("fooCallback");
serverService.registerCallback(callback);
serverService.doWork();
```
Why Use Lingo?

- You need an async API
- You need a remote API
- Not interested in custom thick client
- Exposing JMS API is not appropriate
J2EE Connector Architecture

- Lingo is compatible with Jencks
  - Lightweight JCA container
  - http://jencks.org/
  - Pooling of threads, resources,
  - Provides parallel processing of inbound messages
  - Supports inbound and outbound messaging
    - JMS, JAX-RPC, JBI, JCA CCI and more
  - Provides XA based pooling for JDBC
Hurdles to Adoption

- Developers think in terms of command-control instead of events
- Many developers have little knowledge of messaging
- Developers tend to care only about code
Summary

- Event Driven Architecture (EDA)
- Message Oriented Middleware (MOM)
- Java Message Service (JMS)
- Messaging Crash Course
- Apache ActiveMQ
- Message Driven Beans (MDBs)
- Message Driven POJOs (MDPs)

Lingo
Wrap Up

- Open Source ESB (JBI)
  - http://servicemix.org/

- Open Source JMS
  - http://activemq.org/

- Lingo
  - http://lingo.codehaus.org/

- Support and Services
  - http://logicblaze.com/
Thank You for Attending

- Questions and answers
- Please fill out your surveys