Computer Science • 15 (4) 2014 http://dx.doi.org/10.7494/csci.2014.15.4.351

Karol Grzegorczyk
Vito Baggiolini
Krzysztof Zieliński

COMPLEX EVENT PROCESSING APPROACH TO AUTOMATED MONITORING OF A PARTICLE ACCELERATOR AND ITS CONTROL SYSTEM

Abstract *This article presents the design and implementation of a software component for automated monitoring and diagnostic information analysis of a particle accelerator and its control system. The information that is analyzed can be seen as streams of events. A Complex Event Processing (CEP) approach to event processing was selected. The main advantage of this approach is the ability to continuously query data coming from several streams. The presented software component is based on Esper, the most popular open-source implementation of CEP. As a test bed, the control system of the accelerator complex located at CERN, the European Organization for Nuclear Research, was chosen. The complex includes the Large Hadron Collider, the world's most powerful accelerator. The main contribution to knowledge is by showing that the CEP approach can successfully address many of the challenges associated with automated monitoring of the accelerator and its control system that were previously unsolved. Test results, performance analysis, and a proposal for further works are also presented.*

Keywords automated monitoring, Complex Event Processing, Esper

1. Introduction

A particle accelerator is a large and complex installation whose aim is to speed up subatomic particles, such as electrons or protons, grouped into beams to obtain high energies and then collide them. CERN, the European Organization for Nuclear Research, hosts the accelerator complex, which consists of many devices including the Large Hadron Collider (LHC), the world's largest particle accelerator.

In order to manage and run a particle accelerator, a dedicated control system is needed. The CERN Accelerator Control System, formerly known as The LHC Control System [3], is a complex system consisting of many hardware and software modules and components created with cutting edge technologies. Software components are deployed on about 2500 computers and are represented by some 10 million lines of code and developed by some 80 people.

To be able to monitor and understand the behavior of an accelerator, many sensors must be attached to it. The sensors generate and send logging messages to inform users about the current state of the accelerator. In addition, the control system itself generate logging messages for its own status information.

To ensure the proper behavior of all of the accelerators from the accelerator complex as well as the control system, operators working in the CERN Control Center (CCC) constantly (24 hours per day, 7 days per week) monitor the state of all devices in the accelerator complex as well as the control system itself. If they detect an anomaly, they inform experts or perform some emergency procedure, such as safely discarding ("dumping") the particle beam. In addition, the monitoring data is stored in a database for further offline analysis. Experts can query the database to learn and discover any abnormal behavior of the accelerators.

The ultimate goal of this research is to automate a process of online monitoring by defining, in the form of patterns, those abnormalities that operators are looking for, and by deploying some stream processing engine that will be able to detect those patterns. In case some pattern is matched, it would be expected to automatically: (i) notify experts; (ii) display meaningful information for operators instead of raw data; and (iii) store meaningful information rather than raw data for further analysis. In addition, some of the offline queries experts invoke on the database could be migrated to online patterns as well.

This paper presents a proof of concept (POC) solution to the problems described above using a Complex Event Processing [8] approach.

The paper is organized as follows: Section 2 describes in more detail the problem presented above. The created POC solution is discussed in 3 and 4. Section 5 proposes further works. Finally, 6 describes related works, and 7 concludes the article.

2. Problem description

This paper addresses two similar but slightly different problems. The first problem is related to the monitoring of the states of the accelerator's hardware components,

and the second is related to monitoring the state of the software modules of the accelerator's control system. The problems are further discussed in the following two subsections.

2.1. Monitoring and analysis of logging messages generated by the accelerator

As stated in the introduction, in addition to the online monitoring done by operators, diagnostic information is stored for further processing. The CERN Accelerator Logging Service (CALS) [11], which is a part of the CERN Accelerator Control System, aims to capture and store numeric monitoring information generated by the accelerator and its sensors. About five thousand hardware components (such as collimators, beam position monitors, safety elements, etc) are distinguished in the accelerator complex. These components are called *devices*. Each device produces one or more logging *parameters* containing one or more *fields*. Field values are stored in the Logging Database as *variables*. The Logging Process is a component of CALS that is distributed across several machines and is responsible for receiving parameters from the devices and storing their fields as variables in the database. Each Logging Process instance receives only a subset of available parameters (only those to which it subscribes). Before storing the values in the database, the Logging Process does a simple validation of the parameters and their fields. Both field values and validation errors have to be monitored and analyzed in order to understand the accelerator's behavior. The average rate of monitoring events (received parameters and validation errors) produced by all instances of the Logging Process is about 15 kHz.

