STAR Online Framework: from Metadata Collection to Event Analysis and System Control

D Arkhipkin, J Lauret

Physics Department, Brookhaven National Laboratory, Upton, NY 11973-5000 USA

E-mail: arkhipkin@bnl.gov, jlauret@bnl.gov

Abstract. In preparation for the new era of RHIC running (RHIC-II upgrades and possibly, the eRHIC era), the STAR experiment is expanding its modular Message Interface and Reliable Architecture framework (MIRA). MIRA allowed STAR to integrate meta-data collection, monitoring, and online QA components in a very agile and efficient manner using a messaging infrastructure approach. In this paper, we briefly summarize our past achievements, provide an overview of the recent development activities focused on messaging patterns and describe our experience with the complex event processor (CEP) recently integrated into the MIRA framework. CEP was used in the recent RHIC Run 14 (2014), which provided practical use cases. Finally, we present our requirements and expectations for the planned expansion of our systems, which will allow our framework to acquire features typically associated with Detector Control Systems. Special attention is given to aspects related to latency, scalability and interoperability within a heterogeneous set of services, as with the various data and meta-data acquisition components coexisting in the STAR online domain.

1. STAR Online Meta-Data Collection Framework

The STAR experiment [1] started in 1999, with just one Time-Projection Chamber and a few trigger detectors, but today it is comprised of 18 subsystems. Initially, the STAR Slow Control system recorded ∼40,000 control variables; now it has expanded to over 60,000 variables and is still growing due to the RHIC II upgrade, beam energy scan program, and possible upgrade to eRHIC in future. STAR had just 120 structures to migrate to the calibrations database in the early days of the experiment, and we now migrate over 3,000 structures annually. STAR's Data Acquisition – physics data taking component – was upgraded three times, adding one order of magnitude to the rates each time.

As every experiment has slightly different grouping of internal components, and varying operational meanings for those, let us define a common terminology first. In this paper we define the following cornerstones of "a" physics experiment's backend: Data Acquisition (DAQ), Detector Control System (DCS), Meta-data Archiver, and Alarm system. In this paper, DAQ is assumed to operate on physics signals only, with all accompanied meta-data being handled in concert by DCS, operating on detector control data, and meta-data archiver, recording and storing detector conditions for the data-taking period.

STAR's Messaging Interface and Reliable Architecture framework (MIRA [2]) was created as an attempt to improve meta-data archiver operations in 2010. It relies on an advanced message-queuing middleware, which provides asynchronous, payload-agnostic messaging. We selected AMQP [3] as a messaging middleware standard. It allowed us to design a loosely coupled,

modular framework architecture, resulting in a scalable service, suitable for a highly concurrent online environment.

During the deployment validation phase in 2010, just three subsystems used the MIRA framework. By 2014, all 18 subsystems were completely integrated, with over sixty collector services deployed and continuously monitored by the framework. The total number of messages passing through the system reached 3 billion messages per year, with rates varying between 150 messages per second to over 2,000. On average each message corresponds to a structure of 24 variables, which gives a variable rate of 3,000 to 25,000 channel readings processed per second. MIRA uses a data compression technique to reduce the archived record storage size. The overall disk usage has been moderate, progressing from 30 gigabytes (GB) to just over 200 GB of recorded data per year. In addition to the data archival feature, MIRA provides extensive data visualization capabilities. It allows detector experts to browse archived system states (variables) as structured hierarchical content via a user-friendly responsive web interface. Starting with ∼100 variables in 2010, MIRA was expanded to visualize 1,680 variables in 2014.

In its early stage of development, MIRA provided convenient monitoring, storage and basic flow control over individual streams of data [4], covering for just the Meta-data Archiver part of the fundamental building blocks. To allow further coverage, multi-stream real-time processing capabilities were critical. In the next section, we describe the Complex Event Processing extension of the framework to accomplish this goal, implemented for Run 14.

2. Run 14 MIRA Extension: Complex Event Processing

Integration of the CEP brings the possibility to merge streams, process them according to pre-assembled persistent queries, and produce output streams back to the system for processing via regular means. There are many open-source Event Processor toolkits on the market. We settled on a WSO2 framework [5], enabling usage of the well-known Esper [6] and recently introduced Siddhi CEP engines interchangeably. Typical architecture of the CEP toolkit is presented at Fig. 1. It consists of a broker managing data format conversion, CEP processor instances handling event filtering, and a control component allowing configuration of persistent queries for the processor. CEP at STAR is configured to use existing AMQP topics and queues, provided by Apache *qpid* service [7] as input and output devices, exactly as implemented for MIRA framework services.

