Inaugural-Dissertation

zur

Erlangung der Doktorwürde

der

Naturwissenschaftlich-Mathematischen Gesamtfakultät

der

Ruprecht-Karls-Universität Heidelberg

Korrigierte Fassung

vorgelegt von	Diplom-Informatiker **Camilo Ernesto Lara Martinez** aus Bogota - Kolumbien
Tag der mündlichen Prüfung	19. Januar 2012

The SysMES Framework: System Management for Networked Embedded Systems and Clusters

Camilo Ernesto Lara Martinez

Gutachter: Prof. Dr. Udo Kebschull
Prof. Dr. Michael Gertz

The SysMES Framework: System Management for Networked Embedded Systems and Clusters

Automated system management for large distributed and heterogeneous environments is a common challenge in modern computer sciences. Desired properties of such a management system are, among others, a minimal dependency on human operators for problem recognition and solution, adaptability to increasing loads, fault tolerance and the flexibility to integrate new management resources at runtime. Existing tools address parts of these requirements however there is no single integrated framework which possesses all mentioned characteristics.
SysMES was developed as an integrated framework for automated monitoring and management of networked devices. In order to achieve the requirements of scalability and fault tolerance, a fully distributed and decentralized architecture has been chosen. The framework comprises a monitoring module, a rule engine and an executive module for the execution of actions. A formal language has been defined which allows administrators to define complex spatial and temporal rule conditions for failure states and according reactions. These rules are used in order to reduce the number and duration of manual interventions in the managed environment by automated problem solution. SysMES is based on standards ensuring interoperability and manufacturer independence. The object-oriented modeling of management resources allows several abstraction levels for handling the complexity of managing large and heterogeneous environments. Management resources can be extended and (re)configured without downtime for increased flexibility. Multiple tests and a reference installation demonstrate the suitability of SysMES for automated management of large heterogeneous environments.

Das SysMES Framework: System Management für Vernetzte Eingebettete Systeme und Cluster

Automatisiertes System Management für verteilte und heterogene Umgebungen ist eine derzeitige Herausforderung der Informatik. Gewünschte Eigenschaften eines solchen Systems sind unter anderem eine möglichst geringe Abhängigkeit von menschlichen Arbeitskräften für Problemerkennung und -lösung, Anpassungsfähigkeit an variierende Last, Fehlertoleranz und Flexibilität in Bezug auf die Integration neuer Managementressourcen zur Laufzeit. Vorhandene Tools decken Teile dieser Anforderungen ab, es gibt jedoch kein umfassendes und integriertes Framework, welches alle diese Charakteristiken besitzt.
SysMES wurde als integriertes Framework für das automatisierte Monitoring und Management verteilter Ressourcen entwickelt. Um den Zielstellungen der Skalierbarkeit und der Fehlertoleranz zu genügen, wurde eine vollständig verteilte und dezentrale Architektur konzipiert. Das Framework umfasst ein Monitoring-Modul, eine Rule-Engine und ein Ausführungsmodul verantwortlich für die Ausführung von administrativen Aktionen. Für die Spezifikation von Fehlerzuständen auf Basis von komplexen räumlichen und zeitlichen Zusammenhängen und geeigneter Lösungsmöglichkeiten wurde eine formale Sprache entwickelt. Die so entstehenden Regeln werden von der Rule-Engine verarbeitet und ermöglichen dadurch eine automatisierte Problembehandlung. Dies führt zu einer Reduktion der Menge und Dauer manueller Eingriffe. Die SysMES Implementierung basiert auf Standards und realisiert damit eine weitgehende Interoperabilität und Herstellerunabhängigkeit. Die objektorientierte Modellierung der Managementressourcen ermöglicht ihre Beschreibung auf verschiedenen Abstraktionsebenen und vereinfacht daher den Umgang mit der Komplexität einer großen und heterogenen Umgebung. Dergestalt modellierte Managementressourcen können, im Sinne erhöhter Flexibilität, zur Laufzeit modifiziert und erweitert werden. Multiple Testserien und eine Referenzinstallation zeigen die Erfüllung der theoretischen Anforderungen sowie den praktischen Nutzen des entwickelten Systems für das Management großer, heterogener Umgebungen auf.

Acknowledgements

It is time to submit my thesis and close one of the most important chapters in my life. During the last years I enjoyed the support of many people to whom I want to express my thankfulness today.

I would like to thank Prof. Dr. Udo Kebschull and Prof. Dr. Volker Lindenstruth for the opportunity to work in their department and the trust they put in me. Many thanks to Prof. Dr. Michael Gertz for his willingness to review my thesis.

My most sincere thanks to all the (ex)members of the SysMES group especially to Stefan Boettger, Jochen Ulrich and Timo Breitner for their sensational collaboration, their support for the realization of this thesis and their patience reviewing it.

My very grateful thanks to the members of the ALICE HLT Collaboration especially to Jochen Thaeder, Torsten Alt, Timm Steinbeck and Oystein Haaland for a great time during the realization of this project.

I also want to thank some friends of the Kirchhoff Institute for Physics, who have helped me to feel good at my new home Heidelberg over the last years. Beatrice, Claudia, Thomas, Robert, Helmut, Eike, I will come back.

Thanks to my friends Sebastian Kalcher, Ralf Panse and Sebastian Manz for advice and for enduring me bravely.

Very special thanks to my uncle Gregorio Martinez and to Julia Hofmann who did not give up on improving my English and to Fernando Acuña, Jan de Cuveland and Dirk Huetter, who helped me with the layout of my thesis.

I would like to thank my Colombian family especially my Dad, Javier and Felipe, my German family especially Helmut and Ilsemarie, my family-in-law, especially Ingrid and Andreas, as well as my friends Anja and Antje, who always supported me and who were excitedly waiting for the finishing of my thesis.

At last I want to thank my small family Yvonne and Michelle who always gave me power to continue this way.

Dedicado a las tres mujeres mas importantes en mi vida
Maria Cristina, Yvonne y Michelle Cristina.

Camilo Lara

Heidelberg, May 2011

Contents

List of Figures

List of Tables

1. Introduction

Computer clusters are a consolidation of interconnected servers or nodes working together with a common objective. According to this objective, clusters can be used for increasing the availability of applications (e.g. building redundancy in a high availability cluster), for increasing the load that can be handled (e.g adding new nodes in so-called load balancing clusters) and for increasing the computational power, which can be used to handle load (e.g. by partitioning and parallel processing of jobs in so-called high performance computing clusters).
Concerning their physical infrastructure, computer clusters, can be homogeneous, i.e. all nodes have the same hardware, operating system and software releases, or heterogeneous if there are multiple versions of those.
Computer clusters are counted among the most widely used equipment for both research and industry. A typical usage for clusters arises for example when processing data taken by one of the experiments in CERN (Conseil Europeen pour la Recherche Nucleaire), the largest research facility for particle physics in the world, located near Geneva - Switzerland.

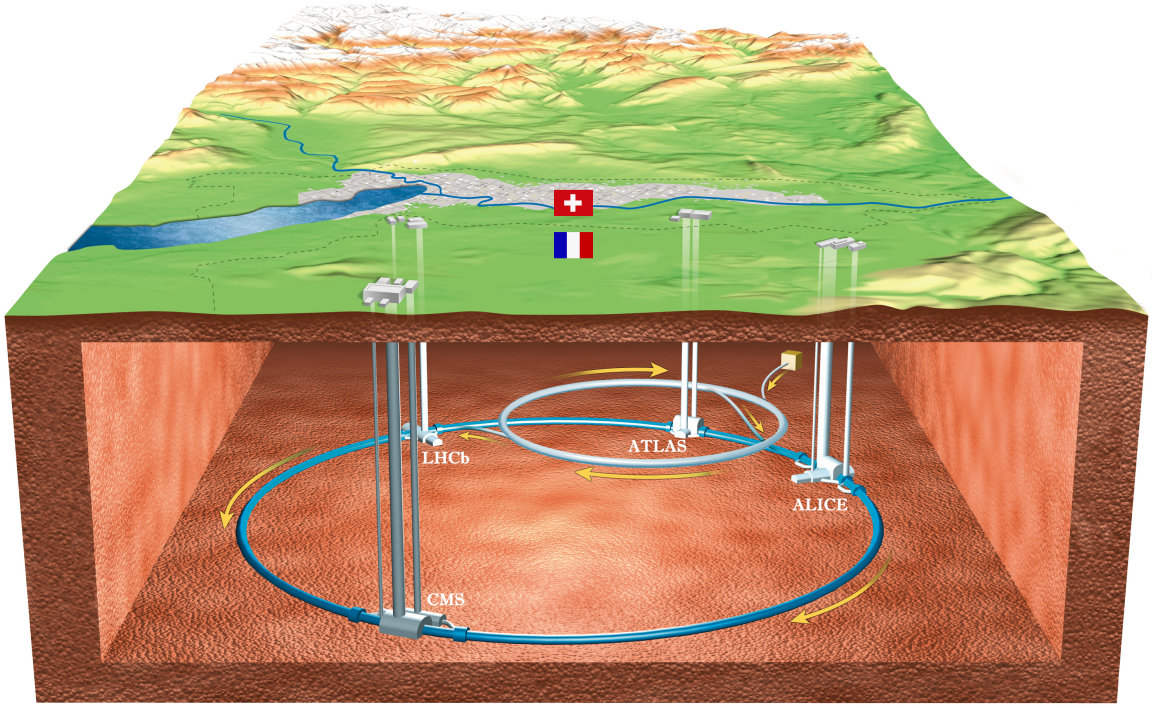
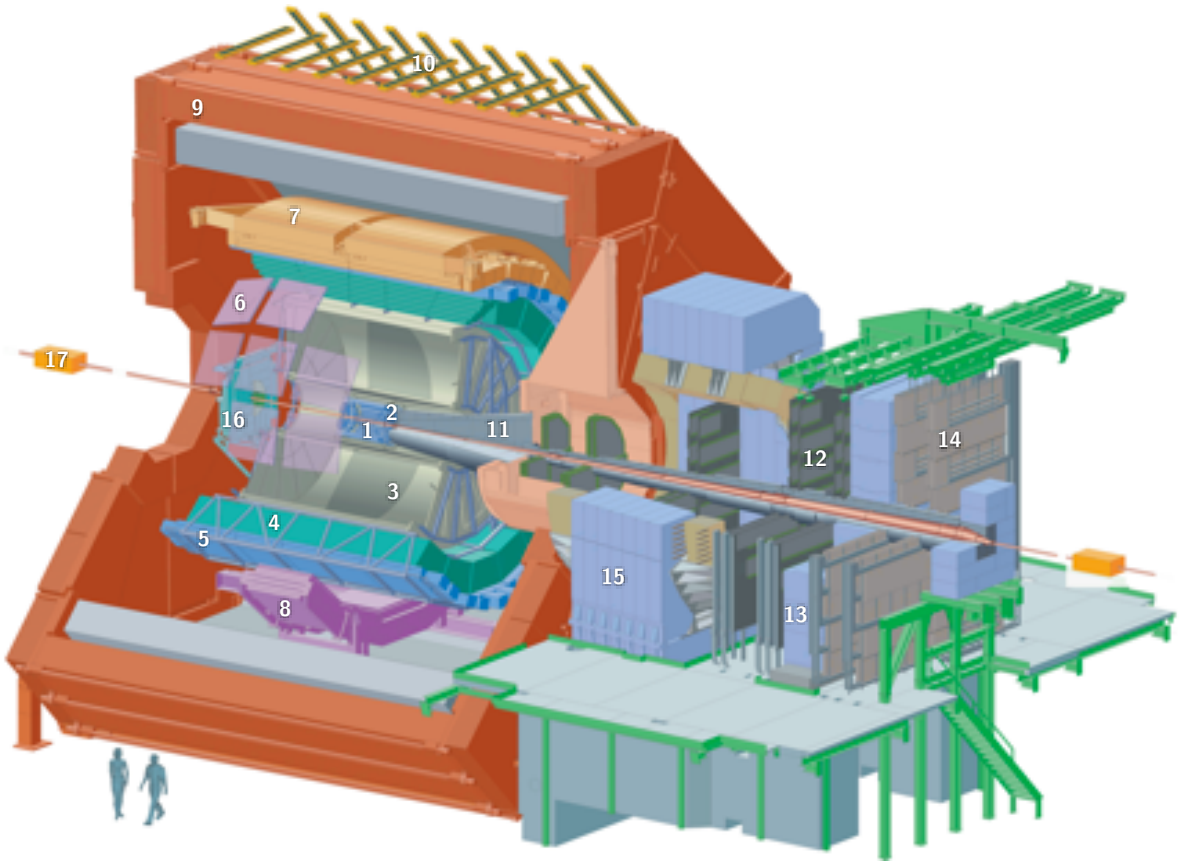


Figure 1.1.: Overall View of the LHC Experiments

The Large Hadron Collider (LHC) is the biggest hadron accelerator and collider in the world [75], [74]. It has been built approx. 100 m underground and it has a circumference of approx. 26.7 km (see



1. ITS (Inner Tracking System)
2. FMD (Forward Multiplicity Detector), T0, V0
3. TPC (Time Projection Chamber)
4. TRD (Transition Radiation Detector)
5. TOF (Time of Flight Detector)
6. HMPID (High Momentum Particle Identification)
7. EMCal (Electromagnetic Calorimeter)
8. PHOS CPV (Photon Spectrometer Charged Particle Veto Detector)
9. L3 Magnet
10. ACORDE (ALICE Cosmic Ray Detector)
11. Absorber
12. Muon Tracking Chambers
13. Muon Filter Wall
14. Muon Trigger Chambers
15. Dipole Magnet
16. PMD (Photon Multiplicity Detector)
17. ZDC (Zero Degree Calorimeter)

Figure 1.2.: The Alice Experiment - Detectors Overview

figure 1.1 [33] [1]). The LHC is in charge of accelerating protons and lead-ions to 99.9999991% of the velocity of light and colliding them in four specific places.
Each collision point is surrounded by of one of the four main experiments, which are: A Large Ion Collider Experiment (ALICE) [2], A Toroidal LHC Apparatus (ATLAS) [1], Compact Muon Spectrometer (CMS) [7] and Large Hadron Collider Beauty (LHCb) [12].
Figure 1.2 [2] visualizes the structure of the ALICE experiment and the required detectors and facilities for collecting collision data.
The ALICE detector system was built to explore the properties of matter at extreme conditions. It is designed to study strong interaction matter. At its extreme, the Quark-Gluon-Plasma (QGP) is supposed to be formed during the very high energy density and temperature of ultra relativistic heavy nucleus-nucleus collisions. Additionally, ALICE has an extensive proton-proton program to obtain reference data for the heavy ion collisions.
The ALICE High Level Trigger (HLT) [10] is a distributed and hierarchical application for online triggering, selecting and compressing of collision data. The triggering functionality is related to ac-

[1] Picture copyright owned by CERN.
[2] Picture copyright owned by CERN.

cept or reject collision data. The selecting functionality is in charge of defining a region of interest (e.g. desired detectors, parts of detectors or a combination of both) for selecting desired data, and if required, to perform triggering on this data. The compressing functionality concerns the reduction of data size without losing physics information. The HLT will be configured before data taking in order to define a combination of those functionalities.
Conforming to the HLT specification and independent of its configuration, it is able to receive 25 GByte/s of data, which has to be analyzed and reduced down to 1,25 GByte/s, which is the maximum value of data which can be stored persistently. Figure 1.3 [3] visualizes the particle tracks in lead-lead-collision in the inner part of the ALICE experiment which represent the data to be analyzed by the HLT.

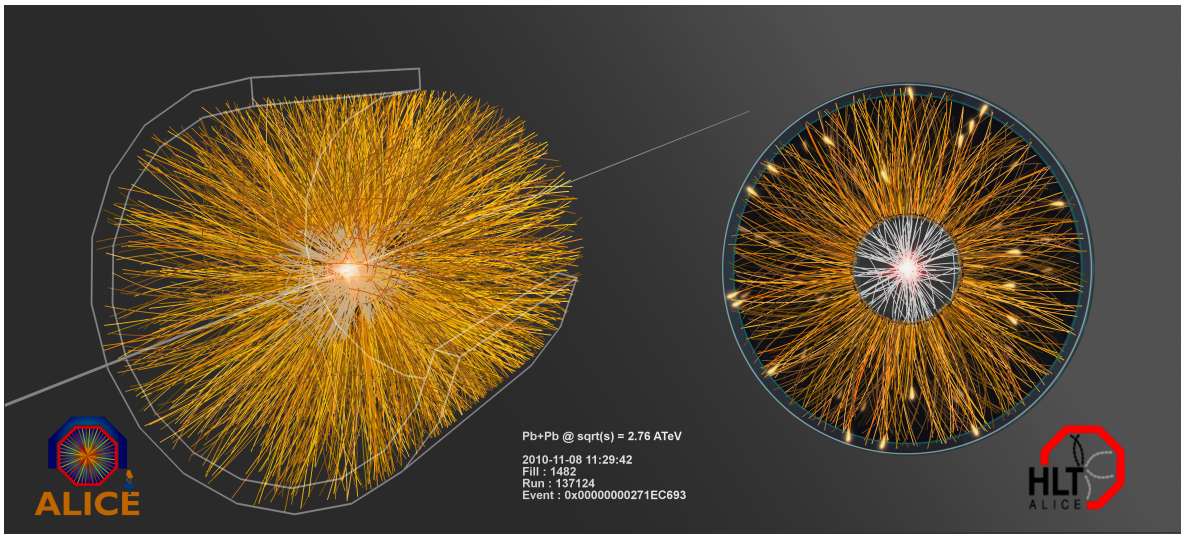


Figure 1.3.: ALICE Experiment - First Heavy-Ion Collisions

The HLT runs in a heterogeneous cluster (the so-called HLT cluster), which is composed of about 200 low-cost commodity hardware nodes and 25 infrastructure servers at the actual commissioning state and will be extended up to about 1000 nodes in the final commissioning state. During the experiment run period, the cluster will be used full time for running the HLT and outside this period for tests and for running offline analysis.
Cluster nodes are divided into two categories: Front-End Processor (FEP) nodes and Computing Nodes (CNs). FEP nodes are used for injecting detector's raw data in the cluster using the so-called High Level Trigger Read Out Receiver Card (H-RORC) [11]. The CNs are used for performing further analysis steps. An extensive description of the HLT cluster follows in section 7.1.1.
The high grade of heterogeneity in the HLT cluster is related to the duration of the ALICE experiment and its multiple commissioning phases. Hardware is bought in batches for each commissioning phase (i.e. according to the required resources) as late as possible in order to get a better value for money. The experiment's duration will be longer than 10 years, which is normally longer than the life time of hardware and therefore nodes have to be replaced (i.e. treatment of legacy hardware). Another aspect of heterogeneity in the HLT cluster is due to the fact that some nodes are dedicated to specific purposes and can not be arbitrarily interchanged (e.g. because the usage of add-on cards such as the

[3] Picture copyright owned by the Alice Collaboration. Designed by A: Manta and N. Emmanouilidis.

H-RORC), the so-called functional heterogeneity.
In the case of the HLT cluster and for every heterogeneous cluster in general, the management complexity increases because of the diversity of hardware and software releases.
The usage of low-cost commodity hardware is another general trend, which is used for a dynamic increase of the computational power if required. Such environments often require extensive system management because there is no redundancy in the system resources (no redundant networks, power supply, cooling, hard disks, etc.) and therefore extensive monitoring and data correlation is required.
Budgets for system management are also limited as normally the usage of commodity hardware is related to a cost-saving strategy which reflects this limitation. Because of the lack of network redundancy, a system management solution should offer some decentralized management functionality for implementing a standalone management strategy on the nodes.
In traditional system management, the analysis and correlation of monitor data is realized by system administrators. They are in charge of interpreting actual status data and recognizing connections between those. This is a task which normally requires lots of time and experience, which is definitely not conform with the widespread trend of managing computer clusters with a lack of man power (i.e. not enough system administrators).
Reasoning on the previous deliberations cluster management is a complex business, which depends on the complexity of the environment to be managed and the amount of management effort to be invested. Effort is measured in funds, personal resources and required technology. As already mentioned, managing heterogeneous environments is always more complex than homogeneous environments. The effort for managing configurations increases because of the required treatment of exceptions and the resulting reduced capability of automatism development. The usage of commodity hardware lowers the cost of acquisition, but the management effort increases and consequently the number of manual interventions to be performed in the managed environment (i.e. the required man power) increases.
Furthermore, there is a long list of requirements that system management solutions have to comply with. Examples of this are a high grade of scalability for handling increased load, fault tolerance for dealing with system crashes, system openness and extensibility for self developing of management strategies, etc.
The SysMES framework, which is subject of this thesis, has been designed and developed based on the previous considerations for reducing management complexity and for supporting system administrators in the execution of their job. It is a scalable, decentralized, fault tolerant, dynamic, rule-based tool set for monitoring and management of networked target systems.
Reducing management complexity with SysMES is related to the manner how management information is organized and distributed. This information can be about management resources (such as monitors), managed resources (such as nodes) and the mapping between those. Further complexity reduction is realized by automatic deployment of the management resources to the managed resources.
SysMES supports system administrators and operators with the following functionality:

- Fast correlation of monitor data for reducing the error recognition time.
- Automated execution of actions which contribute to error solution.
- Reporting performed actions to administrators.
- Interface for manual execution of actions.
- Management environment overview for showing the state of the node.