Motivation

The problem with the current solution is that operators monitor in an online manner only raw data, so they are only able to spot straightforward problems. In order to learn and discover more complex accelerator behavior, queries need to be invoked either by experts or by some scheduling mechanism on time series raw data stored in a database. This data describes the accelerator status at a specified time in the past. It would be highly desirable to introduce an online stream analysis that allows us to (i) extract similar knowledge that currently can only be obtained in an offline manner, (ii) notify experts or users in case of some problem or anomaly, and (iii) store meaningful information in the Logging Database instead of raw data.

2.2. Monitoring and analysis of logging messages generated by the control system

To avoid confusion with logging messages generated by devices, logging messages generated by a control system are called *tracing messages*. At the time of writing, the rate of tracing data produced by all software components of the CERN Accelerator Control System, and the operating systems of machines on which they run, was about 130 Hz. All tracing messages are sent via modern message-oriented middleware or

legacy syslog [5] protocol to a central Splunk Enterprise [14] server, which is a text-oriented NoSQL database with the ability to index messages based on key-value pairs existing in their bodies. Splunk Enterprise will be hereinafter referred to as the *Tracing Database*.

In order to retrieve some state information of a control system, operators or experts query the Tracing Database specifying some time scope. About one hundred queries are defined and stored to ease the analysis of tracing messages.

Motivation

The problem with the current solution is that, in order to query the database, all tracing messages have to be stored and indexed, which is costly. In addition, monitoring is not automated, so it relies heavily on the manual intervention of operators and experts. Sometimes, system behavior is unexpected, and even experts are not able to fully understand it. It would be desirable to (i) decrease the amount of stored messages, (ii) introduce online processing (to capture expert knowledge in the form of some formal patterns), and (iii) automatically detect patterns in a data stream which appear frequently but are not yet well-known and, therefore, cannot be formally defined as in (ii).

3. Proposed solution

To address the requirements presented in the previous section, the Complex Event Processing (CEP) approach was chosen. CEP is a part of Event-Driven Architecture [10] and it enables stream correlation and defining patterns spanning multiple data streams.

Esper [2], the most popular open-source implementation of the CEP approach, was selected as a basis for the POC solution. Data patterns in Esper, called statements, are defined using Event Processing Language (EPL). EPL is a domain-specific language that is similar to Structured Query Language (SQL) but uses *streams* rather than tables and *events* instead of records (rows). As SQL queries can join multiple tables so EPL statement can join several data streams. Each stream has one simple event type associated with it. In addition, data streams can be joined with relational data or even non-relational data (via method invocation). Update listeners or subscriber objects are associated with EPL statements. Listeners and subscribers are notified when a new event matching an EPL query arrives.

What predominantly distinguishes EPL from SQL is its ability to reason over time and to detect patterns in a sequence of events. Therefore, the concept of *view* known in SQL is much more important in the case of EPL. One of the popular views is a time window (a.k.a. sliding window). This view "tells" Esper how long events from a specific stream should be kept in memory while processing.

Due to the different nature of offline and online processing, partially reflected by the differences between SQL and EPL presented above, it is not possible to directly apply the existing predefined offline queries to the online analysis. Some of them

can be adapted, but the majority of statements were created from scratch based on discussions with experts.

Two separate software components were created for the two use cases described in the previous section. Each of them was written in the Java language, run in a separate Java Virtual Machine (JVM), and integrated with the existing software components via interfaces. Both solutions are described in detail in the two following subsections.

3.1. Solution for monitoring and analysis of logging messages generated by the accelerator

As described in section 2.1, the Logging Process receives numeric parameters from devices, does basic validation of them, and stores them in the database. The created CEP solution was integrated with the Logging Process using Java Remote Method Invocation (RMI) technology. The architecture of the solution is presented in figure 1. The Logging Process sends simple events to the CEP engine. The events belong to one of six event types: (1) *parameter processed* – when a parameter has been received and no validation error is found (this event does not provide information regarding the value of the received parameter); (2) *parameter error* – when a parameter has been received but some validation error has occurred; (3) *field processed* – as in the case of parameters; (4) *field error* – as in the case of parameters; (5) *incorrect value* – extends the *field error*, providing a detailed error description; and (6) *value received* – event containing the variable name and its value.