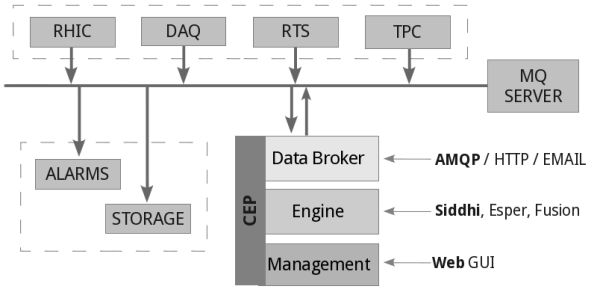

Figure 1. An overview of the Complex Event Processing components, attached to the messaging system: Broker, Engine and Management Console.

We describe here a few practical use-cases occurring in the STAR environment. The first example for CEP is to employ its stream processing power for a smart and flexible alarm system. CEP provides the ability to reconfigure the existing alarm rules or add new sophisticated ones on the fly without stopping the existing monitoring services. This feature improves user experience and facilitates elimination of semi-permanent false or nuisance alarms (see Fig. 2). A second use-case is detection of anomalies in the detector conditions with CEP. By merging multiple streams and applying filters, which discard expected "normal" events, one can get a stream of "abnormal" events, unearthing hidden problems with various components or subsystems of the experiment. For example, we can identify aberrant voltage fluctuations occurring during stable beam operations (data-taking flag set to "on" and non-fluctuating collision rate). Fig. 3 illustrates such behavior. In general, CEP increases modularity of online services by providing "external" stream aggregation. A dedicated Event Processor lifts the burden of manual stream

manipulation from clients, simplifying their data handling code. It also allows pre-testing of services attached to the message-queuing broker by feeding a simulated data stream, such that the response to incoming events can be validated well before the production deployment. Another use-case, whose implementation is expected for Run 15, is to perform triggered data migration from the online to the offline domain. Having a CEP trigger stream, which combines readiness and completion states of many processes, allows decreased resource consumption by eliminating unnecessary online database polling.

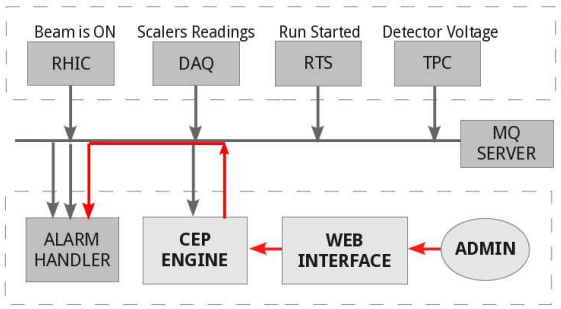

Figure 2. An overview of the Alarm handler use-case, describing the data flow and online configuration of persistent queries. Signals from many sources are combined to provide a clean and clear alarm one signal alone would not provide.

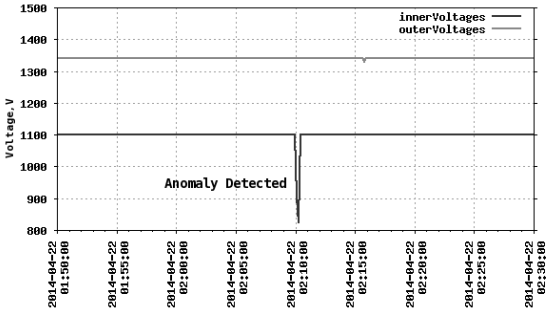

Figure 3. An example of the unexpected voltage drop, happening during data taking process, and registered by the event processing engine. Voltage drop is expected off physics running (and ramping up/down) but not during stable data taking (as in this example).

3. Framework Future: Control System Capabilities

The MIRA framework is still evolving. In the near future, we are planning to add features commonly encountered in the DCS domain: experiment workflow and hardware control. To implement MIRA's expansion effectually, we surveyed the requirements of our users and online software developers during the initial preparation phase. We have identified the following key features: *Scalable Architecture*, *Low-overhead, Inter-operable, Messaging Protocol*, *Payload-agnostic Messaging*, *Quality of Service*, *Improved Finite State Machine*, *Real-time Remote Control*.

