The AMQP Distributed Transaction Classes (Java)

The distributed transaction classes provide support for the X-Open XA architecture. The dtx-demarcation class is used to demarcate transaction boundaries on a given channel that is subsequently used to perform AMQP native transactional work (produce publish messages). Transaction coordination and recovery operations are provided by the dtx-coordination class.

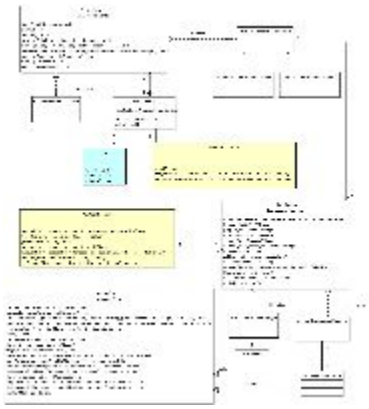
A Transaction Manager uses RM Client XA interface to demarcate transaction boundaries and coordinate transaction outcomes. RM Clients use the dtx-demarcation class to associate transactional work with a transactional channel. The transactional channel is exposed to the application driving the transaction. The application can then use the transactional channel to transactionally produce and consume messages. RM clients use dtx-coordination to propagate transaction outcomes and recovery operations to the AMQP broker. A second coordination channel can be used for that purpose.

More details about can be found at:

- https://wiki.108.redhat.com/wiki/index.php/AMQP:Transaction_SIG_dtx_XML
- http://cwiki.apache.org/confluence/download/attachments/55787/dtx-classes-specification-document-v1.2.pdf
- http://cwiki.apache.org/confluence/download/attachments/55787/dtx-classes-presentation-v0.10-PMC-03142007.pdf

The Qpid Implementation in Java

As shown on the following class diagram, there are two protocol specific dtx classes, that is to say DtxDemarcation and DtxCoordination that are highlighted in yellow.



DtxDemarcation

DtxDemarcation interacts with the corresponding AMQChannel. The operation select creates the corresponding TransactionalContext (a channel has by default a non-transactional context). The operations start and end associate and disassociate a provided xid with the current TransactionalContext that percolates the call to the TransactionManager (operations begin and end respectively). Note that the operation end is responsible for acknowledging the messages against the context i.e. those messages are seen as being consumed under the currently associated xid.

DtxCoordination

DtxCoordination directly interacts with the TransactionManager. Note that it is a requirement that the operation end is called on all involved channels (i.e. all the acknowledged messages have been specifically consumed under the provided xid).

TransactionalContext

There are three flavours of TransactionalContext: the non transactional one, the local and distributed ones. The distributed and local contexts are very similar and both extend the abstract context. Note that the distributed context does not implement commit and rollback as this is DtxCoordination that is responsible for deciding of a transaction outcome.

TransactionManager and MessageStore

Transaction manager and message store are linked as the message store may need to add transaction records to the transaction identified by a given xid. Moreover, the transaction manager may need to use the transactional facilities of the underlying store. This is the case of the JDBCStore and JDBCTransactionManager. The JDBCTransactionManager uses the transaction facilities of the JDBCStore for performing ACID operations during prepare and commit.

This is the responsibility of the MessageStore to recover queues and exchanges and messages. Note that the JDBCTransactionManager delegates the responsibility of getting the list of in-doubt transactions to the JDBCStore but another implementation of TransactionManager may handle that directly. The MessageStore implementation should not load the messages in memory during recovery but only set the messageID. The message header, publish info and payload are lazily loaded. Note that the message payloads are currently loaded in memory. We can however easily implement a direct streaming of message payload on the wire (The MessageStore interface can be extended for supporting that).