How an Event Becomes a Message

Message busses, HornetQ, and the firehose

What is a “Message Bus”?

- A mechanism for sending arbitrary messages between different software components
- (Mostly) Independent of implementation language and transport protocol
- Can have a central broker, or a distributed topology

Queue vs. Topic

Queue

- Point-to-point
- Each message consumed exactly once
- Guaranteed delivery

Topic

- One-to-many
- Every subscriber has chance to consume message
- Potentially lossy (messages not re-delivered)

Queue ~~vs.~~ and Topic

- Queue is the most basic unit of communication
- Topics can be simulated as a collection of Queues connecting publishers and subscribers
- Clever combination of Queues and Topics can generate complex topologies

In Javaland...

JMS – Java Message Service

- A standardized API for interacting with message Queues and Topics
- Implemented by many Java libraries; lowest common denominator of functionality
- Provides mechanisms for getting resources, managing sessions, sending and receiving messages

JMS – Java Message Service

- Uses “dotted” notation for naming queues and topics (e.g. “jms.queue.myQueue” or “jms.topic.eventFeed”)
- Implements a “correlationId” and the ability to add filters to connections; this enables point-to-point-ish communication

JNDI – Java Naming and Directory Interface

- Ties names to resources (objects)
- Utilized by JMS to get references to queues and topics by name (e.g. “/queue/myQueue” or “/topic/eventFeed” naming style)
- Mostly transparent (but can be customized for larger/complex topologies)

HornetQ

- JMS Message Queue implementation from JBoss org
- Does not require a stand-alone broker
  - Ships with integration for Spring apps as well as JBoss AS
- Distributable, persistable, journalable, fast
- Secure w/permissions management

HornetQ

- Can connect directly (HornetQ “protocol”) or via JMS wrapper (included)
- Direct connections can only create Queues
- JMS Topics implemented by HornetQ as coordinated collection of Queues
- Configuration via XML (you were expecting something else?)

HornetQ

- Configuration
  - Queues/Topics – can be specified in `hornetq-jms.xml` or created on the fly
  - Connectors – objects used by the system to connect (as a client) to queues/topics
  - Acceptors – interfaces clients can use to connect to queues/topics
- Connections can be in VM or via any interface supported by Netty (TCP, HTTP, STOMP, STOMP-WS)

STOMP – Simple Text-Oriented Messaging Protocol

- Similar to HTTP, but allows for bi-directional messaging over stateful connections
- Supported by HornetQ, AcitveMQ, RabbitMQ, others
- Works over TCP or even WebSockets

Messaging in OMERO

Firehose – Event Feed

- Hook into `ome.security.basic.EventHandler` to send event messages at the point event logs are saved
- JSON of id, entity type, and action
- Feed available on the “/topic/eventFeed” or “jms.topic.eventFeed” topic

Firehose – Event Backhaul

- Endpoint for clients to get a span of events missed
- Same info as Event Feed, but not real-time
- Available on the “/queue/eventBackhaul” or “jms.queue.eventBackhaul” queue

Firehose – Client Library and More

- Python library (in progress) to consume Firehose, automatically handle reconnection/backhaul filling, scheduling, etc.
- Communication via STOMP (localhost only until security is implemented)
- Example CLI tool (`omero firehose`)

Questions?