Application of Complex Event Processing Software to Error Detection and Recovery for Arrays of Cherenkov Telescopes

T. Holch[*a], **U. Schwanke**[b], **M. Füßling**[a], **M. Gajdus**[b], **T. Murach**[b], **I. Oya**[a], **P. Wagner**[b] **and P. Wegner**[a]

a DESY Zeuthen, Platanenallee 6, 15738 Zeuthen, Germany
b Humboldt–Universität zu Berlin, Newtonstraße 15, 12489 Berlin, Germany
E-mail: holchtim@physik.hu-berlin.de

Data acquisition (DAQ) and control systems for arrays of Cherenkov telescopes comprise hundreds of distributed software processes that implement the readout, control and monitoring of various hardware devices. A multitude of different error conditions (malfunctioning detector hardware, crashing software, failures of network and computing equipment etc.) can occur and must be dealt with to ensure the speedy continuation of observations and an efficient use of dark time. Flexible, fast and configurable methods for automatic and centralized error detection and recovery are therefore highly desirable for the current generation of ground-based Cherenkov experiments (H.E.S.S., MAGIC, VERITAS) and will be important for the Cherenkov Telescope Array (CTA), a more complex observatory with O(100) telescopes. This contribution describes a Java-based software demonstrator that was developed for the High Energy Stereoscopic System (H.E.S.S.) and uses the complex event processing engine Esper for error detection and recovery. The software demonstrator analyses streams of error messages in the time domain and aims to apply recovery procedures that reflect the knowledge of DAQ and detector experts.

The 34th International Cosmic Ray Conference,
30 July- 6 August, 2015
The Hague, The Netherlands

[*]Speaker.

1. Introduction

The data acquisition (DAQ) and control systems for major astroparticle experiments are expected to deliver science data with minimal loss of observation time. Achieving a high data-taking efficiency in a reasonably complex installation with numerous detectors and many interacting distributed software processes requires a system with a high fault-tolerance and a clear strategy for the handling of error situations. There are paradigms for building a fault-tolerant distributed system (e.g. by making sure in the design and testing phase that subsystems can gracefully continue in situations where resources or other subsystems become temporarily unavailable) but it is also clear that not all error situations can be anticipated and resolved in advance. In general, failures will occur due to (i) a malfunctioning of the detectors that must be controlled and read out, (ii) problems with the hardware (computers, network equipment) on which the DAQ system runs, and (iii) internal problems (bugs, inconsistencies) of the DAQ software itself. In a given error situation, it is important to quickly identify the real cause of an error (be it a hardware or software problem) and to distinguish it from a wealth of other possibly unrelated or secondary errors, and to take appropriate recovery actions. Experience shows that this kind of error detection and recovery can be a challenging task for the operators of the DAQ system who often resort to a (possibly time-consuming and not constructive) restart of the entire DAQ system. It is therefore desirable to automatically detect error situations and to apply recovery procedures that combine the knowledge of domain experts (e.g. detector experts, DAQ software experts).

2. Automatic Error Detection and Recovery

A software system for automatic error detection and recovery should be (i) fairly centralized, (ii) highly configurable, (iii) capable to process different input streams (e.g. error messages and results of the monitoring of the network hardware), and (iv) capable of executing a variety of operations to assess the situation. A certain degree of centralization is mandatory since one individual software process that experiences an error state has often too little information about the state of the entire system for suitable error handling. A limited, hard-coded local error handling (e.g. by re-trying in case of connectivity problems) or attempts by several processes in an error state to find out what is going on are in general poor remedies that can even deteriorate the situation. The demand for high configurability derives from the fact that the error detection and recovery system must be easily adaptable to changes in the detector & software configuration and to new information provided by domain experts, all of which evolve over time. Of course, the system must also be reasonably stable in order not to become a cause of trouble itself.

Knowledge-based systems are one way to approach the problem of automatic error detection and recovery. Classic expert systems operate on known facts (e.g. the occurance of a certain error message) and use a set of rules (the expert knowledge) to derive an inference (the classification of the error condition and a suggested remedy). They are used in a wide range of applications but it was realized [1] that their rather static reasoning is not well suited to the analysis of error messages that occur in connection with data acquisition and control systems. Classic expert systems are not designed to operate in the time domain whereas in our application a lot of the information is contained in the rate and sequence (in time) of error messages. An effective approach for the processing and classification of streams of events (i.e. error messages) is therefore the *complex event processing* paradigm [2]. This technique is applied successfully in particle physics experiments [3] and we are exploring it for arrays of imaging atmospheric cherenkov telescopes (IACTs).

3. Complex Event Processing and IACT Arrays

Complex event processing (CEP) is a general approach to the flexible and fast software-based processing of events that occur in time. The events (e.g. stock orders, purchases in an online shop, error messages from a DAQ system) and their properties are defined by the user of the CEP system and fed into a so-called CEP engine. The CEP engine either assigns a timestamp to each event or uses existing timestamps, and allows the execution of many parallel queries on streams of input events. The queries are also defined by the user when the CEP engine is configured and can take advantage of the event properties and the functionality (e.g. filtering, averaging, pattern matching) provided by the CEP engine. If a certain query is successful, an associated user-supplied code is executed and receives information about the event(s) that matched the query.