- Decentralized management for implementing emergency and rescue strategies in case of network failures and the consequent missing connectivity to management servers.

Using the described functionality, system administrators are able to develop automated error recognition, reporting and solution which contribute to minimize manual interventions. The experience, gained while managing clusters, demonstrates that most of the usual failures repeat periodically and occur on many nodes. When such a problem is detected, the system administrator develops a Monitor for automatic recognition of the error in the future. The Monitor is then deployed on the nodes using SysMES. When the error occurs again, the Monitor informs SysMES using Events. Additionally, the system administrator develops Rules that carry out actions to solve or report the error. Furthermore, SysMES offers an interface for system administrators for manually performing administrative tasks.
Some cluster applications are configured, monitored and managed by operators. They normally do not have the privileges to perform administrative tasks in the cluster and are therefore dependent on system administrators for error solution. SysMES adopts this role and offers a GUI which can be used by operators for error recognition in their relevant environment. As an example an HLT operator uses this GUI for recognizing uncorrectable errors in nodes and reacts by excluding these nodes, reconfiguring and restarting the HLT.
In order to optimize system management for the HLT application and the cluster, the SysMES framework has been customized, but it is based on generic algorithms, which allow the management of large and distributed environment. Although lots of effort has been invested in reducing management complexity for the usage in highly heterogeneous environments, the designed and developed methods are also suitable for being used in homogeneous environments for supporting system administrators and reduce manual interventions.
The organization of the remaining chapters of this thesis is as follows:
Chapter 2 describes the design and conceptual goals for the development of the SysMES framework. The following chapter 3 contains a review of the state-of-the-art commercial system management products and research approaches for managing large distributed computer environments. It also compares management solutions of other similar clusters in the area of physics experiments. Chapter 4 discusses design considerations to be taken into account and the corresponding design decisions for the development of the SysMES framework. Chapter 5 is dedicated to the SysMES architecture and functionality. The first part of the chapter discusses the general management algorithm and design. Afterward follows the description of the several management layers and the respective functionality. The interaction between the functionality of several layers is the subject of chapter 6. Furthermore, implementation details are also discussed in this chapter. The following chapter 7 presents different system tests for demonstrating the correctness of the developed functionality as well as scalability and fault tolerance tests. Since the tests were carried out on the HLT cluster, this chapter also contains a description of the cluster as the test environment. Chapter 8 summarizes the process of system design, development and test of the SysMES framework and gives an overview about future improvements and further functionality to be developed. Finally, the Appendix A is for the description of used abbreviations.

Note to the reader: All SysMES specific concepts (such as Rules, Tasks, Events, etc.) are written with an initial capital letter in order to make a precise distinction to other subjects of the real world.

2. Goals

This chapter is dedicated to define the goals concerning the conceptual design and development of a system management framework, which is the main topic of this thesis.
In general, the main goal of this project is the design and development of a system management framework for managing a large environment of networked cluster nodes, embedded microcontrollers and applications.
More specific goals are defined as following:

1. Design goals: This list contains the design requirements for the system management framework.
   a) The system management framework has to be able to manage a very large number of devices and services (nodes, network devices, embedded systems, applications, etc.) of a very high number of different hardware and software versions and releases.
   b) Due to the requirement of managing a large number of devices, the management framework has to be scalable, i.e. in the case of increasing load, the performance of the management system has to improve proportionally to the additionally required system resources in order to handle the increased load.
   c) The system management framework must have the capabilities to deal with crashes, i.e. to be fault tolerant. In other words, the system management services must be kept available independent of failures in parts of the management environment. Such design methods and principles like clustering, redundancy and avoidance of single-point-of-failures have to be taken into account in order to guarantee a nearly uninterruptible management. This goal is related to making the management functionality fault tolerant and it is independent of the fault tolerance of the physical environment where it is running.
   d) The management framework has to be platform and location independent in a manner which allows the interoperability with other systems.
   e) The performance of management services on target devices has to be resource-efficient and should not influence normal usage of the targets for their respective purposes.
   f) An unified management interface has to be offered to the system administrators for the configuration of the management environment and for the interaction with the managed devices. The management interface offers visualization capabilities for displaying information about the managed resources and devices.
2. Functional goals: The following goals define the desired functionality for a system management framework.
   a) Monitoring functionality is required. The system management framework should be able to read relevant information about the managed devices and resources.

      Furthermore, it must be able to distinguish between data that represents a problem which therefore has to be categorized as important for further processing and data with informational content. Persistent storage of the monitor data as well as different processing strategies are required.

b) Monitoring data shows the actual state of individual devices or resources. A functionality for problem recognition and solution based on this data is required.

   The recognition part should be able first to evaluate data from a single or multiple monitors, i.e. the recognition of a single or complex state and second, to recognize local or global states, i.e. states from a single device, from a group of devices or from all devices of a type. The solution part should contribute automatically and unattended to the solution of problems.

   In case of failures by the automatic reaction, a problem escalation strategy is required.

c) A feature for manual interaction with the managed devices is required. The system management framework should offer an interface for a manual interaction with the managed devices in order to perform configuration and administration tasks. Furthermore, this interface should be open to other systems that use the manual interaction functionality for the same purpose.

d) Reporting capabilities are required. The framework reports about undesired monitor data, recognized errors and also about the status of the activities done by a system administrator. Reports should be offered in a way easy to understand in the management interface and should also be sent to a system administrator using common technologies (e.g. Short Message Service (SMS) or email).

e) The system management framework should first offer a capability to define, store and manage the required system management resources and second to perform automatic management configuration i.e. to deploy those resources to the location where they are required.

   The definition of the management resources has to be realized using common technologies and standards in order to contribute to the openness of the system.

f) The management framework should be open for changes. The management framework offers an easy method for the customization of the management environment by the system administrators. Another point is the capability to develop new management resources and to integrate those into the existing management environment (e.g. development of a new monitor for a new device).

   A dynamic (re)configuration of the management functionality and resources is required. The management solution has to be able to change the syntax (i.e. structure, attributes, etc) and semantics (i.e. its purpose and behavior) of the management resources at runtime without downtime.

3. State of The Art

This chapter gives an overview about several products and approaches which may achieve the goals (or part of these) defined in chapter 2.
The first part of the chapter is dedicated to the definition of the relevant characteristics to be reviewed. These have being divided into two categories "design" and "functionality".
The second part of this chapter consists of the review of the selected products and approaches. This part also has been divided into the main categories for commercial products and research projects. Other categories have been introduced for treating monitoring systems, control system, vendor specific systems and the review of the system management solution of the other experiments at CERN.
The following list specifies questions to be answered during the review process:

1. Design and general characteristics
    - Architecture: What is the management solution built on? client/server architecture, publisher/subscriber architecture, centralized or decentralized management, etc.?
    - Scalability: Is the management solution able to deal with increasing load or numbers of devices to be managed? How?
    - Fault Tolerance: This characteristic describes how a system management solution reacts in case of failures or crashes. Are the management services still available? Is the collected information available? Are there any single-point-of-failures?
    - Information Transfer Method: What is the method for exchanging information between manager and managed? Push/pop method? In a transactional or non-transactional way? Which formats have been used? Is there a strategy for traffic reduction?
    - Dynamic Changing of Management Resources Configuration: Is it possible to reconfigure the management resources on the fly without downtime?
    - Interoperability and Extensibility: Are there third-party interfaces? Is it possible for system administrators to develop and distribute new management resources? How easy can this be done?
    - Realization and Implementation: Is the management system to be reviewed platform independent? Which standards are used? How is it implemented?
2. Functionality
    - How is the monitoring capability realized? Which data and sensor types are supported? Is it possible to categorize the monitor data by its importance?
    - How can errors and problems be recognized and how can the system react to them? Is it possible to recognize complex states? Is it possible to correlate multiple Events? Is it possible to react manually? What is the strategy if the automatic reaction fails? Is there any escalation strategy?

- Is it possible to execute actions on the managed devices? Manually? How are administrators informed about the results of the execution? How can a system administrator perform changes? Using a GUI?
- How can a system administrator be informed about recognized states? GUI? Emails or Events?
- Management of Complexity (Heterogeneity): The question to be answered is: how does the management system deal with increasing complexity derived from a higher grade of heterogeneity?

  Another point about complexity management concerns the management of information about the cluster and the management environment. How is realized? In models? Databases? Files? How is the mapping between management resources and physical resources realized?

  Furthermore, it is necessary to find out if there is some functionality for an automatic configuration of the management environment.

3. Others: There is another very important aspect to be taken into account. This is the economic aspect for procuring, customizing and operating a system management framework.

The following sections introduce in detail the most important representatives of each categories. Although the review process provides answers for all asked questions the following sections contained only the most relevant information about the systems and projects.

3.1. Commercial Products and Solutions

This section introduces the leading products for system management of computer clusters.

Bright Cluster Manager [29]: This is a system management product from Bright Computing [27], a fully owned subsidiary of ClusterVision [38]. As described in [28] and [30], the management architecture is based on the definition of two different node types, the head and slave node. Head nodes are the manager unit and the slave nodes the managed. Depending on the desired functionality, there are several particular slave nodes such as failover, compute, login, I/O, provisioning, workload management and subnet management nodes. The failover node is responsible for taking over the functionality of the head node in case of failures. In other words, the head node and the failover node build a master-slave failover strategy. The definition of multiple slave nodes of several types (e.g. provisioning slave nodes) has been made in order to distribute the load generated by specific functionality and consequently to make the management environment scalable
Every node in the management environment runs the Cluster Management Daemon (CMDaemon), which is used for the execution of all management actions.
According to Bright Computing own statements, there is an advanced edition which is able to manage more than 10.000 nodes, but there are no numbers about the required computation or network power for this purpose.
Bright Cluster Manager [1] offers collections of metrics which can be used for accessing device information and monitoring of this data. There are software and hardware metrics and these can be

[1] Bright Cluster Manager is a trademark of the Bright Computing corporation.

categorized into cluster metrics (i.e. such metrics for all nodes) and device metrics (i.e such metrics which can be defined for one individual node). Customer metrics can be easily added.
Each CMDaemon in the cluster is in charge of sampling the monitor data of all defined metrics. The CMDaemon on the head node periodically collects this data and stores it in a raw-data database (normally a MySQL database) [81] and in a consolidated-data database. The storing, sampling and consolidating strategy is subject of the metric configuration.
A very useful tool for managing a cluster with this product is the centralized Cluster Management GUI. This tool builds the interface to all offered management functionality. It communicates with the CMDaemon on the head node for performing configurations on the slave nodes as well as the visualization of the monitoring data and of course the consolidated data. There are several views for metrics and also general cluster views, such as the rack view.
Bright Cluster Manager has a feature for automated cluster management. It is usable for the definition of metric thresholds and to define one or a series of actions to be executed if the threshold is exceeded, i.e. to define Rules. Typical actions are the generation of a message for the GUI or sending an email/SMS. It is also possible to execute operating system commands or scripts as an action. During the review process of this product it was not possible to recognize where the Rule matching occurs and also where the action execution has been initiated i.e. in the head node or in the slave nodes. A capability for correlating monitor data to recognize complex states is not available.
Manual interaction with the cluster is realized by using the Cluster Management Shell. This is the interface for both system administrators and other applications, which can use this Shell for a scripted execution of actions on the cluster nodes. The Cluster Management Shell communicates with the CMDaemon on the head node which delegates the execution to the local CMDaemon on the slave nodes. The result of the action execution is displayed in the system output/error in the shell.
An important reference installation of Bright Cluster Manager is the LOEWE-CSC Cluster of the University of Frankfurt which has been ranked at position #22th in the top500 list of supercomputers [73].

Summary: Bright Cluster Manager is a very interesting tool which facilitates the work of a system administrator. It also offers functionality for node provisioning, user and workload management. The most serious weak points are the limited capabilities for the correlation of monitor data and a centralized execution of actions by exceeding a threshold. Another point is the centralized monitoring and management philosophy, which is in contrast to the stated system scalability. Another weakness is the strategy for complexity management in a heterogeneous environment. This tool offers the capability to define node categories (i.e. groups with the same configuration) and also to define cluster or device metrics. Each node belongs per definition to only one category. The managing of a heterogeneous cluster becomes more complicated because the configuration of an exception (e.g. the inclusion of new hardware types) can only be managed by the definition of new categories and device metrics. Extension of the management functionality is also a weakness because this tool is not open-source and the license fee for a node license (estimated at 200 - 300 USD) has to be included into the cluster acquisition and management budget.

ParaStation V5 [89]: ParaStation is an integrated cluster management and communication solution from ParTec Cluster Competence Center GmbH. It is composed of a high performance communication middleware and a cluster management facility. The most interesting part for the subject of this thesis is the management facility and more exactly the GridMonitor and HealthChecker components [90].
The GridMonitor component is divided into two parts: the Collector, which represents the server

component, and the Agents, which represent daemons for retrieving sensor or device data from different devices and using different methods (e.g. Intelligent Platform Management Interface (IPMI) [61], Simple Network Management Protocol (SNMP) [97], etc.). Agents have to be installed on the respective nodes to be managed. Agents take measurements and make the results available. The Collector (or another client application) retrieves this data and stores it for being displayed in the GUI. Another important task of the Collector consists of checking if gained data exceeds predefined values and reporting this state by the generation of Events.
The second component is the HealthChecker. This component is a test suite used to guarantee that all required resources are available and running properly for a job submission (i.e. before the cluster or single nodes are used for a specific task). This is a very helpful functionality in order to make the cluster fault tolerant because nodes which have not passed the tests will be excluded for job submission.
Concerning the management architecture of ParaStation, it is possible to define it as a centralized architecture (e.g. for small or middle environments) or in a hierarchical distributed manner (e.g. for large environments) which also contributes to a better scalability.
Single actions on the cluster nodes can be performed using a parallel shell.

Summary: ParaStation offers suitable tools for mainly managing Message Passing Interface (MPI) [80] applications and the required cluster nodes. An example of this is the installation on the JuRoPa cluster ranked #23th in the top500 list [67].
There is no doubt about the functionality for managing MPI applications, which is their major area of application. However, there are some weaknesses in using this tool-set for managing large heterogeneous environments. One point is the rudimentarily implemented fault tolerance strategy, which consists of a self-monitoring component for the Collector and a restart action if it fails. Another point is that the event generation (i.e. the triggering capability) and also the initiation of an action execution is located in a central instance - the Collector. This strategy makes it impossible to define local management strategy (e.g. for a local triggered shutdown) important for defining actions in case of connectivity failures to the Collector. Finally, the most important weak point is the missing functionality for the recognition of complex and global states (i.e. correlation of Events) and the execution of actions for problem solving.

IBM Tivoli Software [99]: Tivoli [2] is the system management solution of IBM. It is composed of a large number of different products for specific managing tasks.
The main component is the Tivoli Management Framework (TMF) which is based on the Common Object Request Broker Architecture (CORBA) [39]. This framework builds the interface for the integration of other components and for performing actions on the managed devices.
The Architecture of Tivoli is hierarchical with a top level server called Tivoli Management Region (TMR) server, with optional one or multiple gateways and endpoints running on the managed devices. The endpoints communicate with TMR server directly or via gateway. The function of a gateway is to pre-process data from the endpoints in order to avoid collapse of the TMR server. A Tivoli environment can also have multiple management regions and consequently multiple TMR servers, which communicate with each other. This hierarchical strategy is used for increasing the scalability grade of the Tivoli framework.
Tivoli offers a large list of products for managing every IT related thing in a company, the core components are: Tivoli Enterprise Console (TEC), Tivoli Configuration Manager (TCM) and IBM Tivoli Monitoring (ITM).

[2] IBM and Tivoli are registered names of the IBM corporation.

TEC is a very powerful prolog-based Rule engine used for Event correlation. TCM is in charge of inventorying devices in the management environment and also for software distribution.
ITM is the monitoring component of Tivoli and besides resource monitoring, it is responsible for sending Events to the TEC server and for their visualization. ITM utilizes two models for the proactive monitoring of resources. The first one describes the resource to be monitored and the second one describes the operational description of the resource. Tivoli uses the information stored in those models for the recognition of complex errors and failure signatures.
Besides the extensive functionality of Tivoli, other strong points of this product are their user support, offers of user training, product documentation and a very large knowledge base about known problems. As a typical commercial product Tivoli is closed-source and almost all products are subject to a fee.

Summary: IBM purchased the Tivoli company in the middle of the 90s and included it in the IBM Software Group as a 100% subsidiary. As already described at the beginning, Tivoli has been designed as a corba-based framework and the core functionality was basically the prolog based Rule engine, a distributed monitoring component and a remote control component. More of the current components have been obtained by purchasing other companies and competitors and integrating these into the Tivoli framework. The result of this is a tool-set of applications integrated (more or less successfully) using a out-of-date interface (i.e. TMF) with duplicated, but not redundant functionality (e.g. Event correlation in TEC and ITM).
Some of the Tivoli component suffers from overcomplexity. N. Bezroukov describes in [20] the case of the TEC where Rules have to be programmed in prolog, which is not a widespread proficiency in system administrators and operators and therefore special staff or expensive training is required. Another example of overcomplexity is the product installation which normally is facilitated by Tivoli experts and this increases the costs for the introduction of the system management solution.
The biggest weak point of Tivoli is the architecture because gateway instances are not clustered (i.e. single-point-of-failure) and a failure in one causes unavailability of all clients (called endpoints in Tivoli) connected by this gateway. Endpoints are not able to contact another gateway for delivering their Events. Similar weakness concerns also the TMR server instances.
The scalability of Tivoli is questionable. They claim to be scalable up to several thousand nodes, but both the TMR servers and gateways are bottlenecks for managing the underlying layers or nodes.
The licensing costs cannot be neglected as Tivoli normally charges a basic license fee for a product and an additional fee per CPU core of the server where it is running. Costs amounting to half a million dollars per year for a middle sized cluster are normal. An exhaustive review of Tivoli software can be found in [100].

Hewlett Packard Operations Manager (HPOM) Software [58]: Hewlett Packard Operations Manager (HPOM) is the direct competitor of the previously presented IBM Tivoli Software and in principle it is designed in a similar way. HPOM architecture is also hierarchical and client/server based. Unlike Tivoli, HPOM introduced clustering functionality for servers as well as the capability of building server hierarchies. These features contribute to increasing the scalability and fault tolerance grade of the management solution. Furthermore, servers are able to exchange messages with each other for example for forwarding Events to the server where they are expected.
Although HPOM is closed-source, it is possible to extend monitoring capabilities using shell or perl scripts. The Event correlation functionality is limited to basic operations such as Event suppression, duplication, creation, etc., but it is distributable to the clients for local correlation, which also contributes to the improvement of the scalability of the system. Additionally, Hewlett Packard (HP) has

acquired a new correlation engine called OMi Topology Based Event Correlation [59], which can be used for extended correlation capabilities.

Summary: HPOM is definitely a worthy competitor in the area of commercial system management solutions. The physical architecture has been designed for getting better scalability and failure tolerance. The correlation engine is the biggest weakness because it offers limited functionality and the usage of OMi is not really an alternative because this product runs only on Windows [3] Operating System (OS). A positive aspect is the simple and intuitive correlation building method.
The user support for HPOM is another weakness. Documentation is not sufficient and often outdated. Concerning the costs, the licensing strategy of HP is more user friendly. Charging fees for server and agents are independent of the number of cores. Nevertheless, a big budget is necessary for the installation and yearly maintenance.

