Web Services and Transactions

Jonathan Halliday
JBoss
6500

JAZOON09
THE INTERNATIONAL CONFERENCE ON JAVA TECHNOLOGY
JUNE 22–25, 2009 ZURICH

AGENDA

Transactions Background

Atomic Transactions
- WS- AT
- Transaction Bridging

Long Running Transactions
- WS- BA
- BA Annotation Framework

RESTful Transactions

Transactions Background

A transaction is:

- A set of activities that require some shared properties
- Most important property: consensus of outcome
- Short lived transactions: ACID properties
- Longer transactions: relaxation of some properties

Distributed transactions:

- Involve two or more systems
- Require agreement on protocol for interoperability
- May span organizational boundaries

Web Services and ACID transactions

Web services may be used within an enterprise for system integration

- Same administrative domain
- Fast network

WS-Atomic Transaction: ACID transactions for Web Services

- JTA like behavior
- Suits closely coupled environments
- Short duration transactions due to locking model
- Begin / Commit / **Rollback**

WS-AT specifies the wire protocol only

- No standard Java API (yet)

Using WS-AT on the client side

```
UserTransaction userTx = UserTransactionFactory.userTransaction();
userTx.begin();

    webServiceOne.someBusinessMethod(param);
    webServiceTwo.anotherBusinessMethod(arg1, arg2);

userTx.commit();
or
userTx.rollback();
```

Using WS- AT on the server side

```
TransactionManager tm = TransactionManagerFactory.transactionManager();

tm.enlistForVolatileTwoPhase(myVolatileParticipant);

tm.enlistForDurableTwoPhase(myDurableParticipant);

tm.suspend();
tm.resume();
```

Implementing WS- AT Participants

Users must implement not only the business logic, but the transaction event handling logic too:

```
interface Participant
{
    public Vote prepare();
    public Vote commit();
    public Vote rollback();
}
```

Using WS-AT is hard

JEE containers provide lots of abstraction on top of JTA

- Business programmers hardly ever implement XAResource
- Or even call begin/commit/rollback thanks to EJB3 @TransactionManagment and @TransactionAttribute

Web Services don't benefit from this established infrastructure despite running in the same container

How can we make this easier?

- Allow WS-AT transactions to behave as though they are JTA transactions

Transaction Bridging

Existing JEE code understands JTA transactions

Web Services understand WS-AT transactions

Interoperability and reuse is improved by linking these domains

- Web Services can use existing XA aware resource managers
- JEE code can call transactional Web Services

txbridge does this

- Interposition plus a protocol adapter
- Bi-directional
- Near invisible to the application – just add one standard annotation
- Provides XAResource / Participant implementation and event handling

Transaction Bridging

```
@Stateless
@Remote(Bistro.class)
@WebService()
@SOAPBinding(style = SOAPBinding.Style.RPC)
@HandlerChain(file = "jaxws-handlers-server.xml")
@TransactionAttribute(TransactionAttributeType.MANDATORY)
public class BistroImpl implements Bistro {
    @WebMethod
    public void bookSeats(int numberOfSeats) {
        BistroEntityImpl entity = em.find(BistroEntityImpl.class, someId);
        entity.increaseBookingCount(numberOfSeats);
    }
```

Web Services and Business Activities

Web services may be used between business partners

- Different administrative domains
- Loose coupling, high latency, low reliability

Need to relax ACID properties

- Locking won't work
- Use compensations instead
- Reduced isolation of transactions
- Per- application undo behaviour

WS- Business Activity

WS- BA

Compensation based
- Relaxes isolation
- Changes transaction event model
- Makes participant implementation harder

Transaction events:
- begin()
- complete() - persist changes, log compensation data
- close() - clean up, discard logs
- cancel() - discard changes
- compensate() - undo previously completed changes

Using WS- BA on the client side

```
UserBusinessActivity userTx =
                UserBusinessActivityFactory.userBusinessActivity();
userTx.begin();

    webServiceOne.someBusinessMethod(param);
    webServiceTwo.anotherBusinessMethod(arg1, arg2);

userTx.close();
or
userTx.cancel();
```