In the application of the CEP paradigm to error detection and recovery for a DAQ system the input streams are defined by error messages as well as general log data and possibly further monitoring information (e.g. the available disk space). The expert knowledge is given to the CEP engine in the form of suitable queries which, when successful, would initiate error recovery actions. These recovery actions can comprise suggestions for the DAQ operators and/or automatic execution of recovery routines.

For the DAQ and control systems of current-generation IACT arrays (H.E.S.S., MAGIC, VERITAS) automatic error detection and error recovery have not been a priority since the level of complexity was limited: the number of telescopes was small (2–5) and the telescopes were within easy reach of operators and maintenance personnel. A good data-taking efficiency could be achieved with the help of dedicated DAQ experts advising the operators/shift crews and by a suitable documentation (e.g. in wikis) of known failure modes of the hardware and software.

The Cherenkov Telescope Array (CTA, [4]) is planned as the next-generation facility for ground-based very high energy gamma ray astronomy. Improved flux sensitivity (in the energy band from few 10 GeV to several 100 TeV) and angular resolution will be attained by placing O(100) and O(20) Cherenkov telescopes on sites with an area of O(1 km^2) in the southern and northern hemisphere, respectively. The increase in the total number of telescopes, the operation of different telescope types, more complex observation modes and the nature of CTA as an open observatory pose a challenge for the availability and reliability of the deployed hardware and the DAQ software. It is therefore important to explore the tooling for automatic error detection and recovery and to assess whether it could be combined with the DAQ and control software that is currently being developed for CTA [5].

Among the current CEP tools, the freely available Esper [6] package stands out due its performance, its expressive language EPL (event processing language) for the notation of queries, and its application in DAQ systems of substantial complexity [3]. Though targeting CTA in the long term we have developed a test system for the H.E.S.S. experiment [9] to deal with the error conditions occuring in a real astronomical installation.

4. The Esper CEP Engine

Esper [6] is a Java-based CEP engine running on a Java Virtual Machine (JVM). It parses streams of incoming events for user-defined queries and is able to initialize desired responses. The ingredients for the setup of the CEP engine are:

- *Event objects* describing properties of parsed events
- *Queries* (called statements) defining patterns of interest in arriving events
- *Listener objects* coding responses to firing statements

Event objects can consist of any form of structured reoccurring information, e.g. log messages or similar monitoring information. The current Esper CEP test setup uses a single stream of event objects whose structure is defined by log messages with properties such as log type, sending process and message. EPL provides several approaches to define event patterns for the construction of search queries. To formulate such statements the events can e.g. be grouped by their parameters, aggregation functions (e.g. *mean*, *min*, *max* calculations) and regular expression queries can be applied to filter for certain event properties. There is also the option to define time- or batch-windows which define the consideration time of an event for a search pattern. The EPL statements can be nested in an arbitrary manner which allows a broad range of query definitions. In a listener object the response to a pattern within the event stream is formulated in Java syntax. Therefore the action on a firing statement can be anything from a simple printout on a terminal to a fully automated execution of an error recovery procedure. To initialize a response for a statement one or more listener objects are *added* to the statement. An example is given in Fig. 1.

```
EPStatement findError = cepAdmin.createEPL(
  "context SortByProcess select * from
  LogLine(logType='Error').win:time(60 sec).std:firstunique(logMsg)
  where logMsg regexp '.*Tracking.*'")
findError.addListener(new Listener.ErrorSolution())
```

Figure 1: Schematic example statement demonstrating EPL syntax. The statement selects logs of type "Error" with keyword "Tracking" in the event property logMsg. To suppress multiple output only events with a unique logMsg within one minute trigger the solution process. The context sorts the incoming events by sending processes.

5. A Test System for H.E.S.S.

At the moment, Esper is run within a test environment of the H.E.S.S. DAQ system [7] (a simplified emulation of the real DAQ system) which is a C++/Python-based software package that uses CORBA for internal communication. In this test environment, Esper is implemented in a stand-alone software package which is run as an optional process. It is initialized and terminated by hand and does not interfere with other DAQ processes. The Esper engine is connected to the H.E.S.S. Message-GUI via the Py4J package [8] which makes it possible to access Java code from within Python, a convenient way to transfer log data between the C++/Python-based H.E.S.S. DAQ and the Java-based Esper CEP engine. The schematic structure of the data flow in the system is shown in Fig. 2. The log messages arrive at the Message-GUI via a CORBA message server which manages the communication between processes within the H.E.S.S. DAQ system. When a new message is received, a Python function tries to update a sender object with the message parameters via a gateway server which is provided by Py4J. This sender object then creates a new log event and feeds it into the stream of incoming log events which are then parsed by the CEP engine.