Besides the introduced products, there are other system management solutions which are able to offer parts of the desired functionality or characteristics. Some of those are CA Network and System Management [31], BMC Event and Impact Management [21] and Zenoss [113].

3.2. Vendor Specific Solutions

A new trend in the area of system management is that vendors of computer systems, operating systems and software offer a integrated management solution. Those management tools are optimized for managing the respective supported hardware which minimizes the time for installation and customization.
Examples of such tools are Dell [4] Data Center Management [41] and Microsoft [5] System Center [77]. These kinds of tools are normally not suitable for managing heterogeneous environments due to the lack of interoperability with devices of multiple vendor, the limited extensibility (they are normally not open-source) and often the limited support of different operating systems and architectures. The functionality of these tools is similar to those of the monitoring tools with a GUI for displaying status and performing manual actions on the clients. A secondary argument against its usage is related to the licensing cost for this solution.

3.3. Research Projects

Research projects in the area of system management normally cover a subset of the functionality and features expected from the SysMES framework. In order to organize the most relevant projects the following categories have been chosen:

3.3.1. (Autonomous) Autonomic Computing

This is a paradigm proposed by IBM at the beginning of the 21th century. In principle, it describes a system which is capable of managing itself without human intervention. The role of the system administrator has been shifted from a control role (operator) to the role of a system developer, who defines the policies and rules for the autonomic management. Conforming to [70], [69] IBM frequently cites four aspects of self-management. These are:

[3] Windows is a registered name of the Microsoft corporation.
[4] Dell is a trademark of the Dell corporation.
[5] Microsoft is a registered name of the Microsoft corporation.

- Self-Configuration: automatic component configuration.
- Self-Optimization: automatic improvement of performance and efficiency.
- Self-Healing: automatic error detection and solution.
- Self-Protection: automatic defending capabilities.

An autonomic computing environment is composed of several autonomic components, which use self-monitoring for detecting infringement of policies and a self-regulating capability to keep the environment in the policies range.
This computing paradigm is a theoretical proposal which does not prescribe how the self-management strategy has to be realized. A usual interpretation is to introduce autonomic capabilities to applications and also to define techniques for developing autonomic software from scratch. S. Hariri et al. extended in [54] the autonomic computing paradigm in the year 2006, introducing further self-properties.
The research and development efforts of researchers reflect their interpretation of that paradigm. Each of these research projects is specialized in implementing autonomic computing for specific purposes.

M. Agarwal et al. describes in [9] the AutoMate framework for introducing autonomic capabilities to Grid applications. It is based on the idea of building autonomic applications as the dynamic composition of autonomic components. Each component has an interface which is used for offering information about the component (e.g. sensors offering information about their performance, requirements, etc.). This information is used to define policies which describe the desired component behavior, to check these policies are fulfilled and to execute actions such as component reconfiguration (e.g. actuators which are able to change the structure and behavior of the component). Policies are defined locally in each component in the form of simple Rules (if-then Rules).
The AutoMate framework is composed of a set of tools e.g. the decentralized coordination framework Rudder [114] and the framework Accord [72], which are used for building the autonomic components. Automate describes a suitable method for making applications (or components) manageable autonomicaly. It also describes how it is possible to manage bigger applications, building a management chain of autonomic components. One weak point is that management chains lack fail over strategies. There is no information about how the management resources (such as sensors and Rules) are organized and managed. The most important weakness is the restriction that only applications can be managed. More information about AutoMate can be found in [88].

S. Bouchenak et al. describes in [25] the Jade middleware which can be used for developing self-management capabilities of distributed software environments. Its main task consists of wrapping applications using Fractal [51] components in order to build a uniform management interface which can be used to develop a managing functionality, such as application monitoring and reconfiguration. This framework relies on two main components: a component model of the managed environment and control loops. These loops are in charge of regulating and optimizing the managed system and are composed of sensors, reactors and actuators.
An implementation of a self-optimization manager for clustered Java Enterprise Edition (J2EE) applications [26] serves as proof of concept with the main task of increasing or decreasing the number of clustered instances according to the current load. Another aspect treated is repair management for such J2EE applications [24].
In this case, sensors are used for monitoring performance (i.e. CPU and user-perceived response time), reactors are used for analyzing the sensor data and reacting by increasing or decreasing the number of instances in the application cluster and actuators are used for performing desired reconfigurations.

Due to the low number of sensor data to be correlated, the decision logic has been built using triggers based on thresholds. The actions for repair management are stopping and starting faulty software instances.
The model based management is a very interesting feature of Jade for dealing with software heterogeneity. This framework has been designed for managing software which can be encapsulated by a wrapper (more exactly a fractal Java-based component) and therefore, it is not suitable for managing computer clusters (i.e. hardware).

J. D. Baldassari, C. L. Kopec and E. S. Leshay describe in their bachelor thesis [15] the first steps in the development of the Autonomic Cluster Management System (ACMS) framework. This framework is based on a number of networked agents which are able to communicate with each other. To be more specific, it consists of two configuration agents and one optimization agent for the central functionality, and two general agents per cluster node. Configuration agents are responsible for monitoring any other agent, checking if they are working properly and restarting a failed agent. Besides this, the configuration agents collect system statistics of the cluster nodes. The configuration agent propagates the statistic data to the optimization agent, which is in charge of distributing work to the nodes according to their current load. The general agent on the node executes the actions initiated by the optimization agents in order to perform work.
This project describes an agent based load balancing strategy with the capability to recover itself. Most effort has been invested in making the management tool fault tolerant by building replicas of the agents and guaranteeing a disjunct placing of those on several nodes. The scalability grade of the system is limited by the design decision to use a fixed number of configuration agents (one master and one slave agent) and optimization agents (the only one without a replica). The analyzing capabilities of the optimization agent are hard coded and static. More about this tool can be found in [16] and [101].

Summary: The (autonomous) autonomic computing definition of IBM was the first step towards building a new paradigm of computing and system management. The pioneers in this area designed and developed frameworks based on the self properties. The results of those projects demonstrate the usage of autonomic principles for isolated areas such as load balancing and management of J2EE applications. Noticeable is that the last results on the IBM research side [70] are from the year 2003.
The previously introduced ACMS project describes some efforts for managing large distributed environments in an autonomic way. The cited documents are about five years old and there are no further publications about current results in this area.
In actuality, autonomic research projects do not try to cover all self properties for a general purpose framework. The new developments are dedicated to specific areas. Examples of this are projects for self organizing networks [98], business process management [93] and database management systems [76].

3.3.2. Other Research Areas

This section covers other research projects in the area of system management which utilizes other methods than those presented before.
One trend in the area of management of large environments consists of monitoring devices and performing data mining in order to recognize undesired situations.

A typical data mining project is InteMon [57] and [56] described by E. Hoke et al. This project has the goal to extend typical monitoring systems with a data mining module, which is able to correlate monitor data. The correlations are used first for determining range of values for normal behavior of

the managed devices, and second for the recognition of a problem if current monitor values deviate from the calculated normal range.
InteMon monitoring uses SNMP for periodically retrieving device status information and it stores the data in a customized relational database. The data mining module then works on this database.
This method is useful for the detection of undesired states based on statistical data. It simplifies the problem recognition task for a system administrator who normally has to analyze the monitor data himself using histograms. Another benefit is that this method elimitates the need to calibrate monitors with thresholds.
The data mining component is centralized and therefore its scalability grade is limited. No performance results have been presented. Furthermore, the functionality of InteMon is comparable with an extended monitoring functionality, which is only a subset of the desired functionality of a system management solution.

P. Bodik et al. introduces in [22] the term of a "fingerprint", which describes the current state of the managed environment. The status is composed of the values of monitoring metrics. A subset of those metrics are the key performance metrics, which are used for monitoring Service Level Objectives (SLOs) [6]. A performance crisis is defined as a long term SLO violation.
The focus of this project is to first classify crises, and second to their early detection based on the previous classification. The detection algorithm compares the current fingerprint (e.g after a SLO violation) with the classified crisis fingerprints in order to recognize if the incoming crisis has been seen before and to accelerate crisis management by applying a known crisis solution. Practical results demonstrate that this system management strategy provides system administrators the required information for starting recovery within 10 minutes, which is six times faster than the mean required time for a human crisis recognition.
The main result of this project is the development of a problem classification and identification. This functionality supports the system administrators when working on problems and in failure recognition. However, this offered functionality is not enough to manage a cluster. It lacks automated recovery capabilities and a manual interaction interface. The centralized approach is also critical because the scalability, fault tolerance and the fact that in case of major failures (i.e. network failures) the required metric information will not arrive at the location where the problem recognition occurs and therefore some errors stay undiscovered.

Z. Zhi-Hong et al. describe in [115] the self-configuring and self-healing capabilities of the Fire Phoenix cluster management software which is for managing the Dawning 4000A supercomputer [40]. Both features are based on an agent-based architecture. The complete managed environment is divided into partitions and each partition has a Leader (i.e. a master node) and a Prince (i.e. a slave node) for managing the rest of the partition members.
Master nodes are grouped into a Meta-Group with a Leader and a Prince. All those nodes build a management ring and each node is in charge of sending a heartbeat to the next node. If one node does not receive the heartbeat of the neighbor then it reports this to the Meta-Group Leader, which starts a recovery strategy restarting failed processes or replacing the failed node (e.g. the prince node of the partition). The self-configuring capability is in charge of setting up the new node with the desired configuration. Self-healing for the self-configuration module is realized in the same way as other applications. Locally, each node is able to monitor applications and to restart them in case of failures. This procedure is realized by a locally installed Watch Daemon.
The presented heartbeat based management offers limited capabilities for correlating data of the applications and resources and also limited reaction capabilities. The positive review point of this project is

[6] SLO is a quantified service delivering agreement between a service provider and its customer.

the capability to react locally at the managed node site. Other performance results have unfortunately not been published.

3.4. Monitoring

This section has been dedicated to typical monitoring systems which are widespread in the area of system management. Normally, these tools are designed and optimized for retrieving information from devices (e.g. cluster nodes, switches, Rack Monitoring System (RMS), etc.), checking if the values are in a predefined range, reacting to a value deviation with a simple action (e.g. generation of an Event or report, email or SMS) and visualizing the gained data. The strengths of such systems are in a well defined visualization strategy, which offers a real time status of the monitored devices, their customization (i.e. configuration and extension) and of course, the low costs, because many of them are freeware and open-source.
Representatives of the most well known monitoring systems are Nagios [7] [82], Ganglia [52], LHC Era Monitoring (Lemon) [71].

Summary: According to the defined reviewing criteria the weaknesses of these tools are the system architecture (e.g. centralized by Nagios), the method for retrieving monitor data (e.g. polling by Nagios), the usage of an unreliable communications method (e.g. UDP based communication by Ganglia), the lack of fault tolerance and a complexity management strategy for heterogeneity (e.g. in Lemon) and the most serious weakness is the missing functionality for the recognition of complex states. Furthermore, the usage of monitoring systems requires the employment of a greater number of system administrators and operators, which have to interpret the data contained in the monitoring histograms and graphs.
However, these tools are very useful for getting information about the managed resources and for the real time visualization of the cluster state. Other monitoring systems are Pandora [86], Zabbix [110], MonALISA [79], etc.

3.5. Industrial Control Systems / Scada Systems

Control systems are mainly used for monitoring and managing hardware devices, which are part of scientific or industrial equipment. The reason for including this in this chapter is because they are specialized in collecting, monitoring and visualizing a vast number of sensors, which can only be managed by a scalable system.
Some of the most used products are Experimental Physics and Industrial Control System (EPICS) [46], Prozessvisualisierungs und Steuerungs System (PVSS) [8], FactoryTalk View (FTV) [9] and TACO New Generation Objects (TANGO). A very actual and extensive review of those frameworks has been realized by O. Barana et al. and can be found in [17]
One of these systems, EPICS, has been extensively reviewed because it offers an interface for monitoring computers using SNMP, so that the methods for the management of industrial equipment can be easily transfered to computer clusters.

[7] Nagios is a registered name of the Nagios corporation.
[8] PVSS is the German translation of process visualization and control system
[9] FTV is a registered name of the Rockwell Automation corporation.

EPICS: It is a decentralized monitoring and control system mainly used for managing scientific equipment, such as particle accelerators and telescopes. It is based on a entity called Input/Output Controller (IOC). Each IOC can assume the role of a server for offering information or client for consuming information. An EPICS server stores information about the state of a device in a structure called record. A record is composed of an unique name, fields (e.g. input and output fields for information holding) and an action (e.g. "calc" for performing calculations on the fields). A collection of records is known as a database. The combination RecordName.Field is known as Private Variable (PV) and has to also be unique in the EPICS environment. EPICS clients access the information contained in a PV using the Internet Protocol (IP) based Channel Access (CA) protocol. A typical client is a GUI, which displays the status of the equipment to the operators.

Other relevant characteristics of EPICS are that ,it is open-source and there is a large community of users and developers, who contribute in making its usage easy. EPICS is used e.g. for monitoring more than 30000 PV of the High Acceptance DiElectron Spectrometer (HADES) detector [53] at the GSI Helmholtzzentrum fuer Schwerionenforschung GmbH (GSI) in Darmstadt (Germany).

The most important strength of EPICS is its excellent scalability grade because it is fully distributed. Another point which also contributes to this is the configuration capabilities in the IOCs, which allow it to perform trigger tests in order to avoid uninteresting data propagation and consequently network load. It is also possible to configure the clients to get data only if it is currently required e.g. in case of a GUI, the client requests, collects and visualizes data only if the GUI is open and in use.

EPICS allows the correlation of data in two ways. The first one is the correlation of data in a database through record linking (i.e. An output value of a record is the input value of an other record). The second one is realized by a client called sequencer, which is able to execute compiled State Notation Language (SNL) [10] code.

Some weaknesses of EPICS are related to these correlation capabilities. Both correlation methods are static. The required logic for this purpose and its configuration can not be changed at runtime. For performing changes the involved IOC and/or sequencer has to be stopped, the new logic has to be compiled and loaded. The IOC has to be restarted.

Another important point is the missing failure tolerance capability of the IOC. This has been excluded for the EPICS design because the IOC is designed for monitoring and managing hardware without redundancy. It is possible to implement workarounds by including redundant records in different IOCs, but this creates the problem that the IOCs have to be used in a way where one is active and the other one is in standby to avoid multiple execution of actions.

The management of the records and databases is left to the system developer and administrator. There is no standardized way of building models and mappings between the records or databases and the devices to be managed.

The way of managing clusters with EPICS started with the development of devSNMP [68] which is an interface for putting SNMP data in EPICS records.

Summary: It can be said conclusively that EPICS is a very good option for managing hardware (i.e. static environments), but the required effort for managing large heterogeneous and dynamic clusters is beyond the scope of its design and development. The inclusion of small clusters in an EPICS environment is recommended whenever data correlation is rarely required and used.

[10] SNL is a language similar to C used for programming sequential operations.

3.6. System Management by the other CERN Experiments

The SysMES framework, which is the main topic of this thesis, has been developed as a generic tool for automated system management for large and heterogeneous clusters. However, the main application at the moment is managing the ALICE HLT Cluster. This section gives an introduction of the management efforts of some of the other experiments at CERN and also the CERN Computer Center.

CERN Computer Center: The CERN computer center builds the infrastructure for offering computational services to CERN users, staff and collaborators. Those services can be divided into administrative services (e.g. management of user accounts), physics services (e.g. Worldwide LHC Computing Grid (WLCG) [106]) and technical services (e.g. Engineering & Equipment Data Management Service (EDMS) [44]). For this purpose the computer center is equipped with several thousand computer nodes, cartridge tape libraries/robots and Storage Area Network (SAN) equipment. The management of the computer center has been realized using multiple tools for specific purposes. System monitoring is realized by the usage of Lemon [71], configuration management by the usage of Quattor [92] and visualization and action execution by the usage of CluMan [96]. Some numbers about the Computer Center are as follows:

- $\approx$ 8000 servers
- $\approx$ 13800 processors
- $\approx$ 50000 cores
- $\approx$ 50000 disks
- $\approx$ 45 PB raw disk capacity
- $\approx$ 45 PB available tape capacity
- $\approx$ 45000 tapes
- $\approx$ 10000 mounts/day
- 5 tape libraries for LHC data

ATLAS TDAQ Cluster: This cluster is used by the ATLAS experiment for reading out the detector front-end data and hosting the ATLAS HLT. The current installation state of this cluster consists of approximately 1200 nodes, which are managed by more than 60 dedicated servers [6], [43]. Monitoring of the cluster resources is realized using Nagios, which stores the collected data in about 25800 Round Robin Database (RRD) [11] files with a total size of about 4.5 GByte. Nagios offers reaction capabilities in the form of SMS/email for reporting the exceedance of a predefined threshold. Remote hardware management is realized using IPMI. A Configuration management tool called ConfdbUI has been developed. Using this tool, system administrators are able to perform actions for providing new nodes with the required system and management configuration. Another functionality of the ConfdbUI tool is the capability to execute IPMI and system commands on the managed nodes. Conforming to [6] the ATLAS Trigger and Data Acquisition (TDAQ) cluster is managed by a group of system administrators, who belong to the ATLAS TDAQ SysAdmin Group.

[11] RRD is a file based database for storing time-series data.

The CMS Online Cluster: The CMS online cluster is used for hosting the CMS Data Acquisition (DAQ), HLT and the CMS run control system. At the moment, it consists of more than 2000 computer nodes and 120 networking devices. Depending on the functionality of the nodes, they are connected to the CERN campus public network, the LHC technical private network, the CMS service network, the CMS data network or to the central data recording private network. The management of this cluster is also realized by the usage of different tools for specific purposes. Nagios is used for monitoring the cluster resources, IPMI is used for monitoring and management of the hardware, Quattor is used for configuration management, etc. Failure tolerance and load balancing has been realized by the definition of a one-master/n-replica strategy for the IT services. Load balancing is realized by explicit segregation which means that the cluster is partitioned and each partition contains a primary and secondary server. For managing the CMS online cluster there are 5 full-time system administrator positions available [19].

Summary: All the introduced clusters and data centers are managed in the same manner. There is a group of system administrators responsible for the network and computational environment and also for developing tools which contribute to a better management. Open-source and in-house developed tools and scripts are used for specific purposes. The main management topics of those management projects are monitoring, security and configuration management. All those groups are doing a good job and the main differences to the SysMES framework and the management of the ALICE HLT cluster are related to the expected grade of automatic management, as well as to the capabilities for the recognition of complex and global states.

3.7. Evaluation

The previous review of several system management tools, products and research projects has been done in order to find out if any of those is able to achieve the Goals of chapter 2.
The first insight gained by the previous review is that none of these systems are able to achieve all the goals. In principle, there are two methods of how system management has been used in the last years. The first one is the method of commercial products where system management solutions are offered normally for an expensive price. The functionality of those products is very extensive because the vendors are interested in covering all areas of the IT infrastructure.
The advantage of such a solution is that the vendor assumes the responsibility for the installation and customization of the management environment and also for training of the system administrators and operators. Apart from the high licensing and maintenance price, there are other disadvantages, such as limited extensibility due to closed-source and vendor dependence.
The second method is the use of multiple tools (normally open-source tools) each for a specific purpose. Those tools are often from different vendors and normally not integrated in a unified management environment and GUI. Lack of functionality in some areas is the consequence and often the motivation for self-developing. Plus points are the almost infinite extensibility capabilities and the low cost for licensing and maintenance (except personnel costs).
N. Bezroukov describes this trend in [20] as the *"Best of breed" mix of low cost proprietary and open source scripting-friendly products*. This method has been observed for managing the other experiment clusters and the data center in CERN, although such a method increases the required manpower in terms of required number of system administrators and operators. However, all these clusters are running stable and are well managed.
Almost all solutions ignore the importance of Event correlation, which is a key point for minimizing the administration effort with automatic error recognition and solution.

Almost all reviewed systems utilize message transmission in a non-transactional way. Messages (such as Events) can get lost because they are mainly used for displaying an undesired state and the next message will report it again. The situation gets critical if those Events are excepted in a chronological order or in a specific count for a correlation.
In the area of research, there are interesting and promising projects which can contribute to a better manageability of large distributed environments in the future.
The dissatisfaction with state of the art products was the initial and deciding factor for the development of the SysMES framework, which is subject of this thesis.

