AMQP Illustrations

2 December 2008 – Post Transport/Model Integration

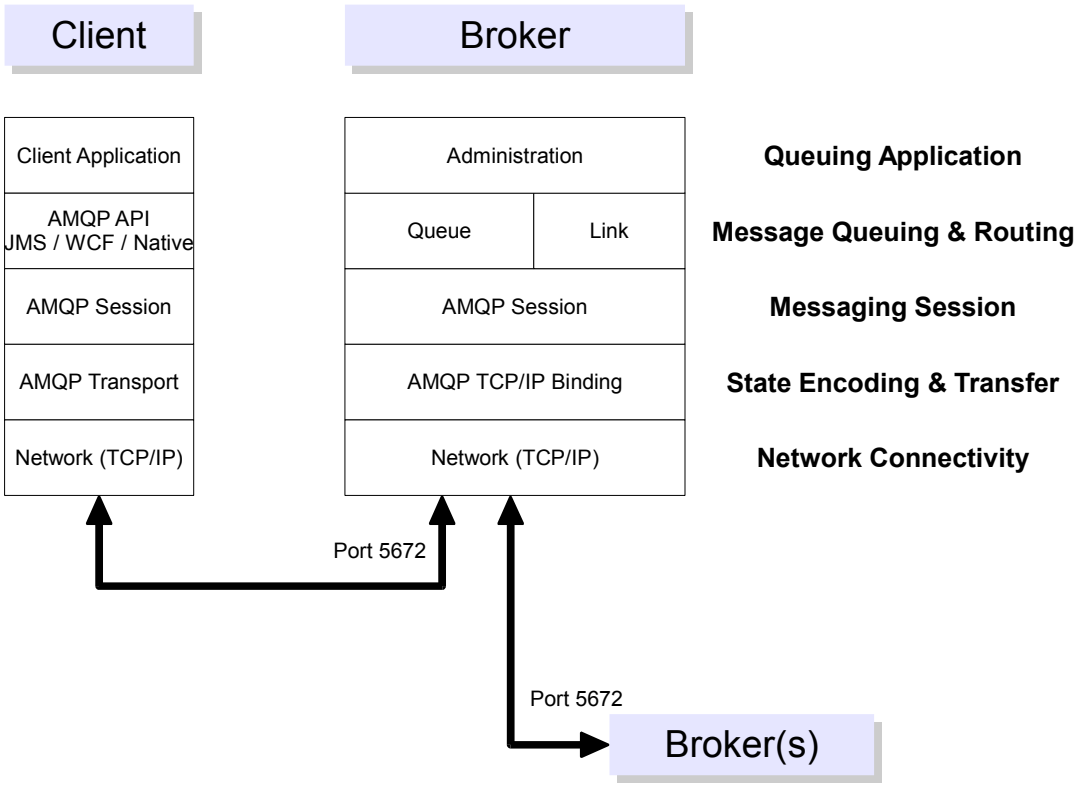
Corresponds to Model Draft 91

Table of Contents:

Note: This document specifically does not address the creation of LAN Federations or HA Broker configurations; these may be outside the remit of protocol standardisation.

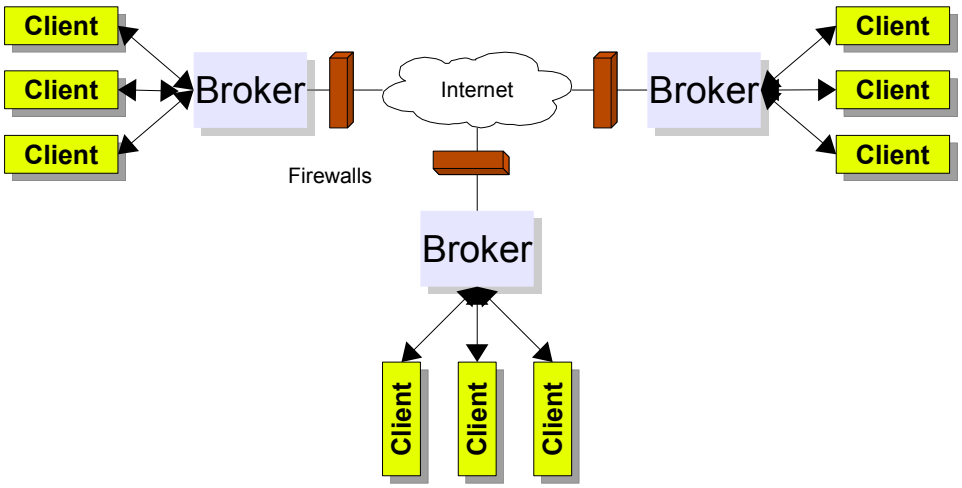
AMQP Transport Peer-to-Peer Topology

The API implementations see more detail, under the covers of AMQP. In addition to MOM messaging, they have to deal with the details of maintaining and securing a network connection between the client and the broker. This is how the lower level transport stack looks to a systems engineer.



