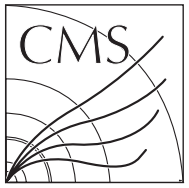
Available on CMS information server

CMS CR -2009/064



The Compact Muon Solenoid Experiment

Conference Report

Mailing address: CMS CERN, CH-1211 GENEVA 23, Switzerland

08 May 2009

An Assessment of a Model for Error Processing in the CMS Data Acquisition System

Schahram Dustdar [2)], Johannes Gutleber[1)], Roland Moser[1,2)], Luciano Orsini[1)]

Abstract

The CMS Data Acquisition System consists of O(20000) interdependent services. A system providing exception and application-specific monitoring data is essential for the operation of such a cluster. Due to the number of involved services the amount of monitoring data is higher than a human operator can handle efficiently. Thus moving the expert-knowledge for error analysis from the operator to a dedicated system is a natural choice. This reduces the number of notifications to the operator for simpler visualization and provides meaningful error cause descriptions and suggestions for possible countermeasures. This paper discusses an architecture of a workflow-based hierarchical error analysis system based on Guardians for the CMS Data Acquisition System. Guardians provide a common interface for error analysis of a specific service or subsystem. To provide effective and complete error analysis, the requirements regarding information sources, monitoring and configuration, are analyzed. Formats for common notification types are defined and a generic Guardian based on Event-Condition-Action rules is presented as a proof-of-concept.

Presented at *17th International Conference on Computing in High Energy and Nuclear Physics,21 - 27 March 2009,Prague,Czech Republic,15/05/2009*

[1)] CERN, Geneva, Switzerland

[2)] Technical University of Vienna, Vienna, Austria

An Assessment of a Model for Error Processing in the CMS Data Acquisition System

S Dustdar[1], J Gutleber[2], R Moser[1,2] and L Orsini[2]

[1] Technical University of Vienna, Karlsplatz 13, 1040 Vienna, Austria

[2] CERN, 1211 Gevena 23, Switzerland

E-mail: dustdar@infosys.tuwien.ac.at, johannes.gutleber@cern.ch, roland.moser@cern.ch, luciano.orsini@cern.ch

Abstract. The CMS Data Acquisition System consists of O(20000) interdependent services. A system providing exception and application-specific monitoring data is essential for the operation of such a cluster. Due to the number of involved services the amount of monitoring data is higher than a human operator can handle efficiently. Thus moving the expert-knowledge for error analysis from the operator to a dedicated system is a natural choice. This reduces the number of notifications to the operator for simpler visualization and provides meaningful error cause descriptions and suggestions for possible countermeasures. This paper discusses an architecture of a workflow-based hierarchical error analysis system based on Guardians for the CMS Data Acquisition System. Guardians provide a common interface for error analysis of a specific service or subsystem. To provide effective and complete error analysis, the requirements regarding information sources, monitoring and configuration, are analyzed. Formats for common notification types are defined and a generic Guardian based on Event-Condition-Action rules is presented as a proof-of-concept.

1. Introduction

The CMS Data Acquisition System [6] consists of O(20000) interdependent services. A system providing exception and application-specific monitoring data is essential for the operation of such a cluster.

Due to the number of involved services the amount of monitoring data is higher than a human operator can handle efficiently. Thus moving expert-knowledge for error analysis from the operator to a dedicated error processing system is a natural choice. This reduces the number of notifications to the operator for simpler visualization and provides meaningful error cause descriptions and suggestions for possible countermeasures.

2. Technologies

The CMS data acquisition system follows a service-oriented architecture (SOA) [1][8] where each service provides a SOAP control interface [10]. High-level data acquisition applications have been implemented using the XDAQ framework [7], which also provides fundamental infrastructure services for error processing, such as a distributed monitoring system. The XDAQ monitoring and alarming system (XMAS) [2] infrastructure is based on a scalable and distributed publish/subscribe eventing system [3] and currently handles O(100000) notifications per second.

Continuing a service based approach, we implemented an error processing system with Web Workflows. We chose the ActiveBPEL workflow engine [4], which combines workflows with SOAP based web services.