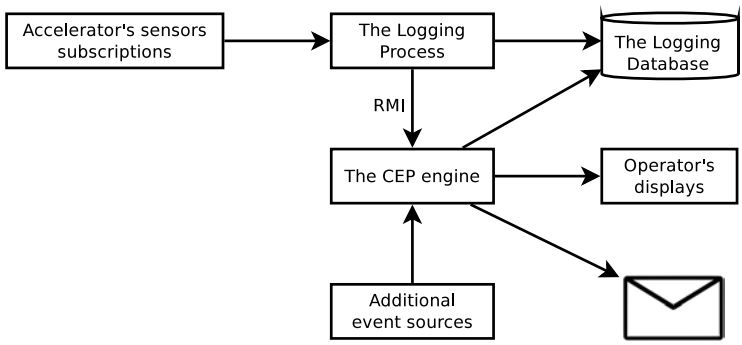

Figure 1. Architecture of the CEP module for the CERN Accelerator Logging Service.

In addition to the events that are sent from the Logging Process, the CEP engine can receive events from other sources. Those sources are illustrated with the box on the bottom of Figure 1. An example of such an event is a *beam mode change* event. The beam mode describes a state of the accelerator and is relevant for event processing. In some modes, the accelerator stays only for a short period of time (e.g., *beam injection* or *beam dump*). In others, it can stay for several hours (e.g., *stable beam*).

Apart from the events mentioned in the two preceding paragraphs, two relational data providers were created: *beam mode provider* and *parameter configuration provi-*

der. The first one returns a current mode of a beam (the *beam mode change* event mentioned in the previous paragraph is sent only when the accelerator switches from one mode to another, but sometimes it is necessary to know what the current mode is), and the second returns a list of categories to which a parameter belongs (to ease monitoring and management, each parameter is assigned to one or more categories).

Based on these seven event types and two data providers, some thirty EPL statements were created. The statements are classified into one of two groups: those relaying on numerical values received from devices (*value received* event) and those based on validation events (the remaining five coming from the Logging Process).

The statements are further divided into two groups: periodic and aperiodic. The periodic statements produce output at every specified time interval. The aperiodic ones produce output only when an anomaly is detected. Since, in most cases, a particular event can be considered an anomaly only in the context of a beam mode, aperiodic statements often join simple events with the *beam mode provider*.

Multi-level time window

Most of the periodic statements adhere to the so-called *multi-level time window* pattern. The pattern is as follows: for each simple event, *a rate* is calculated as a sum of events in a one-second time window. The rate is inserted as a new event to an intermediate stream and, based on it, *an average rate* is calculated for a longer window (the size of the window is defined as a variable). Following the same pattern, *a long run average rate* is calculated based on the average rate. A simplified example of multi-level time window pattern for *parameter processed* simple event is presented below[1]:

```
insert into ParametersProcessedRate
select deviceId, parameterName, count(*) as number
from ParameterProcessed.win:time(1 sec)
group by deviceId, parameterName
output all every 1 sec;

insert into AverageParametersProcessedRate
select deviceId, parameterName, avg(number) as avg
from ParametersProcessedRate.win:time(10 sec)
group by deviceId, parameterName
output all every 10 sec;

insert into LongRunAverageParametersProcessedRate
select deviceId, parameterName, avg(avg) as avg
from AverageParametersProcessedRate.win:time(30 sec)
```

[1]In reality, those statements are a bit more complicated. For example, the first one is run in predefined EPL *context*, which segments events by device ID. A context is an Esper concept that takes a cloud of events and classifies them into one or more sets, which are called *context partitions*. The use of contexts significantly improves performance. A detailed description of them can be found in chapter 4. of the Esper reference documentation.

```
group by deviceId, parameterName
output all every 30 sec;
```

An *all* keyword, which is present in the last line of each of the statements, will guarantee that information regarding rate, average rate, or long run average rate will be produced for all of the parameters that have ever arrived. If there is no ParameterProcessed event in the last interval, the listener will be notified that the rate for a particular *deviceId, parameterName* tuple is zero.

