ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THPHA198

DEVELOPMENT OF MQTT-CHANNEL ACCESS BRIDGE

J.Fujita[†], M. Cherney, Creighton University, Omaha, NE, USA
D. Arkhipkin, J. Lauret, Brookhaven National Laboratory, Upton, NY, USA

Abstract

The integration of the Data Acquisition, Offline Processing and Hardware Controls using MQTT has been proposed for the STAR Experiment at Brookhaven National Laboratory. Since the majority of the Control System for the STAR Experiment uses EPICS, this created the need to develop a way to bridge MQTT and Channel Access bidirectionally. Using CAFE C++ Channel Access library from PSI/SLS, we were able to develop such a MQTT-Channel Access bridge fairly easily. The prototype development for MQTT-Channel Access bridge is discussed here.

INTRODUCTION

Most of the STAR (Solenoidal Tracker At RHIC) Experiment Control System has been based upon the EPICS (Experimental Physics and Industrial Control System) from the beginning. Currently roughly 60,000 parameters are controlled and monitored with EPICS. Recently, MQTT [1] has been chosen to integrate the STAR DAQ, Offline, and Control Systems. As there was no MQTT-Channel Access bridge available, we needed to develop one.

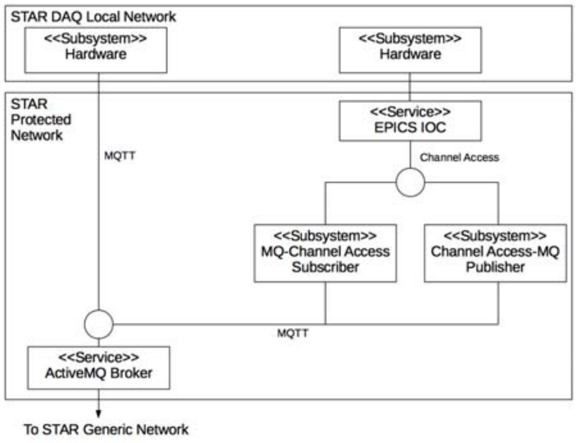

Figure 1: Schematics of the STAR Control Integration plan.

Figure 1 shows the schematic diagram of the STAR Integration plan. [2] The MQTT Broker further publishes the data to the STAR Generic network and the data will be accessible via WebSocket such that it can be viewed via a web browser.

At the moment, the existing EPICS infrastructure will remain as it is. In the STAR Control Room, we use various EPICS tools such as Alarm Handler [3], MEDM [4], and Control System Studio [5] during the commissioning and the data-taking period of the experiments as we have been doing for the past 17 years. EPICS to MQTT is a first stage transition but new detector sub-systems seem to be going the native MQTT way.

MESSAGE QUEUE TELEMETRY TRANSPORT: MQTT

MQTT was initially developed by IBM and Eurotech in 1999. Originally, it was intended for an oil pipeline to satellite communication link. It was internally used and maintained by IBM until 2010. In 2010, the version 3.1 was released royalty free. In 2013, International Standards Organization OASIS begin officially advocating MQTT as a lightweight, open source solution for device to device communications. In 2016, ISO officially approved MQTT version 3.1.1 as an ISO standard (ISO/IED 20922). MQTT has been very popular choice among Internet of Things (IoT) as the communication protocol.

MQTT protocol typically runs on top of TCP/IP as well as UDP and Zigbee. It is relatively simple and easy to implement. It is lightweight and bandwidth efficient. It is based upon a publish and subscribe architecture and the Quality of Service (QoS) is built-in to the protocol. Unlike Channel Access, it requires a Message Broker that functions as a communication hub on the network. In MQTT, the data are called as messages. The message is in ASCII based format. MQTT also use topic as filter and categorize the messages in the broker. The topic could be used by the clients to get only the information they need. Figure 2 is the MQTT schematic diagram illustrating the overall MQTT concept.

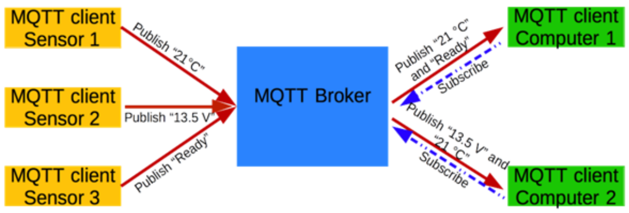

Figure 2: MQTT concept diagram.

MQTT-CHANNEL ACCESS BRIDGE

The prototype MQTT-CA bridge was written using standard Channel Access C library. While it is possible to accomplish the same in Python or Java, C was chosen, as that provided more powerful sets of libraries as well as the request from one of the STAR experts from a different system. Since we needed an MQTT library, we chose the Paho [6] library, which supports C/C++ as well as Python, Java and many others. For EPICS Channel Access, we used the standard EPICS portable Channel Access C library that comes with EPICS Base 3.14.

The message broker needed for the MQTT had already been chosen by the time the Control Group was involved for the project; the STAR Experiment has adopted Apache ActiveMQ Apollo [7] for the message broker.

The first prototype was written in about two weeks, including the time to setup the broker and understand the basics of how MQTT works. We started with very little knowledge in MQTT, but were able to write something that worked fine in a relatively short time. This also was the first time the STAR Control Group wrote a standalone application with the Portable Channel Access C Library.