3. Requirements

We have identified the following requirements to build an extensible error processing system for the CMS data acquisition system:

- Access to **run-time** and **configuration information** in a standardized format.
- Allow extension with custom error processing components through a **defined interface**.

3.1. Run-time and configuration information

Run-time information **represents the actual condition of the running system and can be categorized as shown in**

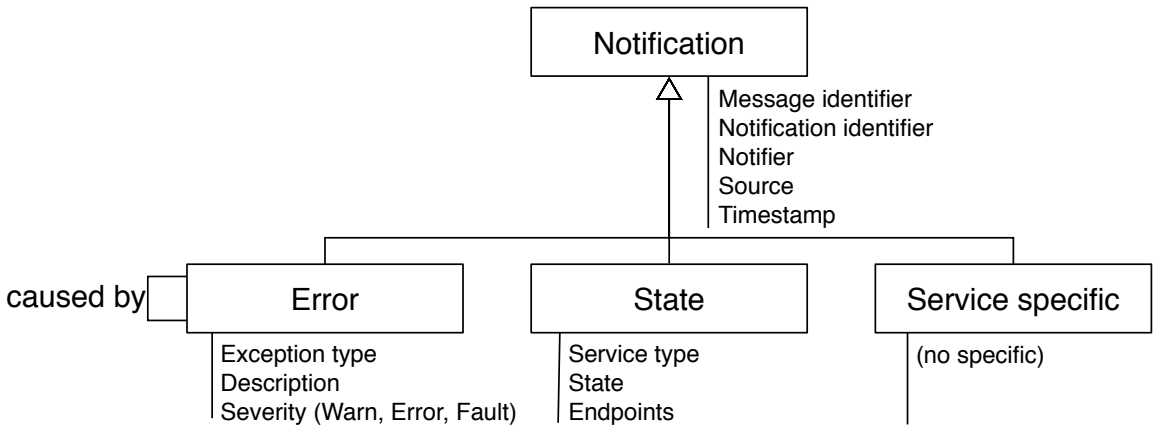


Figure 1:

State information **contains information about the actual state of services. With hierarchical states as defined in ASAP [5] we can impose general states for visualization and error processing and allow refinement when necessary for control (**

- Figure 2).
- **Error information** describes exceptions, which could not be handled locally by services. It embeds a complete exception trace for debugging. In addition custom properties can be added at each level of the exception trace to provide further information for error processing in an automated fashion.
- **Service information** contains dynamic data ranging from statistics to configuration data not known a priori. It is freely definable and usually specific to applications.

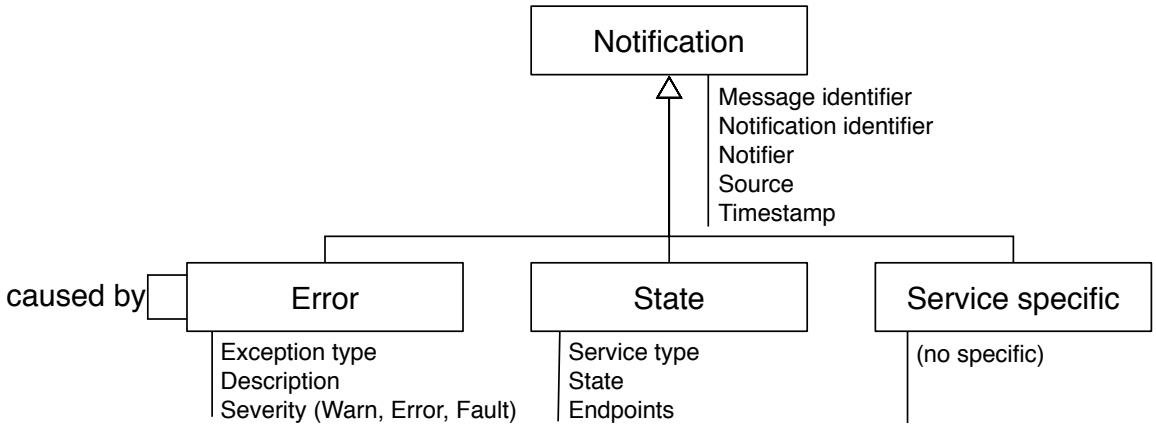


Figure 1 Notifications and their primary properties.