Most of the aperiodic statements catch deviations from the respective levels of multi-level time windows. For example, statement *deviation from average parameter processed rate over limit* will match if the rate is much higher or lower (above or below a given threshold) than the last average rate. A simplified version of the statement is presented below. Variables *updateRateUperLimit* and *updateRateLowerLimit* are set at runtime to define the thresholds.

```
select n.deviceId, n.parameterName, n.number
from ParametersProcessedRate n unidirectional,
AverageParametersProcessedRate.std:lastevent() a
where n.deviceId      = a.deviceId
and   n.parameterName = a.parameterName
and (n.number >= updateRateUperLimit  * a.avg_number or
     n.number <= updateRateLowerLimit * a.avg_number);
```

One of the requirements for monitoring is to know a periodic value (e.g., average or long run average) for some parameter while the accelerator was in a particular beam mode. In order to achieve this, a couple of statements join a stream of *beam mode changed* simple events. When the mode changes, a snapshot of the last values from the previous mode for each parameter is generated.

Additional message processing

Finally, for some of the monitored parameters, it is expected to have more comprehensive statistical information than just a simple mean (average). A few statements were created that take advantage of several univariate statistic functions (sample variance, sample standard deviation, population standard deviation) that are built-in into Esper.

For the purpose of testing, all statements were associated with a subscriber that writes an output to a log file and to a Java Management Extensions (JMX) bean. In a production deployment, as depicted in Figure 1, additional subscribers that write an output to the Logging Database, operator displays, and send notifications will be added.

3.2. Solution for monitoring and analysis of logging messages generated by a control system itself

As stated in section 2, before introducing CEP processing, all tracing messages were stored in the single Tracing Database. After the CEP solution was deployed, messages coming from all message-oriented middleware and legacy syslog sources were sent to

the CEP engine and not directly to the Tracing Database anymore. Only an output of statements (output generated by statement subscribers) is stored in the Tracing Database. The architecture of the solution is depicted in Figure 2. Integration with the database is done using Splunk's streaming scripted input.

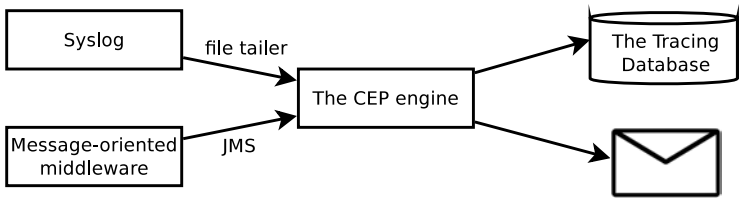

Figure 2. Architecture of the CEP module for automated monitoring of the CERN Accelerator Control System.

Message reduction

To address the first requirement from section 2.2 (message reduction), the notion of message similarity has to be introduced. Two messages are considered to be similar if they differ only in numbers (decimal or hexadecimal). In other words, if all numbers are stripped, the resulting messages are equal.

Further, two EPL statements were defined. The first one matches a message only if there is not a similar message inside a time window (a similar message has not arrived for the window-size period). In this case, the tracing message is immediately forwarded to the Tracing Database. A simplified version of the statement is presented in the following code listing:

```
select date, host, msg
from TraceEvent.win:time(30 sec)
group by host, msgTemplate
having count(*) = 1;
```

The second statement matches only if there is at least one similar tracing message in a time window. In this case, the tracing message is not directly forwarded to the Tracing Database. Instead, one summary message per group of similar messages is generated at regular time intervals. The statement is presented below in a simplified version:

```
select max(date), host,
aggregateTraces(msg) + ', REPETITIONS=' + count(*)
from TraceEvent.win:time_batch(30 sec)
group by host, msgTemplate
having count(*) > 1;
```

The term *aggregateTraces*, seen in the second line of the statement, is not part of EPL, but is a custom aggregation function. For each event (tracing message) entering a group, the function parses the message body and splits it into parts. There are two main types of body parts: variable parts (e.g., real numbers, hexadecimal numbers,

IP addresses) and constant parts (the remaining message body). Constant parts are equal for all the messages entering the group (we will describe later how to ensure that only messages having the same constant parts enter the same group). Variable parts differ between messages, so they have to be stored for the window-size time and for each variable part; a separate aggregate summary is generated when some message leaves a group or, in the case of batch windows, an interval ends. In order to ensure that similar messages go together to common groups, the *msgTemplate* property of the *TraceEvent* event type (depicted in the penultimate line of the above statement) was introduced. This property simply removes all variable parts (described above) from the message.