Scalable Architecture is needed to accommodate the growing number of detectors and channels. We expect STAR to double the number of channels in the next five years, hence, system scalability is our primary objective. An easily-implemented, low-overhead, inter-operable messaging protocol is highly desired to allow steady migration from a wide variety of existing legacy hardware (with non-standard architectures from early days of the experiment) to the modern detector control equipment. It implies that there are no publicly available, ready-to-use messaging libraries for this type of equipment. Payload-agnostic Messaging is needed to allow simultaneous handling of a multitude of custom control protocols. The existing in-house-developed protocols are predominantly text-based, and there is a strong desire to use and test legacy and upgraded components in parallel during the transition phase. Eventually, all archaic protocols will be upgraded to use JSON notation (text) or MsgPack (binary JSON equivalent) data serialization formats.

There is a clear need to provide and regulate different Quality of Service levels for three distinct groups of clients: (a) sensors, which provide continuous streams of data, are tolerant to sporadic loss of messages, but require maximum throughput; (b) data storage backend services, which need a "receive message at most

once" mode of operations, to avoid data duplication; (c) control services, which operate in a "receive at least once" mode to ensure persistent control behavior. Our preliminary survey also indicates that existing Finite State Machine (FSM) implementations lack features like state stack, complex state handling and sub-states. We plan to investigate the possibility of implementing an improved hierarchical, stack-based FSM for the expanded MIRA. For Real-time Remote Control, we plan to study the possibility of making a soft real-time, low-latency Human-Machine Interface using newest web technologies like WebSockets [8] from the HTML5 technology stack, in combination with the messaging libraries. Our goal is to provide a consistent interface for all client types from desktops to smartphones and tablets. The development and transition to the fully upgraded MIRA is expected to happen in three years.

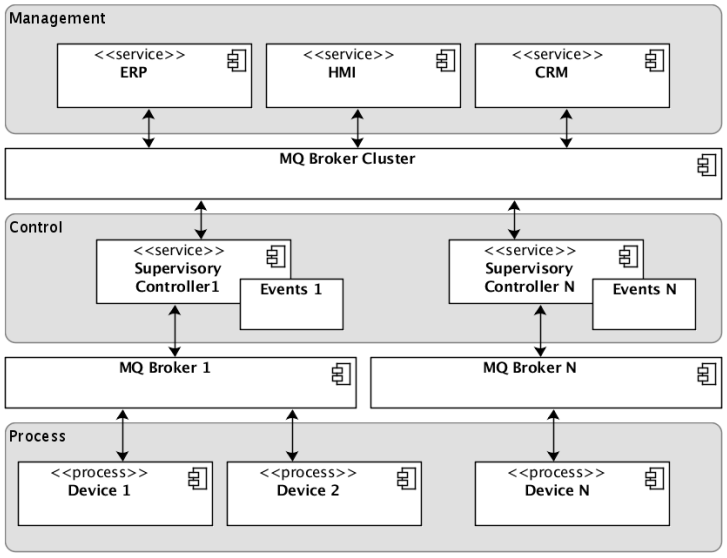

Figure 4. An overview of the preliminary version of tiered Control System architecture. Management, Control and Process tiers are communicating using messaging middleware services.

With the requirements described above, our next goal is to choose the architecture of the DCS. There is a choice between industrial SCADA architectures, based on the OPC-UA standard [9], and a common High Energy or Nuclear Physics approach. Both options have their unique strong and weak points. For example, OPC-UA features an informational model, based on complex objects with dependencies, promoting a centralized architecture approach. On the other hand, Control Systems developed for HEP/NP experiments, feature flat address space, the so-called "single variable without dependencies" model, promoting a decentralized, highly distributed approach. We describe our design as a "fusion" of both approaches. It will leverage standardized message-queuing protocols like AMQP and MQTT [10] (Fig. 4), and define a "structure" as an informational exchange unit, conceptualizing it as a "first class citizen" programming model, a compromise between the complexity of large objects, and the "single variable" approach.

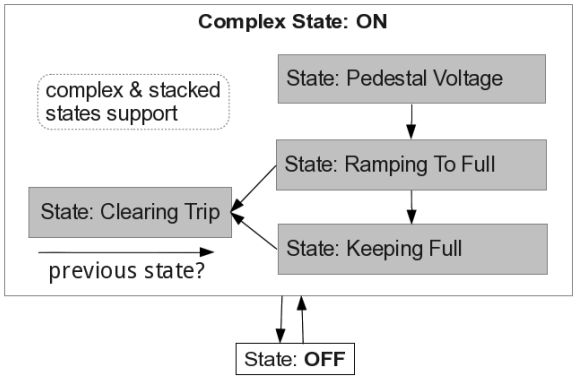

Figure 5. Illustration of the issue with the Finite State Machine having no stackable state support. Implementation of the exit state handlers using non-stackable state machine is possible, but is complicated compared to stack-enabled FSM.

