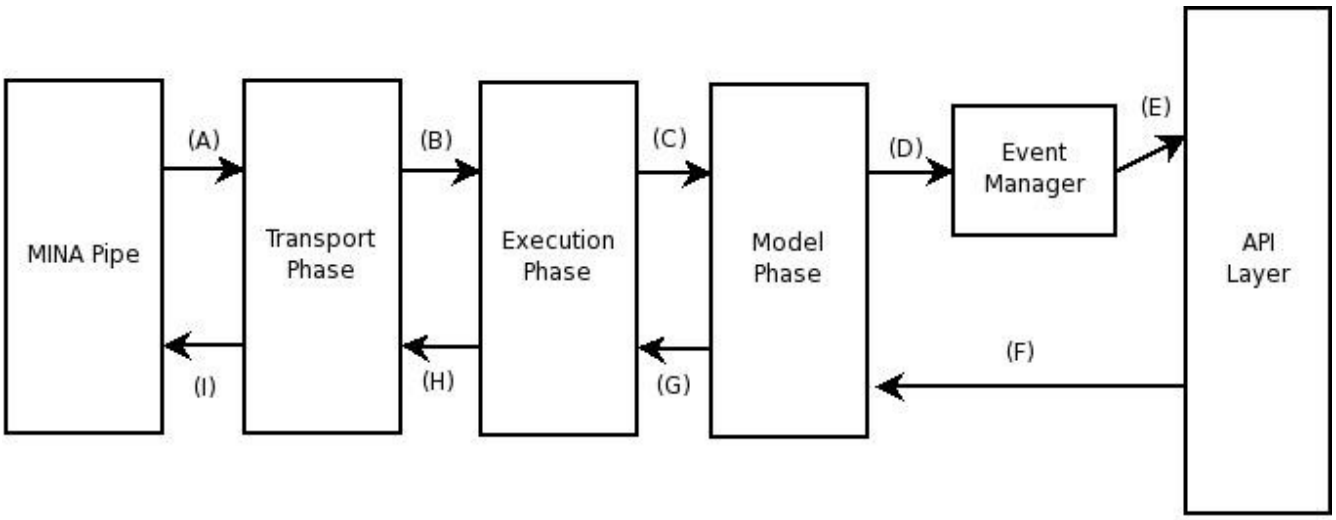
AMQP Java Client API Architecture

The following diagram provides a high level overview of the architecture from a call flow perspective. (A) to (I) describes message flows. An important point to note is that there is an instance of a pipe per physical connection. So all interactions are scoped by a connection. In summary the client consists of a well defined pipeline of phases and an API layer closely modeled on the AMQP client API.



Phase pipe

The phase pipe is introduced to archive a clear separation of concerns and consist of an arbitrary number of Phase objects. There is a clear contract between each phase, and changes within each phase is insulated so that it doesnt affect any other phase.

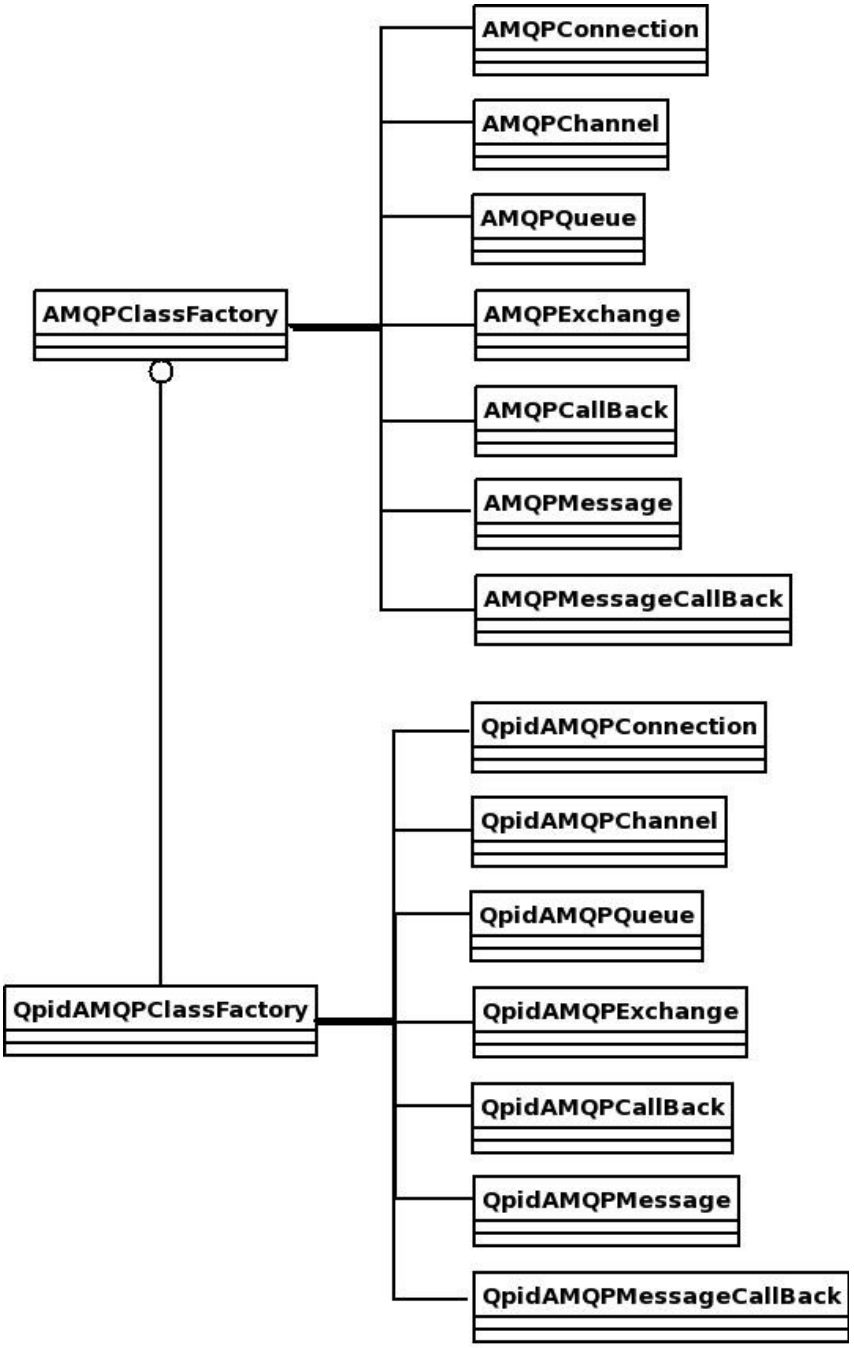
- Currently there are 3 phases and they correspond directly to each layer in the AMQP protocol.
- The Transport phase deals with AMQFrames and deals with MINA layer for NIO.
- The Execution phase deals with the request/response paradigm.
- The Model Phase works at the AMQPMethodBody level.

