A MESSAGING-BASED DATA ACCESS LAYER FOR CLIENT APPLICATIONS*

James Patrick#, Fermilab, Batavia, IL 60510, U.S.A.

Abstract

The Fermilab Accelerator Control System [1] has recently integrated use of a publish/subscribe infrastructure as a means of communication between Java client applications and data acquisition middleware. This supersedes a previous implementation based on Java Remote Method Invocation (RMI). The RMI implementation had issues with network firewalls, misbehaving client applications affecting the middleware, portability to other platforms, and lack of authentication. The new system uses the RabbitMQ implementation of the AMQP messaging protocol and broker architecture. This decouples the client and middleware, is more portable to other languages, and has proven to be much more reliable. A Java client library provides for single synchronous operations as well as periodic data subscriptions. This new system is now used by the synoptic display manager application as well as a number of new custom applications.

FERMILAB CONTROL SYSTEM

The Fermilab accelerator control system, generally referred to as ACNET, is a unified system controlling all accelerators in the complex. This includes the primary accelerator chain delivering beam to physics experiments, and well as test facilities for future accelerators based on superconducting RF cavities. It is a three tiered system with front-end, central service, and application layers. Front-end computers directly communicate with hardware over a wide variety of field buses. Console applications provide the human interface to the system. Central service computers provide general services such as a database, alarms, application management, and front-end support. Communication between the front-end and central layers is carried out using a connectionless protocol also named ACNET over UDP. Data from front-ends is then delivered by the central layer to applications by either a shared memory protocol for C-language applications that run within a console, or the Java Remote Method Invocation (RMI) protocol for Java language applications.

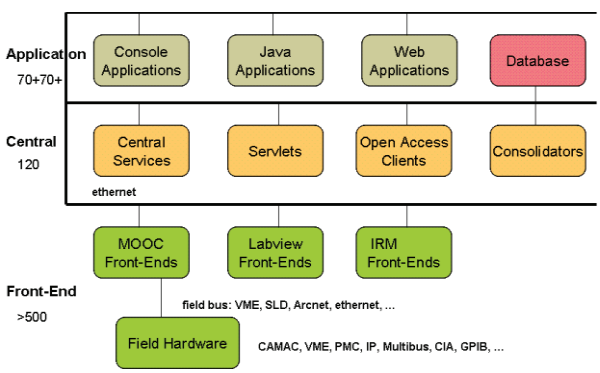

Figure 1. Architecture of the Fermilab Control System.

A goal of the control system is to allow applications to be run from anywhere by a properly authenticated user. The ACNET communication protocol is not suitable for use outside of the control system network. See Fig. 1.

The Java RMI implementation had several significant issues. The central layer Data Acquisition Engines (DAEs) deliver data to clients by a callback mechanism. The port and network interface used could vary so it was challenging to develop firewall rules for systems outside the core network. The system did not have a proper load balancing mechanism to distribute data acquisition jobs among the DAEs. There was no failover mechanism if a DAE stopped functioning. And there was no way to authenticate users running application programs on non-control system computers.

To address these issues, a system was developed ("Secure Controls Framework") with an additional server between the application and DAE simplifying firewall issues and requiring authentication for connections. Also a prototype implementation was done using the ZeroC "ICE" product. Neither of these proved satisfactory. It was decided to try the publish/subscribe messaging paradigm, and use a system that supported clients in multiple languages unlike the Java Messaging Service (JMS).

AMQP AND RABBITMQ

The AMQP (Advanced Message Queuing Protocol) is a standard that defines a publish/subscribe messaging system based on a central broker architecture. Producers send messages and consumers receive them. Messages are sent to "exchanges" defined in the broker and routed to "queues" based on an associated key or topic. The standard defines several possible strategies for the message routing. Consumers bind to a queue and may then receive messages from that queue.

The advantage of this architecture is that producers and consumers need not know each other's exact identity or computer host, they just need to know the message routing key or topic. The infrastructure also supports recovery of broken connections if a broker is restarted. Firewalls need only pass traffic for a single port for the few broker machines. It also supports clusters of brokers with load balancing and failover.