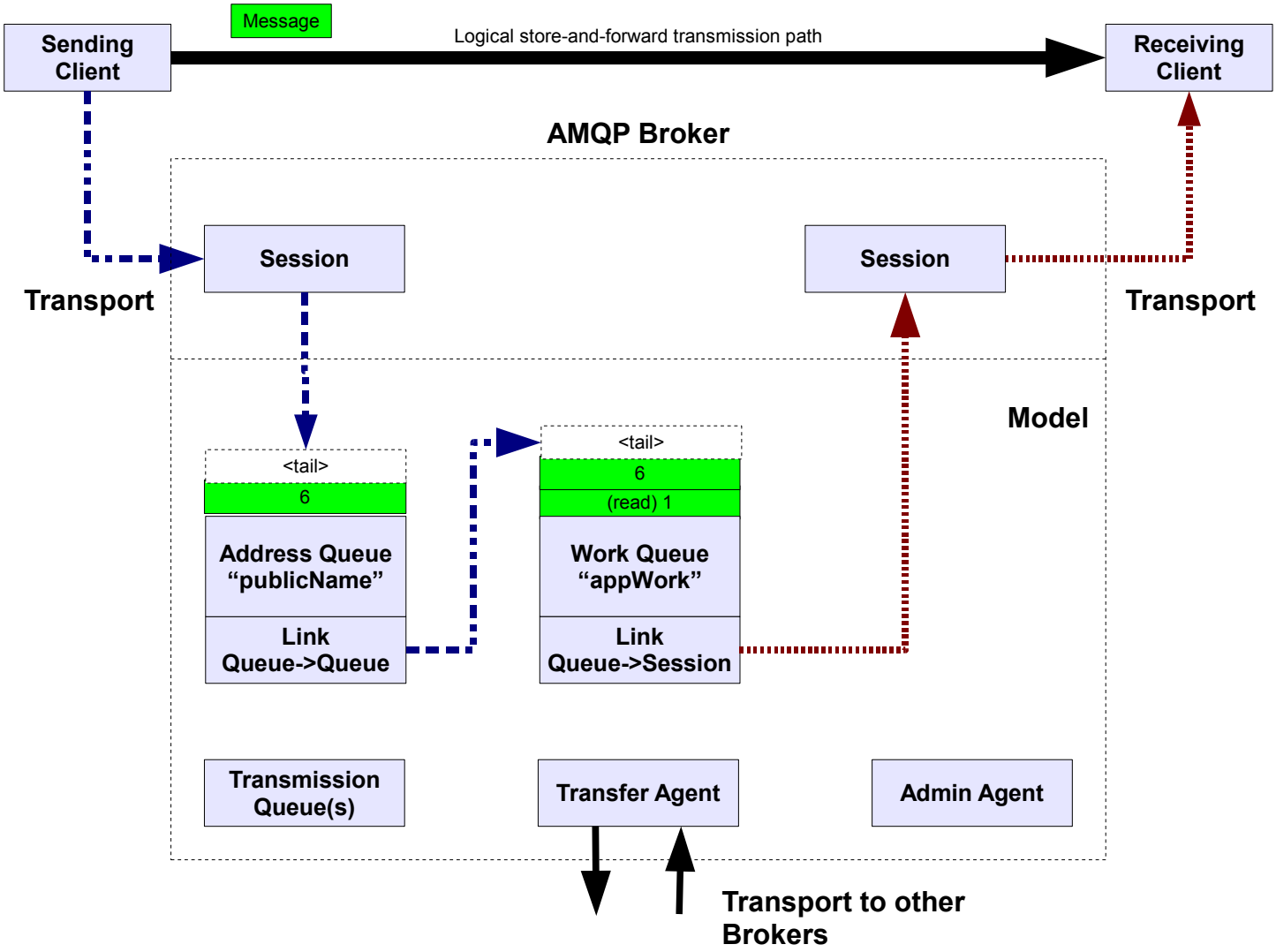
AMQP Broker-to-Broker Global Forwarding

Global addressing will relay messages from local queues on a client-local broker to queues resident on the destination organisations broker. This is very similar to SMTP. Also, as with SMTP relaying is strongly discouraged as it may compromise security.



AMQP End-to-End MOM Topology (WORK+)

This is the message transmission path as the sending and receiving applications perceive it.
It is important to note that the Message transmission path is a game of two halves; the deposit in a queue half, and the collect from the queue half.
This decoupling of sender and recipient is the prime purpose of message oriented middleware.



Notes on Addressing, Queues and Links (previously Bindings):

An Address is just a name made known to a publishing Client by those who would like to solicit messages from that client.

An Address is the fully-qualified name of a Queue (queue_name@broker-host). By using more than one Queue (as shown) an organisation can decouple where a message is sent to from how and where it is processed. This is useful to enable administrators to modify the topology without their client(s) knowledge. So, while every Queue in AMQP has a name and can be directly accessed for enqueue and dequeue operations, only naïve applications with simple needs will do this.

The Addressing format of queue-name@host-name or queue-name@domain-name is intended to draw a deliberate comparison with email addressing. However, for local delivery on the same Broker, no domain name need be used in the Address. Domain names are only required for relaying to remote Brokers, discussed elsewhere.

The Address is opaque to the sending Client, but behind that Address, the owner of the Broker creates Links (either administratively or dynamically) to deliver Messages sent to that Address to one or more Message Queues on the same or different Brokers, and from there to receiving Clients.

Links automatically route messages between Queues (from a Source to a Target) optionally filtering using a predicate. Links are also used to transfer Messages from a Queue to a Client. Messages sent to an Queue will be considered by the Links which have that Queue as their Source. Links may inspect Messages from their specified Source to determine whether to transfer or copy it to the Target.

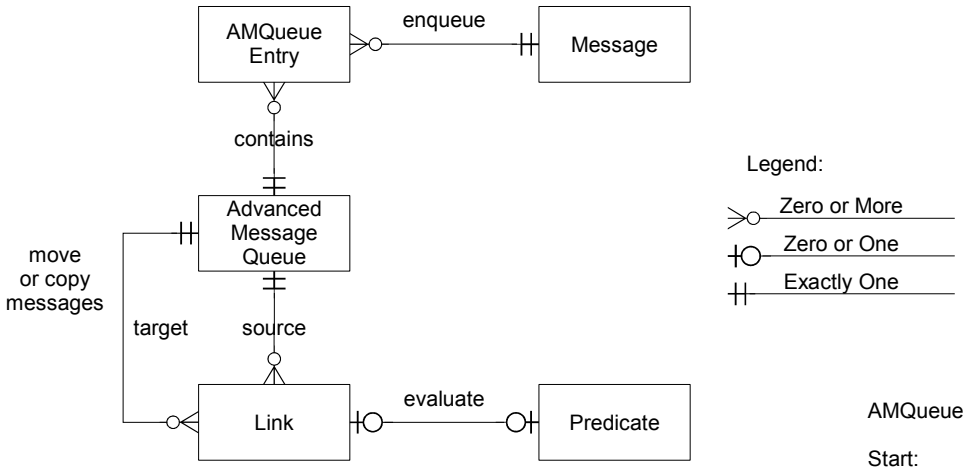
A Queue can have many Links transferring Messages to many Targets (one Target per Link). The parts of the Message which get inspected by a given Link are the "routing key(s)" of the Link. Each Link may have different "routing keys" from other Link on the same Source. Link predicates are specified using a subset of SQL 92 (though vendor extensions can be applied here to permit other ways of specifying the predicate).

