OOI-CI Prototype Exchange-Space Implementation Report

Tony Garnock-Jones <tonyg@lshift.net>

5th January 2010

Abstract

Final summary of the prototype phase of the OOI-CI RabbitMQ integration project, listing deliverables, design alternatives, and open issues.

1 Introduction and background

This document reports on the outcome of the RabbitMQ part of the prototype OOI-CI AMQP messaging work. It should be read as a companion to the following documents:

- The RabbitMQ OOI Statement of Work;
- The OOI-CI Prototype Exchange-Space API documentation; and
- General OOI-CI context documents on exchange-spaces, exchange-points and federation.

There are three main deliverables: a preconfigured RabbitMQ EC2 AMI; support for AMQP 0-9-1 in the RabbitMQ server and selected client libraries; and a prototype implementation of exchange-spaces and exchange-points, along with documentation and examples. Each is briefly described below, along with a summary of its status and pointers to further information.

2 RabbitMQ EC2 images

A collection of RabbitMQ AMIs is now available for use. Links and instructions are available at `http://www.rabbitmq.com/ec2.html`. The maintainer of the AMIs is Marek Majkowski (majek@lshift.net).

- Both 32- and 64-bit images are available.
- Images are based on Ubuntu 9.10.
- RabbitMQ v1.7.0 is preinstalled.

3 AMQP 0-9-1 interoperability

RabbitMQ's support for the 0-9-1 variant of the AMQP protocol is now largely complete. Three-way interoperability between RabbitMQ, Qpid, and OpenAMQ has been demonstrated through the use of standard off-the-shelf examples shipped with each system, and also through the 0-9-1 test suite developed by Michael Bridgen (mikeb@lshift.net). The status of our interoperability testing is kept up-to-date at `https://dev.rabbitmq.com/wiki/Amqp091Interoperability`.

Support for 0-9-1 is available on the `amqp_0_9_1` branches of the various RabbitMQ mercurial repositories. The following shell commands check out and build an 0-9-1-compatible broker, C client, and Java client:

```
hg clone http://hg.rabbitmq.com/rabbitmq-codegen
hg clone http://hg.rabbitmq.com/rabbitmq-server
hg clone http://hg.rabbitmq.com/rabbitmq-c
hg clone http://hg.rabbitmq.com/rabbitmq-java-client
(cd rabbitmq-codegen; hg up amqp_0_9_1)
(cd rabbitmq-server; hg up amqp_0_9_1)
(cd rabbitmq-c; hg up amqp_0_9_1)
(cd rabbitmq-java-client; hg up amqp_0_9_1)
make -C rabbitmq-server
(cd rabbitmq-c; autoreconf -i; ./configure; make)
(cd rabbitmq-java-client; ant dist)
```

At this point, you can start the server,

```
make -C rabbitmq-server run
```

use the java client library in `build/dist`,

```
(cd rabbitmq-java-client/build/dist; \
  sh runjava.sh com.rabbitmq.examples.SendString)
```

and run the C example programs:

```
./rabbitmq-c/examples/amqp_sendstring
```

To run the 0-9-1 test suite,

```
ant test-suite-prepare test-functional
```

The results of the suite will be available in `build/TEST-*.txt`.

4 Exchange-space prototype implementation

The prototype takes the form of a new RabbitMQ plugin, available via git on `amoeba.ucsd.edu`, and a set of

supporting modifications to the RabbitMQ server. Instructions for building and running the prototype are available in the file `README.md` in the plugin source code checkout.

4.1 Design alternatives

Three potential alternative designs were considered:

Layered NIPCA-style networks. [1]As part of the background research for this project, the idea of unifying AMQP with NIPCA-style network operations was discussed extensively. This is still an extremely promising line of investigation, and should be revisited in future phases of the project because of its formidable descriptive power.

Exchange-space extensions to the AMQP vocabulary. An experimental new class, `ExchangeSpace`, was considered. It would have permitted full control over exchange-space functionality without being restricted by limitations of the existing AMQP vocabulary; however, it would have also required changes to client libraries in order for them to use the new functionality, which would have made experimenting with the new features prohibitively difficult.

New AMQP exchange types. This was the option chosen in the end: it helped draw out the similarities and differences between the existing AMQP vocabulary and the vocabulary required for controlling exchange-spaces, and worked seamlessly with existing AMQP client libraries. While it does not result in a perfect fit, leaving some exchange-space features unaccessible to AMQP client libraries (but controllable via out-of-band methods), it does permit flexible and rapid experimentation with the new functionality.

4.2 Retrieving and building the code

To retrieve the plugin,

```
git clone \
    git@amoeba.ucsd.edu:exchange_space_prototype.git
```

The changes to the RabbitMQ server are available on the branch named `bug22019`. For more information on retrieving, building, configuring and running the system, see the plugin's `README.md` file.

Example programs using Pika[2] are provided in the `examples` directory of the plugin source code checkout, but similar examples can be written for any AMQP client library that can support the provision of custom exchange-type names and exchange declaration argument tables; the prototype was specifically designed to be compatible with unmodified existing AMQP client libraries. See the prototype API documentation for more information.

4.3 Open issues and next steps

The experimental implementation, with its new exchange-types and otherwise unmodified AMQP vocabulary, is a stop-gap on the road to full integration between AMQP and NIPCA-style concepts. A synthesis of the two systems seems likely to be able to uniformly and flexibly describe and manipulate many common configurations of networks small and large, including several that are not amenable to description in any manner within the bounds of today's network programming APIs.[3] Many open questions in this area remain; among others:

- What is the relationship between associations (roughly analogous to connections) and subscriptions?
- What is the relationship between associations and individual messages?
- What is the relationship between name-binding, directory, and subscription?
- What is the relationship between AMQP's exchanges and queues and NIPCA's names and addresses?
- Where do applications and application instances fit in AMQP? How do they relate to applications and application instances in NIPCA?

In terms of the experimental implementation itself, one interesting issue is that the existing AMQP 0-9-1 vocabulary is not rich enough to describe all exchange-space and exchange-point operations: in particular, it can neither describe options for managing buffers and transfer-of-responsibility within the network nor express the difference between exchange-point deletion and exchange-point unsubscription. If the current implementation is to be taken forward, either an out-of-band control channel will need to be provided for controlling such things, or the core AMQP vocabulary itself will have to be altered, perhaps in a manner guided by future work on the AMQP/NIPCA synthesis.

[1]Network IPC Architecture, NIPCA: a powerful network architecture described in John Day's book, *Patterns in Network Architecture*, 2008.

[2]A python AMQP client library. `http://github.com/tonyg/pika`

[3]*"Whereof one cannot speak, thereof one must be silent."* Firewalls, just to pick one important example, are not represented in the BSD sockets API.

The experimental implementation provides only the barest rudiments of a system for management and monitoring of exchange-spaces and exchange-points. Future work will need to explore the options for improving the management and monitoring situation.

Finally, federation is one of the areas being worked on as part of the development of the AMQP 1.0 specification. The design ideas explored in this project could potentially contribute to the design of standardised federation for AMQP, especially in the area of managing the scoping of addressing and naming.

5 Conclusion

All the project deliverables are complete, with respect to the goals outlined in the Statement of Work and refined by subsequent email and face-to-face communication. OOI's internal prototyping work can proceed, based on the experimental exchange-space implementation provided, and clear lines of potential future investigation exist.