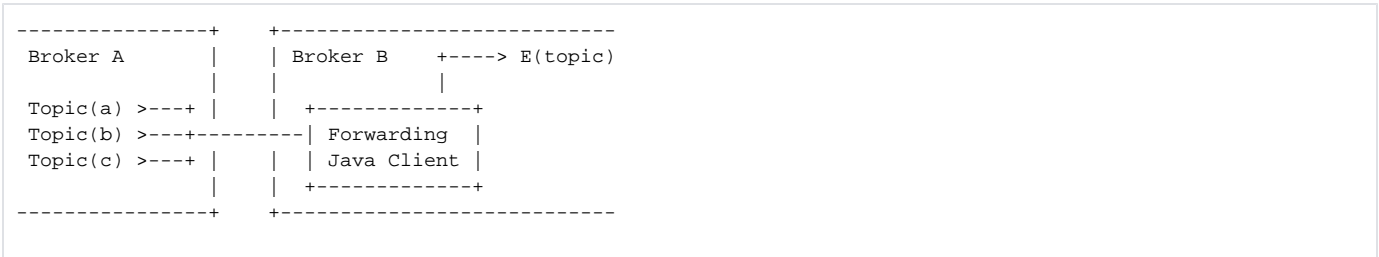# Java Federation Design Proposal

Message Federation Design Proposal

The following proposal is only to address two type of message federation. This is aimed to be implemented in the Java broker before we fully upgrade it to support AMQP 0-10.

The two types of message feddration that we will consider. Topic Fanout and Remote Queues.

Topic Fanout

```
---------------+    +--------------------------
 Broker A      |    | Broker B     +----> E(topic)
               |    |              |
 Topic(a) >---+ |   |   +------------+
 Topic(b) >---+---------| Forwarding |
 Topic(c) >---+ |   |   | Java Client|
               |    |   +------------+
---------------+    +--------------------------
```

Broker B has a process where it subscribes to various topics on Broker A forwarding the messages on to the Specified Exchange on B.

New Dynamic Topic Exchange

An additional topic exchange type can could be added so that any *bind* request is added to the existing set of subscriptions. This would remove the need to explicitly configure any forwarding and so ensure that any new clients that joined would receive messages on the topic they requested.

Embedded Client

While the client could be a much simpler client than the existing Java Client. By utilising the existing java client reduce the replication of functionality. If the performance overhead of using the full client is shown to be to high then we can revisit this situation.
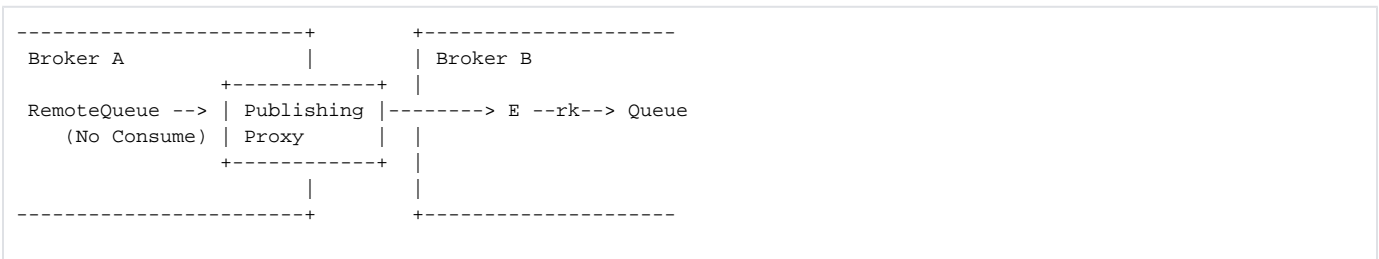
There are several features that the Forwarding Client should perform:

- 'Topic Reduction', 'weather.europe.scotland.*' and 'weather.europe.#' can be reduced to 'weather.europe.#'
- Handle reconnection to Broker A.
- Loop detection

Loop Detection

If a tag identifying the broker is added to the each message that is forwarded then the client can tell if a message has already been forwarded and so silently drop the looped message. This detection could be made more proactive by having the Client subscriptions use a selector to specifically exclude all messages with the Broker B's ID.

Remote Queues

```
-----------------------+        +--------------------
 Broker A              |        | Broker B
             +-----------+  |
 RemoteQueue --> | Publishing |--------> E --rk--> Queue
    (No Consume) | Proxy      |  |
             +-----------+  |
                    |       |
-----------------------+        +--------------------
```

The Remote Queue case has a Publishing Proxy that will push all the messages that arrive on the 'RemoteQueue' to the specified Exchange (E) and RoutingKey (rk).

The 'RemoteQueue' should potentially prevent any client subscriptions requests from directly consuming from the Queue on Broker A. Attempts to consume from this queue could utilise the the redirect method to send the requesting client to Broker B.

Publishing Proxy

All messages should be sent via transaction to maintain the reliability of the transfer.
Just as with the Forwarding Client this proxy must be capable of reconnecting in the event of connection failure.