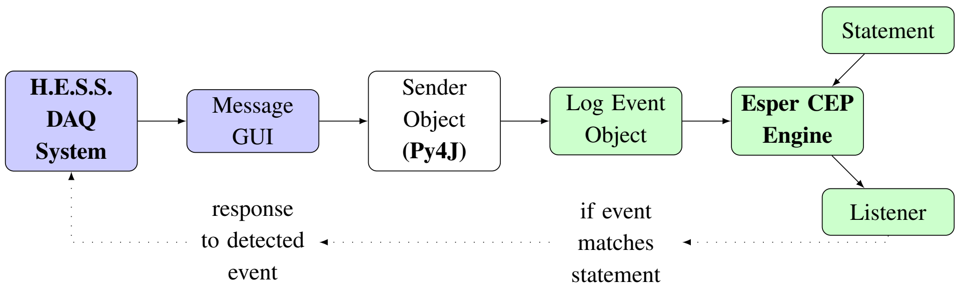

Figure 2: Schematic data flow between the C++/Python-based H.E.S.S. DAQ (blue) and the Java-based Esper processes (green) in the test setup.

If a statement fires and activates a listener, the underlying response is executed. In the test system, the response is for now limited to a printout of descriptive guidelines for the H.E.S.S. shift-crew to recover the system from an error state. The information is gathered in a database in which the subsystem experts provide solutions for given error scenarios. For the future it is conceivable to allow activated listeners to execute scripts which then perform the suggested recovery steps automatically.

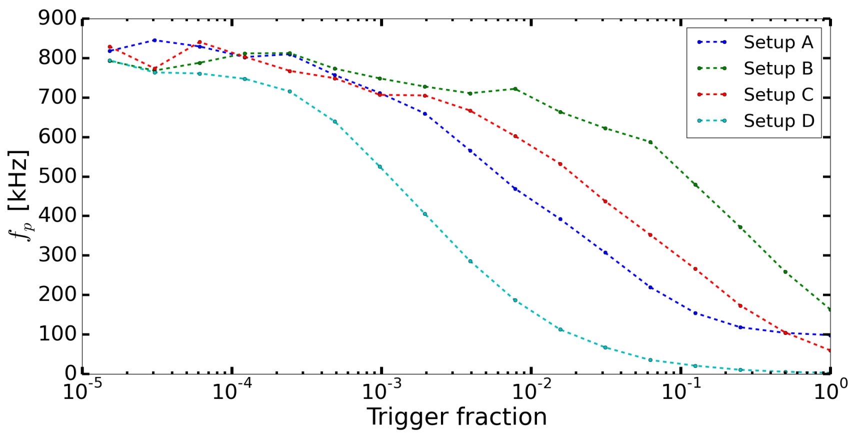

Figure 3: Performance benchmark plot with processing frequency f_p plotted vs. trigger fraction, the fraction of error events in the parsed stream which trigger a response. f_p is the rate at which events are parsed by the CEP engine. Firing statements print logs in a Java-Swing-GUI (*Setup A*: one statement parsing for error messages, *Setup B, C, D*: 5, 20, 200 statements firing on different unique error messages within a time window). It is notable, that all setups reach a maximum processing frequency of O(800 kHz) for low trigger rates.

The rate at which events can be processed and the memory consumed by Esper depend on the listeners and statements used in a setup. As seen in Fig. 3 setup A, a system with one statement querying for error messages and displaying these in a Java-Swing GUI, the processing rate (dual-core processor up to 3.6 GHz) is of the order of 100 - 800 kHz depending on the rate of error messages in the parsed stream. Setup B, C, D on the other hand resemble a more application-oriented setup of 5, 20, 200 statements respectively which parse the stream for different error messages via regular expressions and suppressing multiple printouts within a given time window (as in Fig.1). All four setups show a maximum processing frequency of O(800 kHz) for lower trigger rates which shows that for the tested setups, the performance occurs to be dominated by the reaction process initialized by the listeners rather than the number of statements. The overall performance of a system is also given by the performance of the running JVM and can therefore be optimised by tuning the JVM appropriately.

6. Summary and Outlook

Within the H.E.S.S. test environment, Esper has shown promising results regarding performance, flexibility and the broad range of different constructs provided by EPL. An application of the developed software package as a shift-support tool within the DAQ system used at the H.E.S.S. site in Namibia is in preparation. In the first stage the tool will provide recovery information to the shifters in certain fault-states of the system. With enough practical experience automated error recovery can be explored. This setup is viewed as a key test for applications of an Esper CEP-engine in CTA.

References

[1] Avolio, G. et al., *J. Phys.: Conf. Ser.* **396** (2012) 012003.

[2] Luckham, D., Addison-Wesley, ISBN 0-201-72789-7.

[3] Kazarov, A. et al., *J. Phys.: Conf. Ser.* **368** (2012) 012004.

[4] Acharya, B. S. et al. [CTA Consortium], *Astroparticle Physics* **43** (2013) 3.

[5] Wegner, P. et al., Proceedings of ICALEPCS 2013, San Francisco (2013).

[6] http://www.espertech.com/esper/

[7] Balzer, A. et al., *Astroparticle Physics* **54C** (2014) 67–80.

[8] http://py4j.sourceforge.net/

[9] Aharonian, F. et al. [H.E.S.S. Collaboration], *Astron. Astrophys.* **457** (2006) 899.