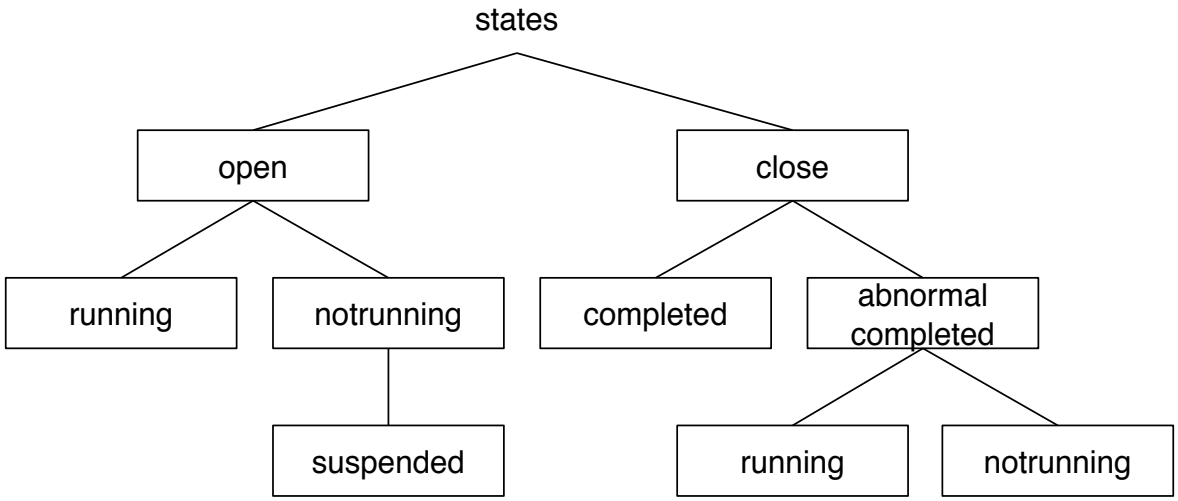


Figure 2 Hierarchical states allowing refinement and generalization.

Configuration information represents the nominal condition of the running system. It can be categorized in hardware and software information. Hardware information describes the setup of hosts, devices and networks. Software information specifies applications, services and communication endpoints.

4. Error Processing Architecture

A high-level error processing system is responsible to detect the cause of errors on startup and operation of the monitored system. Therefore it analyzes differences between actual and nominal condition. The general architecture of our error processing system is depicted in

Figure 3. The data layer contains services, which may emit data into the monitoring system and alarming system. The logic layer contains the monitoring and error processing system and the visualization layer contains the graphical user interface the operator interacts with.

The *error processing system* contains of two kinds of services, an *Error Processor* and *Guardians*. In our system the **Error Processor** is an intermediate, which subscribes to the monitoring system for retrieving error notifications and forwards them to error processing components, called Guardians.

Based on their responses the Error Processor forwards notification to the operator and monitoring system accordingly. If exceptions could not be matched to an error cause it informs the operator to refine the rules for notifications based on their unique identifier.