4. SysMES Design Considerations and Decisions

This chapter describes the design considerations for the development of the SysMES framework and the design decisions that have been made in order to achieve the goals that have been set in chapter 2.

4.1. Distributed and Location Independent System Management

Consideration: System management services are often attached to dedicated servers. In this case, consumers of system management services have to know the servers and the services offered by these. Availableness of services without downtime is hardly even possible to be realized.
It is not important for the SysMES framework where which service is offered. It is only important that a sufficient number of instances to deliver system management services always exist.
Decision: The SysMES framework is designed as a distributed and multi-layered system. Each layer has a specific functionality. Layers are composed of a number of instances, which offer the same functionality and are able to substitute for each other. The number of instances in a layer can be changed by adding or removing instances at runtime. This specific number of instances depends only on the expected load and desired degree of availability of the framework. Several functional layers can be located at the same physical server. It is also possible to distribute parts of the functionality to the SysMES targets.
Distributed and location independent system management in terms of SysMES means that system management services are offered by several instances of a layer and each of these instances is able to replace any other. The SysMES framework delegates work to several instances according to their utilization. In case of failures in one or multiple instances of the same layer, pending work is distributed to the other layer members.

4.2. Decentralized System Management

Consideration: The common way for system management in the most popular tools (see section 3.1 and 3.4) is to collect monitoring information from the targets and send it to the centralized management servers. The servers have to store the collected data and process them in order to recognize failures or undesired states. In a very large environment, this method is not suitable because the network bandwidth and the server processing power will cause bottlenecks and single-point-of-failures. Targets in traditional system management solutions are used for monitoring and have limited resources and functionality for active system management i.e. process and correlation of monitor data and automatic reaction.
Decentralized management in terms of SysMES means that the management functionality can be distributed across the management environment. Both servers and targets have basic functionality to store and process monitoring data.

Decision: The SysMES framework has been designed for the management of large distributed environments. Management services are distributed depending on their priority and complexity, on different layers either on the targets (e.g. in extremely critical cases) or in a management server hierarchy (for more complex cases). Decentralized methods treat storage and processing of target data as follows.

- Decentralized Data Storage: The term "data" represents the system and application information collected during the monitoring of the targets. The SysMES framework implements different options for the storage of these data either locally on the targets or in a clustered database on the server side. Storage of data on the targets occurs locally in a data cache and depending on its importance, uses different strategies. This data cache strategy avoids information loss due to data transfer or network connectivity failures and establishes the first design principle for the scalability strategy described below. The other possibility is permanent storage of the data in a fault-tolerant database cluster.
- Decentralized Data Processing: Similar to the data storage, the SysMES framework is able to process the monitoring data on targets or servers. The reason for the decentralized data processing is the need for dynamic processing according to the load on the targets, servers and network. Critical data can be processed locally on the targets if these do not have connectivity to the servers. An example of this is the processing of a high temperature value in an offline phase in order to issue a shutdown without server interaction.

Another advantage of this decentralized management is the local storage of configuration data and management objects. Each target has its own local repository, which contains the management objects (e.g. Monitors, Rules and Actions) and information required for recovering a previous management status during startup.

4.3. Scalability

Consideration: There are several definitions of the term scalability concerning the design and conception of software systems, some of which are contradictory.
The basic statement in the research about scalability is that it is the ability to cope with a growing load. According to the scalability discussion of C.B. Weinstock and J.B. Goodenough in [105] there are two basic statements about how a scalable system deals with additional load. The first one is that scalability is the ability of a software system to handle increased workload without adding resources, whereas according to the second one scalable systems are allowed to add new resources.
In case of the SysMES framework as a distributed system, growing load depends on multiple factors. These factors can be divided into target side factors, i.e. the number of targets to be managed and the amount of target data to be stored and processed, and server side factors, i.e. the number and kind of provided services, the number of servers as well as the available networks and databases.
In case of the SysMES framework, the scalability definitions introduced above are not sufficient and therefore it has to be extended as follows: SysMES scalability is defined as the ability of a system to master additional load by the extension of the system resources or by dynamic relocation and redistribution of the work. In this context, relocation means the vertical distribution of the load on different management layers instead of the horizontal distribution of the load (i.e. insider a layer), for example by a load balancing strategy.
Scalability can be categorized according to [95] in various dimensions, such as:

- Load scalability: The ability for a distributed system to easily expand and contract its resource pool to accommodate heavier or lighter loads. Alternatively, the ease with which a system or component can be modified, added, or removed, to accommodate changing load.
- Geographic scalability: The ability to maintain performance, usefulness, or usability regardless of expansion from concentration in a local area to a more distributed geographic pattern.
- Administrative scalability: The ability for an increasing number of organizations to easily share a single distributed system.

Decision: The SysMES framework has been designed to achieve all requirements of the scalability dimensions described above and therefore pursues the following strategy: In order to realize the load scalability, the framework has been developed using the clustering of the system management capabilities such as Event and Rule management. Due to the clustering, it is possible to extend the framework by adding a new server on the fly. The new server is able to run the required management capabilities or extend the whole functionality of an overloaded server.
Another capability of the SysMES framework is self-management which is used for the monitoring of its own infrastructure, such as servers and databases, and for reacting to undesirable states. For example, this feature is used for recognition of overloaded servers and to activate an additional server automatically. Using the same procedure, it is also possible to remove a server if it is not needed. Another desired capability is load balancing, which distributes the load to the management servers according to their utilization.
The design considerations and decisions for scalability, reliability and availability are defined for the development of the SysMES framework and its offered management services and not for the design or implementation of the physical infrastructure. Therefore, other extension possibilities such as target hardware upgrade or the extension of network infrastructure, have been neglected.
The basic idea of geographical scalability is that the overload of one components (targets, network or databases) can be avoided by relocation of the data storage and processing places. In case of an overloaded target, the data storage and processing will be transfered to the servers. Similar to the previous example if the databases are overloaded, the targets will be reconfigured and assume control of data storage and processing without data transfer to the databases. Network overload will be reduced through the development of a triggering strategy to decide which of the data has to be processed and sent from the targets to the management servers. Remember that data represents the system and application information collected during the monitoring of the targets. It is also possible to reconfigure the triggers on the fly in a dynamic way.
In order to achieve the administrative scalability, the SysMES framework defines a set of targets' identifiers such as DeviceID, GroupID and FirmID, which define the affiliation of a target to a firm or organization. The management servers have been implemented in order to deliver the management services according to the membership of the targets and to allow the reusability of the management infrastructure for different organizations.
Summing this up, the SysMES framework proposes the delivery of management services in a cost efficient way, due to the extension and relocation of the management capabilities.

4.4. Dependability

Consideration: Dependability is one important aspect to be considered by the development of a distributed management system. Dependability has been defined as the ability of a system to deliver

service that can justifiably be trusted and to avoid failures that are more frequent or more severe, and outage durations that are longer than what is acceptable to the users [13]. It is a general concept, which can be specified in different attributes depending on the system to be designed. The most common attributes are: reliability, availability, maintainability and safety.
The main focus of the design of the SysMES framework is dedicated to the attributes reliability and availability.
The Reliability of a component or system is its ability to function correctly over a specified period of time [94]. That means that reliability deals with continuity of service [62].
Availability is the probability that a system is operating correctly at a given time instant [91]. That means that availability deals with readiness for usage [62].
Decision: In order to achieve reliability and availability for the SysMES Framework, there are two strategies to be pursued: these are fault prevention and fault tolerance. Fault prevention consists of a set of design and implementation arrangements for avoiding faults and fault tolerance consists of a set of techniques to be used in order to guarantee the continuous and correct supply of services if a fault occurs. A concrete definition of error, fault and failure can be found in [62].

4.4.1. Fault Prevention

In managing a distributed environment, the most performed activity is data exchange between the components of the management environment (i.e. managers and managed devices). The fault prevention strategy includes a transactional information exchange, which ensures complete information delivering.
Another fault prevention procedure consists of input and output data control, checking data types, range of values and data consistency. This method avoid faults resulting from errors in data.

4.4.2. Fault Tolerance

The fault tolerance strategies are based on decentralized management, avoiding single-point-of-failures. Decentralized management offers the capability to define management strategies in the targets (e.g. definition of an emergency management strategy), which can be performed independently of the servers.
Avoidance of single-point-of-failures is realized by clustering the management functionality in each management layer. Each member of the layer is able to take over the work of any other.
An Exception handling strategy is also relevant for the treatment of exceptions without crashes of the management servers and targets.
The fault prevention and tolerance methods described in this section correspond to the approach to make the SysMES Framework fault tolerant (i.e software fault tolerance of the provided management services). The requirement for this purpose is a fault tolerant physical environment with redundant hardware and network connectivity to cope with failures in this layer (hardware fault tolerance).

4.5. Development Based on Common Standards and Technologies

Consideration: The system management architecture has to be developed using common and open standards and technologies, which allow a better interoperability and manufacturer independence, and therefore become immune to changes and platform independent.

Decision: In addition to the modeling standards such as Unified Modeling Language (UML), Common Information Model and Web Based Enterprise Management (WBEM) described in the following section 4.9, the SysMES framework has been designed based on the following standards and technologies:

- eXtensible Markup Language (XML): This is a standard of the World Wide Web Consortium and it is defined by them as follows: XML is a simple, very flexible text format derived from SGML (ISO 8879). Originally designed to meet the challenges of large-scale electronic publishing, XML is also playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere [109].
- Enterprise Java Beans (EJB): The EJB technology [45] is the server-side component architecture for Java Platform, J2EE [63]. EJB enables rapid and simplified development of distributed, transactional, secure and portable applications based on Java technology.
- JBoss Application Server (JBoss AS) [1]: This is the most widely used Java application server on the market. A certified Java platform for developing and deploying enterprise applications, JBoss AS [65] supports both traditional APIs and J2EE APIs and includes improved performance and scalability through buddy replication and fine grained replication. According to their own statements [64], JBoss Application Server provides a complete Java platform by integrating Apache Tomcat as its web container along with capabilities for data caching, clustering, messaging, transactions, and an integrated web services stack that simplifies development of web services via web services metadata.

The SysMES framework has been developed based on the Java and EJB specification for the implementation of the system management methods. The implemented applications will be hosted on the JBoss AS, which allows clustered and transaction based data processing. XML will be used as the message format in which servers and managed targets communicate with each other.

4.6. Management Close to the Targets

Consideration: The decentralized data processing capability and the location independent management establish the starting point for another design strategy which defines that the processing of the data should occur as close as possible to the data source. The background of this is the effort to perform failure detection and solution as soon as possible.
Decision: In case the targets have the abilities to solve the recognized failures, they carry out the data processing. Otherwise the data has to be sent to the management servers to be processed. The data processing on the server side is carried out in a similar way, as the server closest to the corresponding target executes the needed reactions for problem resolution. If this server does not have all the information and capabilities for this purpose, the data will be sent to the next server layer to be processed and so forth.
The decision which data has to be processed in which layer is an individual decision of the system administrator. He has to trade off the management load caused by the targets against the cost for sending this data through the network and the needed response time. The SysMES Framework provides the services for this purpose and is able to process the data in each layer. More about the system architecture and the different physical and logical layers will be explained in chapter 5.

[1] JBoss is a registered name of the Red Hat corporation.

4.7. Centralized Operator View

Consideration: Although the decision for a decentralized system management has been made, it is also important to get the whole status of the environment to be managed in a central point even without knowing the specific physical infrastructure. The system administrator should interact with the management system without having to care about where the management services are running and where the data is stored.
Decision: The SysMES framework implements a system administrator view as a GUI, hiding the decentralized aspects of storage and processing of data. This allows the operator to interact with the management environment by executing management tasks regardless of the specific location of stored data or the server which processes it.

4.8. Modular Functionality

Consideration: As already mentioned in the previous sections, the SysMES framework has been designed as a multi-layered system for a better distribution of the management functionality. A suitable mapping is required between specific functionality and the (sub)layer where this has to be located.
Decision: In order to achieve the whole required functionality described in the goals in chapter 2, the whole management functionality has been divided into the following modules or subsystems.

- Monitoring Subsystem: This subsystem is in charge of monitoring system resources, i.e. to make measurements or read out the current value of system resources. This functionality is located in the targets.
- Event Management Subsystem: Our experience gained monitoring clusters demonstrates that about 90% of the values measured by the Monitors describe a normal status of a system resource and only the remaining 10% have to be processed. Therefore, the Event Management subsystem is in charge for analyzing the monitor data and for finding out if this has to be treated as a failure or an (un)desired state or not. Monitor data, which represent a problem or (un)desired state are called Events. Another point concerns the storage of the data which can be done in a decentralized way in the targets or the server layers.
- Rule Management Subsystem: This subsystem evaluates the monitor data in order to recognize (un)desired states, problems or errors. It is divided into a Simple Rule Management subsystem for the evaluation of single monitor values and a Complex Rule Management subsystem for the correlation of monitor values and to reason a complex or global state.

  The problem escalation strategy is also realized by Rules. If the execution of an action for the solution of a problem fails then a new Event with a higher priority will be generated. It is also possible to define other Rules for processing this new Event and for initiating a new reaction.
- Task Management Subsystem: The automatic reaction is one of the main topics of this subsystem. The second topic concerns the functionality for executing Tasks manually on the SysMES targets. The reporting capabilities are also included in this subsystem because they have been treated as reactions for informing system administrators about what is happening.
- Modeling Subsystem: This subsystem is in charge of the definition and management of a model, which contains information about the environment to be managed, the management resources and a mapping between both. Its realization is treated in detail in the forthcoming section.

4.9. Object-Oriented Modeling of the Management and the Business Environment

Consideration: One of the most important design determinations concerns the object-oriented modeling of the management environment (SysMES management capabilities), the environment to be managed (physical resources such as computer, servers, networks, etc. - the so-called business environment) and the relationships between them. Object-oriented modeling has been chosen in order to reduce the management complexity in a large and heterogeneous environment, using features like derivation, aggregation and association and the possibility to reuse modeled resources.
Decision: The first part is the decision to develop a UML [102]based model, which represents the management and physical resources, as well as their characteristics and relationships. It is one of the tasks of a system developer to model the structure, behavior, semantics and relationships of the environment to be managed. In case of the SysMES Framework, the model has been built using UML class diagrams. These diagrams consist of a set of attributes, which represent the state of an object of this class and a set of operations which can be executed on these objects.
The second part concerns the choice of a common object-oriented standard to generate and manage objects of the UML model. The CIM [34] has been chosen for this purpose.
The CIM is a standard for the object-oriented description of system management data and methods. It was developed by the Distributed Management Task Force (DMTF) [42], an independent organization, which was founded by a large number of major software and hardware companies. Additionally, it is freely available to the community, widespread and well known in the field of distributed system management.
For the class instantiation, the object storage and management, and the intercommunication of objects, OpenWBEM has been chosen. It is an implementation of the WBEM [103] standard and the CIM Object Manager (CIMOM). More information about these technologies and their applications can be found in the chapter 5.5.1
The specific models and the path from modeling to object creation and deployment will be explained in section 5.5.1.1

4.10. Automatic Device Update and Status Recovery

Consideration: There are two factors which cause some management components (such as system management clients and servers) in a distributed environment to not be updated or their status has to be recovered. The first factor is target unavailability provoked by hardware or software failures, or maintenance. The second factor is when the targets are not available because of network or connectivity failures.
Decision: The SysMES framework has been designed to ensure an automatic device update and status recovery after the failures have been resolved and targets become available. The targets affected by the failure will send their current configuration status to one of the management servers and will automatically get the corresponding updates, for example after an offline phase. In case of hardware failures, crashes or system reboots, the management target automatically deploys the locally saved configuration and reports its status to be updated if necessary. In a large distributed environment it is conceivable that new targets are added or substituted. These targets would be started with a basic configuration and would be updated automatically in a similar way. To sum up, this design decision is a suitable method for keeping the whole environment updated without the intervention of administrators. The update and recovery functionality concerns the state and configuration of

the SysMES services, as well as any kind of change, extension or reconfiguration performed by the SysMES framework.

4.11. Dynamic System Management

Consideration: A dynamic system management framework should be able to extend, reduce, activate or deactivate the management and reacting capabilities of targets and servers in a dynamic manner without downtime, i.e. it offers the possibility to reconfigure the management framework at runtime.
Decision: Dynamic management in the SysMES framework considers three aspects: dynamic configuration, extension and allocation.
Dynamic configuration is realized in the SysMES framework through the design and development of management objects such as Monitors, Rules and Actions, which can be reconfigured at runtime. The reconfiguration will be controlled using other management objects called Tasks.
The SysMES framework offers a collection of Monitors, Rules and Actions with an initial configuration, but it is possible to change the configuration, semantics and behavior of these if required by changing object attributes values.
Furthermore, the management framework allows system administrators to develop new management objects according to their needs. These management objects can be distributed at runtime to desired targets in order to extend their management capabilities.
In section 4.1 it was mentioned that the SysMES framework has multiple functionality layers. One aspect of dynamic allocation concerns the capability to increase or decrease the number of instances in a functionality layer. Another aspect is realized by the capability to change the location for system management services to servers and targets, i.e. to change the location where target data has to be processed from the target to a server.

5. The SysMES Architecture

This chapter describes how the SysMES Architecture has been developed according to the design considerations described in the last chapter. The first part introduces the basic management algorithms for the development of a management framework. The second part is related to the design of the SysMES framework and specifies the architecture and its layers, their responsibilities and the communication between these. The third part introduces each architectural layer. In particular, the functionality of the management capabilities, the algorithms for handling the management resources and the distribution of these across the whole architecture are explained.

5.1. General Management Algorithm

The first step in the development of a distributed system management framework is to define a basic algorithm which describes the activities needed to provide management services.
The system management algorithm is visualized by the following use case diagram 5.1 and the sequence diagram 5.2.

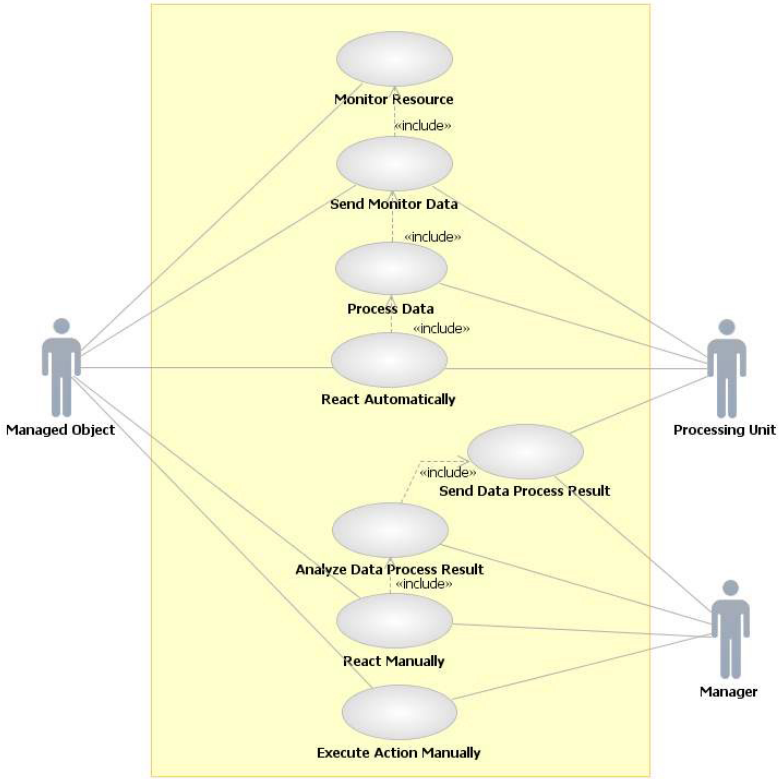


Figure 5.1.: System Management Use Case Diagram

The SysMES general algorithm involves three actuators according to figure 5.1. These are: the Managed Object, which requires the management services, the Processing Unit, which analyzes the Managed Object data and determines the next management steps, and the Manager, who decides how to interact with the Managed Object. The Managed Object represents all kinds of system management requestors, such as cluster nodes, servers, databases or applications. The Processing Unit represents the system management framework, which offers the management services and interacts with the administrators. The Manager can be a part of the system management framework in case of unattended system management or a person, such as an administrator, who is able to execute a reaction or a stand alone action manually on the Managed Objects. The execution of a stand alone action is described as when the Manager initiates the interaction with the Managed Object without the previous exchange and processing of Monitor data.