A disadvantage of this architecture is the additional latency incurred in going through the broker for all data transfers. The applications that use the system do not perform time critical control operations so this is not a problem. Another issue has been the evolution of the

*Fermilab is operated by the Fermi Research Alliance under contract to the US Department of Energy
#patrick@fnal.gov

ISBN 978-3-95450-139-7

AMQP standard. Prior to version 1.0, the standard defined both the wire protocol and the broker architecture. AMQP 1.0 is simpler, doing away with exchanges and queues and primarily defines the wire protocol. This work uses AMQP versions 0.8 and 0.9.1. The implementation selected is RabbitMQ [2]. This includes a broker implementation written in the Erlang language as well as client APIs in several languages including Java. RabbitMQ is an open source product now owned by VmWare. The RabbitMQ broker is relatively compact and by default does not support other protocols. Its performance is excellent compared with other implementations. RabbitMQ plans to support the pre-1.0 versions of the AMQP protocol indefinitely.

USAGE IN ACNET

The control system is configured with three RabbitMQ brokers each connected to two data acquisition engines. DAEs subscribe to a topic associated with new data acquisition requests. The application client creates a new dedicated exchange/queue for a request, and then publishes a "Request Init" message to the DAE via "topic" exchange in a randomly selected broker. Requests can be for single readings or settings, or a subscription for periodic data from some device or set of devices. The DAE performs the operation on behalf of the client and sends status and data back to the client via the dedicated exchange.

We do not use the broker clustering feature in AMQP as attention has to be paid to the distribution of and resources required by data acquisition jobs running on the DAEs. Primitive load balancing and clustering is done by choosing a random broker for the init message. The low level client software sets up a heartbeat mechanism between the client and DAE. If the client stops receiving data for some timeout period, it will automatically restart the data acquisition job on a different DAE. If the DAE detects the client is no longer present, it will stop the data acquisition job and release the associated resources. Thus the system can deal with a missing broker and/or DAE and jobs will transparently survive restarts of either.

To authenticate users, a standard Fermilab Kerberos ticket is passed from the client to the DAE as a property in the message header. The DAE validates the ticket and determines if the user is authorized to access the control system, and has the appropriate privileges to set devices if requested. Although AMQP supports authentication of broker connections we do not use this feature. SSL connections to the broker are also supported by AMQP but again we do not currently use them.

Structured messages are serialized using the ACNET Protocol Buffer system [3] rather than AMQP system. This is done to be consistent with other message passing functionality in the control system.

The DAEs make available internal statistics and information on the jobs they are running via the Java Management eXtensions (JMX) mechanism. A set of web pages provides an overall summary of the system and can provide detailed information on each job. The RabbitMQ broker also exposes management information via http queries, development of web pages to display this information customized for our usage is in progress.

ACNET APPLICATIONS

The synoptic display system in ACNET [4] now uses the RabbitMQ based data acquisition exclusively. These are displays generated by a drag and drop builder. Usage of this package has dramatically increased in recent years. Also a standalone client library in Java for custom applications is available. A variety of applications have been written using that including camera data acquisition and analysis, beam steering for the Fermilab linac, and a general waveform plotter. This standalone library is trivially accessible from Matlab, and several substantial applications have been written using that.

EXPERIENCE

This messaging system has proven to be much more robust in transferring data between DAEs and clients than the previous Java RMI based system. The primary issues with that system are resolved. Most problems encountered have been with errors in our implementation. The core RabbitMQ system has worked extremely well. As usage of the system has increased, we have experienced problems with excessive load on the DAEs and/or brokers. As data acquisition requests may contain a wide range in number of devices, data sizes, and rates, a better load balancing system is needed than randomly distributing jobs to brokers. Work on this is in progress.

CONCLUSIONS

A publish/subscribe messaging system based on RabbitMQ has been implemented for the Fermilab accelerator control system ACNET to transfer data between middle layer processes and high level applications. This provides a robust, secure, firewall friendly method of accomplishing this and is a great improvement over the previous method based on Java RMI.

ACKNOWLEGEMENTS

The original implementation of this messaging layer was done by Andrey Petrov.

REFERENCES

[1] Cahill, K., et al, "The Fermilab Accelerator Control System," ICFA Beam Dyn. Newslett. 47 (2008) 106-124 FERMILAB-PUB-08-605-AD.

[2] RabbitMQ: http://www.rabbitmq.com

[3] R. Neswold and C. King, "Generation of Simple, Type-Safe Messages for Inter-Task Communication," ICALEPCS'2009, Kobe, Japan, October 2009, TUP024, p. 137 (2009); http://accelconf.web.cern.ch/accelconf/icalepcs2009/papers/tup024.pdf.

[4] A. Petrov and T. Bolshakov, "Drag and Drop Display Builder", ICALEPCS'2007, Knoxville, TN, USA, October 2007, ROPB03 (2007).

ISBN 978-3-95450-139-7