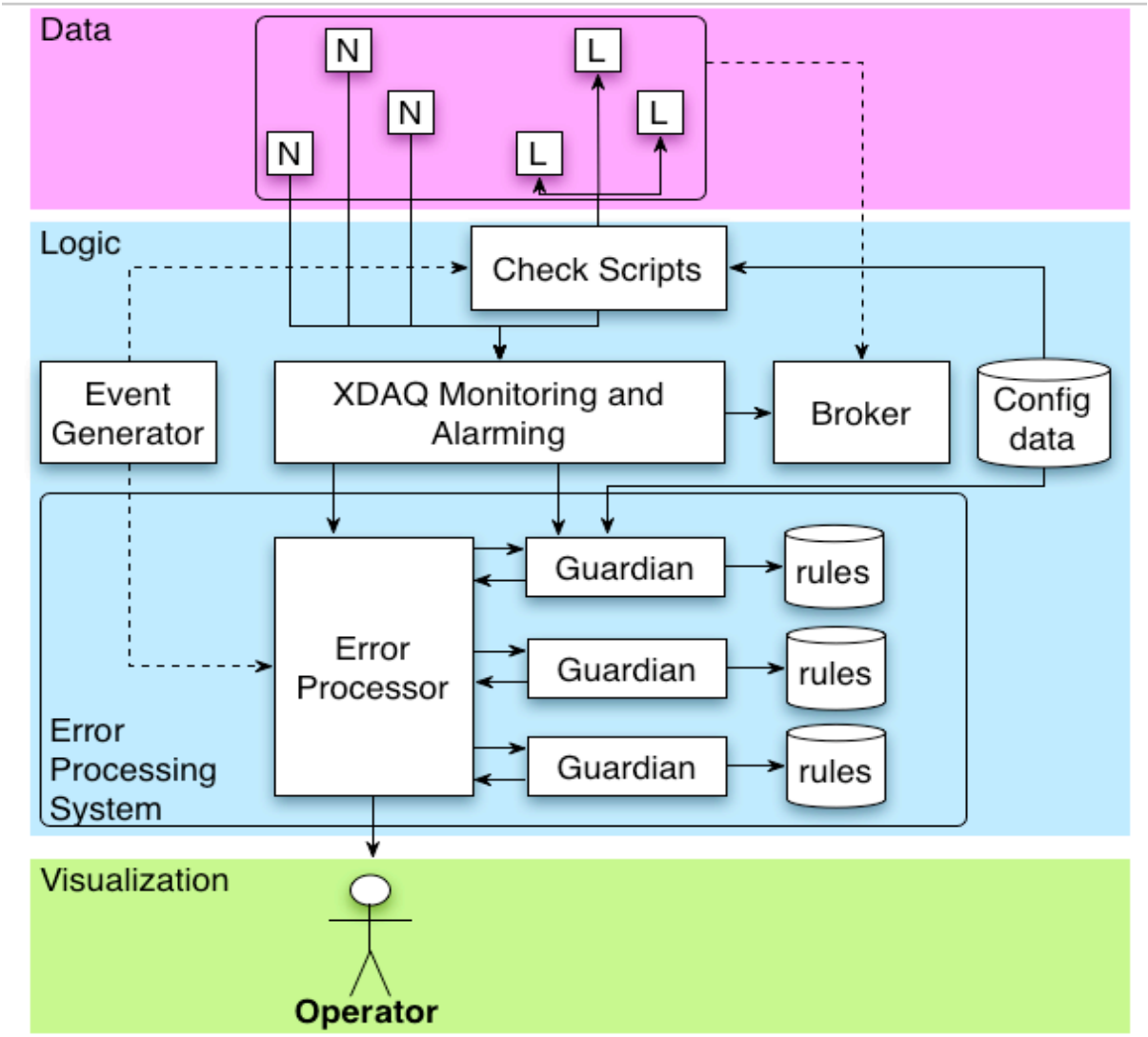


Figure 3 Principle Architecture containing native XMAS services (N) and legacy services (L).

Guardians are hierarchically ordered error analysis components and contain expert knowledge about specific services or subsystems (Figure 4). The lowest layer guards observe specific services whereas the higher ones observe groups of services. In case a guardian cannot identify the cause of an error directly it may emit an exception, which is passed to a higher-level Guardian. Error processing should always be done on the lowest possible layer without incorporating knowledge about other subsystems or services. This keeps the higher-level guardians abstract and confined to their respective group of applications. In case a Guardian could identify the cause of an error it may return an operator notification, which is forwarded, to the operator by the Error Processor.

All guardians provide the same SOAP interface and as such may be implemented in any language. This allows integration with already existing rule-based systems or custom error processing code in case a generic Guardian is insufficient. The request message to the Guardians contains a list of error

notifications and a list of URLs of monitoring data servers, which may be queried for more information. The response message contains notifications for an operator if an error cause could be identified or an error notification, which is propagated, to a higher level Guardian. It additionally encloses a list of matched notification identifiers.