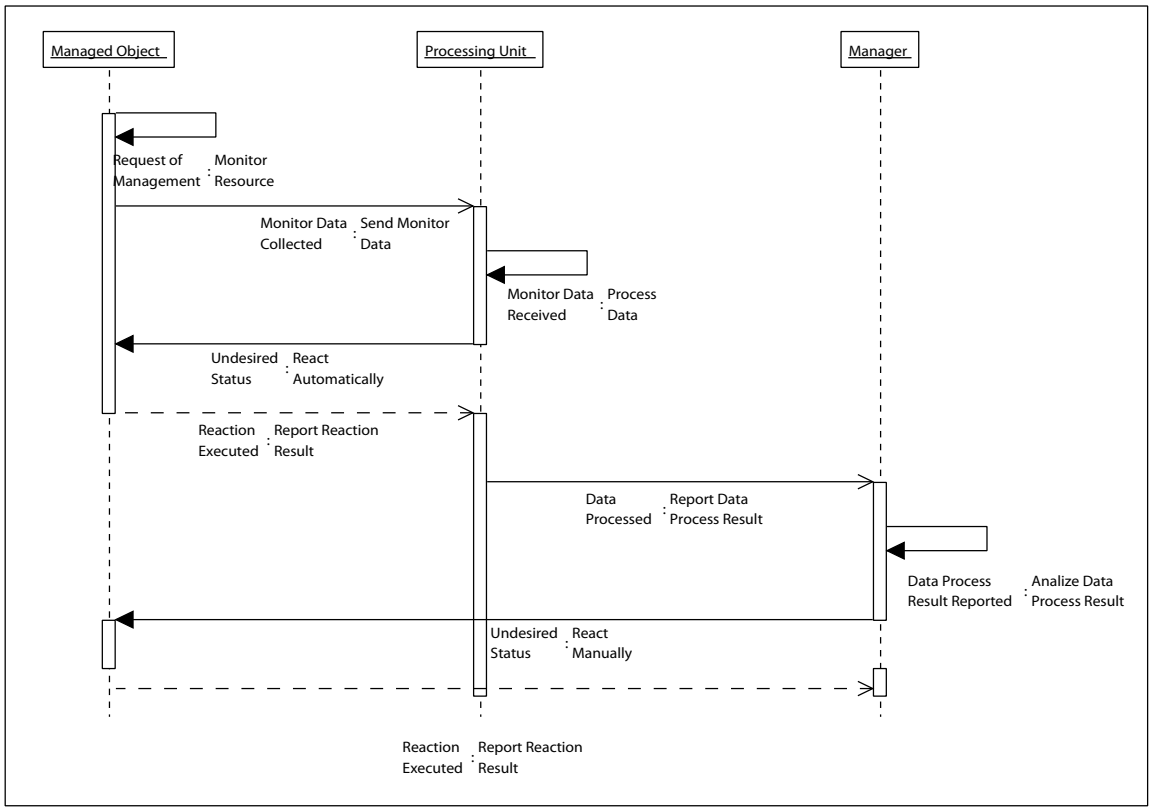


Figure 5.2.: System Management Sequence Diagramm

Figure 5.1 visualizes the basic tasks needed for system management. These are collecting data (i.e. monitoring), reporting, processing, analyzing, reacting and execution of actions. These activities can be performed by one actuator, e.g. the monitoring task, or by more than one, e.g the react task where one actuator (i.e. Manager) initializes the reacting process and the other (i.e. Managed Object) executes the desired action.
The time related work flow of the algorithm is described in the sequence diagram 5.2. The system management algorithm begins with resource monitoring on the Managed Objects side. Although this task provides an overview about the status of the resources, it is necessary to send the collected data to a Processing Unit in order to analyze it and to decide if further management tasks and more precise

reactions have to be considered.
After the processing task, the Processing Unit is in charge and decides to react automatically or to report the result to the Manager according to the processing result. The Manager receives the processing result and analyzes it in order to determine the next reaction strategy. Both the automatic and manual reaction should lead to the solution of recognized problems. The executive Managed Object will report the result of the reaction execution to its initiator.
The processing of the reaction results is not taken into account for the sake of simplicity, but this data can be analyzed and processed in the same way as the data obtained by monitoring.
The SysMES Framework is based on this simple algorithm and provides the whole functionality to achieve the system management task described in this section.
The next section will provide an overview of the design and will introduce the management architecture as well as its characteristics.

5.2. General Design

In principle, the SysMES Framework is built on three interconnected layers according to figure 5.3. This design has been chosen due to the different physical and functional characteristics of the three actuators of the management algorithm. The targets layer is located on the bottom of the figure, and contains the Managed Object, i.e. several system management consumers. The middle layer includes all the system management services offered by the Processing Unit, i.e. the system management provider side and interfaces the Operator Layer responsible for the interaction between the Manager and the management framework.

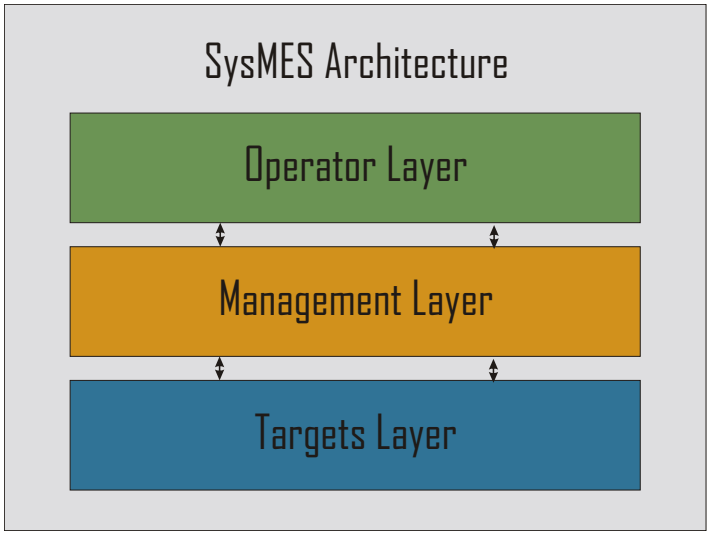


Figure 5.3.: General SysMES Architecture Diagram

The design of the physical architecture of the SysMES Framework is based on the following considerations:

- Vertical and Horizontal Distribution: Due to the requirements of scalability and dependability, the physical architecture distinguishes two different distribution methods. The vertical distribution describes the management capability to distribute the functionalities in different layers.

Each of these layers offers a set of managements tasks and is able to communicate with each other in order to exchange information or process results. The horizontal distribution thereby means that for every management unit hosting specific functionality of a layer, a set of identical replicas exist, which are able to substitute for each other.

- Management Tasks Close to the Initiator: As a characteristic of the management algorithms, the communication can be initiated in two different manners. The Managed Objects (Target Layer) initiate it in order to send monitoring data to the Processing Unit (Management Layer), or the Manager (Operator Layer) initiates the communication in order to execute stand alone actions on the Managed Object. According to the vertical distribution, the functionality for each initiator will be placed either on the top or the bottom of the hierarchy. This functional distribution is fundamental to achieve a system management close to the Managed Objects.

  Based on these considerations, the SysMES architecture described above has been extended to a multi-layered architecture in order to achieve the vertical distribution. Each of these three layers is divided into one or more sublayers. One special characteristic of each sublayer is the clustering of its functionality due to the horizontal distribution.

  Figure 5.4 visualizes the different layers and sublayers which belong to the SysMES physical architecture. The lowest layer is the Target Layer and includes SysMES clients (just called clients). This is the only layer where the management functionalities are not clustered, which means that each member of the layer is a different Managed Object. The next upper layer is the Management Layer consisting of access point and server sublayer. The access point layer is responsible for the communication between different client implementations and the servers. The server sublayer consists of the Local Area Management (LAM) layer and the Wide Area Management (WAM) layer. On the top of the hierarchy is the Operator Layer, which is divided into the Modeling Layer, where the management model is hosted, and a Graphical User Interface (GUI) layer.

  The physical infrastructure fulfills the requirements of scalability and dependability because it supports the vertical and horizontal distribution of load. This begins with the reduction of the amount of data to be processed on the clients and ends on the server side with the distribution of the management activities on different layers. Another design consideration attached to the physical layer is that system management should be located as close as possible to the initiator. This is reflected in the strategy to process the Monitor data first on the clients - where the monitoring has been initiated - and second on the servers. Similar to this, the distribution of management objects at the top of the Server Layer occurs close to the Modeling Layer.

- Transparent Management: The SysMES framework has been designed for the management of one or more different physical environments through the logical transparent separation of management infrastructures running on one physical management infrastructure. More explicitly, it is possible to use one instance of the SysMES framework for the system management of different enterprise infrastructures. This is possible due to the assignment of different IDs to each participant of the framework regardless of the fact whether it is a server or a client. There is a FirmID for the assignment to one explicit firm or enterprise, a GroupID for the organization of multiple devices of the same enterprise and distribution to this and DeviceID for the identification of each device. In conclusion, each device has one unique DeviceID, one or more GroupIDs and one FirmID for its identification.

- Object-Oriented System Management Resources: According to the design considerations (see

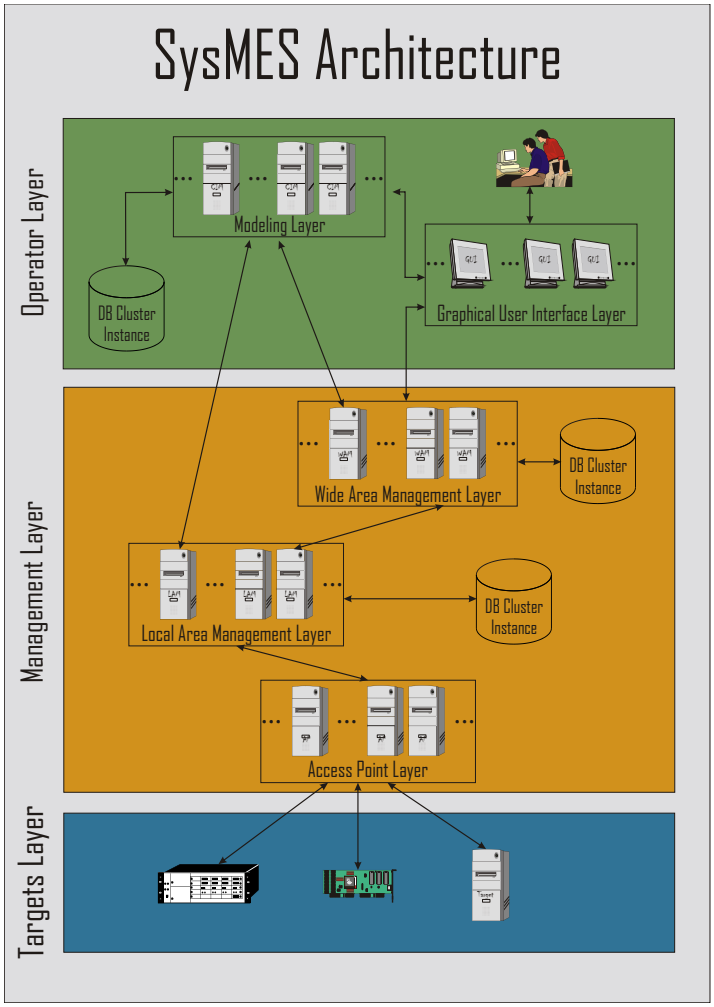


Figure 5.4.: The SysMES Physical Architecture Diagram

chapter 4) the management environment is based on an object-oriented model. In case of the SysMES framework, a UML based Model has been designed and developed. This is the RBEM, and its basic classes can be found in figure 5.5

To simplify matters and in order to visualize the RBEM Model, only the basic management resources are introduced but the extended description of these, as well as the model of the environment to be managed and the relationships between both, can be found in 5.5.1.1

The RBEM Model introduces a root class named RBEM_ManagedElement and all the other classes inherit attributes from it. This class has four basic attributes: ElementName, which represent the primary and unique key of each object, Caption, utilized for the visualization of management objects, CreationClassName, which contains the name of the class which the objects belongs to, and Description, which is a textual description of the object and it is used for the visualization.

The main derived classes are attached to the SysMES functionalities and are RBEM_Monitor, RBEM_Action, and RBEM_EventClass for Monitoring, RBEM_TargetMask and RBEM_Task

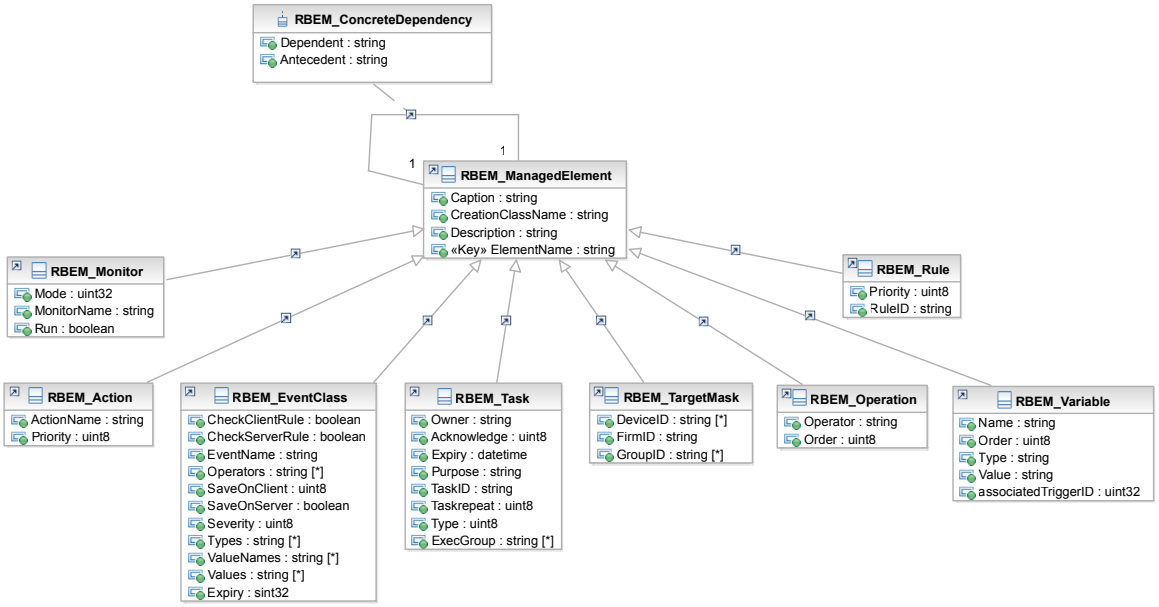


Figure 5.5.: RBEM System Management Model - Basic Classes

for Task management, and finally, RBEM_Rule, RBEM_Operation and RBEM_Variable for Rule management.

The association classes RBEM_ConcreteDependency is located near the top of figure 5.5 . This class is used to model the relationship between the other classes [1].

- Inter Layer Communications and Data Format: The communication between several SysMES layers is carried out by exchanging messages in XML format. In detail, the clients send the collected and pre-analyzed Monitor data to the servers in form of XML documents and the servers transmit the Actions to be executed using this format as well. The SysMES framework uses only two types of XML messages. The top-down communication is realized by exchanging the XML representation of the Task objects as described in section 5.3.4 and the bottom-up communication by Events as described in section 5.3.2.

  The reason for the choice of a XML based communication format is that XML is a highly flexible and versatile language, it is widely considered an accepted standard for data description and it is platform independent. Furthermore there are multiple standards such as XML-ENC and XML-DSIG for a flexible encryption and authentication of the XML documents or the data contained in it.

The next sections discuss the three introduced layers in detail, their physical and functional properties and the interaction with each other. The following description starts with the client layer at the bottom of the architecture up to the middle with the Management Layer and more specifically, with the Access Point, LAM and WAM layer and finally ends with the Operator Layer on the top of the architecture.

[1] For readability reasons the prefix "RBEM_" of the class name will be omitted in further sections of this thesis.

5.3. Client Layer

In general, the tasks of a SysMES client comprise on one side monitoring of any system resources and the corresponding measured values and on the other side the execution of management objects, i.e. Actions.
At this time it is important to clarify what a system resource is. It is a part of a managed object (such as cluster nodes, embedded devices, servers, network devices, etc.) which requires monitoring and management services. A system resource can be a hardware device (e.g. sensors, hard disk, CPU, memory, etc.) and applications (e.g. daemons and other kind of resources such as log files).
This section discusses in detail the tasks from figure 5.6 and the several interactions between clients and servers.

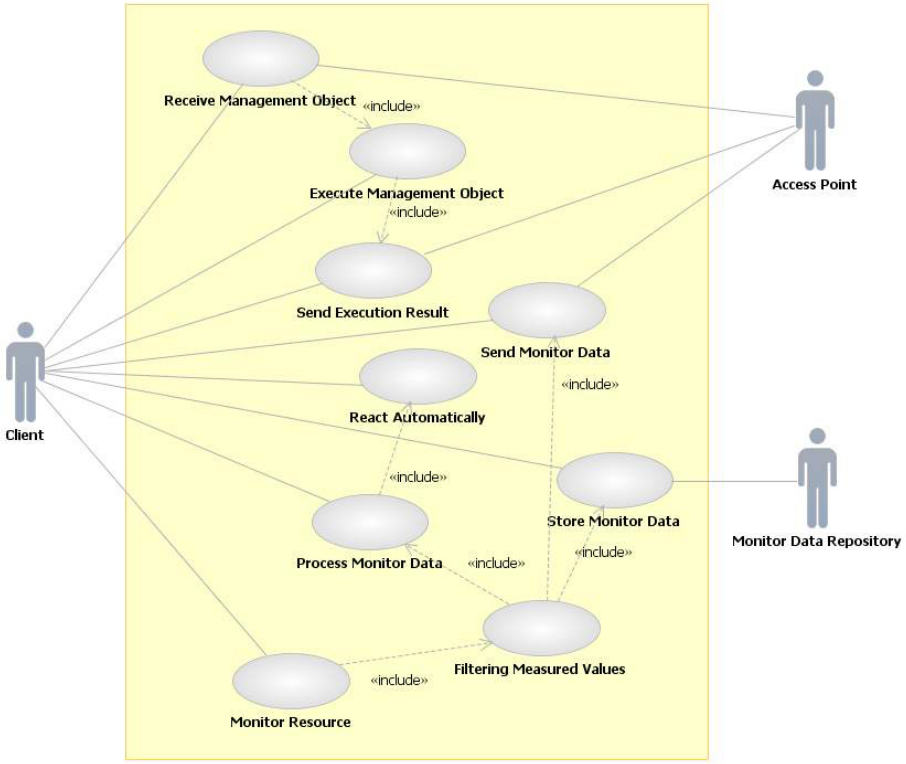


Figure 5.6.: System Management - Client Side Use Case Diagram

5.3.1. Distributed Monitoring

The monitoring capability determines the process to extract information from the system resources in order to identify a critical status. Furthermore, Monitor data is used as input for triggering the execution of an action, which will be used to resolve a problem and to return the monitored system to a healthy state.
The concept of "Distributed Monitoring" is usually used for the centralized online monitoring of physically distributed managed objects. This kind of monitoring offers a near real-time status of the monitored resources, but in fact causes a lot of network load for the transmission of the measured values to a central processing unit and does not offer the capabilities to resolve problems automatically.

The SysMES framework extends this to the capability of monitoring the system resources of the distributed objects from different locations using different strategies. In order to introduce the monitoring strategies, it is necessary to define what a Monitor is.
A Monitor is a management resource, which observes the behavior of a system resource by reading out its attributes and properties. As shown in figure 5.7, each Monitor contains a set of attributes, which determine when and how often it has to be executed, a set of triggers (the so-called Event Class which will be introduced and explained in the next section 5.3.2) for deciding if the measured value needs to be processed and a monitoring action object. Actions are management objects similar to the Monitors and describe how the attributes and properties can be read out.