A common example of a Link predicate is in Publish/Subscribe Messaging, where a Message's Subject will matched on a pattern to determine which Clients will get the Message. Such a Link would have the routing key of "Subject" matched against a Client supplied pattern argument. Note that any field could have been the routing key, it didn't have to be "Subject".

So, when Addressing a Message, the Address forms the primary means of directing the Message, but other Message Properties may be involved as well. In such cases, the Sending Client will need a complete specification of how to populate the Message they send; but this forms a natural part of the interface contract between sending and receiving user applications.

AMQP Model Entity Relationship Diagram

This diagram shows the perceived entities which the client an discover in a compliant broker, and the relationships between them. A compliant broker may be implemented differently, but MUST exhibit these same perceivable entities.

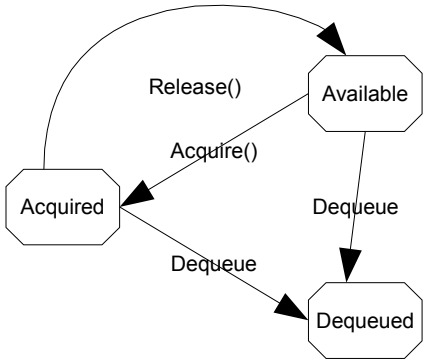


AMQueue entries may have the following states:

Start:
Acquired: Y/N
End: Y/N

{ optional (DTX like) transport protocol command :
park seqno
unpark from seqno}

Queue Entry Detail (WORK NEEDED)

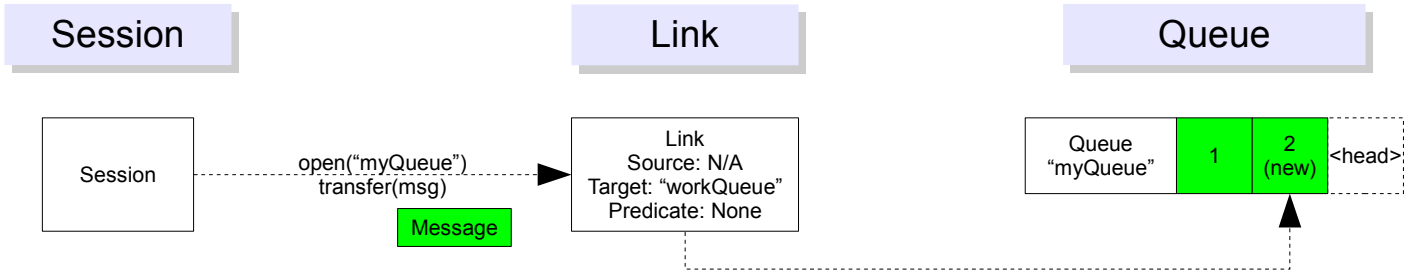


Link Detail (WORK NEEDED)

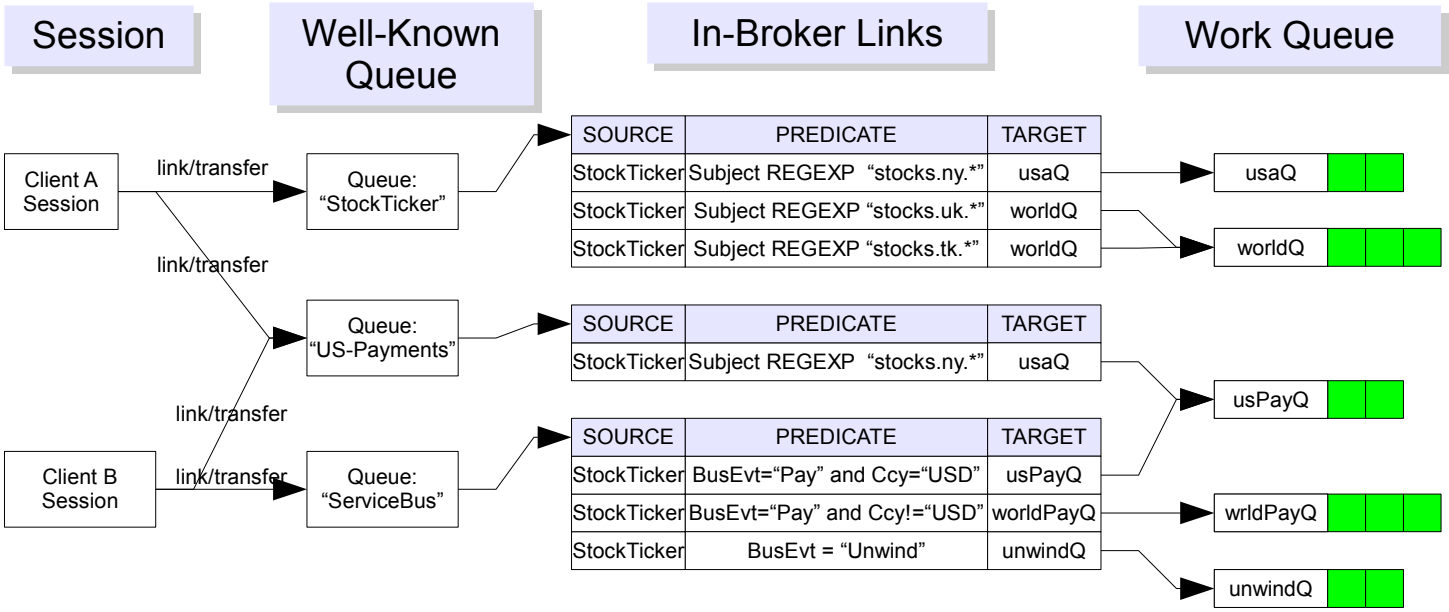
Delivery into Message Queues

Delivery into queues is the first half of the end-to-end MOM delivery process.