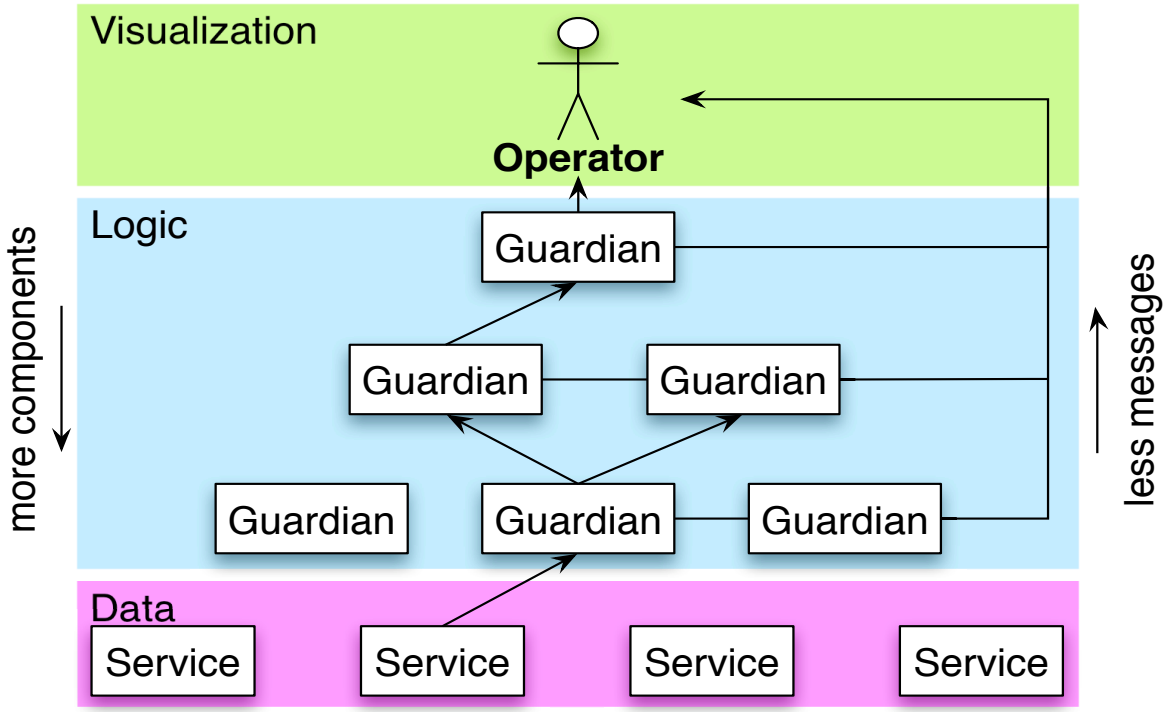


Figure 4 Error propagation (arrows in the middle) and operator notifications (arrows on the right).

We chose to implement error processing using BPEL as it already provides powerful languages for filtering (XPath) [12] and querying (XQuery) [11] XML data. Using those features we implemented also a generic Guardian, which processes *Event-Condition-Action* (ECA) rules [9]. A rule that checks the diskUsage of our computers is shown in Figure 5. This is an example of a rule which is not triggered by an error notification but triggered periodically and checks service-specific information.

```
<eca xmlns:tns="http://xdaq.web.cern.ch/xdaq/wsdl/2008/guardianeca-10.wsdl">
 <source type='flashlist' name='diskInfo'>urn:xdaq-flashlist:diskInfo</source>
 <rule>
  <condition>/*/source/diskInfo/table/rows[ diskUsage/rows[xs:double(usePercent/text())>90] ] </condition>
  <action>
   <inform>
    <message>free disk space below 10 percent</message>
    <services source="condition">/*/rows/context</services>
   </inform>
  </action>
 </rule>
</eca>
```

Figure 5 ECA rule for generic Guardian detecting low disk space.