To better understand this, consider the following sequence of syslog kernel messages, informing about active network connections and arriving one after another:

```
Sep 1 16:22:17 server01 kernel: NET: Connection with host 172.16.0.2 active
Sep 1 16:22:18 server01 kernel: NET: Connection with host 172.16.1.3 active
Sep 1 16:22:20 server01 kernel: NET: Connection with host 172.16.0.6 active
Sep 1 16:22:21 server01 kernel: NET: Connection with host 172.16.1.1 active
```

In the first place, messages are partitioned into arrival date, host, and body. Next, the body is split into three parts: text part, IP address part, and one more text part. Because messages are arriving at short intervals, and because they differ only in IP addresses, they will be replaced with one aggregate message in the form:

```
Sep 1 16:22:25 server01 kernel: NET: Connection with host 172.16.*.* active
```

Additional message processing

In addition to the general reduction mechanism described above, several domain-specific patterns were defined, which automatically match some sequences of messages and help operators and experts to easily recognize important operational scenarios. Examples of these patterns are: *a machine failed to boot after restart*, *a machine cleanly rebooted*, and *a machine not active for a long time*. When those patterns are matched, some action, like sending a notification, can be performed. In addition to the notification, a message describing the problem is stored in the Tracing Database.

The third challenge is to detect new, previously unknown patterns in the data streams. As opposed to the previous challenge where a human expert has to specify EPL queries that match well-known patterns, in this third challenge new patterns that might be interesting are detected. The goal is to spot small groups of lines that always appear together and, thus, have some common meaning. This is not a trivial problem, and it is described in more detail in section 5. As a part of this work, some quasi-intelligent solution was created. The basic idea is to look at how many times lines are repeated. If several reduced messages described above have the same repetition number, then one more summary message is generated, suggesting that those reduced messages belong together.

4. Performance analysis

In order to process a data stream and be able to do some computation on it, some data needs to be kept in memory for some period of time. An example of the computation can be the correlation with data from other streams or data from the same stream arriving later. If the rate of arriving events is high and the sliding windows used in statements are long, then the memory (heap) usage can be expected to be high. Also, the garbage collector will be used heavily to remove events whose timeout has expired.

According to its official documentation, Esper "exceeds over 500 000 event/s on a dual CPU, 2 GHz Intel-based hardware (...) with 1000 statements registered in the system". This seems to be a lot, but the result depends not only on the rate of incoming events and number of statements, but on the statements' internal structure and design. For example, a statement that joins several data streams is much heavier for the engine than one that does not join.

There is also a risk of creating a statement that will keep some objects in memory forever, preventing them from being eligible for garbage collection which, in turn, can lead to a memory leak. This can happen, for example, if a statement uses the *group by* clause and the group-by expression has an unlimited number of possible results.

For each of the two solutions presented in section 3, a testing environment was prepared and a set of integration tests was performed. All experiments were conducted on the CERN Beams Department's infrastructure between July and September 2013, while LHC was in the first long shutdown stage (a two-year period during which the accelerator is prepared for higher energy and luminosity). Experiments were conducted based on version 4.9.0 of Esper and version 1.7 of Oracle Java Virtual Machine (JVM).

4.1. Performance analysis of a solution for monitoring and analysis of logging messages generated by the accelerator

For testing purposes, a dedicated instance of the Logging Process subscribed to around 900 parameters (out of 9,000 existing parameters) was deployed on a virtual machine having one Intel Xeon CPU (E5645 2.40G Hz) and 4 GB RAM assigned. Scientific Linux CERN 6 (SLC6) Linux distribution was running on the machine[2]. The average rate of events coming from the Logging Process to the CEP engine was about 1.5 kHz. This is a lot, but it is still much less than what Esper is capable of (see above). A memory consumption (heap) was stable at the level of 200 MB. The CPU usage of the machine was 7%.

[2]Scientific Linux CERN is a Linux distribution built within the framework of Scientific Linux which, in turn, is rebuilt from the freely-available Red Hat Enterprise Linux (Server) product sources

4.2. Performance analysis of a solution for monitoring and analysis of logging messages generated by a control system itself

Tests were performed on the production tracing machine which had dual Intel Xeon CPU (X5660 2.80G Hz) and 12 GB of RAM. As in the previous case, the operating system was SLC6 Linux distribution.