Using WS- BA on the server side

```
BusinessActivityManager tm =
        BusinessActivityManagerFactory.businessActivityManager();

tm.enlistForBusinessAgreementWithCoordinatorCompletion(myParticipant);

tm.enlistForBusinessAgreementWithParticipantCompletion(myParticipant);

tm.suspend();
tm.resume();
```

Implementing WS- BA Participants

Users must implement not only the business logic, but the transaction event handling logic too:

```
interface Participant
{
    public void close();
    public void cancel();
    public void compensate();
    public void complete();
}
```

BA Framework

Writing Business Activity code is hard

- Compensations are application specific, unlike rollbacks
- More work for the business logic programmer
- How can the container help?

BA Framework provides high level annotations

- Ideas taken from EJB3, JSR-181
- @CompensatedBy()
- Easy for JEE programmers to pick up

BA Framework

- Container provides transaction plumbing
  - Serialization, concurrency control, locking, versioning of data, crash recovery
  - Business logic does not respond directly to transaction control events or implement Participant interface
- Separate business logic from transaction management as much as possible
  - But compensation logic belongs on the business side
- Declarative approach
- Near transparent runtime, much like EJB3
  - Automatic execution of compensations
  - AOP based, compile time or runtime instrumentation
  - Automatic Participant enlistment, ordering (reverse!) and serialization of compensations+ parameters

BA Framework

Annotations describe the relationship between actions and their compensations

Annotations are WS-BA specific, but the approach is generic enough

```
@BACompensatedBy(“cancelRoom”)
public int bookRoom() { ... }

public int cancelRoom() { ... }
```

BA Framework

@BAParam and @BAResult for wiring of parameters are return values

General purpose per-tx persistent map for storage of values needed for compensation

```
@BACompensatedBy(“cancelRoom”)
@BAResult(“reservationNumber”)
public int bookRoom(@BAParam(“clientID”) String client) { … }

public void cancelRoom(@BAParam(“clientID”) String who,
                      @BAParam(“reservationNumber”) int resID) { … }
```

RESTful Transactions

WS-AT and WS-BA are good for SOAP
... but what about Web Services that use a REST architecture?

Even with REST, you still need consistency and reliability between systems
So you need a coordination protocol (or two)
JAX-RS standardizes some aspects of REST, but not transactions

Model the transaction coordinator and participants as resources
Transaction context propagation standard is also required

RESTful Transaction Coordination

Transaction Coordinator

```
POST .../transaction-coordinator/begin
PUT .../transaction-coordinator/<TxId>/commit
GET .../transaction-coordinator/active
```

What it looks like with JAX-RS:

```
@PUT
@Path("transaction-coordinator/{TxId}/commit")
public Response commitTransaction(@PathParam("TxId")String txId) {...}
```

RESTful Transaction Participants

Enlist a participant in a transaction

- PUT .../transaction-coordinator/<TxId>/
- The body identifies the participant URL

Operations on Participants

- GET .../participant-server/<ParticipantId> : status
- POST .../participant-server/<ParticipantId>/prepare

The service must implement appropriate behavior for prepare/commit/rollback

- Transaction bridging?

RESTful Transactions

Sometimes you do need transactions

It's possible to do transactions in the REST style

But there is no standard protocol for it yet

- We have specs for ACID and forward compensation based transactions

Implementation using JAX-RS is relatively straight forward

- We have a prototype on RESTeasy

Interoperability will have to wait for mass adoption

But you can use it internally now

Summary

Transactions are a useful tool for structuring data manipulations

ACID transaction are not suitable for all cases

- Sometimes you need a lock-free, forward compensation model

WS-AT and WS-BA provide standard, interoperable transactions

- But only the protocol, not the Java API

Easy of use requires going beyond the standards

- Transaction bridging and BA Framework

Transactions are possible and sometimes necessary in a REST architecture

Protocols and prototypes available now, but no REST standards yet

Jonathan Halliday

JBoss

www.jboss.org/ jbosstm

jonathan.halliday@redhat.com