
SUMMIT
JBoss
WORLD
PRESENTED BY RED HAT
LEARN. NETWORK.
EXPERIENCE OPEN SOURCE.
www.theredhatsummit.com

Does REST need middleware?

Bill Burke
Fellow, Red Hat

Speaker's Qualifications

- RESTEasy project lead
  - Fully certified JAX-RS implementation
- JAX-RS JSR member
  - Also served on EE 5 and EJB 3.0 committees
- JBoss contributor since 2001
  - Clustering, EJB, AOP
- Published author
  - Books, articles

Agenda

What does Enterprise SOA need from REST?

What's missing?

Some ideas on RESTful interfaces for middleware services

Just as many questions as answers…

What are the goals of SOA?

SOA Goals

- Reusable
- Interoperable
- Evolvable
  - Versioning
- Governable
  - Standards
  - Architectural Guidelines and Constraints
  - Predictable
- Scalable

What system has these properties?

The Web!

Can REST be applied to Enterprise SOA?

REST and Enterprise SOA

- SOAP tried to bring the Web to IT
  - It turned into just tunneling over HTTP with XML
  - Never really leveraged HTTP or the principles of the Web

REST and Enterprise SOA

- Enterprise SOA requires read-write applications
- Integration and coordination between many services
- Sometimes complex interactions

REST and Enterprise SOA

- REST really shines in read-only applications and has scaled easily and simply
- Mostly browser-based applications take advantage of REST
- RESTful Read-Write applications usually one-off simple client-server interactions
  - Most break the stateless property of REST

REST and Enterprise SOA

What does this mean?

- We are only at the initial stages of applying REST to Enterprise SOA
- Machine-based clients will have different requirements than browsers
- There's still a lot of kinks to work out

Can middleware fill in the blanks?

Messaging

Transactions

Workflow/BPM

Security

What's Missing?

Security?

- The Web runs pretty well on HTTPS
- Between basic, digest, and client cert, authentication protocols pretty solid
- OAuth provides mechanism to authorize third-parties
- OpenID provides decentralized authentication
- multipart/encrypt and multipart/signed for payload protection

What's Missing?

Messaging?

- Atom provides Publish/Subscribe patterns and forma
- Is it just another SOAP?
- There is no real solution for p2p. (queues, work management)

What's Missing?

Transactions?

- RESTafarians say that ACID transactions don't belong in a distributed system
- They just don't scale
- Transactions aren't RESTful (break stateless requirement)
- Can't avoid them sometimes
- What about compensations (do/undo)?
- Its is ***THE*** most common question asked in REST talks

What's Missing?

- Workflow/BPM?
  - Nothing really for coordination/orchestration
  - Is hypermedia enough to provide the "flow" apps need?

Red Hat driven REST Standardization Effort

- From the perspective of our open source projects and communities

Attempts to answer some of these questions

- RESTful interface for common middleware patterns
- Open Process (anybody can interact)
- Open Source IP

Specifications

- Transactions (2pc and compensation)
- Messaging (p2p and pub/sub)
- Workflow
- Caching

REST-*.org

Goals

- 80/20 - keep things simple to implement and use
- Use conneg to support vendor extensions and edge cases
- Publish additional links for vendor extensions
- Avoid payload formats like SOAP
- Leverage full HTTP

Let’s show some details…

REST-* Messaging

REST-* Messaging

Atom is text based (XML)

Not great for binary media types

Designed really for pub/sub (blogging), not queues.

Design really to be consumable by humans (through rendering)

No real guaranteed message delivery or message acknowledgement protocols

REST-* Messaging

Doesn't require a payload format for single messages

Leverage Atom for Link relationship/metadata

- Published via Link headers instead
- Easily allow binary formats

Leverage Atom format for batch text transfers

multipart/* + Link headers for binary batch transfers

Defines guaranteed messaging and acknowledgement protocols over HTTP

Supports Queueing

Reliance on Link headers

Define/publish links through an HTTP Header

Easy way to link contextual information and metadata

Allows us to avoid payload formats

Easier for “intermediaries” and generic services and frameworks to process

They don’t have to look into message body for links

```
Link: <http//example.com/messages/111>; rel="next";
                                        type=application/xml
```

Message Posting

Message Posting

Destination has two posting links

- post-message - simple factory pattern
- post-message-once - reliable posting pattern

Message Posting

```
Request:

POST /destinations/test HTTP/1.1
Host: example.com
Content-Type: application/whatever

<body>
```

Message Posting

```
Request:

POST /destinations/test HTTP/1.1
Host: example.com
Content-Type: application/whatever

<body>

Response:

HTTP/1.1 201 Created
Location: /destinations/test/messages/111
```

Post Once Exactly: Avoiding Duplicates

Empty POST to the *post-message-once* link

Returns a “create-next” link that is a one-off URL

If you POST more than once you get a 405 Not Allowed response

Reponse contains a new “create-next” link

Post Once Exactly: Avoiding Duplicates

```
Request:

POST /destination/test/messages
Host: example.com
```