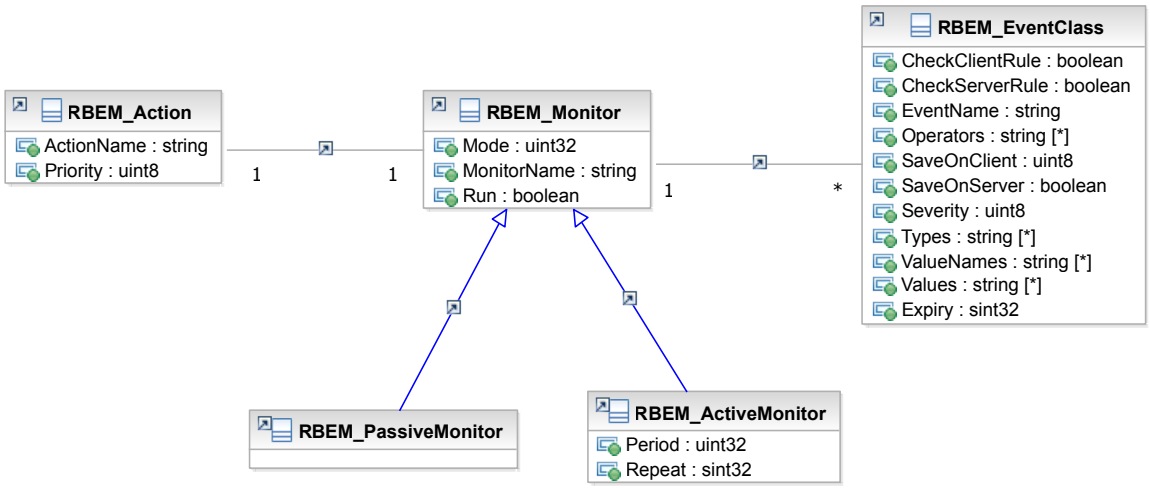


Figure 5.7.: SysMES Monitor Classes

The development and storage of Monitor and Action objects takes place at the Operator Layer and they are distributed from there to the targets through the Server Layer. This method allows the extension of the monitoring capabilities at any time because the needed binaries or their configurations will be developed according to the monitoring requirements and distributed to the targets. The targets are also able to deploy the new Monitors and - more interestingly - to execute the new Actions. In a similar way it is also possible to reconfigure the Monitors on the fly without downtime of the targets. One typical and often used online reconfiguration is the activation and deactivation of the Monitor by setting the Run attribute to `"true"` or `"false"`.
The SysMES client monitoring strategy is based on the idea that every programmable Action can be used for the development of a new Monitor. However, it is also possible to develop Monitors which do not need to actively measure the values of the system resources, but get these from other monitoring systems. According to the measuring strategy, Monitors can be developed as active or passive.
The second important strategy determines how a Monitor result has to be calculated. Monitors have the capability to calculate one single result using one or a series of measurements. According to this strategy, Monitors can be persistent and non-persistent. The persistent strategy accepts series of measurements as input and calculates one result using an Operator specified for the Monitor object and the non-persistent strategy utilizes exactly one measurement as a monitoring result.
There are four different Monitor types as a result of the combination of these strategies, these are Non-Persistent Active Monitors, Non-Persistent Passive Monitors, Persistent Active Monitors and

Persistent Passive Monitors. The most common Monitors are the Non-Persistent Active Monitors and Non-Persistent Passive Monitors, which are for simplicity named Active Monitors and Passive Monitors. Throughout the following chapters and sections, the terms "measurement" and "Monitor result" are used as synonym in order to simplify matters.
The syntax and semantics of the four Monitors are described as follows

- Active Monitor: This represents a SysMES client Monitor object, which is able to read out values of the system resources in an active way. That means that it is featured with the capabilities, more exactly the binaries or executables, needed to measure the value of the monitored resources. In principle, there are two possibilities. The first one is the execution of binaries, which are located on the targets and the second one is that the monitoring action contains the binary code (which is encoded in base64 [18]) needed to measure the properties.

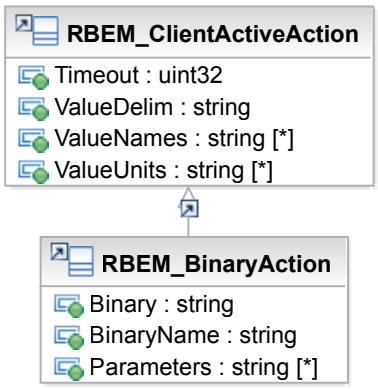


Figure 5.8.: SysMES Binary Action Class

Active Monitors possess the attributes Period and Repeat, which describe their time behavior. Period describes a time interval (in milliseconds) in which the Monitor has to be executed. This is a mandatory attribute of each Active Monitor. Repeat describes how often a Monitor has to be executed. The default value of Repeat is `"-1"` and means that the Monitor will be executed infinitely.

Active Monitors are associated with an Active Action, more exactly the Binary Action shown in figure 5.8, which contains either the attributes Binary (binary code in base64 coded) and Parameters, or the name of a local binary to be executed set on the attribute BinaryName. The SysMES clients are able to receive, save and decode the binaries and to use these for the monitoring of system resources. A sample of a Monitor object including a Binary Action can be found in figure 5.9.

Figure 5.9 describes a Monitor object utilized for getting the value of two CMOS [2] settings and its associated Binary Action and Event Class objects. The Binary Action contains the base64 code utilized for reading out the system resource. This Monitor is deployed on the Computer Health and Remote Management (CHARM) card [87] [3] and is used to get the values of the

[2]CMOS is the abbreviation of Complementary Metal-Oxide Semiconductor and in the area of computing science it describes a battery-powered memory chip for storing Basic Input Output System (BIOS) configuration.
[3]The CHARM card is a remote management add-on card used for monitoring and management of the computer where it is plugged in.

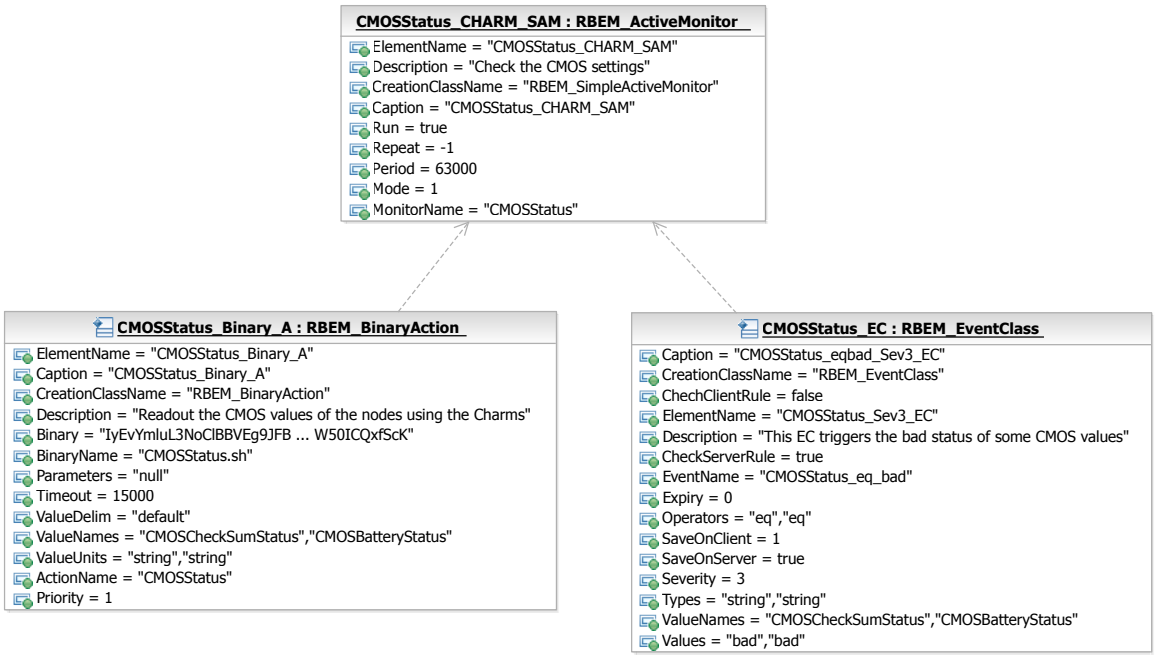


Figure 5.9.: SysMES Monitor Object

CMOS battery status and the CMOS checksum status. The significance of the Event Class is described in the next section.

Another important attribute of the Binary Action is Timeout, which defines a time interval to terminate the execution of the action in case of failures. In case of multiple measurements, the Binary Action object has the attributes ValueDelim, which represents a delimiter between several measurement values (default value is " " i.e. a blank space), ValueNames to label each individual measurement, and ValueUnits containing the data type of the measurements. If the Monitor returns only one measurement, then it is not necessary to set these attributes. The Binary Actions are the most used action type for the interaction with the targets and are used in relation with other management objects such as Tasks, which are introduced in upcoming section 5.3.4.

One of the most important characteristics of the Active Monitors is flexibility because each Monitor and its behavior can be reconfigured, changing the binary of the associated Binary Action. The SysMES Client supports each type of binary such as scripts (.sh, .py, .bash, etc.) or executables.

- Passive Monitor: This describes the needed information to receive and process information from other monitoring systems.

  The SysMES client implements the functionality for getting the measured values from third party monitoring systems such as Lemon [71], Ganglia [52] or Nagios [82]. This will be carried out by making a UDP and TCP listener available. The only restriction for supporting third party monitoring is the message format, which is formally defined as follows:

$$
\begin{aligned}
message &= header,\ \{sample\}-;\\
header &= msg\_version,\ "\ ",\ msg\_method;\\
msg\_version &= "A1";\\
msg\_method &= "0";\\
sample &= nodename,\ "\#",\ metric\_id,\ "\ ",\ timestamp,\ values,\ "\#";\\
nodename &= string;\\
metric\_id &= string;\\
timestamp &= uint;\\
values &= \{"\ ",\ value\}-;\\
value &= string \mid number;\\
string &= \{character\}-;\\
number &= ["-"], uint, ["."], uint];\\
uint &= \{digit\}-;\\
character &= letter \mid digit \mid "\wedge" \mid "!" \mid "\$" \mid "\%" \mid "\&" \mid "/" \mid\\
&\quad "(" \mid ")" \mid "=" \mid "?" \mid "+" \mid "-" \mid "*" \mid "\ " \mid\\
&\quad "\_" \mid "." \mid ":" \mid "," \mid ";" \mid "|" \mid "@" \mid "\{" \mid "\}" \mid "[" \mid "]";\\
digit &= "0" \mid "1" \mid "2" \mid "3" \mid "4" \mid "5" \mid "6" \mid "7" \mid "8" \mid "9";\\
letter &= ?\ A..Z,\ a..z\ ?;
\end{aligned}
$$

The following example visualizes a message from Lemon Monitoring, where each line contains the measured data about the utilization - and other values - of single system resources, in this case the hard disk partitions.

```
A1 0 apoll#9104 1207228171 / ext3 rw 38456308 90 -1 -1 10 0#
apoll#9104 1207228171 /home reiserfs rw 39061264 58 -1 -1 -1 0 #
apoll#9104 1207228171 /tmp reiserfs rw 121415040 51 -1 -1 -1 1#
```

The amount of information contained in the sample message makes clear the necessity of new attributes in order to parse and map the collected information. The Passive Monitor associates a Passive Action, which has these attributes and contains the configuration for managing the data from external monitoring tools. A sample of a Passive Action can be found in table 5.1.

MetricId	9104
RowID	/home
RowIDPosition	0
ValueNames	[fs, size, used]
ValueUnits	[none, KB, perc]
ValuePositions	[1, 3, 4]

Table 5.1.: SysMES Monitors - Action Sample

In contrast to Active Monitors, this kind of Monitor do not own the time related attributes, such as Repeat and Period, because the SysMES client does not influence the measuring behavior of the external monitoring system.

The MetricId is related to the Monitor identifier used by the third party Monitor. The RowID specifies a particular system resource. The RowIdPosition indicates which of the data included in the message has to be interpreted as the RowID. Besides the attributes for the specification of the monitored system resource, it is also necessary to interpret the values contained in the message. This is done using the attributes ValueName, ValueUnit and ValuePosition.

The SysMES client utilizes the mentioned attributes to parse the message extract the components described in table 5.2 and to calculate a result collection for the desired Monitors, discarding unnecessary information. In the case of the introduced example, the result collection is described in table 5.3

Nodename	**MetricID**	**Timestamp**	**Values Position:**								
			0	1	2	3	4	5	6	7	8
apoll	9104	1207228171	/	ext3	rw	38456308	90	-1	-1	10	0
apoll	9104	1207228171	/home	reiserfs	rw	39061264	50	-1	-1	-1	0
apoll	9104	1207228171	/tmp	reiserfs	rw	121415040	51	-1	-1	-1	1

Table 5.2.: Third Party Monitoring - Parsed Values

ResultCollection		
valueName_	valueUnit_	value_
nodename	txt	apoll
fs	txt	reiserfs
size	KB	39061264
used	perc	50

Table 5.3.: Third Party Monitoring - Result Collection

- Active and Passive Persistent Monitors: Independent of the measurement strategy, both kinds of Persistent Monitors have the capability to make a calculation with one or a series of measurements and to return one Monitor result. The Monitor makes a series of measurements of the size described in the attribute SetSize. This attribute has to be set to `">= 2"`.

  As described at the beginning of the section, a Monitor measurement can contain more than one value. These different values can also be different data types, which complicates the application of a specific Operator to a set of values. Therefore the SysMES Persistent Monitors define two attributes (vectors) Operators and ValueNames, which are used to identify related tuples (Operator, Value) where the Operator can be applied to the data type of the value. One restriction is that it is only possible to define one tuple per value. At the moment, two Operators have been developed. These are `"Gradient"` and `"Average"`.

  However, the SysMES client is designed to be open for the development and integration of other Operators. Analog to the other Monitors, the Period attribute is responsible for the periodicity of the measurements and Repeat for the number of repetitions.

5.3.2. Event Handling

As discussed in the previous section, the SysMES client uses different ways and strategies for the collection of monitoring data. Contrary to traditional monitoring systems, where the main task is to build a real time database containing a history and the actual state of the monitored system resources, the attention of the SysMES framework is turned to the processing of monitoring data which represents an undesirable state, an error or a special state, the so-called Monitoring Events. The SysMES client also implements other kinds of Events, the so-called Administrative Events and the Application Events, but in order to simplify matters the general term Event relates to the Monitoring Events.
The processing of Monitor data on the client side is carried out by the Simple Rule Management functionality 5.3.3, which is detailed in the next section. The pre-processing of the Monitor data consists of checking if this data is relevant and reacting creating a new Event.
This pre-processing capability is the first strategy for the reduction of load and more explicitly, the increase of scalability according to the design considerations of chapter 4.
The main idea is to recognize which of the measured values represent an undesirable or special state of the monitored resource and to process only this information. All other data will be discarded automatically on the client side and therefore carry no weight for the next steps of the client side management. However, SysMES Monitors also can be configured for collecting statistical data and processing each measurement, if required.
In order to recognize these states, the SysMES client uses Event Classes introduced by the Monitors 5.7. An Event Class is a management object, which contains a Trigger and additional information. The Trigger is utilized to decide whether for a specific measured value an Event of this class has to be generated. The additional information is used for the further processing of the generated Events. Figure 5.9 shows an example of an Event Class and its attributes.
A Trigger is defined as a management object formed by a set of Conditions, which are AND related. The attributes ValueNames, Operators and Values define these Conditions. In the case of the sample, Event Class shown in figure 5.9, there are two Conditions: `"CMOSCheckSumStatus=bad"` and `"CMOSBateryStatus=bad"`. The triggering algorithm of the SysMES clients utilizes a set of Operators which will be discussed in section 5.3.3. Types define the data type of the Values attribute (e.g. Integer, Double, Float and String).
EventName represents an identifier for generated Events. The attribute Severity stands for the importance and urgency of the Events. There is a scale of four different severities, which represent four different Event types. These are Immediate Event `"Severity=1"`, which implicates a very urgent Event which has to be processed immediately, Critical Event `"Severity=2"`, Service Event `"Severity=3"` and Information Event `"Severity=4"`, for the collection of data which do not represent a serious problem.
The SysMES client implements other Severities used for administrative purposes and for generating Events concerning other applications. For example, the client generates special Events in order to send application log messages to the server Log Event `"Severity=5"` or to report its online status, i.e. Alive Event `"Severity=8"`. Events can also be used to send some information concerning the execution of Actions such as the return value, i.e. TaskReply Event `"Severity=6"`, execution or transmission errors, i.e. Error Event `"Severity=7"` and the receive and execute acknowledge, i.e. Ack Event `"Severity=9"`.
The CheckClientRule and CheckServerRule attributes describe the different possibilities for the further processing of the Events. The SysMES client enables and/or disables the processing of the Events on the clients (local Events processing) and servers (remote Events processing) depending on the setting of these attributes. It is also possible to change the values of these attributes on the fly, which

allows a dynamic adjustment of the processing strategy.
Event storing strategy is defined by setting the attributes SaveOnClient for saving Events locally on the client or SaveOnServer on the servers or in both locations.
The Expiry attribute of the Event Class is used in order to define a time related validity of Events. The default value is the value of the Monitor attribute Period, where the Event Class is associated, but it is possible to set this attribute to any value greater than zero.
For each Severity there is a default configuration as can be found in table 5.4. For the Monitoring Events it is possible to reconfigure the Event Classes with values which differ from the defaults.

Severity	**SaveOnClient**	**SaveOnServer**	**CheckClientRule**	**CheckServerRule**
Immediate	1 (bevor send)	true	true	true
Critival	2 (after send)	true	true	true
Service	0 (no)	true	false	true
Information	0 (no)	true	false	false
Log	0 (no)	true	false	false
TaskReply	0 (no)	true	false	false
Error	0 (no)	true	false	false
Alive	0 (no)	true	false	false
Acknowledge	0 (no)	true	false	false

Table 5.4.: Severities: Default Strategies

Another SysMES client functionality to be mentioned is the Inject Interface. This interface allows the injection of Events into the SysMES client and therefore avoids the local triggering decision. Such a scenario can be the monitoring of devices where a SysMES client cannot run or be installed, i.e. network devices such as switches or rack monitoring systems. These devices are able to Trigger themselves and to send Events to a SysMES client using its inject interface. The Events injected through this interface will be processed in exactly the same way as the native SysMES Events. The interface user defines the Event storage and processing strategy by setting the discussed attributes. It is also possible to set the Severity so that depending on this, a default strategy is offered.
After the Event generation, the SysMES client executes the next two phases in parallel. The first one concerns the storage and forwarding of Events and the second one the processing of these on the client side.
As a part of the distributed and decentralized strategy, the SysMES Client is able to store Events locally and persistently. This is strictly necessary because the scalability requirement can only be met if it is possible to reallocate the storage of Events. Another important aspect is the avoidance of information loss resulting from failures in network connectivity to the servers. The clients are able to save the Events and to send these to the servers when they go online again.
The storage and forward activities will be executed according to the values of the SaveOnClient attribute. There are two different methods depending on the reliability requirements for the delivery of Events. The first method `"SaveOnClient=1"` saves the Event to the local Event Cache and then tries to send it to a SysMES server. The Events will be kept saved while the server receives and stores them. In case of network or transmission failures, the SysMES client is able to resend it. This method should be preferred only for high priority Events because the delivery of these will be delayed due to the storage procedure. However, this method offers higher reliability concerning the information loss. The second method `"SaveOnClient=2"` works on the reverse order. The Event will be sent to the servers and afterwards stored in the local Event Cache, which accelerates its delivering. It is also possible to

configure the Event Classes so that no Events will be stored on the clients "SaveOnClient=0".

The Events will be stored in an Event Cache, which implements a table containing the Event information and its state. The Events in the Event Cache can have different states, which describe whether they are already and successfully sent "Status=sent" or queued (i.e. in case of a huge numbers of Events) and waiting to be sent "Status=open". The size of the Event Cache is part of the SysMES client configuration and in order to guarantee its functionality and to avoid a cache overflow, a self monitoring and management algorithm has been developed.

The algorithm uses two different strategies to ensure an Event Cache utilization beneath a specific threshold "default=85%". The first method is the compression of sent Events of a single specific Severity, beginning with the highest, which means the most unimportant (As a reminder this is "Severity=4"). This compression generates a new Event from a number of Events to be compressed. Thereby, the number of Events determines the value for the Count attribute for the new Event. The FirstOccurrence and the LastOccurrence attributes are set according to the oldest and newest occurrence of the Events to be compressed. For non-compressed Events the value of both attributes are equal.