Simple scenario...



Comprehensive scenario and walk through...



Clients create Messages and Address them.

The client then uses its Session to Open a Link to a Queue. It can then Transfer the Message over the Link to the Queue on the Broker. This process is repeated for each separate Queue the Client wants to directly address;. This is very similar to SMTP.

The Broker evaluates the Message and related commands against its configuration and Access Control Lists. The Broker may reject the Clients request to access certain named Queues.

After a Message as been transferred onto a Queue via a Link the Message has been accepted for delivery. Subsequently the Broker can perform onward routing and copying via Queue-to-Queue Links the administrator may have preconfigured within the Broker.

In AMQP, an Address is essentially just a durable Queue with a well known name..

Clients publish Messages to these well known queues.

It is good practice to separate well known names from the queues an application may actually use to process Messages – this allows the application to change its topology at a future date without impacting addresses used by Clients (who might be external to your organisation). Think of these public Queues as being akin to PO Boxes.

The Broker can use internal Links to forward messages to other internal queues for your applications benefits based on Message properties, etc.

Since some Queue Names are well known to Clients (they may well be hard coded in applications, for example) the have value. Hence there are implementation specific ACL checks on which Clients may send to a given Queue, and which Queues may Link to other Queues (why? Imagine if I could request the post office to redirect mail for 10 Downing Street to my home instead!)

A Link also controls how Messages are transported from one Queue to another within the Broker.

Links route/copy Messages destined for a given source Queue into the associated target Queue optionally based on a Predicate.

For a given Queue, all Links which have that Queue as a Source and which match the Predicate, and which have capacity in their Target Queue will result in copied Message deliveries to the associated Target Queues.

A Queue holds messages until all interested Clients have had the opportunity to consume them.

In a high throughput system it is desirable to arrange for Queues to contain as few messages as possible. This implies that there are enough subscribing Clients consuming quickly enough to match or exceed the rate of the publishing Clients.

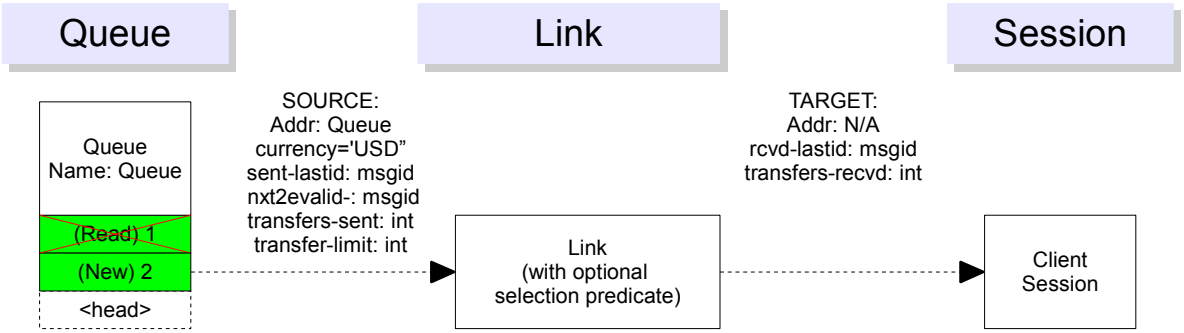
In a more batch-oriented system a Queue may be deliberately allowed to fill until some predetermined time or number of Messages held triggers a Client jobs to begin and dequeue and process the messages.

It is important that an AMQP implementation support both styles of client application usage.

