Section I - Broker behaviour

1 Rationale

1.1 What is a broker?

A broker is, generally speaking, a trusted intermediary. For an AMQP 1.0 broker (henceforth just "broker") this entails

- taking responsibility for messages on behalf of clients
- acting as a transaction resource, and possibly transaction co-ordinator
- routing and distributing messages

An AMQP broker is also a computer system, and as such, has operational aspects, some of which may be usefully standardised. For example, means of configuration and monitoring are set out in the Management specification.

Implementers need the behaviour of a server to be specified such that AMQP 1.0 can be retrofitted to legacy messaging solutions, or incorporated in non-traditional solutions. Applications need behaviour to be tightly-enough specified that they can reliably write useful applications that can be counted on to function correctly. These needs motivate our definition of a messaging intermediary model, outlining a set of requirements that can characterise the behaviour of existing systems, and allow general purpose AMQP 1.0 clients to interact with them.

There is also the matter of what it is to be an AMQP broker specifically; this is covered in the broker model, which provides a set of primitives covering most messaging use cases, building on the intemediary model.

1.2 Relation to other specifications

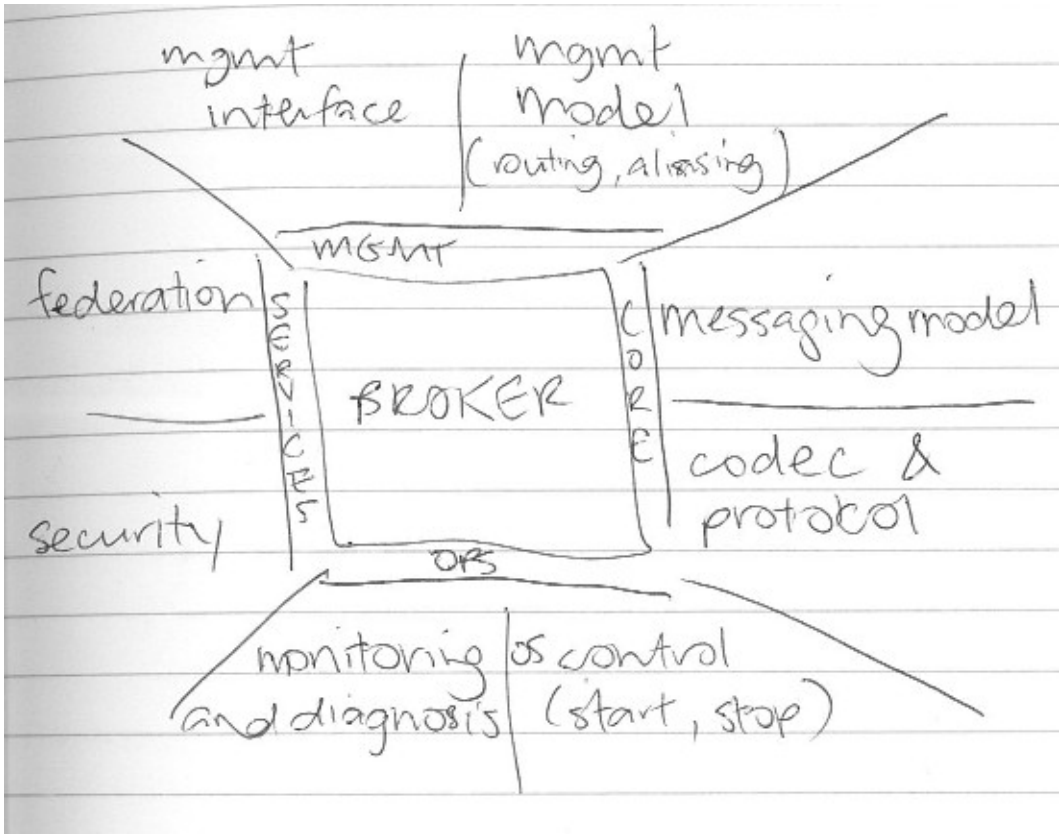
Books II and III define a type system and codec; an abstract protocol, using the type system, for transferring messages between peers; and a concrete protocol, using the codec, over TCP.

This document adds to the model the notion of an intermediary, and a model for trusted intermediation of message transfer. This messaging intermediary model supplements the messaging model primitives with attributes representing application and operational concerns; for example, persistence of messages, and ordering semantics.

