Enterprise Integration

Enterprise Service Bus
Java Message Service

Presented By
Ian McNaney

University of Colorado at Boulder

Motivation

- Enterprise context
- Many different systems
  - Varying ages
  - Varying technologies
  - Varying business owners
- They need to communicate
  - Billing to fulfillment
  - Fulfillment to inventory
  - Just about everything to accounting

Motivation

- Organizations and their offerings develop over time
- As do their software systems and environment
  - Corporate mergers
  - Acquisitions
  - Changing development methodologies over time
  - New products and systems to back them

Legacy Systems

- A polite term for systems nobody dares touch
  - Too big or risky to replace, too important to discard
  - Maybe the last person who knew anything about it quit four years ago
- The decision has been made to leave it alone
- It still has to receive a data feed from the new billing system somehow

Direct Communication Problems

- Brooks' Law - More systems leads to more potential communication channels
- Different systems can be based on
  - Different architectures – endianness issues
  - Different languages – makes rpc and serialization/deserialization tricky
- Reliable delivery issues
  - What if the other system is down or there's network disruption?

Direct Communication Problems

- Modifying systems to handle these issues leads to a large amount of code that is
  - Unrelated to the business logic
  - Repeated in each system

Direct Communication Problems

- Many systems cannot (or will not) be modified to directly communicate with other systems
  - Closed source 3rd party apps
  - Legacy systems
- Even if we could modify each system for direct communication, do we want to?
  - New release cycle for every communicating system whenever its communication partners change

A Solution

- Message Oriented Middleware (MOM)
  - Each application sends and receives through an intermediary
  - Separate system responsible for reliable delivery, message routing, etc.
- Systems can focus on what they do, allowing the complexity of communication to be handled elsewhere
- Consolidate communication configuration in one place

Other Options

- Won't be presented, but other options exist
  - Standardize on one language and use its RPC mechanism
    - Still need to worry about reliability and retry
  - Web Services with lookup
    - Can be thought of as a subset of MOM
    - Lookup resolves the destination service
    - HTTP with retry provides the transport and some degree of reliability

JMS and ESB

- I'll talk about
  - Java Message Service (JMS) as a foundation for an...
  - Enterprise Service Bus (ESB)

JMS

- Java Message Service 1.1
- Defines a messaging API and a set of behaviors, not a wire level protocol
- Different implementations are generally not compatible with one another at the wire level.
- Many implementations provide HTTP and/ or SOAP interfaces, so non-Java languages can play too.

A JMS Message

- Headers
  - Message metadata relevant to delivery – destination, reliability mode, TTL, message ID, etc.
- Properties
  - Custom defined additional headers – intended for use in filtering messages.
- Body
  - The content of the message.

JMS Message Body

- Several types are defined by the standard, many of which are Java-specific.
- Of interest to us in a heterogeneous environment are the non Java-specific types:
  - TextMessage – the payload is literally just text. Handy for XML. Recommended.
  - BytesMessage – the payload is a stream of raw bytes.

JMS Destinations

- JMS supports two types of message detinations
  - Queue – for point-to-point communication
  - Topic – for broadcast communication
- Every message must have a destination

JMS - P2P

- JMS Queues provide point-to-point (p2p) communication.
  - Each message is sent to one destination queue
  - Messages in the queue are delivered in order under normal circumstances
  - Messages are delivered to one consumer, but multiple consumers can consume from a single queue – automatic load balancing if order doesn't matter
  - Analogous to a post office box – messages pile up until they are consumed

JMS - Pub/Sub

- JMS Topics provide for broadcast to multiple consumers
  - Publish/Subscribe
  - Consumers establish a subscription to the topic
  - Each subscriber receives a copy of each message that arrives on the topic
  - Disconnected clients miss messages unless a Durable Subscription is established
  - Analogous to a radio station – a consumer is either listening or it isn't

JMS - Reliable Delivery

- Reliable delivery can be requested for messages sent to queues
  - Persistent – the JMS provider persists the message to disk so that it will be retained in case of a failure. Once-and-only-once.
  - Non-persistent – persistence to disk is not required, reducing messaging overhead. The message may be lost in case of a failure. At-most-once.

JMS - Message Ordering

- The oldest message is always delivered first, but this does not mean that messages are delivered in order that they arrived
  - Redelivery in the case of an unacknowledged message
    - Redelivered header will be set in this case
  - Clients can filter messages based on headers or properties, skipping over some older messages

Broker

- The running software instance responsible for managing queues and topics
- Keep in mind that no delivery guarantee is absolute
  - Physical destruction of the backing store after a broker outage will prevent delivery of any pending messages

Clustering / HA

- Brokers, being software on computers, go down for maintenance or due to hardware/ software faults.
- To improve uptime establish a high availability (HA) replicated cluster
  - Primary/Failover approach is common
    - Primary is “active” until a fault, replicating its state to the failover.
    - When the primary fails the failover becomes active and accepts connections

Failover

- Clients need to reconnect to the failover broker. Sometimes the JMS driver automatically handles this.
- Failed node's persisted state will become stale almost immediately, while the failover becomes “correct”
- Some implementations (SonicMQ) claim to re-synchronize state when the primary come back up.
- Others (ActiveMQ) require manual sync and full cluster restart.