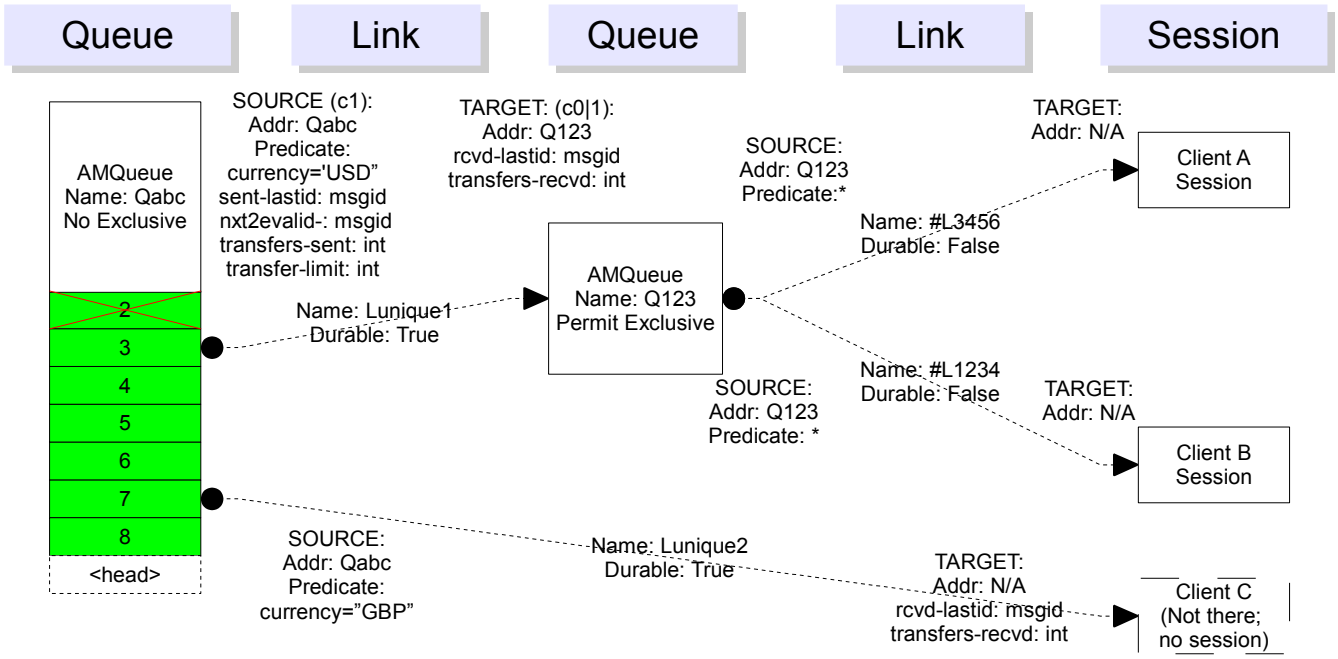
Consumption from Message Queues

I'm using the name “Stable Queues” (a la Stable Storage) to encompass the idea of a safety-deposit-box style drop-point.
Collecting Messages from Queues in a configurable way is the second half of the MOM end-to-end process.

Simple scenario...



Comprehensive scenario and walk through...



An AMQueue holds Messages until all Links have had the opportunity to inspect them.

AMQueues also apply ordering, capacity limits, TTL, persistence and various admission and disposition policies (which may be implementation specific).

An AMQueue owns its outbound Links.

AMQueues manage message locking and disposal. A AMQueue should not dispose of a Message until all Links have had the opportunity to inspect them, though the precise mechanism is implementation specific

A Link is an iterator over the underlying AMQueue combined with a Predicate which picks from its Source end the Messages of interest to its Target end.

There can be more than one Link originating from an AMQueue and they are independent; this enables an AMQueue to in effect have many Heads (but only one Tail).

Messages are always logically transported through a Link.
The AMQueue manages the allocation of Messages to Links (this how messages on one Queue can be shared out between multiple subscribers each with **arbitrary** criteria).

When a Link is first created, it usually first inspects the Message on the Tail of the AMQueue, which has the effect of considering only Messages subsequent to that Message.

A Link may also start any arbitrary point in the AMQueue.

Client Session creates a Temporary Link in order to dequeue message from a Queue.

The AMQP Transport controls the AMQP Session to effect transfers off the Link across the network.

In the “topic” case it will be common for the Temporary Link to initialise at the Tail and in the “queue” case it will be common for the Link to initialise at the Head.

When a Session is destroyed, it first ensures all the Temporary Links it created are destroyed.

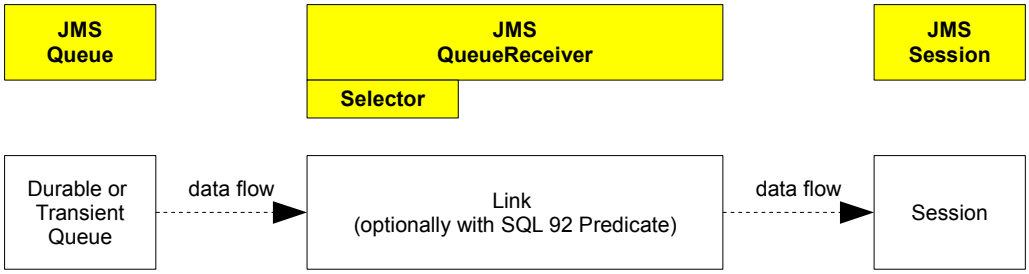
Clients may have many Consumers to many Selectors on many Queues.

JMS Concepts Modelled in AMQP

AMQP supports more Message distribution patterns than JMS specifies. This slide illustrates how JMS behaviours can be achieved through AMQP constructs.

MessageConsumer / QueueReceiver

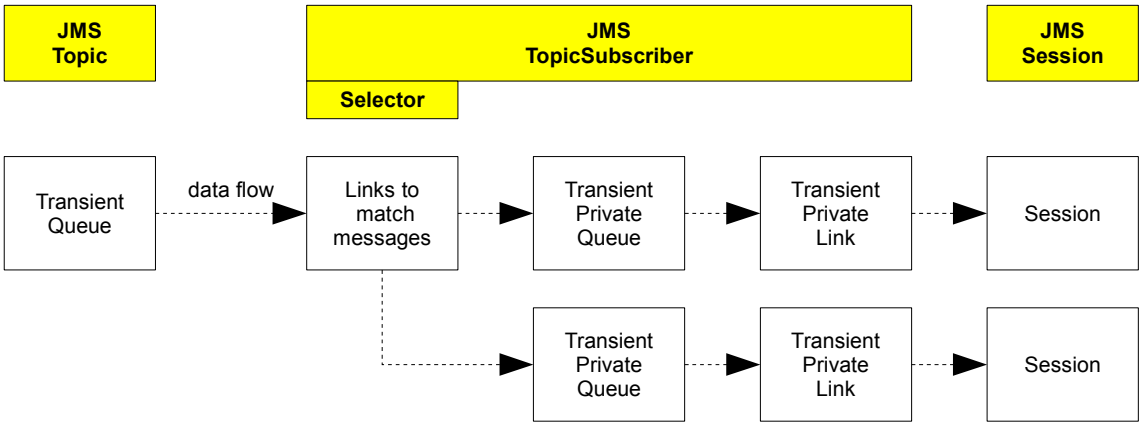
Here, the JMS Selector concept is modelled using AMQP Selector+Consumer to pick Messages from the Queue.