In AMQP transfer protocol terms, a broker provides an TCP/IP server to service Connections, and keeps track of Sessions and Links created by the client. It resolves sources (for outgoing links) and targets (for incoming links), and fulfils certain contracts with respect to acknowledgement.

> Relation to management: that spec defines the mechanism for manipulating the broker model, and all sorts of operational controls.

A broker may also provide services for applications that are orthogonal to the messaging model; for example, authentication and federation. These are not discussed here.



<!-- Spiffy version of multi-facet diagram goes here →

1.3 Relation to previous versions of AMQP, and other messaging specifications

Talk about trying to admit all the useful messaging patterns; broadly speaking, work distribution, relaying and pub/sub.

In general, we try to account for the important bits, and admit the other bits. E.g., a 0-9-1 exchange has these 1.0 attributes, and some additional semantics that aren't ruled out by 1.0.

2 Messaging intermediary model

The messaging intermediary model introduces a set of attributes for describing messages, links and nodes that address messaging network concerns. Central to this model is the idea of taking responsibility, and its implications of persistence and buffering.

2.1 Message format

Book IV describes a message format that includes transport headers (values important for delivery) and message properties (immutable properties of the message). In general it is not necessary to discuss these here, other than to require that intermediaries must not change message properties; specific properties with implications for intermediaries are mentioned in the sections below.

> Define only what is necessary to support what is described below; probably not much besides headers e.g., return-to, reply-to (sans semantics).
>
> ".. and may change transport properties."? Do any any these make sense to change?

2.2 Responsibility and transfer failures

The messaging model admits varieties of responsibility transfer: at most once, at least once, and (to an approximation) exactly once. The other end of the link may disappear without responding and not come back (i.e., time out); we must account for what happens to messages that have been committed for transfer over that link, but which are not known to have been transferred successfully.

Generally speaking, there are four options:

1. Forget about the delivery;
2. Try to deliver the message to some other link from the node, or requeue it;
3. Send the message to a "Dead letter queue";
4. Return the message to the publisher, given a suitable return route.

The source type in book IV has an orphan-outcome field. This should be respected in the absence of overriding configuration; however, note that it may affect delivery semantics.

> Orphan-outcome "release" means anything could happen, basically.
>
> It's worth noting these are not all mutually exclusive; it may be desirable, for instance, to respect "return-to" as well as dead-lettering when it's not present.
>
> What needs to be required here, if anything?

2.3 Buffering and ordering

Taking responsibility for messages on behalf of peers downstream implies buffers. However, there is no inherent sense of ordering, except to say: Putting aside transfer failures, returns, and routing which introduces races, messages should observably propagate over links and nodes in the order in which they are published.

This constrains the order of message propagation over a single route; however, we may also wish to serialise messages at a node such that all outgoing links observably receive the messages in the same order. Since this is not always necessary, we make this an attribute of nodes.

Priority indicates that a message may "overtake" other messages with lower priority; i.e., be delivered before despite being published after. It is otherwise not defined here.

2.4 Message durability and TTL

Informally speaking, an intermediary should make sure no message is lost unless it knows it does not need to. Sending a *non-durable* message effectively means "Please trade recoverability for speed"; it is a hint that the intermediary can optimise for this case, e.g., by acknowledging straight away, avoiding hitting the disk, and not requiring acknowledgement downstream.

> This is an operational concern. If memory was orthogonally persistent, there would be no need for durable flag – it's only meaningful when you want to turn it off.
>
> TODO: TTL – discard if the message is not what? Transferred?
>
> "Transmit-time" obviously is problematic.

2.5 Lifetime and exclusivity

Containers delimit lifetime for nodes; i.e., a node located at a container necessarily is destroyed when the container is destroyed. We may wish to give nodes a lifetime delimited by the applications using the node.

For this purpose, a node may be marked as *autodelete*, which means it may be destroyed as soon as the last incoming or outgoing link is detached.

> Another possibility is to mark a node as *exclusive*, restricting its visibility to a particular session. In this case, the lifetime of the node is necessarily bound to the session lifetime.
>
> TBD: Should this go in broker model maybe.

2.6 Browsing

Various use cases require the ability to receive messages from a node without interacting with its distribution of messages. In book IV this is encoded by link sources specifying whether they will move or copy messages; move meaning that the link is considered when distributing messages, and copy meaning that it is considered in addition to distributing messages. Intermediaries may or may not support this mechanism.