5. Enhancements

During evaluation of existing workflow engines we identified some shortcomings of BPEL and missing components necessary for integration with our system:

- BPEL workflows can only be triggered through SOAP messages and not through timers or even more complex rules.
- ActiveBPEL natively supports only SOAP based protocols.
- BPEL does not support to model an organizational perspective [15] and mapping of services to invoke activities must be modelled explicitly.

To overcome those shortcomings we implemented several additional services as depicted in Figure 3.

The **Event Generator** is a service, which sends SOAP messages based on predefined rules. Rules may match on workflow engine, timing and external user events. This allows periodic triggering of workflows and avoids ever running workflows, both concepts that are not supported by BPEL natively. The rule in Figure 6 shows a timing event emitted (MinuteTimer:trigger) once every 60 seconds. The timer is started based on the internal start event that is emitted as soon as the servlet engine in which the Event Generator is running is started. The second rule presented in Figure 7 starts a workflow which checks if all discovery services daemons [14] in our cluster are running and fully functional. The rule specifies that specific SOAP request message to be sent to a web service based on the previously mentioned timer event. This allows calling web services with without enforcing a specific interface on them.

```
<?xml version='1.0'?>
<netflow:event xmlns:netflow="http://xdaq.web.cern.ch/xdaq/xsd/2006/netflow-event-10">
  <netflow:component activated="true" changeable="false" class="ch.cern.cms.wf.event.Timer">
    <netflow:item name="name">MinuteTimer</netflow:item>
    <netflow:item name="type">timer</netflow:item>
    <netflow:item name="description">Timer for executing scripts once per minute </netflow:item>
    <netflow:item name="maxinstances">1</netflow:item>

    <netflow:item name="period">PT60S</netflow:item>
  </netflow:component>

  <netflow:bind xpath="/netflow:event[@name='internal' and @command='start']">
    <netflow:event name="MinuteTimer" command="start">
      <!-- contains SOAP message to send out if component supports that -->
    </netflow:event>
  </netflow:bind>
  <netflow:bind xpath="/netflow:event[@name='internal' and @command='stop']">
    <netflow:event name="MinuteTimer" command="stop">
      <!-- contains SOAP message to send out if component supports that -->
    </netflow:event>
  </netflow:bind>
</netflow:event>
```

Figure 6 Event Generator rule of a timer emitting an event once per minute.

```
<?xml version='1.0'?>
<netflow:event xmlns:netflow="http://xdaq.web.cern.ch/xdaq/xsd/2006/netflow-event-10">
  <netflow:component activated="false" changeable="true" class="ch.cern.cms.wf.event.Workflow">
    <netflow:item name="name">slpcheck</netflow:item>
    <netflow:item name="type">workflow</netflow:item>
    <netflow:item name="description">Script for checking if SLP daemons</netflow:item>
    <netflow:item name="maxinstances">1</netflow:item>
  </netflow:component>

  <!-- Event Bindings between internal components -->
  <netflow:bind xpath="/netflow:event[@name='MinuteTimer' and @command='trigger']">
    <netflow:event name="slpcheck" command="start">
      <ns1:StartServiceRequest xmlns:ns1="http://xdaq.web.cern.ch/xdaq/wsdl/2007/wfcheck-10.wsdl"/>
    </netflow:event>
  </netflow:bind>

  <!-- External (User) emitted events -->
  <netflow:emittable from='user' to='slpcheck' description='activate'>
    <netflow:event name="slpcheck" command="activate"/>
  </netflow:emittable>
  <netflow:emittable from='user' to='slpcheck' description='deactivate'>
    <netflow:event name="slpcheck" command="deactivate"/>
  </netflow:emittable>

</netflow:event>
```

Figure 7 Event Generator rule for triggering a web service (workflow) based on a timer event.

The **Broker** is a component for dynamically allocating resources and services according to *Quality of Service* (QoS) requests. It works with models for different scenarios. For example, the model for the monitoring system implements a load balancer for periodically allocating monitoring services to O(20000) services. This model itself relies on monitoring information, e.g. CPU load, to provide a scalable monitoring infrastructure. Another example where a model would be useful is the assignment of services to hosts based on QoS attributes instead of statically assigning services to hosts. This can provide improved fault-tolerance and better resource usage in the data acquisition cluster. It will also simplify our workflows, as they will not need the informational perspective to model the organizational one [16].