PubSub can be implemented in 2 ways

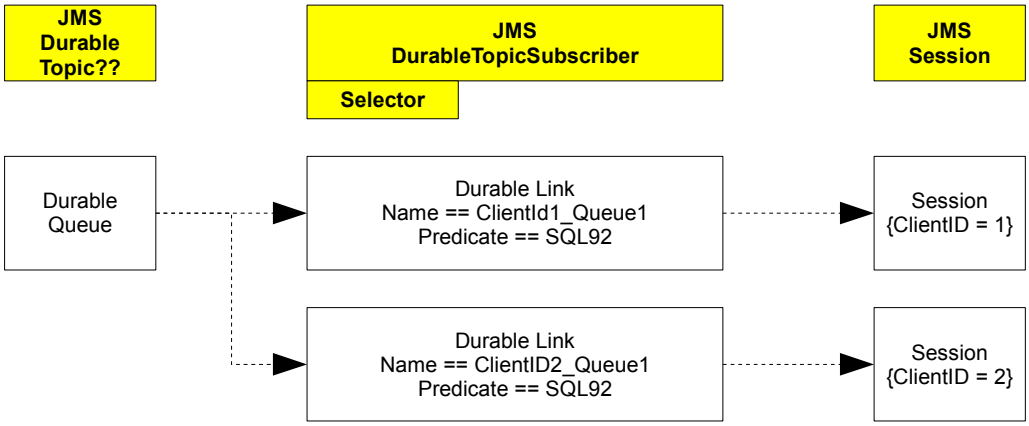
MessageConsumer / TopicSubscriber

Here, the JMS Selector concept is modelled using a Queue to represent the Topic and AMQP Bindings to route Messages from the Topic using Topic Bindings to copy the Messages into Private Queues, one Private Queue for each subscriber to the Topic.



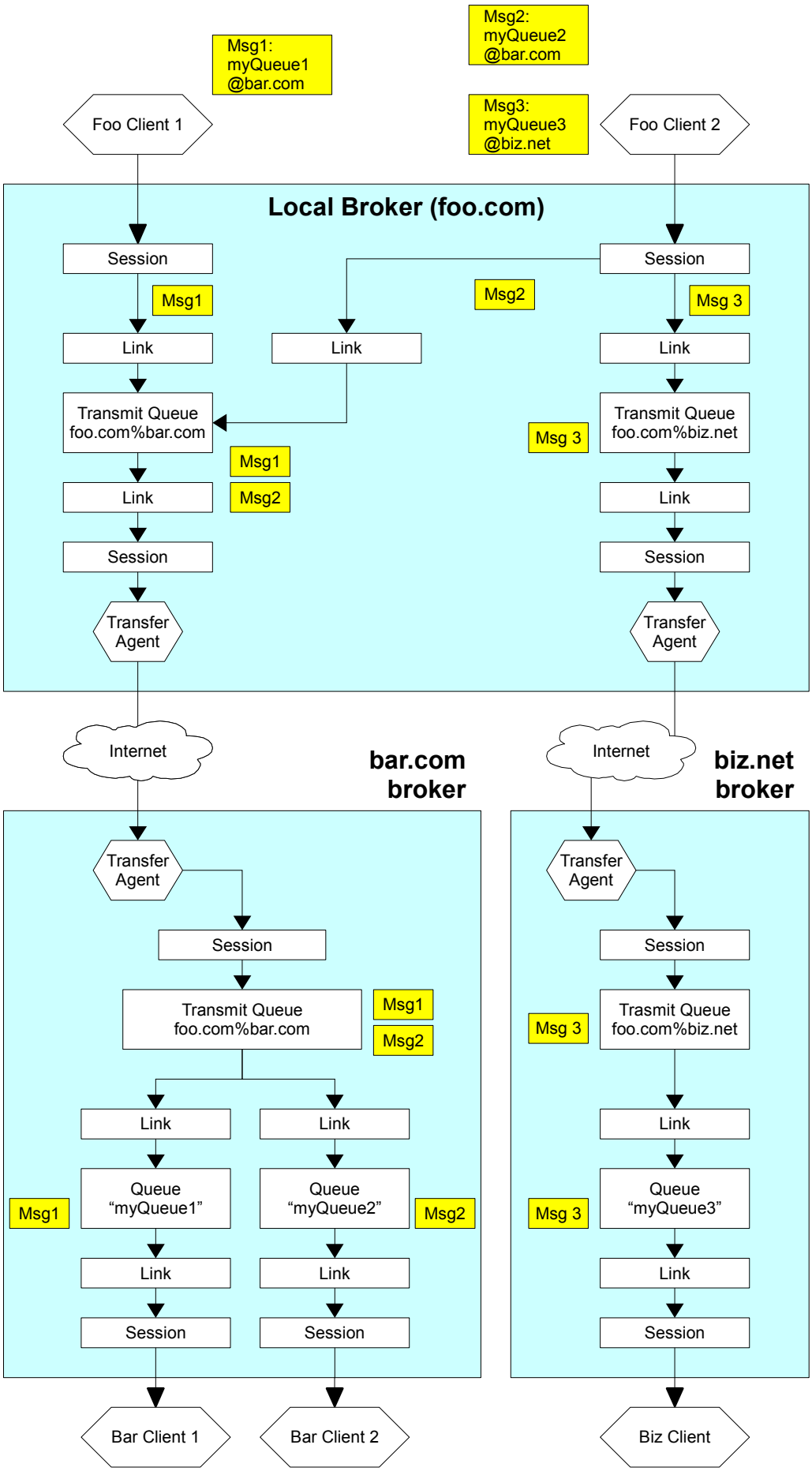
MessageConsumer / DurableTopicSubscriber

Here, the JMS Selector concept is modelled using a Queue to represent the Topic and AMQP Bindings to route Messages from the Topic using Topic Bindings to copy the Messages into Private Queues, one Private Queue for each subscriber to the Topic.