Broker-Broker Connections

- Systems at separate, disconnected sites need to communicate.
- Brokers can establish an SSL tunnel between sites, or between different brokers on-site.
  - With network administrator cooperation
  - Will need to allow the inbound/outbound connections at the firewall
- Secure messaging pipe between sites
- Not exactly a cluster since queues/topics aren't shared, just remotely accessed.

Administrative Issues

- Slow/Offline consumers
  - Persisted messages consume system resources
  - Eventually resources become exhausted
  - All bets are then off – persistent messages may be dropped.
- JMS Broker crash/restart
  - Loss of non-persistent messages.
  - Clients need to detect and reconnect or they'll become offline consumers.

Non-Java Interoperability

- Many JMS providers allow for non-Java clients to produce or consume messages
  - HTTP post/get or Web Service interface
  - Monitor file system locations for messages to send, drop contents of a queue to a directory
- Others are not fundamentally Java-centric
  - IBM WebSphere MQ is decades old – JMS was bolted on later

Popular JMS Implementations

- Apache ActiveMQ
- Fuse (based on Apache ActiveMQ)
- Progress SonicMQ (closed source)
- IBM WebSphere MQ (closed source)
- JBoss JMS
- Oracle (closed source)
- Many J2EE application servers contain limited JMS functionality for communication between apps on the server.

Enterprise Service Bus

- Build on top of the reliable messaging services provided by JMS
- Provide additional capabilities
  - Content based routing
  - Message replication
  - Message transformation or translation
  - Synchronous or asynchronous invocation of external web services
  - And more...

What Do We Gain?

- Allows us to route messages along a specific processing path based on its details – polymorphism for messages
- A new subscription message may be handled differently than a cancellation message
- All without senders or receivers knowing or caring what happens in between

XML Messages

- Internally the ESB routes standardized XML messages
- At the edges, where clients connect, the message may
  - Already be in the canonical format
  - Be transformed from a language or platform specific format into the canonical format
- You don't **have** to use XML, but I highly recommend something like it

Content Based Routing

- The message is inspected by a service on the ESB
- It is dispatched to one or more destinations based on its type and content
- Configured within the ESB itself, often through a drag/drop GUI

Message Replication

- Create copies of a message and forward each to its own destination
- Useful when a known set of systems each need a separate copy
- Can also be accomplished with topics and durable subscriptions
  - That can be problematic for non-Java clients
  - Also problematic for buggy Java clients that don't reconnect with the same name

Message Transformation

- With XML messages this is simple
  - XSLT
  - Can call out to external services for portions of the data
- Usually only relevant at the edges of the bus, to translate into or out of the canonical format
- Also useful when converting from one message type to another somewhat related one

Invocation of Web Services

- Messages that arrive at a destination can be translated and forwarded on as HTTP requests to Web Services.
- The WS response can be forwarded on to the next step in the flow (synchronous)
- The WS can post back later via JMS or HTTP for asynchronous operation
- This came in handy for submitting orders to external vendors

Case Study - FindLaw

- Corporate business systems hosted in one data center
  - SAP
  - Order entry and tracking
  - Inventory control
- Web site and related products hosted in a separate data center
  - Web Site
  - Ads and fulfillment

Use Case – Purchase an Ad

- A lawyer calls the sales rep and works out a deal
- The rep enters the information into Tracker, an in-house inventory tracking application
- “Magic” happens
- SAP starts billing the customer according to the terms of sale
- Fulfillment systems activate the ad on the appropriate pages, and acknowledge having done so