Post Once Exactly: Avoiding Duplicates

```
Request:

POST /destination/test/messages
Host: example.com

Response:

HTTP/1.1 200 Ok
Link:
     <http://example.com/destination/test/messages/111>;
         rel=create-next
```

Post Once Exactly: Avoiding Duplicates

```
Request:

POST /destination/test/messages/111
Host: example.com
Content-Type: application/json

[SomeJsonMessage]
```

Post Once Exactly: Avoiding Duplicates

```
Request:

POST /destination/test/messages/111
Host: example.com
Content-Type: application/json

[SomeJsonMessage]

Response:

HTTP/1.1 200 Ok
Link: <http://example.com/destination/test/messages/112>
          rel=create-next
```

Message Posting

- Specification also describes similar batch submission of messages
- Different posting protocols encapsulated as links published by the destination

Messaging Consuming: Topics

Pull Model

Messaging Consume: Pull model

Client pulls published messages from the destination

Atom *first*, *last*, and *next* links reused through published link headers

Clients are responsible for “bookmarking” their place in the topic/subscription

Message Consuming: Find Links

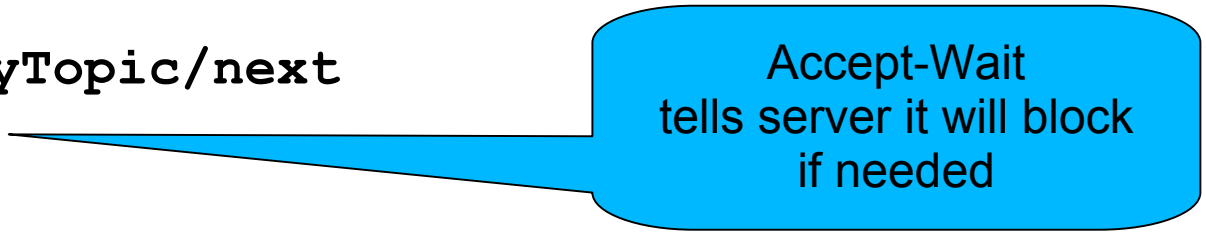
```
Request:
HEAD /destination/myTopic

Response:
HTTP/1.1 200 Ok
Link: <…/last>; rel="last",
      <…/next>; rel="next",
      <…/first>; rel="first"
```

Message Consuming: Pull Message

```
Request:
GET /destination/myTopic/next

Response:
HTTP/1.1 503 Service Not Available
Retry-After: 5
```

Message Consuming: Pull Message

```
Request:
GET /destination/myTopic/next
Accept-Wait: 100

Response:
HTTP/1.1 200 OK
Link: <…/messages/222>; rel="next",
      <…/messages/111>; rel="self"
Content-Type: application/json

[some posted JSON message]
```



Message Consuming: Pull

A bookmarked next link allows client to have a placeholder into the topic

In many MOMs, like JMS, this information is stored in a session on the server

The next link pattern allows any number of client to receive a sequenced ordering of messages in a lightweight manner

Messaging Consuming: Topics

Push Model

Message Consuming: Push Model

- Client registers a atom:link with provider when creating a push subscription
- Link defines forwarding semantics
  - Simple post?
  - Post once exactly?
- When message is published into topic or queue, server forwards request based on registered link semantics

Push model

```
Request:
POST /mytopic/subscribers
Content-Type: application/atom+xml

<atom:link rel="post-message-once"
           href="http://foo.com/somewhere" />


Response:
HTTP/1.1 201 Created
Location: http://…/mytopic/subscribers/111
```

Messaging Consuming: Queues

Pull Model

Queues

- Delegation of work
- One and only one client can consume a messag
- Once consumed the message can be garbage collected or archived
- Pull model has acknowledgement protocol

Message Consuming: Find Links

```
Request:
HEAD /destination/myQueue

Response:
HTTP/1.1 200 Ok
Link: <…/poller>; rel="poller"
```

Message Consuming: Consume Message

```
Request:
POST /destination/myQueue/poller

Response:
HTTP/1.1 200 Ok
Link: <…/messages/333/ack;token=3211>; rel="acknowledge"
Content-Type: application/json

[Some json document]
```

Message Consuming: Acknowledgement

- Server wants to guarantee that client received and processed message
- Client POSTs to acknowledgement link
- Server will re-enqueue the message if client doesn't acknowledge

Message Consuming: Acknowledgement

```
Request:
POST /destination/myQueue/messages/333/ack;token=3211
Content-Type: application/x-www-form-urlencoded

acknowledge=true
```

Message Consuming: Acknowledgement

```
Request:
POST /destination/myQueue/messages/333/ack;token=3211
Content-Type: application/x-www-form-urlencoded

acknowledge=true

Successful Response:
HTTP/1.1 204 No Content
```

Message Consuming: Acknowledgement

```
Request:
POST /destination/myQueue/messages/333/ack;token=3211
Content-Type: application/x-www-form-urlencoded

acknowledge=true

Unsuccessful Response (Message got re-enqueued):
HTTP/1.1 412 Preconditions Failed
```

Messaging Wrap-up

Send/Receive content without a envelope format

Use link headers

No footprint required on client or server

Simple? I hope…

REST-* Transactions

Does REST need transactions?