AMQP Global Addressing – Message Relay

Remote address scenario...



Walk Through...

When a Client Addresses a message using *address@host* notation, the portion after the @ is used as a host name where the Broker may be found which understands the address component.

If the IP address matches one of the IP addresses of the host where the broker is running, or is localhost, or blank, then the message is delivered to a local Address within the Broker, as normal.

If the host is on a different machine then the Broker's Linking mechanism will enqueue the Message on a Transmission Queue specific to the remote host.

Logically speaking, there is one Transmission Queue for each remote host. All Messages which have Addresses which reference the same remote host (see spec. for details) are placed onto the same Transmission Queue. This is to ensure that the relative ordering of Message delivery both with and between queues (esp. within a Transaction) is retained.

An entity known as a Transfer Agent dequeues Messages from each Transmission Queue and enqueues them to the appropriate remote Brokers at some time in the future. Whether the sending or receiving Transfer Agent initiates the request is administrator configured, as is the duration/frequency of connection duration (batch communication is legitimate use case).

Transfer Agents interact with the local and remote Brokers exactly as if they were normal receiving/sending Clients. The Transmission Agent ensures that Messages are delivered at least once, with duplicate suppression based on unique Message Identity (to prevent the need for 2 phase commit). Ideally source Transaction Boundaries would be preserved during the Transfer process, but this is not mandated.

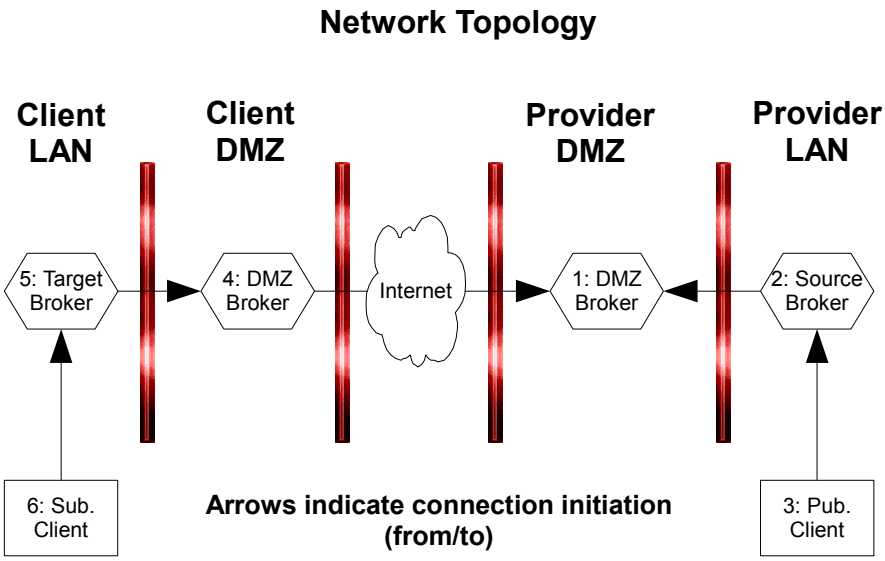
A relayed Message retains its Message Identity on all Brokers visited. The number of Brokers to be visited should be minimised – MOM routing is not a substitute for IP routing.

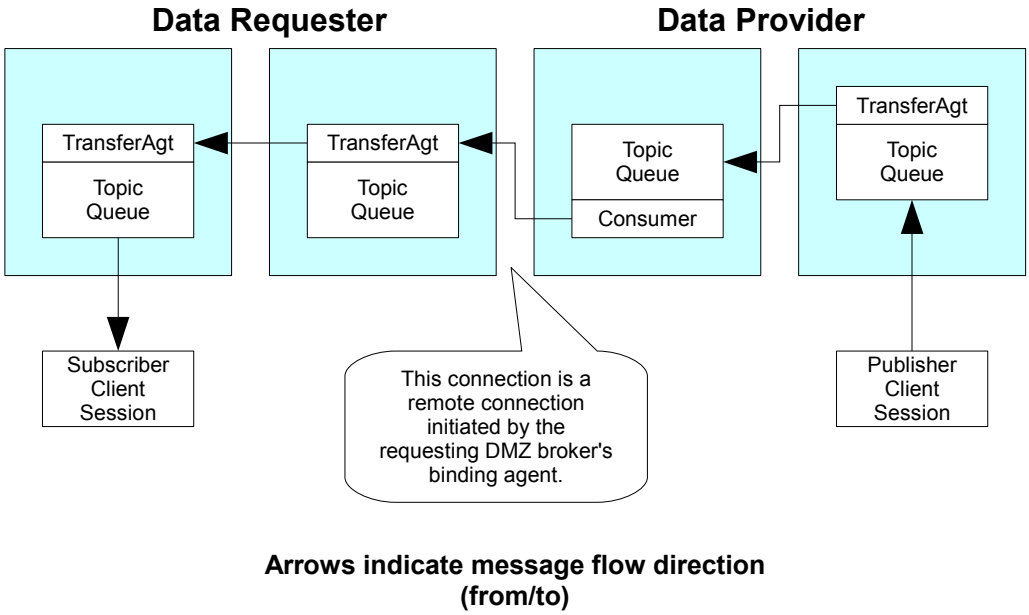
Transfer Agents must authenticate with the remote Broker, and the Message published the Message on the receiving Broker will be associated with the User account of the Transfer Agent as configured in the receiving Broker.

In the receiving Broker, the Message is processed for delivery to Queues via Links just as usual; or indeed further relaying.

Between firms, it is envisaged that DNS SRV records will be used to identify the gateway host providing AMQP services to the inbound connections from the Internet.

AMQP Global Addressing – Internet Subscription





Walk Through...

The aim of Global Subscription Relay is to enable de-queuing of information from a well known Address Queue within another organisation, whilst adhering to good Internet Security practices.