Use Case – Purchase an Ad

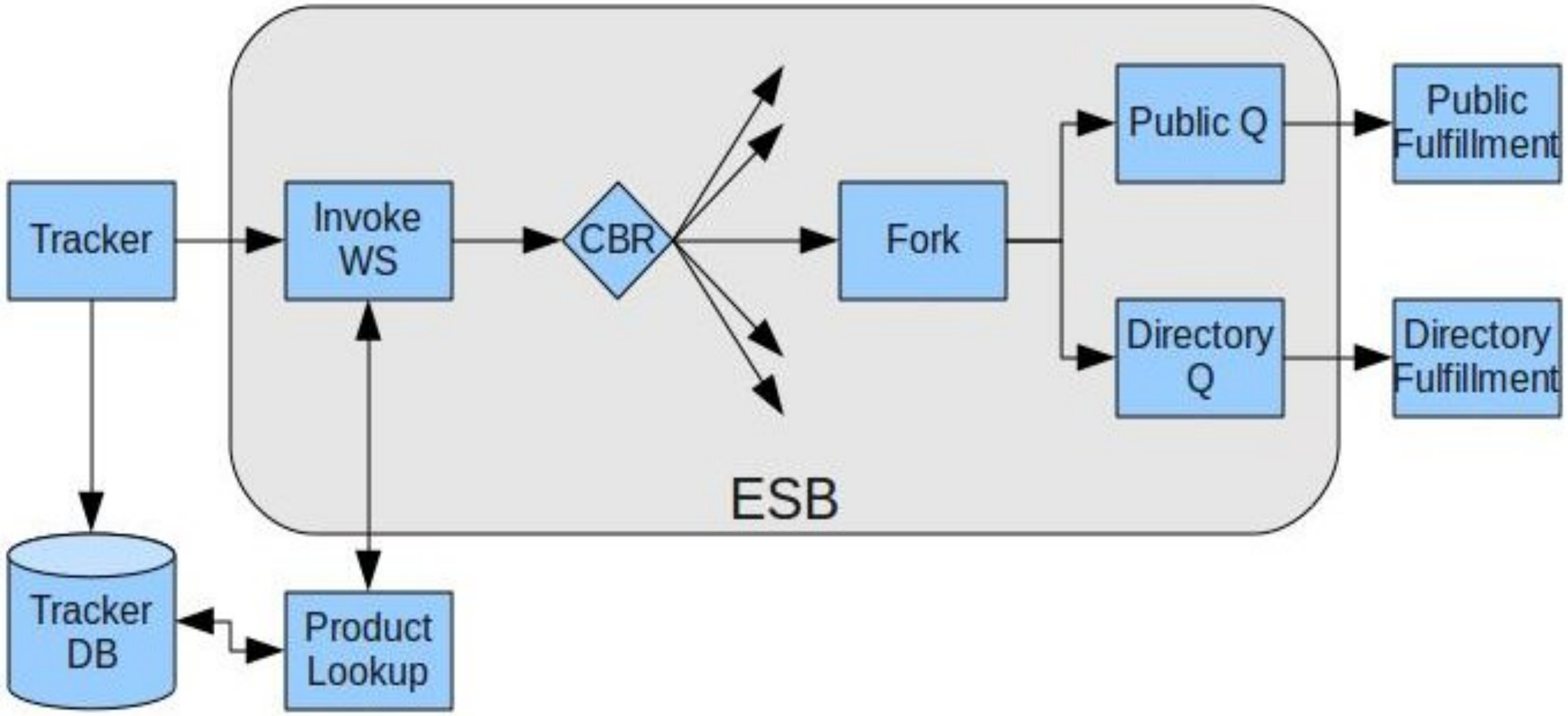
- Tracker to Fulfillment
  - When the rep presses “submit” in Tracker an empty shell of a “new inventory” message is sent to the bus.
  - This shell is routed to a synchronous web service by the ESB to retrieve order details from the Tracker DB and flesh out the internal XML message
  - The reply is forwarded to a content based routing service that routes it to the appropriate message flow for its product

Use Case – Purchase an Ad

- Example product message flow – new on all pages for a specific geographic region
  - Message is replicated and sent to a queue for each major component system of findlaw.com
    - Lawyer directory
    - Public channel
    - Etc.
  - Each system handles its message and returns an acknowledgement message

Use Case – Purchase an Ad

- Acknowledgement
  - All acknowledgement messages are of the same format, and are routed to the business systems data center
  - A JMS consumer consumes the message and updates the Tracker DB with details of which system fulfilled which products at which times.


Tracker
Invoke WS
CBR
Fork
Public Q
Public Fulfillment
Directory Q
Directory Fulfillment
ESB
Tracker DB
Product Lookup

Use Revision Control

- Always keep your current configuration under revision control.
- The last thing you want if the bus hardware fails catastrophically is to try to figure out what all the routes were and the destination names were while you're recovering everything else.

JNDI

- Use it
- In a properly implemented ESB it allows clients to identify themselves by name and have all of their connection parameters set appropriately
- Removes concern about destination names and connection settings from the application code
- Allows these settings to be updated dynamically, without re-releasing the app

Gotchas

- Watch out for slow/offline consumers
  - Will exhaust resources, disrupt reliable delivery
- A good diagramming tool is invaluable
- An ESB with auto-diagramming functionality is even better
  - Diagrams won't go out of date
- Durable Subscriptions can cause issues if clients don't properly reconnect
  - New DS for every name they connect with

Politics

- The overall system is much more flexible and manageable than individual apps directly connecting, but...
- It can be hard to convince development groups of this when they're focused on their own app, and it only has to talk to one other app.

Additional Reading

- Progress Software's Sonic ESB site has some good white papers
- *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions* by Gregor Hohpe and Bobby Woolf