The MQTT-Channel Access bridge consists of two parts; the Subscriber Component and the Publisher Component. In both cases, in the MQTT message is formatted in JSON format.

The Subscriber Component subscribes the data values via MQTT, then, publishes the value to EPICS Channel Access, so that it could be monitored and archived using the EPICS tools. Figure 3 shows the how the MQTT message is converted to an EPICS database record by the Subscriber component.

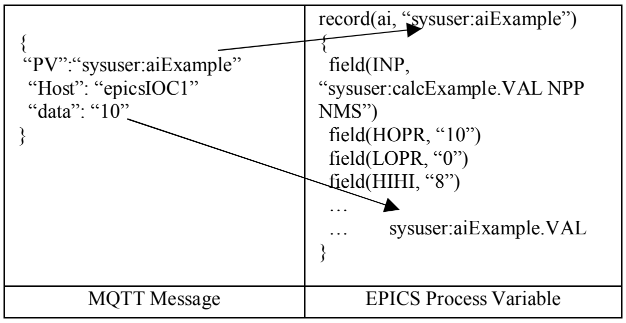

Figure 3: MQTT-Channel Access Subscriber Component Schematic.

The Publisher Component gets the EPICS Process Variables via Channel Access, and then publishes the values to the MQTT Broker. This is basically the reverse of the Subscriber Component. Figure 4 below shows how the EPICS database record is converted to the MQTT message.

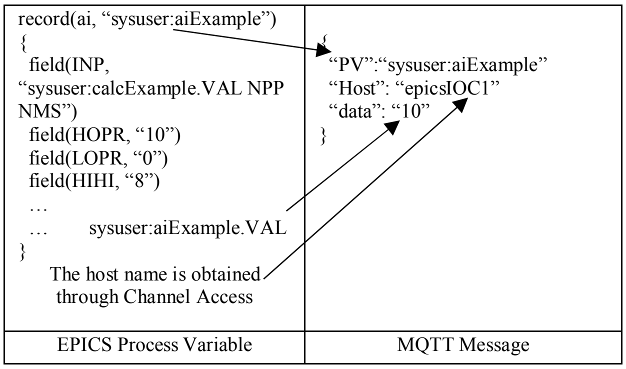

Figure 4: MQTT-Channel Access Publisher Component Schematic.

CAFE

Channel Access bridge. Since the Control Group wants students to participate in the project, it is important to make the code easily understandable by the students involved in the research project. Having worked with the EPICS standard portable Channel Access library in C, the group felt that it was not necessary the easiest library to use. In addition, we have found that for many students, C++ is a preferred language of choice over C.

As we reviewed the different options for the Channel Access library, we came across CAFE (Channel Access interFacE) [8]. It is a C++ interface that provides a modern, multifaceted interface to the EPICS Channel Access library. Once compiled, the CAFE library has proven to be one of the easiest libraries to use, as most function calls behave just like any other from standard C++ libraries.

While rewriting the code with CAFE took some time for the Control Group to accomplish, we were able to complete the task in a month or so. Most of the issues encountered had to do with to understanding what function call was needed for CAFE in C++.

PERFORMANCE EVALUATION

Some performance testing has been attempted during the prototype phase. We prepared four EPICS IOCs on two Linux computers and one VME CPU running RTEMS on one network subnet. One of the Linux computers also had the MQTT-Channel Access bridge programs installed. On a different subnet, we prepared the MQTT broker. On the third subnet, we prepared MQTT clients that receive and send the data.

Since this involved three separate subnets and several computers (and a VME board), we were not able to make a quantitative measurement that would have meaning outside of this specific context. The MQTT-Channel Access bridge programs operated successfully at a rate of 1000 messages/second, which is exceeds the current requirements of the STAR Control System.

In the current production system, around 5000 messages per second are going through the system at peak times, and about 1000 message per second on average.

CONCLUSION

MQTT allows the integration of the existing STAR Control with the rest of the STAR systems. Given the simplicity of MQTT, it was not very hard for a MQTT beginner to implement into a standalone portable Channel Access application. By using CAFE, we believe we were able to lower the initial learning curve of Channel Access for inexperienced students to some level of Channel Access programming. Using MQTT also possibly allows the deployment of different devices such as IoTs to be easily incorporated into the existing control system.

ACKNOWLEDGEMENT

This work was supported in party by the Office of Science of the United States Department of Energy and the College of Arts & Science of Creighton University. We would also like to thank the STAR Collaboration and the EPICS Community. In particular we would really like to thank Dr. Jan Chrin from Paul Scherrer Institute for working on CAFE library.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THPHA198

REFERENCES

[1] MQTT, http://mqtt.org/.

[2] D Arkhipkin and J Lauret, "STAR Online Framework: from Metadata Collection to Event Analysis and System Control", *J. Phys.: Conf. Ser.,* vol. 608, p, 012036, 2015, doi:10.1088/1742-6596/608/1/012036

[3] Alarm Handler, http://www.aps.anl.gov/epics/extensions/alh/.

[4] MEDM, http://www.aps.anl.gov/epics/extensions/medm/.

[5] Control System Studio, http://controlsystemstudio.org/.

[6] Paho, http://www.eclipse.org/paho/.

[7] Apache ActiveMQ Apollo, https://activemq.apache.org/apollo/.

[8] CAFE, https://ados.web.psi.ch/cafe/.