REST-* Transactions

Transactions are used for coordination

2PC is a vote to change state

- TM is the vote taker and voting machine

Transactions guarantee a state transition will happen

REST-* Transactions

Simple coordination isn’t the hard part

Fault tolerance

Crash Recovery after failures

This is the non-trivial part of transactions

REST-* Transactions

- Transactions need not hold database locks
- Transactions don’t even have to be 2PC
- Compensation is a viable pattern for long running interactions
  - Do/Undo
  - Consistency and failure recover still an issue

Are Transactions RESTful?

Interactions with a transaction manager can be

Hopefully show it in following slides

Are Transactions RESTful?

- Does using transactions make an application unRESTful?
  - Break stateless requirement?
- If the tx is modeled as a state change?
  - IMO, app is still restful
- Does it hold DB locks?
  - App becomes session oriented
  - Stateless constraint gets broken

Are transactions RESTful?

Who cares if they are RESTful or not?

Do you need the guarantees?

shrug

Single most asked question in my JAX-RS talks

TX Spec

Strive to be simple to use and implement

So any simple language or platform can use them

Treat Transactions as a service

2PC and Compensation protocols

Let's look at 2PC

Create an 2PC Transaction

- POST to a TransactionManager resource
  - Reliable post-message-once could be used too

Create a Transaction

```
Request:
POST /transaction-manager
Host: tm.org
Content-Type: application/x-www-form-urlencoded

timeout=300s
```

Create a Transaction

```
Request:
POST /transaction-manager
Host: tm.org
Content-Type: application/x-www-form-urlencoded

timeout=300s

Successful Response:
HTTP/1.1 201 Created
Location: http://tm.org/transactions/3322
```

Transaction Resource

Doing a GET returns application/tx+xml

Simple media type specifies status of transaction

- Active, Committing, RollingBack, Committed, RolledBack

Links to other resources and actions

- *participants* - resources participating in the transaction
- *Commit/rollback* - action resources to commit or rollback the transaction

commit and *rollback* links provided only if transaction is Active

Transaction Resource

```
Request:
GET /transactions/3322
Host: tm.org

Successful Response:
HTTP/1.1 200 Ok
Content-Type: application/tx+xml

<transaction>
   <status>Active</status>
   <atom:link rel="participants" href="…" type="…"/>
   <atom:link rel="commit" href="…"/>
   <atom:link rel="rollback" href="…"/>
</transaction>
```

Registering TX-Aware Participants

POST to the *participants* link of the transaction

- post-message-once pattern can be re-used

Content is an atom:link to callback to the participant

Registered link defines interaction semantics

We provide default media types for interaction

No reason you can't support more

Register Tx-Aware Participant

```
Request:
POST /transactions/3322/participants
Host: tm.org
Content-Type: application/participant-reg+xml

<participant>
   <link rel="participant" href="…"
         type="application/participant+xml/>
</participant>

Successful Response:
HTTP/1.1 201 Created
Location: http://tm.org/transactions/3322/participants/00
```

Registering TX-Unaware Participants

We're working on a TX-Unaware protocol

Participants can be created with links for prepare/commit/rollback (or do/undo)

Representations can be stored for each of these actions

Completing a Transaction

- Client does an empty POST to *commit* or *rollbac*
  link
- Transaction Manager calls back to participants

Complete a Transaction

```
Request:
POST /transactions/3322/commit
Host: tm.org

Successful Response:
HTTP/1.1 200 Ok
Content-Type: application/tx+xml

<transaction>
   <status>Committed</status>
</transaction>
```

Change Participant State

```
Request:
PUT /someparticipant
Host: somewhere.org
Content-Type: application/participant+xml

<participant>
   <status>prepare</status>
</participant>

Successful Response:
HTTP/1.1 204 No Content

Unsuccessful Response:
HTTP/1.1 412 Preconditions Unmet
```

Transaction Propagation?

- Forward a *transaction* link when creating or updating a coordinated resource
  - Resource would register itself with TM
- Resource could instead return a *participant* link and the client could register it with the transaction
- Client handles all interactions with TM
  - Uses TX-Unaware protocols

Transactions Wrap-Up

- Transactions provide state transition guarantees
  - Failure recovery untrivial to hand-roll yourself
- People ask for them
  - Whether they need it or not, is IMO, not our business
- REST-* Transactions attempts to provide a simple interface

References

Links

http://rest-star.org

O'Reilly Books

"RESTFul Java with JAX-RS" by me

"RESTful Web Services"

"RESTful Web Services Cookbook"

FOLLOW US ON TWITTER

www.twitter.com/redhatsummit

TWEET ABOUT IT

#summitjbw

READ THE BLOG

http://summitblog.redhat.com/