Call flow

(A) The MINA layer hands over an AMQFrame to the Transport phase
(B) The Transport phase hands over the AMQPFrame to the Execution phase
The execution phase handles the request or response logic
(C) The Execution phase hands over one or more AMQMethodEvents to the Model phase
(D) & (E) The Model phase distributes these events to the API layervia the Event Manager
(F) The API layer hands over AMQMethodEvents to the Model phase
(G) The Model Phase hands over to the Execution phase
(H) The Execution phase handles the correlation,creates AMQFrames and hands it over to Transport phase.
(I) The Transport phase hands over AMQFrames to the MINA layer.

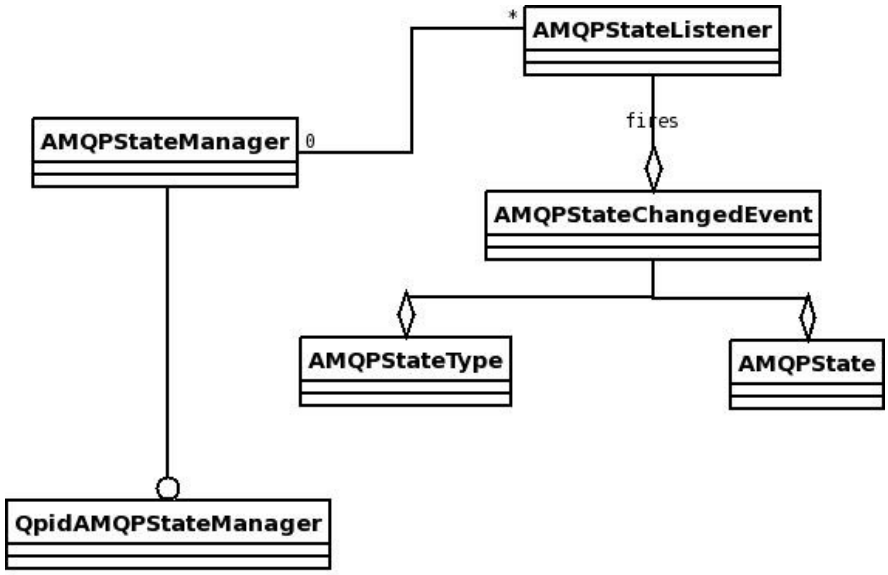
API Layer

The API layer is modeled on the AMQP protocol classes, which provides a more natural programming model. All state transitions are driven by the client API or by broker events. Users will programed to interfaces and not the concrete implementations. Abstract AMQPClassFactory will provide a concrete implementation of the AMQPClassFactory.



State Transitions

- AMQPChannel and AMQPConnection class implementations are finite state machines
- State change events are notified to listeners via the AMQPStateListener interface.
- All state transitions are verified and method invocations on AMQPConnection and AMQPChannel are only allowed in the sequence defined by the AMQP protocol.
- Listeners can register based on the type of events (Ex either channel, or connection)



Extensibility

The API provides several ways to extend the programming model.

- The phase pipe is configurable
- Can provide a different implementation of the o.a.q.amqp package.
- Can register for protocol events via the AMQPMethodListener interface.
- Can register for state change events via the AMQPStateListener interface.