Another important principle of DCS implementation is the Observer Model methodology: clients are completely decoupled from data producers, and any service attached to the messaging broker may observe any process in the system. This allows us to embed rich monitoring, logging and reporting functionalities into the core of the framework. Also, we are going to implement this functionality as an extension of the MIRA framework, allowing us to reuse existing automatic archiving and visualization services.

The messaging part will be split into two distinct areas, corresponding to two distinct

requirements: the *bottom layer* requires low-latency communications, while the *top layer* demands a rich transport protocol functionality and may not be satisfied by the same technology choice. We have selected the MQTT and AMQP protocols respectively. MQTT is a good choice for the sensor equipment due to its simplicity, while AMQP has larger overhead, but has a full stack of features required for reliable queue management. For the engine core we are going to implement a brokered, hierarchical, stack-based finite state machine. In our future studies, we plan to do a detailed review of the open-source implementations of FSMs used in HEP/NP experiments, with the objective to provide an implementation capable of resolving multi-exit scenarios, including complex-state and sub-state support (Fig. 5). Our project is going to use MQTT protocol over WebSockets to implement a Human-Machine Interface for the Control System. This allows utilization of a single technology stack for local, remote and web clients. For the messaging middleware we have selected Apache Apollo, an open-source multi-protocol broker, written in Scala. This broker handles MQTT, AMQP and WebSocket connections, allowing us to reduce the maintenance to the single node, or a single cluster of brokers, removing the necessity to maintain several independent brokers.

4. Summary and Outlook

We have presented an update on the status of MIRA, the STAR Meta-Data Collection Framework, and its recent achievements, current state and plans for near-term development and extensions. Message-Queuing services became an instrumental part of the STAR online infrastructure. The MIRA framework, featuring a flexible, loosely coupled system of components, was very well accepted by STAR's collaborators and detector experts, covering the meta-data collection and monitoring needs of all 18 STAR subsystems, which gradually moved to the new framework. MIRA provided STAR with a solution to handle the growing number of channels (x15), and data structures (x25), facilitating smooth operation during Runs 10-14.

In 2014 we extended MIRA with the stream-based Complex Event Processing capability, which successfully passed our tests. A few alarms implemented for Run 14 saved months of work for the core team and collaborators. More use-cases will be implemented for Run 15 and beyond, expanding the usage of CEP to automatic data migration handling and extensive error conditions detection.

In addition to the CEP features, we have extensively described our future plans for extending MIRA to enable Control System capabilities. This extended framework will benefit from multi-tiering, a scalable architecture, improved Finite State Machine functionality and a real-time remote control interface.

Acknowledgments

This work was supported by the Office of Nuclear Physics within the U.S. Department of Energy's Office of Science.

References

[1] The Solenoidal Tracker At Rhic (STAR) experiment [Online] URL `http://www.star.bnl.gov/`
[2] D Arkhipkin, J Lauret and W Betts 2011 *J. Phys.: Conf. Ser.* **331**
[3] Advanced Message Queuing Protocol [Online] URL `http://www.amqp.org/`
[4] D Arkhipkin, J Lauret, W Betts and G Van Buren 2012 *J. Phys.: Conf. Ser.* **396**
[5] WSO2 Complex Event Processor [online] URL `http://wso2.com/products/complex-event-processor/`
[6] ESPER: Event Stream and Complex Event Processing [online] URL `http://esper.codehaus.org/`
[7] RHEL6 High Performance Network with MRG - MRG Messaging: Throughput & Latency (*Preprint* `http://www.redhat.com/f/pdf/MRG-Messaging-1Gig-10Gig-IB-Xeon-v2.pdf`)
[8] RFC 6455: The WebSocket Protocol [online] URL `http://tools.ietf.org/html/rfc6455`
[9] OPC Unified Architecture [online] URL `https://opcfoundation.org/about/opc-technologies/opc-ua/`
[10] MQ Telemetry Transport [Online] URL `http://mqtt.org/`