As stated in section 2.2, the rate of tracing messages generated by the control system was about 130 Hz, which is 10 times smaller than in the previous case. The frequency of messages leaving the CEP engine and going to the Tracing Database is about 5 Hz. Therefore, it can be concluded that the storage reduction rate is about 25 times (for the 30 seconds time window size used in the experiments). During the experiments, the heap usage of JVM was 60 MB and the CPU usage was almost unnoticeable.

As stated before, these measurements were taken during a long shutdown. When the accelerator will be fully operational, data rates are expected to be higher. In addition, the rate of incoming events can increase significantly when there is some accelerator failure.

5. Further work

The main focus of the works described in this paper was to formally define known patterns that experts and operators are looking for in logging and tracing messages, and to automate the process of finding them using a continuous query mechanism. In addition, in section 2, the need for automated discovery of unknown but frequent patterns was outlined. A simplified quasi-intelligent solution to this problem was described in section 3. Unfortunately, this solution is limited and error-prone. Much better suited for this kind of problems are techniques collectively called frequent pattern mining, which is a subfield of data mining. As a continuation of this work, it is planned to create a stream data mining module that will be equipped with some frequent pattern mining algorithms tailored to online incremental processing.

Another trait of the stream data mining module would be the detection of unknown anomalies or outliers in a data stream (the current solution relies solely on a continuous query mechanism and, therefore, allows us to detect only well-known anomalies). We believe that a clustering-based approach to anomaly detection is best suited to a log analysis domain. This approach assumes working in an unsupervised manner and utilizing some machine learning algorithms. As in the previous case, the algorithm would have to be tailored to online single-pass processing.

The stream data mining module does not eliminate the need for the CEP module. The CEP module can be seen as a preprocessing component for the intelligent module that aims at normalizing both the format and frequency of the data, which are the input for the mining module.

It is worth mentioning that using artificial intelligence (machine learning, in particular) for event processing is not a novel approach. For example, the HOLMES

project [15] uses some unsupervised machine learning techniques to detect anomalous patterns in monitoring data streams. This project is focused on the monitoring of data centers, which is a different domain from the one described in this article. Therefore, we believe that this work still need to be done for the accelerator's control system domain.

The most prominent research applying machine learning for console log processing was done by Wei Xu et al. at Berkeley in the late 2000 s [17]. These authors used frequent pattern mining and anomaly detection as well, but they assumed that they had source code of the system that produces the logs, and they found places where logs were generated by scanning the source code and inferring a "metamodel" of the logs. In other words, they knew which part of a log message is the static template and which parts are variables. We think that this assumption is too restrictive, and in our work, we will infer the template and variable parts of a log message based solely on log message analysis.

6. Related works

In addition to Esper, there are a few technologies which support the Complex Event Processing approach. The two best-known research implementations are SASE [16] and Cayuga [1]. Unlike Esper, these implementations are prototypes and cannot be considered as production-ready. Alternatively, Drools Fusion [4] is a Complex Event Processor that is included in the popular open-source business logic platform JBoss Drools, which has many successful deployments.

CEP can be applied to a variety of domains. For example, the paper [12] presents a solution for formalized medical treatment procedures. In this solution, Esper is used to transform raw data produced by personal medical devices (e.g., pressure gauges, glucose meters) into an aggregated/correlated form, as required by treatment procedures.

At CERN, the Complex Event Processing approach, and Esper technology in particular, have been used for some time but in a slightly different context from the one presented in this paper. The other context is briefly described below.

As stated in section 1, the main aim of the particle accelerator is to collide particles at high energies. Detectors are built along accelerators to analyze the physical phenomena that occur after the collisions. At CERN, the detectors are managed using dedicated control systems separate from the CERN Accelerator Control System. Operators constantly monitor a detector's status from the dedicated detector control centers, separate from the CERN Control Center.

ATLAS is one of two general-purpose detectors at the LHC. The ATLAS TDAQ [7] system is a large, heterogeneous system consisting of a wide variety of software and hardware components. Its overall goal is to gather "interesting" physical data from the ATLAS detector. Two years ago, TDAQ was equipped with a CEP module as part of the AAL project [6, 9]. Since then, it has been extensively used.

The accelerator domain is different from the domain of detectors, which is why the AAL module could not be used for the CERN Accelerator Control System.

