ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOMPR002

IMPROVING USER INFORMATION BY INTERFACING THE SLOW CONTROL'S LOG AND ALARM SYSTEMS TO A FLEXIBLE CHAT PLATFORM

M. Ritzert*, Heidelberg University, Heidelberg, Germany
on behalf of the Belle II PXD Collaboration

Abstract

Research groups operating large experiments are often spread out around the globe, so that it can be a challenge to stay informed about current operations. We have therefore developed a solution to integrate a slow control system's alarm and logging systems with the chat system used for communication between experimenters. This integration is not intended to replace a control screen containing the same information, but offers additional possibilities:

- Instead of having to open the control system's displays, which might involve setup work (VPN, remote desktop connections, ...), a web interface or an app can be used to track important events in the system.
- Messages can easily be filtered and routed to different recipients (individual persons or chat rooms).
- Messages can be annotated and commented on.

The system presented uses Apache Camel to forward messages received via JMS to Rocket.Chat. Since no binding to Rocket.Chat was available, this interface has been implemented. On the sending side, a C++ logging library that integrates with EPICS IOCs and interfaces with JMS has been designed.

IMPLEMENTATION

The gateway combines data from the BEAST alarm system [1] and the message log. Both systems are configured to publish their messages via the ActiveMQ message broker, that is central to the distribution of all messages in the system. The gateway is registered as a listener on the respective channels. It receives all messages, filters them and forwards to the Rocket.Chat server.

All messages are also archived in an Elasticsearch database. Archived log messages from this database are combined with live messages received via JMS to provide a fast and comprehensive overview in the CSS GUI, that is also used to control the alarm system.

The outline of the system is shown in Fig. 1. The gateway between ActiveMQ and Rocket.Chat is implemented as an Apache Camel application. Camel is a message routing engine implemented in Java [2]. Its modular design means that it is easy to provide new functionality on all levels interesting for this work: message reception, transformation, routing and output. Messages received in Camel applications are passed through several different modules connected together

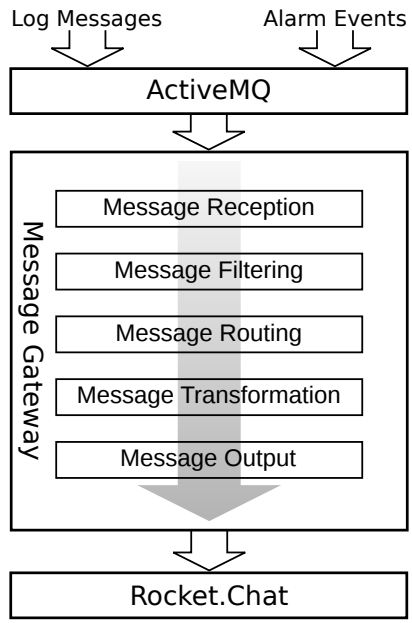

Figure 1: Processing Steps for Log and Alarm Messages.

```
from("activemq:topic:LOG")
.filter().method("LogFilter")
.bean("LogConverter")
.to("rocketchat:https://my.chat:⟵
#channel?accessToken=token&userId=user");
```

Figure 2: Example Camel Route for Log Messages.

to form a route. From the framework's point of view, messages are passed around as anonymous Java objects. It is the responsibility of the application to ensure that the output and input expectations of connected modules match. Modules can discard messages, or modify them to the extend that an object of a different class is passed on.

In a typical configuration, there are at least two routes, one for alarm messages, and another one for log messages. Figure 2 shows the Java code to create a route for log messages from the `LOG` topic to `#channel` on a Rocket.Chat server. The modules described below are implemented in the `LogFilter` and `LogConverter` classes, and made available as the `rocketchat:` output module.

Message Logging

In order to log messages from an IOC, a C++ library, Logfile, has been implemented. The most important functionality is logging to an ActiveMQ server via the STOMP protocol. The messages will be converted to the MapMes-

* michael.ritzert@ziti.uni-heidelberg.de

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOMPR002

sage format, as it is also used by the BEAST alarm system [3], by ActiveMQ. The core of the library itself is independent from EPICS. IOC Shell commands to configure the logging setup at runtime are provided when the library is used within EPICS IOCs. The logging can either be implemented within the device support code itself, or accessed from sequencer code or via `sub` records. A threaded design is used to decouple the main thread from the potentially time-consuming output operations, and avoid blocking the normal operation of the IOC.

Message Reception

On the input side, a suitable module to receive messages from ActiveMQ is readily available [4]. A standard configuration is used to connect to the JMS server and subscribe to the interesting channels: Alarm events are posted to the `..._SERVER` topic, and a dedicated topic `LOG` is used for log messages. For the message format used by the BEAST and logging systems, the output of the module is a message represented as a Java `Map<String, String>`, i.e. key/value pairs.