Integration: As not all services publish directly into XMAS we added custom workflow scripts which query the states of those services over SSH and publish their information into XMAS through SOAP messages. In addition some services use a custom, binary protocol for performance reasons.

Although WSDL allows defining interfaces independent of transport protocols, the ActiveBPEL engine only supports SOAP over HTTP as a protocol by default. ActiveBPEL solves this problem by providing InvokationHandlers, which translate between internal workflow engine data representation (XML) and custom formats and protocols and therefore allowing seamless integration with our system at hand.

6. Summary

This paper summarizes requirements and pitfalls during design and implementation of a generic error processing system using the CMS experiment as a case study. The presented error processing architecture relies on workflow and Web service technologies, which allow seamless integration into the existing environment. We implemented a generic workflow-based Guardian, which performs error processing, based on ECA rules. Tests of the system were performed in the operational environment of the CMS data acquisition system and different kinds of error causes have been successfully identified.

Due to the standardized notification formats integration with other existing monitoring systems is feasible and would allow extending the scope of error processing beyond the core data acquisition applications. In addition providing a standardized interface for Guardians will allow us to take advantage by integrating distributed rule business rule engines [13] and already existing error processing components in the future.

References

[1] Booth D et al 2004 Web Service Architecture http://www.w3.org/TR/ws-arch

[2] Bauer G et al 2009 Monitoring the CMS Data Acquisition System *Proc. International Conference on Computing in High Energy and Nuclear Physics in Journal of Physics: Conference Series*.

[3] Box D et al 2006 Web Services Eventing (WS-Eventing) http://www.w3.org/Submission/WS-Eventing/

[4] ActiveBPEL Engine – official homepage http://www.activevos.com/community-open-source.php

[5] Fuller J, Krishnan M, Swenson K, Ricker J 2005 Asynchronous Service Access Protocol (ASAP) Version 1.0 http://www.oasis-open.org/committees/asap/

[6] Gutleber J, Murray S, Orsini L 2003 Towards a homogeneous architecture for high-energy physics data acquisition systems *Elsevier Comp. Phys. Comm.* **153(2)** 155-163.

[7] Gutleber J, Moser R, Orsini L 2007 *Data Acquisition in High Energy Physics* Proc. Astronomical Data Analysis Software and Systems (ADASS) XVII, **394** 47.

[8] CERN 2002 *Data Acquisition & High-Level Trigger, Technical Design Report* CMS TDR 6.2, LHCC 2002-26 (ISBN 92-9083-111-4).

[9] Chen L, Li M, Cao J, Wang Y 2005, An ECA Rule-based Workflow Design Tool for Shanghai Grid, *2005 IEEE International Conference on Services Computing* **1** 325-328.

[10] Gutleber J et al 2005 HyperDAQ Where Data Acquisition Meets the Web *Proc. 10th Intl. Conf. Accel. and L. Exp. Phys. Control Sys.* (Geneva, Switzerland, 10-14 October 2005).

[11] Scott B et al 2007 XQuery 1.0: An XML Query Language http://www.w3.org/TR/xquery/

[12] Berglund A et al 2007 XML Path Language (XPath) 2.0 http://www.w3.org/TR/xpath20/

[13] Nagl C, Rosenberg F, Dustdar S 2006 VIDRE - A Distributed Service Oriented Business Rule Engine based on RuleML *Proc. 10th IEEE International Enterprise Distributed Object Computing Conference*. EDOC 2006: 35-44.

[14] Guttman E, Perkins C, Vaizades J and Day M 1999 Sevice Location Protocol Version 2 Internet RFC http://www.ietf.org/rfc/rfc2608.txt

[15] Russell N, ter Hofstede A, Edmond D, van der Aalst W 2004 *Workflow Data Patterns.*

[16] Zur Muehlen M 2004 *Workflow-based Process Controlling: Foundation, Design, and Application of Workflow-driven Process Information Systems*, Logos, http://books.google.com/books?id=EpgxaWJwkFQC