Efforts have been undertaken even to add an intelligent module to the ATLAS TDAQ system, similar to the module presented in section 5. The Ph.D. thesis by J. Slopper [13] describes this in detail. Unfortunately, this work was mainly theoretical and cannot be directly applied to the accelerator domain.

7. Conclusions

The two presented solutions prove that the CEP approach is appropriate for the problem of automated analysis of logging data generated by an accelerator and its control system. However, shown in the experiments, Esper requires carefulness and experience in order to create memory-efficient statements.

The majority of the created EPL statements are domain-specific and related only to accelerators and their control systems. However, a few of them are general patterns and can be considered as a contribution to knowledge. Examples of such statements described in section 3 are the multi-level time window pattern or the message reduction pattern.

At the time of writing this article, both of the created solutions were not yet production-ready, but the work is planned to be finalized soon. It is also expected that more statements will be added in the future, covering other aspects of monitoring and diagnostic information analysis.

Acknowledgements

The authors would like to thank Alastair Bland, Łukasz Burdzanowski, Chris Roderick, Steen Jensen, and other experts from the CERN Beams Department for their knowledge sharing and fruitful discussions. All of the research leading to the creation of this paper has been financed by CERN.

References

[1] Demers A. J., Gehrke J., Panda B., Riedewald M., Sharma V., White W. M., et al.: Cayuga: A General Purpose Event Monitoring System. In: *CIDR*, vol. 7, pp. 412–422. 2007.

[2] Esper. URL `http://esper.codehaus.org`.

[3] Frammery B.: The LHC Control System. In: *Proceedings of ICALEPCS*. pp. 10–14. 2005.

[4] Drools Fusion. URL `http://drools.jboss.org/drools-fusion`.

[5] Gerhards R.: *The syslog protocol*. Tech. rep., IETF, 2009. RFC 5424.

[6] Kazarov A., Lehmann Miotto G., Magnoni L.: The AAL project: automated monitoring and intelligent analysis for the ATLAS data taking infrastructure. In: *Journal of Physics: Conference Series*, vol. 368, IOP Publishing, 2012.

[7] Kordas K., et al.: *The ATLAS Data Acquisition and Trigger: concept, design and status*. Tech. rep., CERN, Geneva, 2006.

[8] Luckham D. C.: *The power of events*, vol. 204. Addison-Wesley Reading, 2002.

[9] Magnoni L., Lehmann Miotto G., Luppi E.: *Intelligent monitoring and fault diagnosis for ATLAS TDAQ: a complex event processing solution*. Ph.D. thesis, Ferrara University, 2012, presented 30 Mar 2012.

[10] Michelson B. M.: *Event-driven architecture overview*. Tech. rep., Patricia Seybold Group, 2006.

[11] Roderick C., Billen R., Gaspar Aparicio R.D., Grancher E., Khodabandeh A., Segura Chinchilla N.: *The LHC Logging Service : Handling terabytes of on-line data*. Tech. rep., CERN, Geneva, 2009.

[12] Skałkowski K., Zieliński K.: Applying formalized rules for treatment procedures to data delivered by personal medical devices. *Journal of biomedical informatics*, vol. 46(3), pp. 530–540, 2013.

[13] Slopper J. E., Mapelli L.: *Error Management in ATLAS TDAQ: An Intelligent Systems approach*. Ph.D. thesis, Warwick University, Warwick, 2010, presented 2010.

[14] Splunk Enterprise. URL `http://www.splunk.com`.

[15] Teixeira P.H.d.S., Clemente R. G., Kaiser R. A., Vieira Jr D. A.: HOLMES: An event-driven solution to monitor data centers through continuous queries and machine learning. In: *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems*, pp. 216–221, ACM, 2010.

[16] Wu E., Diao Y., Rizvi S.: High-performance Complex Event Processing over Streams. In: *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, SIGMOD '06, pp. 407–418, ACM, 2006.

[17] Xu W., Huang L., Fox A., Patterson D., Jordan M.I.: Detecting Large-scale System Problems by Mining Console Logs. In: *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, pp. 117–132. 2009.

Affiliations

Karol Grzegorczyk
AGH University of Science and Technology, Department of Computer Science, Krakow, Poland

Vito Baggiolini
Beams Department, CERN, the European Organization for Nuclear Research

Krzysztof Zieliński
AGH University of Science and Technology, Department of Computer Science, Krakow, Poland

Received: 19.03.2014
Revised: 24.04.2014
Accepted: 2.06.2014