Message Filtering and Routing

The first action applied to incoming messages is the filtering, since at this stage, the full message is still available.

For alarm messages, the filter condition is that the current severity and the alarm severity match. This accepts only events created when the alarm condition is triggered, while messages created when acknowledging or releasing an alarm are ignored.

For log messages, the most used filter looks at the severity of the message, to accept e.g. only `WARNING` or `SEVERE` messages that should be broadcast to the wider audience available via the Rocket.Chat server. Of course, it is also possible to filter by any other property of the message, including the actual text of the message.

Routing of messages to different recipients can be implemented as a separate module, or integrated in the filtering step. In any case, when messages are routed to different recipients depending on the content, the information where to route the messages to has to be attached to the message by simply adding it with a well-known key to the map.

Message Transformation

When entering this module, all messages are still in a key/value pair format. To allow posting to Rocket.Chat, they have to be converted to a plain text format, with only few additional options, namely the icon to display next to the message, and the display name of the sender. This conversion is done in custom Java modules for log and alarm messages, respectively. After completing this stage, the message is now in the format suitable for the Rocket.Chat output module.

The default configuration is to use special symbols for log messages at the warning and severe levels, as well as for alarm messages. The latter are also tagged with `@here` to trigger a notification in the Rocket.Chat clients.

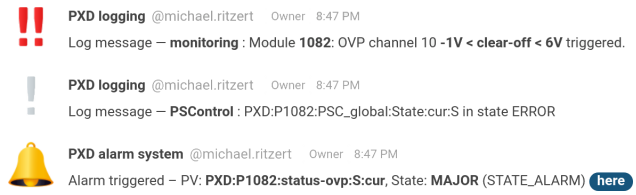

Figure 3: Rocket.Chat displaying messages from the alarm and logging systems.

For messages where the recipient is not hardcoded but stored in the message, this information is simply copied to the new object.

Message Output

No module to interface Camel to Rocket.Chat was available, so a lightweight module, camel_rocketchat, has been implemented.

Rocket.Chat offers several APIs for programmatic access. The REST API [5] has been chosen, because it is easy to use, and essentially stateless: No TCP connection has to be kept alive (HTTP/2 can still be used, but will fall back to opening a new connection, when required), and no information beyond the login token has to be stored between requests. In combination with a personal access token obtained once, there is exactly a single HTTP request per message to be posted, and the implementation is straightforward.

The actual implementation consists of two parts: A small, generic module implementing the posting of messages to Rocket.Chat, and a second module that wraps around it to provide the interfaces as required by Camel.

RESULTS

The system as described above has been implemented for the Belle 2 PXD subdetector. A significant fraction of the hardware used in the system is custom built, so that log messages are available directly from the device support layer. For other IOCs, logging has been integrated in Sequencer programs, or interfaced to via `sub` records. In total, around 50 IOCs provide log messages. A single Camel application with several routes handles the output of all log messages with at least `WARNING` severity, and all alarm messages to a dedicated Rocket.Chat channel, as well as the routing of a few filtered messages to channels dedicated to special topics. Figure 3 shows the rendering of two log messages and one alarm event displayed in the Rocket.Chat channel in the web browser.

CONCLUSION AND OUTLOOK

We presented a simple approach to integrate a slow control system's logging and alarm information into the chat platform also used for communication between the shifters. The solution is based on Apache Camel, that provides the framework for the messages processing pipeline, and the functionality to receive messages from ActiveMQ. Small modules to filter, route and convert messages have been implemented.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOMPR002

Thanks to the modular nature of Camel, switching from Rocket.Chat to other output modules could be implemented with only minimal changes to the code.

While, at present, the filtering and routing configuration is statically built into the executable, one could envision a system where each user can configure their own filter for messages posted either directly to the user's account or to a channel the user set up only to receive log messages.

REFERENCES

[1] K.-U. Kasemir, X. H. Chen, and E. Danilova, "The Best Ever Alarm System Toolkit", in *Proc. 12th Int. Conf. on Accelerator and Large Experimental Control Systems (ICALEPCS'09)*, Kobe, Japan, Oct. 2009, paper TUA001, pp. 46–48.

[2] Apache Camel, https://camel.apache.org

[3] Control System Studio Guide, http://csstudio.sourceforge.net/docbook/ch14.html#idm140287028562704

[4] ActiveMQ component, https://camel.apache.org/components/latest/activemqcomponent.html

[5] Rocket.Chat REST API, https://rocket.chat/docs/developerguides/rest-api/