All the other Events but the newly generated one will be deleted and as a result of the first method, the Event Cache contains only one Event of a specific triple (Severity, MonitorName, EventName). The second method is the deletion of all sent Events of a specific Severity. Both methods are used recursively and alternately and before each iteration, the Event Cache utilization will be checked in order to execute the compression and deletion methods only if necessary.

As described above, the Event forwarding to the SysMES server can occur before or after the persistent storage. The client sends the Events independently from the fact whether they have been processed or not. The reason for this is the fact that the servers possess complex processing capabilities, which are able to correlate more information in order to recognize more complex states. The exchange of data between the clients and servers occurs by sending XML documents. These documents contain information about the target where the Events have been generated and also about the Monitor and the respective measured value. As an example of the Event syntax, a XML document is shown in figure 5.10.

The Event document sample visualizes the syntax and semantics of SysMES Events. It shows the capability to deliver information about different system resources, which have been monitored by the same Monitor, in this case "CMOSCheckSumStatus=bad & CMOSBatteryStatus=bad". The complete XML sample can be found in figure 5.17. Other important information is coded in the CurrentClientTID tag, which represents the current configuration identifier needed by the servers to recognize its inventory and configuration status, more information on this can be found in sections 5.3.4 and 5.4.2.5. Additional information about the originator is also included in the Event document. An Event always contains a unique client identifier "DeviceID=10.162.128.231", an identifier for the assignment to a firm "FirmID=6666" and the name of the host where the client is running "Hostname=feptpcao10-charm". Finally, the Events inform the servers about the client architecture "Type=x86_64", its operating system "System=Linux" and the installed SysMES client version "ClientVersion=3.17".

After the introduction of the Event Class concept and before the further processing of Events has been introduced, is necessary to describe the last attribute of a Monitor object, the so-called Mode. All Monitors implement this attribute, regardless of its nature (active, passive or persistent). The SysMES client distinguishes between two different Modes, which will be used for configuring it in order to send each occurred Event "Mode=0" or to send the Events if the Event Class and more exactly its Severity changes "Mode=1".

```
<?xml version="1.0" encoding="utf-8"?>
  <!doctype ClientMessage>
  <ClientMessage>
     <CurrentClientTID>1220856000829.10.162.15.226</CurrentClientTID>
        <DeviceInfo>
           <ManagedEntityID>
              <DeviceID>10.162.128.231</DeviceID>
                        ........
           <EventID>1216640975326.10.162.128.231</EventID>
           <Info>
              <AttrName>CMOSCheckSumStatus</AttrName>
              <AttrName>CMOSBatteryStatus</AttrName>
              <Value>bad</Value>
              <Value>bad</Value>
              <Unit>string</Unit>
              <Unit>string</Unit>
           </Info>
           <ProcessedRule></ProcessedRule>
           <Expiry>121000</Expiry>
        </Event>
     </Events>
  </Clientmessage>
```

Figure 5.10.: XML Event Document

5.3.3. Simple Rule Management

Up until now, the process of monitoring resources and how to determine whether the measured values represent errors and also the capability to send these to the SysMES servers has been described. This section discusses the processing of Events, i.e. the recognition of special or undesirable states using the client Rules and the capabilities to react automatically, if needed.

In general, the SysMES framework has been designed for the processing of Events using a three-layered Rule-based system. The first layer is the Client Simple Rule Layer, the second layer is the Server Simple Rule Layer and finally the Server Complex Rule Layer. The first of these layers is, as the name suggests, attached to the SysMES clients and will be discussed in this section, the other two layers are located on the server side and will be discussed in section 5.4.2.4.

One of the most important characteristics of the Client Simple Rule Layer is the capability to process the Events locally without server interaction and to react automatically, which implies the capability to work standing alone in case of network problems, such as connectivity loss or server crashes. An example showing the importance of this feature is a Monitor, which observes the temperature of the CPU and reacts with a shutdown of the device if the temperature exceeds a predefined threshold (i.e. 65° C). The shutdown action has to be executed independent from the fact whether the client is connected to a server or not, in order to save the hardware from damage. Another advantage is the processing speedup due to the avoidance of communication overhead. However, the processing capability is limited to take only the current Event into account in favor of fast problem recognition and reaction. The need for the correlation of Events in order to recognize complex or global states justifies the Complex Rule Layer.

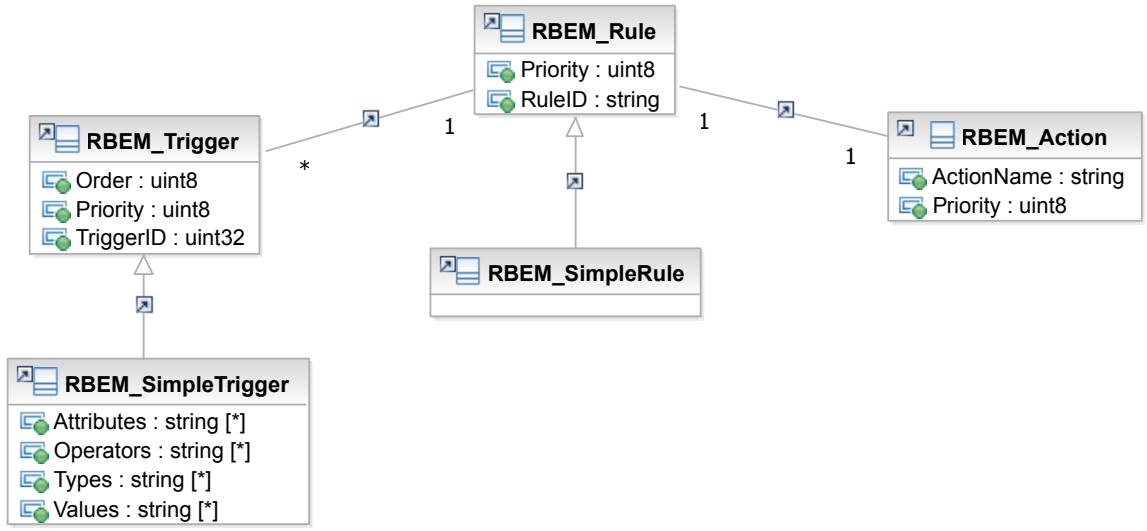


Figure 5.11.: SysMES - Simple Rule Classes

Rules are the management objects utilized for the processing of Events. Figure 5.11 visualizes their structure. In general, Rule objects are If-Then clauses, which describe on the left side (LS) a system status and on the right side (RS) Actions to be executed when the left side - the Triggers - are fulfilled. The SysMES Rule system is based on two different kind of Rules, the Simple Rules and the Complex Rules. The difference between these is merely related to the complexity of the left side and the capability to evaluate only one Event (Simple Rules) or a set or Events (Complex Rules).
The Client Simple Rule functionality will be explained in the following.
A Simple Rule is composed of a set of Simple Triggers building the left side and one Action building the right side. The properties of the Simple Trigger class visualized in figure 5.11 are used for the definition of Conditions. These are: Attributes, Operators, Values and Types. These properties can be arrays, which means that a Simple Trigger can have multiple Conditions. A Simple Trigger is then fulfilled if all Conditions are fulfilled.
The first formal definition of a Trigger and its evaluation function for the SysMES framework can be found in [23]. The following definition extends the initial definition as follows:

Be LS a set of Triggers (more exactly a set of Simple Triggers) $LS = \{Tr_1, Tr_2, ..., Tr_m\}$ with the properties:

$Attributes = \{A_1, A_2, ...A_n\}$
$Operators = \{O_1, O_2, ...O_n\}$
$Values = \{V_1, V_2, ...V_n\}$
$Types = \{T_1, T_2, ...T_n\}$

A Condition is defined as a quadruple $C_i = (A_i, O_i, V_i, T_i)\ i \in (1...n)$ and consequently a Trigger is a set of Conditions and formal defined as $Tr = \{C_1, C_2, ...C_n\}$

Be H a class of functions with

$h_{O_iT_i} \in H : T_i \times T_i \rightarrow \{True, False\}$

Be $F : \{f_{Integer}, f_{Float}, f_{Long}, f_{Double}, f_{String}\}$ a class of functions for converting a string value in the respective type, with

$f_{Integer}: \quad String \rightarrow Integer$
$f_{Float}: \quad String \rightarrow Float$
$f_{Long}: \quad String \rightarrow Long$
$f_{Double}: \quad String \rightarrow Double$
$f_{String}: \quad String \rightarrow Integer$

Be E an Event and $g : A_i \times E \rightarrow String$; a function which extracts the value attached to an specific attribute from an Event

A Condition C_i is fulfilled by an Event E if, and only if,
$h_{O_iT_i}\left(f_{T_I}\left(\, g\left(A_i, E\right)\right), f_{T_I}\left(V_i\right)\right) = True$

consequently a Trigger Tr is fulfilled by the Event E if, and only if,
$\forall\, C_i \in Tr \;\; i \in \{1...n\} \;\; h_{O_iT_i}\left(f_{T_I}\left(\, g\left(A_i, E\right)\right), f_{T_I}\left(V_i\right)\right) = True$

And finally the left side $LS = \{Tr_1, Tr_2, ..., Tr_m\}$ is fulfilled by the Event E if, and only if,
$\exists\, Tr_i \in LS \; i \in (1...m) : Tr_i$ is fulfilled.

The values of the Operators' attributes and their semantics can be found in table 5.5. There is a special one, the `"always operator (a)"`, which always returns true. This Operator is mostly used in order to collect information about the behavior of a specific Monitor and to calibrate the pre-filtering capabilities. Another usage of this Operator is to check the correct functionality of Monitors.

Operator	Reference value	Test value	Semantics
a	always operator		return always true.
eq	x	z	true if $z = x$
ne	x	z	true if $z! = x$
gt	x	z	true if $z > x$
ge	x	z	true if $z \geq x$
lt	x	z	true if $z < x$
le	x	z	true if $z \leq x$
b	xay	z	true falls $x \leq z < y$

Table 5.5.: SysMES Client: Trigger Operators

Summarizing, a SysMES Client Simple Rule object is composed of the Modeling Layer of two object types: One or more Simple Trigger objects and one Action. Each Simple Trigger object contains one or more Conditions, which are a quadruple such as:

$$[Attributes(i), Operators(i), Values(i), Types(i)]$$

A Simple Rule, which has been deployed on the client side, is a management object composed of a left side with an OR-concatenation of Simple Triggers and one action as the right side. Each Simple Trigger object is a conjunction of Conditions. Therefore, the Simple Trigger is fulfilled if all Conditions are fulfilled. A Simple Rule is fulfilled if at least one of its Simple Triggers is fulfilled.

A faster processing of Events through Simple Rules implies the possibility to react faster. More important is the capability to react locally and independently from a server. In case of network failures, the client is able to manage itself in a stand alone mode using Client Simple Rules. The SysMES client reactions correspond to the action objects of figure 5.11 and will be used in order to contribute to bringing the system back to a stable status and to inform the responsible administrator or operator.
As seen in the cardinality between Rules and Actions in figure 5.11, the right side of the Client Simple Rules is one action object. The SysMES client uses the so-called Binary Action (see figure 5.8) as a special class derived originally from the class Action and derived directly from the class Client Active Action. In principle the SysMES client is responsible for executing these Actions and sending Events to the servers to inform them that a Simple Rule has matched, a Binary action has been executed and also to report the execution result.
The practical Simple Rule evaluation algorithm used on the client side is the same as the one used on the server side and will be discussed in section 5.4.2.4.1.

5.3.4. Client Task Management

The previous sections discussed the management capabilities of the SysMES clients concerning the monitoring of system resources, the Event generation and processing by Rules. It was also assumed that the required management objects were deployed and the SysMES client was configured and calibrated for this purpose. In fact, it was discussed that it is possible to calibrate or reconfigure the Monitors, Rules and Actions dynamically and also to extend the reaction capabilities by deploying new management objects, but not about how this occurs.
This section describes how the SysMES clients can be set, configured and, if needed, reconfigured. Furthermore, the possible ways of interaction between system administrators or operators with the managed environment is described.
The SysMES framework utilizes the management objects named Tasks. In principle the Tasks are responsible for the execution of any kind of Actions on all the members of the SysMES framework. The top-down communication from the servers to the targets is realized by the transmission of Task objects and, more exactly, their XML representation.
The structure of a Task XML document is described in figure 5.12.
Depending on its purpose, each Task can contain another management object such as Action, Rule or Monitor, and one Target Mask object, which describes the receiver. In case of the previous Task document sample, this Task is used to deploy the Monitor `"MonitorName=CMOSStatus"` to a client `"DeviceID=10.162.128.231"`.
Note that this is the Monitor described in figure 5.9 with `"ElementName=CMOSStatus_CHARM_SAM"`.
According to their purpose, two different types of Tasks have been developed, the Configuration Task and the Administrative Tasks. Table 5.6 shows a partial exemplary Task selection of both types. All developed Tasks can be found in the diagram of figure 5.37.
The Configuration Tasks will be used in order to distribute the other management objects, such as Monitors and Rules to the SysMES clients, and also for changing their configuration. The Administrative Task will be used in order to execute general actions and can be divided into Static Tasks and Dynamic Tasks.
Each type of Task has the attributes visualized in figure 5.12. When deploying a management object on the client side, only the TaskID and Acknowledge attributes are relevant. The meaning of the other attributes is the subject of section 5.4.2.5.
TaskID is a unique Task identifier which will be set on the server. The assignment of TaskIDs to subsequent Tasks increases strictly monotonically. After the successful execution of a Task, the SysMES

```
<?xml version="1.0" encoding="utf-8"?>
<!doctype Tasks>
<Tasks>
    <Task>
      <Acknowledge>1</Acknowledge>
      <TaskID>1225803468941.10.162.13.102</TaskID>
      <Deployer>lara</Deployer>
      <TaskRepeat>2</TaskRepeat>
      <Expiry>20091018062122.207000+060</Expiry>
      <Type>2</Type>
      <TargetMask>
         <GroupID>3.321</GroupID>
         <DeviceID>10.162.128.231</DeviceID>
         <FirmID>6666</FirmID>
      </TargetMask>
      <ActiveMonitor>
         <Repeat>60</Repeat>
         <Run>true</Run>
         <Mode>1</Mode>
         <MonitorName>CMOSStatus</MonitorName>
         <ElementName>CMOSStatus_CHARM_SAM</ElementName>
         <BinaryAction>
            <Binary>"IyEvYmluL3NoCl ... Q0fScpCg=="</Binary>
            <ValueNames>CMOSBatteryStatus</ValueNames>
            <ValueNames>CMOSChecksumStatus</ValueNames>
                  ...
         </BinaryAction>
         <EventClass>
            <ValueNames>CMOSBatteryStatus</ValueNames>
            <Operators>eq</Operators>
            <Values>bad</Values>
            <Severity>3</Severity>
                  ...
         </EventClass>
         <EventClass>
            <ValueNames>CMOSCheckSumStatus</ValueNames>
            <Operators>eq</Operators>
            <Values>bad</Values>
            <Severity>3</Severity>
                  ...
         </EventClass>
      </ActiveMonitor>
   </Task>
</Tasks>
```

Figure 5.12.: SysMES Task XML Example

Configuration Tasks	
RBEM_ClientSetMonitor	Deploy the new Monitor and start it
RBEM_ClientDelMonitor	Remove the Monitor and all associated configuration
RBEM_ClientSetRule	Deploy the new Simple Rule
RBEM_ClientSetRule	Remove the specified Simple Rule
Administrative Tasks	
RBEM_ClientStartMonitor	Start the pre-deployed or expired Monitor
RBEM_ClientStopMonitor	Stop a Monitor before it expires
RBEM_ClientGetAllMonitors	Return the whole configuration and status of all Monitors
RBEM_ClientGetCache	Get Events contained in the client Event Cache

Table 5.6.: SysMES Client: Tasks

client stores the TaskID and reports it to the server, which is therefore able to recognize the configuration and execution state of the clients.
The Acknowledge attribute defines whether an Ack Event [4] has to be sent. This attribute is used to make a compromise between a very fast Task execution `"Acknowledge=0"` and a safer, but slower execution `"Acknowledge=1"`. For the SysMES client, a two-phase acknowledge algorithm has been developed. In the first phase, the client sends an Ack Event if the Task has been received successfully and in the second phase, the client sends an Ack Event to guarantee its successful execution. In case of errors during the execution of Tasks, the client sends an Error Event reporting the problem.
As described above, there are Configuration Tasks and Administrative Tasks. The first type is used in order to deploy and change the management objects. The second type, the Administrative Tasks, gives the possibility to execute administrative and operative actions. Tasks execution can be initiated automatically or manually by a component or an administrator. The execution results will be sent to the administrators using TaskReply Events.
The Simple Tasks are a flexible and dynamic way used by the operators to interact with the environment to be managed. The execution of a Simple Task implies the execution of any desired binary or command and consequently will be utilized for administrative and operative purposes. The Simple Tasks are used, for example, to start/stop the cluster nodes (reboot, shutdown), to get information about system resources (e.g by the execution of OS commands like `"df"`, `"ps"`, `"du"`, etc.) or to change the characteristics of the system resources (e.g. clean shared memory). These Tasks are associated - similar to Monitors - with a Binary Action (see figure 5.8) containing the binary code or the command to be executed, as well as required parameters. The SysMES client executes the Simple Task and sends TaskReply Events to the server containing the return value of the action. In case of problems during the Simple Task execution, the SysMES client stops this procedure. The same applies for a possible timeout, depending on the value of the timeout attribute specified in the Binary Action object. In both cases an Error Event is generated and sent to the server.

Summary of the section:

The SysMES client implements different kinds of Monitors, in particular, the active and passive ones according to the measurement strategy and the persistent and non-persistent Monitors according to the strategy for the calculation of Monitor results. The particular feature of Persistent Monitors is the capability to calculate a single result from a set of measurements and to forward it to the Event handling

[4] As a reminder an Ack Event is used for reporting a successful or failed Task execution.

facility, for further processing. The SysMES clients are able to decide if the Monitor results need to be handled or if these can be discarded. The decision is made according to the Conditions described in management objects called Event Classes and the values which fulfill these are the Events.
There are Monitoring Events, Administrative Events and Application Events. The Administrative Events do not contain Monitor data and are used to inform the server about the status and return information from the execution of Actions. The Application Events inform the server about the state of applications. The clients have an Injection Interface, which can be used to evade the system, which means that the trigger decision will be made outside of the client and the injected Events will be interpreted as Monitoring Events. The clients send the Events to the management server for further processing in the form of XML documents. These contain the measured and triggered value and more information about the originator such as Type, ClientVersion, CurrentClientTID, etc. Furthermore, the clients have also a Simple Rule System in order to decide whether the Events represent an undesirable or special status and to react, i.e. to contribute to the solution of the problem. Another important feature of the clients is the Task Management functionality, which is used for the active interaction with the targets. The Tasks are divided into Configuration Tasks, which are used to set and reconfigure the Monitors and Rules, and the Administrative Tasks to get information about the clients. The system administrator is equipped with other kinds of Tasks, the so-called Simple Tasks, which can be used to execute any binaries and commands and to return the execution result to the servers to be evaluated and displayed.
The most important Action type on the client side is the Binary Action, which contains the binary code and parameters to be executed. The SysMES framework offers the capabilities to change, configure and extend the semantics of the Binary Actions and to distribute these to the desired targets. The Binary Actions are used for monitoring as well as for the Rule Management and Task Management.
The SysMES client has been designed as a light-weight client due to the fact that the client has to run on cluster nodes as well as on devices with limited system resources. This light-weight design needs the relocation of some management features from the client side to the server side in order to reduce or minimize the additional management overhead. The implementation of the client will be explained in section 6.5

5.4. Management Layer

As described in the previous section5.3, the SysMES clients are capable of monitoring the system resources, to generate and save Events as well as to processes these in a restricted way. In this section, the SysMES server capabilities, i.e. the extended system management capabilities, are introduced. Similar to the clients, the Server Layer is involved in the Event processing and storage and also acts as an interface for the system management operators to interact with the managed objects (cluster nodes) using the SysMES framework.
In order to cover the all of the server functionalities, this section has been divided into the following three subsections, the Access Point, the Server Layer and the Operator Layer.