> TODO: semantics of outcomes when copying

2.7 Transactions and acknowledgement

A messaging intermediary always acts to settle outcomes; either from senders, by accepting transfers with a settled state, or from receivers, by settling the posited outcome. In the absence of overriding configuration, an intermediary should propagate the delivery semantics with a message; e.g., if an

acknowledgement is required by the producer, the intermediary should require acknowledgement from consumers.

An intermediary may act in the roles of transactional resource and co-ordinator, enacting units of work atomically. What “atomically” entails depends on the properties of the source or target node.

Acknowledgement and transactions are intimately related; an intermediary must not send a settled disposition for a message transfer until it has successfully completed the (possibly implicit) transaction in which the message transfer is enlisted.

Messages that the intermediary has transferred to a consumer as part of a transaction must be redelivered if the transaction fails; the precise behaviour depends on the node.

3 Broker model

The broker model extends the messaging intermediary model to add requirements specific to an AMQP broker. Since it does not presuppose an incumbent model (as for messaging intermediaries in general), this layer necessarily takes a position on what messaging primitives are useful for application developers.

The classes of behaviour that nodes at a broker must support are as follows: topics, that route messages to links; and queues, that distribute messages among links. Both may in general be addressed directly by both producers and consumers. Being so addressed by a consumer as a source may imply buffering on behalf of the consumer.

In the following subsections, “broker node” is understood to refer to AMQP 1.0 nodes so far as they are defined within this model. We use “link” to refer to an abstract message flow relation (this includes both publishing relations, consuming relations, and routing relations), and outbound and inbound refer to links to consumers and from producers respectively.

3.1 Names

A broker must resolve addresses given in link targets and link sources consistently along with addresses used elsewhere – e.g., in a management interface – such that addresses shared by applications observably refer to the same broker node. This is to ensure, for example, that an application can use an address generated by the broker as a reply-to header can expect any replies to appear along a link sourced at the address.

How can this be expressed more formally?

The simplest interpretation of this is that source and target names are in a single namespace and each broker node has a single name by which it may be addressed.

Aliasing: not exposing the internal structure. Semantics should be easy – aliases can be resolved at link time. What are the most useful semantics for destroying or moving aliases? Can aliasing be independent of, e.g., implicit buffers. Aliases should be admitted but perhaps not required.

In circumstances in which the broker is obliged to generate a name, the name should be randomly generated; i.e., it should be hard to guess.

3.2 Topics and Queues

A broker node either *routes* each incoming message to all links (that will accept the message); or, for each message chooses a single link (that will accept the message) to which to *distribute* the message.

Routing behaviour is identified with Topics. Distributing behaviour is identified with Queues. The behaviour implies a difference in acknowledgement semantics.

Topics equate outcomes to transmission acknowledgement; as such, only *accept* has a defined meaning, which is that the transfer in question must not be made along the link again. The semantics in the event of *reject*, *release* and *modified* outcomes is left undefined.

Queues, on the other hand, treat outcomes as instructions. Specifically,

- *accepted* instructs the queue to not transfer the message again;
- *release* instructs the queue to redistribute the message;
- *reject* instructs the queue to invoke rejected message handling e.g., sending the message to a dead letter queue;
- *modified* instructs the queue to reconsider the message for distribution.

3.3 Bindings

Bindings represent relations between broker nodes. Bindings may have filters appropriate to the node named as the source, and specify whether messages are to be copied or moved from a queue.

> Bindings are considered for distribution or routing in the same way as outbound links. The outcome of transfers along bindings are, implicitly, *accept*; i.e., responsibility implicitly transfers to the target.

3.4 Visibility

A node may be for the purpose of accepting messages from publishers, routing messages, delivering messages to consumers, or some combination therein. Visibility controls the (transfer protocol) link roles – source and target – in which a node is allowed to be addressed.

Visibility takes the values

- Outbound (addressable by outbound links only)
- Inbound (addressable by inbound links only)
- Both (addressable by outbound and inbound links)

and defaults to Both.

> TODO Suitable errors

4 Services

4.1 Management

TODO: Wait and see what happens with the management spec.

5 Examples

5.1 Exchanges and Queues

5.2 Pub/Sub

5.3 RPC

5.4 Zero-order hold