The direction of connection initiation should always be from higher trust zones to lower trust zones.

Because of the active nature of this activity, concerns to robustness and prevention of resource abuse are paramount.

To this end, the data provider administratively creates a Queue to act as a sink for events published (3) on a Source Broker (2).

Bindings would then be used to “push” all Messages published to the Topic into a Broker in a DMZ (1).

The broker in the DMZ (1) will hold Messages on its Topic queue awaiting the client DMZ broker (4) to connect to it.

The provider DMZ broker (1) has an administratively created Consumer with a well known name and appropriate ACL awaiting the client connection.

The client DMZ Broker (4) connects to the provider DMZ Broker (1) using a Transfer Agent where the “from” end is remote.

The client DMZ broke (4)r then uses normal connection semantics to replay and dequeue messages from the provider DMZ broker (1).

Once the messages are in the client DMZ broker (4) they can be similarly propagated into the clients LAN environment (5) for use by the ultimate subscribing clients (6).

Strictly speaking Brokers (5+2) are not needed, however, they will be commonly created to ease administration of onward data flows unless latency requirements are demanding.

NOTES:
Check vs nntp subscription model, imap model. Key difference how current is determined/stored and how replay is handled.

AMQP as a Directed Graph

(NOT READY FOR REVIEW)

Process
Process graph breath first.
For each Source, for each Binding
The Binding Acquires a message from the Source; this is the Current Binding Message

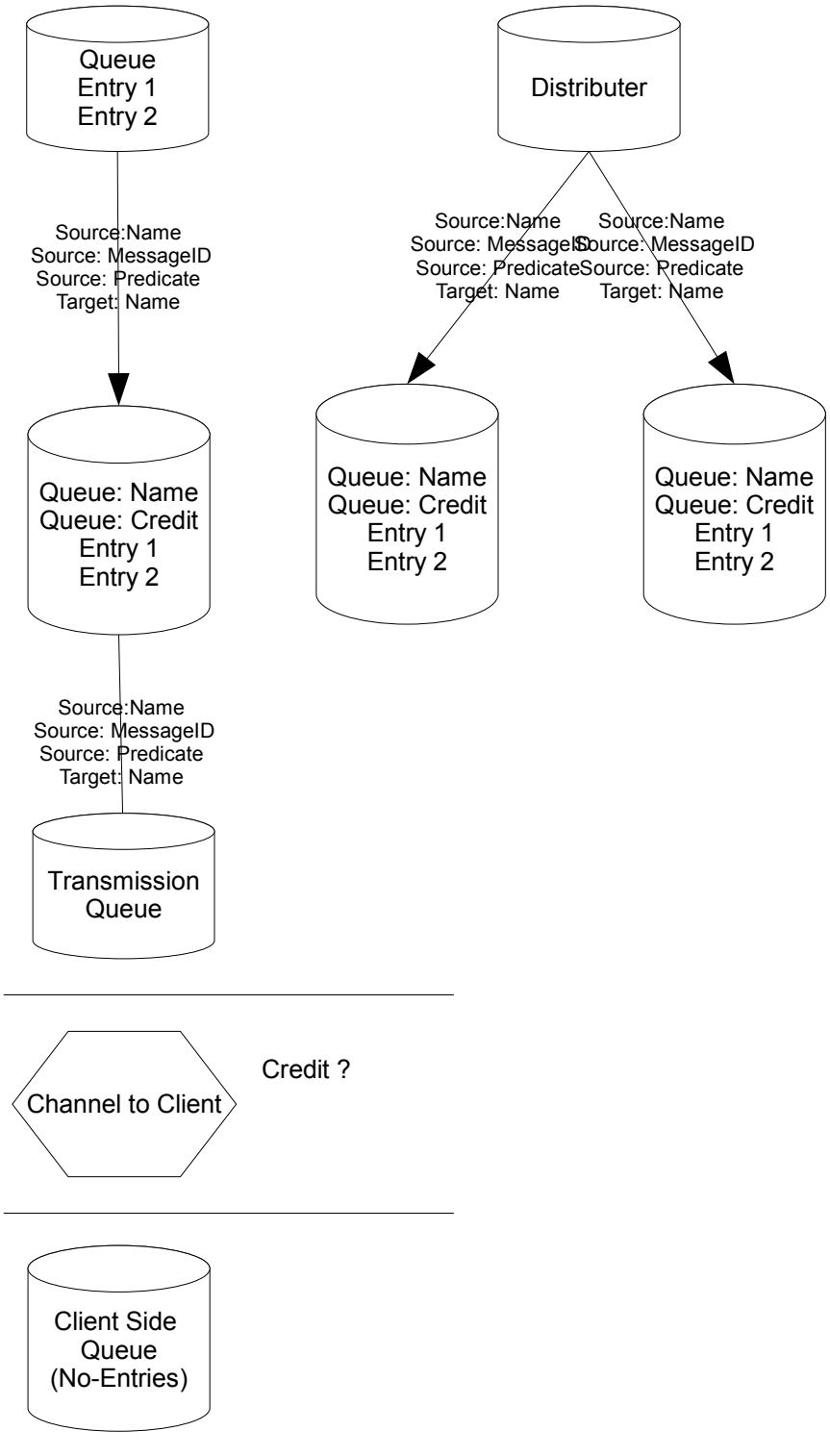
For any given Link to Flow:

Predicate & All(Credit)

Each Binding must maintain its own source queue cursor.

Questions: Persistence of Binding information.

Arrows indicate message flow direction (from/to)



Bindings Acquire Message from Source
Identifies Sinks
Checks Credit Availability on Sink
Copies Message to Sink if Credit
Decrement Credit in Sink
Ack Acquired Message in Source

Garbage Collect All Sources and free up Credit

Repeat for all Links

Client → Server: Flow 1
{Server Side Representation of Channel:
Credit = 1

PROCESS GRAPH