5.4.1. Access Point and Communication Algorithm

To keep the light-weight characteristics of the SysMES clients and to comply with the requirement concerning the minimal system resource utilization, a new component has been introduced. The SysMES Access Point is a communication element, which is located between the Client Layer and the Server Layer, and its primary purpose is to enable a transaction-based transmission of data between

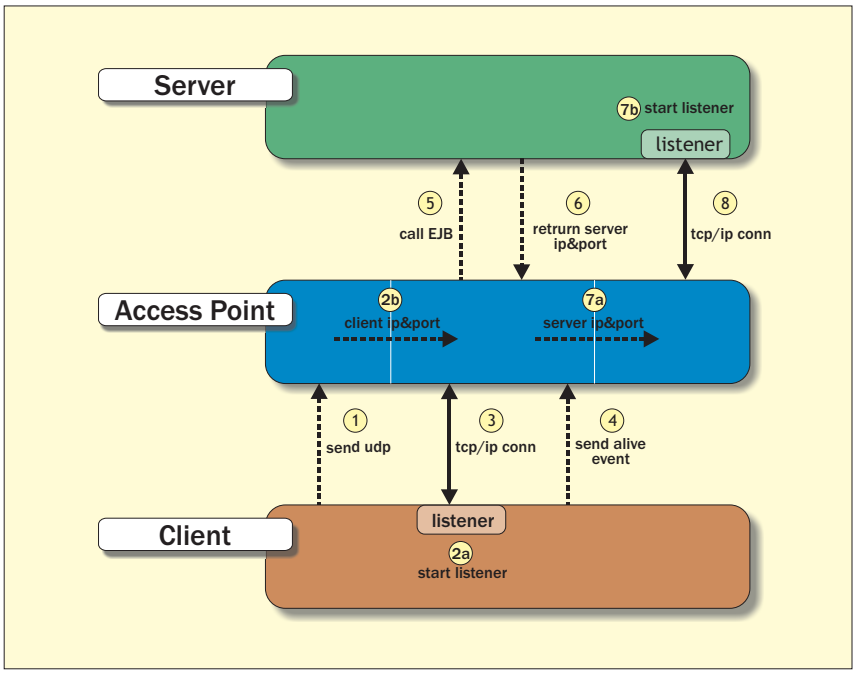


Figure 5.13.: SysMES Access Point

clients and servers using a load-balancing strategy.

In the SysMES framework, the client is responsible for establishing and managing connections conforming to the algorithm described in figure 5.13.

As a part of its startup routine, the client sends a User Datagram Protocol (UDP) broadcast ① to the Access Point containing its own address and a port where the client starts listening ②a. The Access Point parses this message ②b and establishes a Transmission Control Protocol (TCP) connection to the client to the specified port ③. In the next step, the client sends an Alive Event to the Access Point④.

The transmission of the Alive Event from Access Point to one arbitrary server is done using the communications capabilities of Java, i.e. Remote Method Invocation (RMI) in the EJB context. The Access Point calls a server bean which receives the Event ⑤ and returns the address of the server and a specific port ⑥ where the server-side listener process is started ⑦b. The Access Point receives and parses the returned data ⑦a and connects to the server ⑧.

The transmission of data between Access Point and client is also implemented in a transactional way. Clients cache Events until the Access Point confirms the successful invocation of the server bean. In the case of failure, the client tries several times to deliver the Event. In the case of further errors, the client cuts the connection and tries to find another Access Point.

This layer makes the clustering of several Access Point instances available. In case of an Access Point cluster, each instance receives the UDP broadcast packet and tries to connect to the client. The reaction time of each Access Point instance depends on its current load and therefore the instance with the lowest utilization will successfully connect to the client. Subsequently, the Access Point sends a request to the server cluster and an EJB instance on the server with the smallest load is returned automatically. Figure 5.14 visualizes a possible distribution of the connections when multiple access point and a server instances are clustered available.

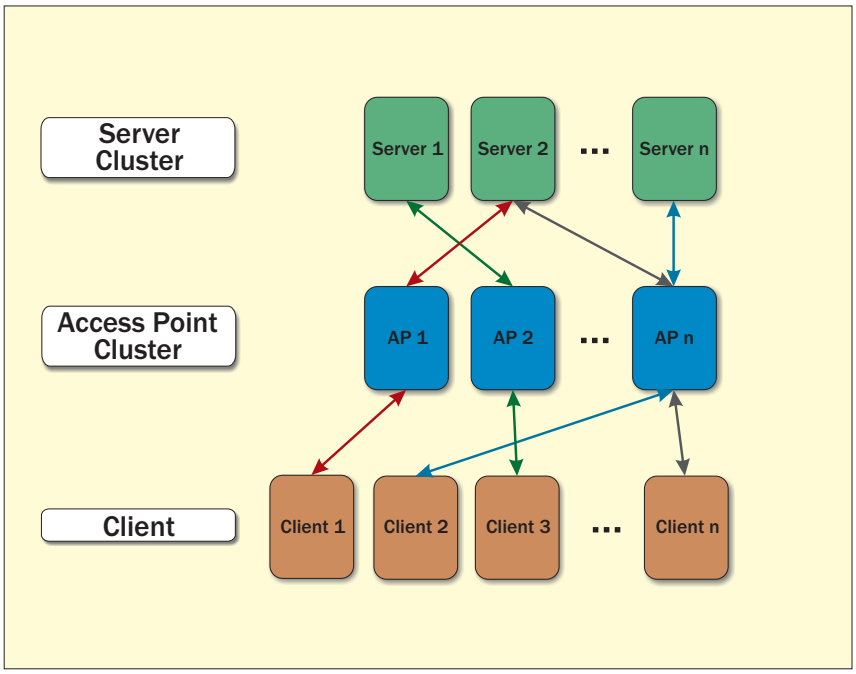


Figure 5.14.: SysMES Access Point - Connections Overview

The clustering capability is strictly necessary in a distributed and decentralized system management framework in order to react to failures, to distribute the connections as well as load dynamically and consequently, to become scalable.

5.4.2. Server Layer

The Server Layer is the main component of the SysMES framework. It is in charge of most of the system management activities, which involve interaction between managed objects and managers; it is also responsible for the analysis of the monitoring data and carries out the reactions.
According to the general design section 5.2, the SysMES physical architecture is based on a vertical and a horizontal distribution and the system management activities should be carried out as close as possible to the initiator. Therefore, the Server Layer is divided into two sub layers (see figure 5.3) with separated competencies. These layers are the Local Area Management (LAM) Layer and the Wide Area Management (WAM) Layer. The next sections introduce both server sublayers and following that, the complete server-side system management functionality.

5.4.2.1. Local Area Management (LAM) Layer

The first Server Layer is the LAM Layer. This layer is responsible for managing the bi-directional communication to the clients using the Access Points. Furthermore, it is involved in the gathering and processing of Events, as well as in the distribution of other management objects to the targets, such as Tasks.
Figure 5.15 shows the more common use cases related to the LAM.

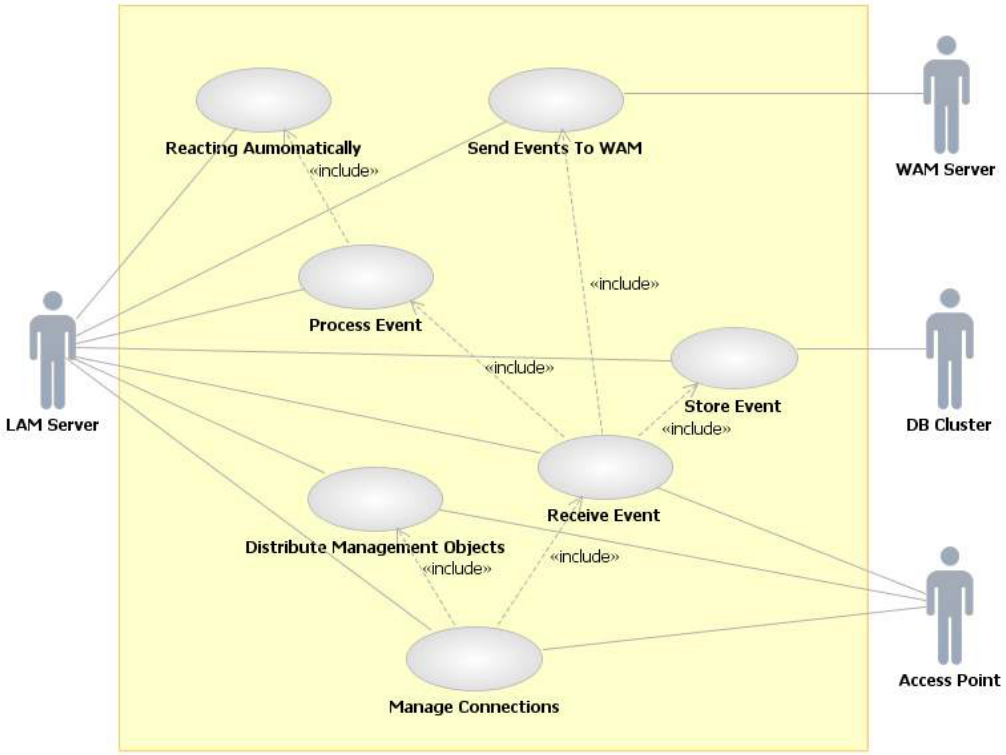


Figure 5.15.: System Management - LAM Server Side Use Case Diagram

- Management of Connections: Each LAM server has a connection interface, which allows the clients to build a one-to-one connection to an arbitrary server member of this layer using an arbitrary Access Point. The LAM is in charge of managing these connections in a manner that allows the localization of each client in order to deliver the specific management objects and to receive the monitoring data.

- Receiving and Storing of Events: According to the figure 5.15, the LAM is involved in the use cases "Receive Event" and "Storage Event". It receives and analyzes the incoming Events to detect which of these have to be saved persistently. This procedure reduces the amount of data in the databases and contributes making the server side scalable. As described on the section 5.3.2, the Event Class has an attribute SaveOnServer, which describes the storage strategy for each generated Event type. The LAM utilizes this information to initiate the storage procedure. More about Event receiving and storage can be found in section 5.4.2.3

- Processing and Forwarding Events: Similar to the clients, this layer is responsible for the evaluation of Events through the Simple Rule system. The evaluation consists of two steps, the first one is the evaluation of the Event to find out if it is necessary to react automatically and the second one is the process of deciding whether the Event has to be forwarded to the WAM layer to be evaluated by the Complex Rule system.

- Reacting Automatically: Granted that a server-side reaction for the Event is required, the LAM is in charge of the execution of the Actions, which contribute to the solution of the recognized

errors. Server-side reactions are especially required when the clients do not have the capabilities to perform these or are overloaded . For this purpose, it is equipped with the Task Management functionality to execute Actions on the different SysMES participants such as servers or clients.

- Distribution of Management Objects: Using the self-managed connections, the LAM is able to distribute the management objects to the clients. The distribution of management objects from servers to targets (note that targets can be SysMES servers and clients) is realized by the means of a management object called Task. Depending on the purpose, a Task object can be used in order to deploy Monitors or Rules and also to execute arbitrary Actions on targets.

5.4.2.2. Wide Area Management (WAM) Layer

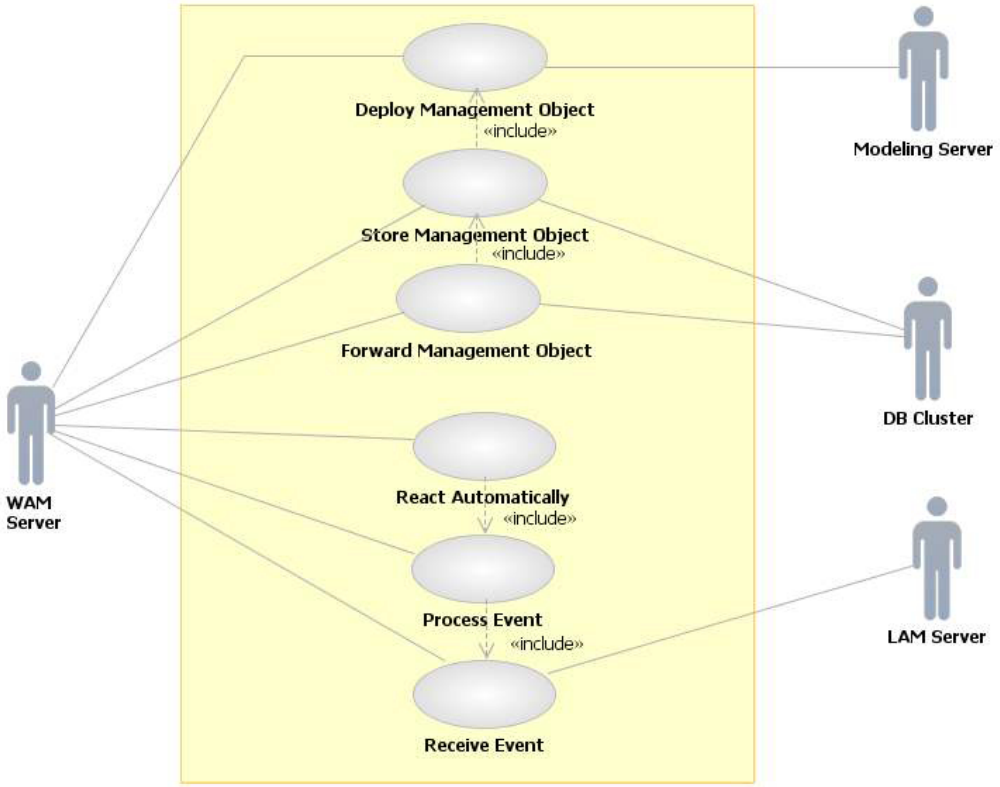


Figure 5.16.: System Management - WAM Server Side Use Case Diagram

According to the vertical distribution, the WAM builds the top of the Server Layer and it is the interface to the Operator Layer. The use case diagram shown in figure 5.16 presents a set of common activities attached to the WAM Layer, as well as the shared activities with the Modeling Layer and LAM Layer.

- Processing of Events: The WAM layer is able to accumulate Events from different Monitors and clients and to correlate these in order to detect a complex or global state. For this purpose, the WAM has a Complex Rule system, which processes the single Events and decides whether these need to be taken into account for the recognition of a complex state. The Complex Rule system builds the last part for the processing of Events in the SysMES framework and it is

able to react automatically, similar to the other processing capabilities such as the Simple Rule system in the LAM side. Detailed information about the Rule Management system can be found in 5.4.2.4

- Deployment of the Management Objects: The WAM is the interface between the Modeling Layer - as a part of the Operator Layer - and the Server Layer, and therefore it is involved in the deployment of management objects(i.e. Task deployment). It receives the XML representation of the Tasks and builds different SysMES specific management objects. The deployment of Tasks is a part of the automatic reaction routine of the Rule Management system, as well as of the manual interaction with the targets.

- Storage and Distribution of the Management Objects: After the deployment of the Tasks, the WAM layer is in charge of their persistent storage on databases and also of propagating the necessary information to the LAM in order to send the Tasks to the targets.

- Interfacing the Managers: As described at the beginning of this chapter (see 5.1) a Manager, can be, but is not necessarily a person, so that the WAM Layer defines an interface to other parts of the system management framework or external tools which utilize this in order to react manually or automatically. This interface is used by the system administrators and operators and it allows them to react manually, according to their management activities using a predetermined set of Actions.

The distinction of the Server Layer in LAM and WAM was made in order to achieve the requirements concerning scalability and dependability, i.e. those layers should be located on separate physical servers. However, the configuration of the SysMES Server Layer allows the placement of WAM and LAM Layer on only one physical server. The following sections describe the functionalities of the Server Layer and the positioning of these in the LAM and WAM Layer in detail.

5.4.2.3. Event Management

The Event handling section of the clients (see section 5.3.2) overviews shortly how these are created and sent to the servers. This section discusses the server functionality concerning the management of Events. The section is divided into three different parts. It begins with the explanation of the purposes of Events and Event Management, followed by a description of the characteristics of the SysMES Event Management and finally, it explains the Event Management functionality.

Purpose: The SysMES Events are used for the transmission of the pre-processed Monitor data to the SysMES framework. They are used for the client/server communication, as well as for the communication between servers. Another usage of Events is reporting of errors, i.e. client or server implementation errors or errors while executing Actions.
Events provide the information needed by a Processing Unit in order to make the decision whether it is required to react. As a reminder, the Processing Unit can be a system administrator who uses a GUI where the Events are displayed and reacts manually, or a Rule system, which processes and reacts automatically. The Event Management subsystem of the SysMES framework is in charge of the persistent storage of the Events and delivering these to the different Processing Units. The storage of Events is used for statistical purposes derived from the long-term observation, such as the recognition of hardware errors.

Characteristics: The SysMES Event Management features the following properties:

- Decentralized Event Management: The SysMES Event Management capability is designed decentrally and therefore appropriate for the management of both a huge number of targets (i.e. embedded systems or cluster nodes) and a huge number of Events. The decentralization is the key property to fulfill the requirement of scalability and dependability. The Event Management functionality is distributed over several LAM instances, which are clustered and can replace each other. The consequences of a decentralized Event management are a better load distribution in the management environment, as well as the high availability of the system Management services.
- Location Independence: Event Management is performed by any LAM server independent of its location. The relevant aspects for deciding where in the LAM Layer Events are processed are the amount of load of the servers and their availability. This decision is made by the clustering and load balancing capabilities of the application server, which hosts the SysMES framework. As introduced in section 4.5, the SysMES framework is based on JBoss AS and utilizes its offered load balancing strategies. Another reason for a location independent Event Management is the capability to react to a software or hardware server failure, because any other server is able to substitute for it.
- Dynamic and Flexible Extensibility: There are two other cases which have to be taken into account. The first case is when a server fails and the remaining servers are not able to process the load generated by the incoming Events. The second case concerns the reduction of the number of SysMES servers in order to save resources when the servers are not working at full capacity. In both cases, it is important to react according to the need for Event Management resources, i.e. increasing or reducing the number of server instances dynamically. These procedures have to be carried out without stopping the SysMES Event Management capabilities.
- Transactional Event Management: Events are used for information transfer. The transmission of this information through the different participants of the SysMES framework is carried out using transactions, which ensure that the Events are delivered to the desired destinations. Another usage of transactions is for the storage of Events in the database.

Functionality: The SysMES Event Management component in general is in charge of the reception of Events and to make these available to the other management components, such as the Operator Layer or the Rule Management component. This section discusses the functionalities needed to achieve the purposes described above. Before introducing these functionalities, the term "Event" needs to be explained and classified.

Event Definition and Classification: On the client side, the term Event was introduced as *...monitoring data, which represent an undesirable state, an error or a special state, the so-called Monitoring Events...* (see section 5.3.2). At that time, it was important to pay attention to the generation of Events from the Monitor results. Now on the server side, the definition of Event has to be extended in order to include other aspects.
An Event represents data originating from a SysMES target (such as clients running on cluster nodes or servers) or a component (such as the Rule Management component). It is used to inform a destination about a state or something observed and recognized at the originator side. An Event destination can be any SysMES server (LAM or WAM), the database or the GUI. The data contained in an Event

can be monitoring data, states or errors to be reported, as well as a processing result to be taken into account in further processing.

```
<?xml version="1.0" encoding="utf-8"?>
<!doctype ClientMessage>
<ClientMessage>
   <CurrentClientTID>1220856000829.10.162.15.226</CurrentClientTID>
   <DeviceInfo>
      <ManagedEntityID>
         <DeviceID>10.162.128.231</DeviceID>
         <FirmID>6666</FirmID>
         <Hostname>feptpcao10-charm</Hostname>
      </ManagedEntityID>
      <Type>x86_64</Type>
      <System>Linux</System>
      <ClientVersion>3.17</ClientVersion>
   </DeviceInfo>
   <Events>
      <Event>
         <Severity>1</Severity>
         <MonitorName>CMOSStatus</MonitorName>
         <EventName>CMOSStatus_eq_bad</EventName>
         <Count>3</Count>
         <Status>open</Status>
         <CheckRule>true</CheckRule>
         <SaveOnServer>true</SaveOnServer>
         <FirstOccurrence>1216640860285</FirstOccurrence>
         <LastOccurrence>1216640975326</LastOccurrence>
         <EventID>1216640975326.10.162.128.231</EventID>
         <Info>
            <AttrName>CMOSCheckSumStatus<attrname/>
            <AttrName>CMOSBatteryStatus<attrname/>
            <Value>bad</value>
            <Value>bad</value>
            <Unit>string</unit>
            <Unit>string</unit>
        </Info>
        <ProcessedRule></ProcessedRule>
        <Expiry>121000</Expiry>
      </Event>
   </Events>
</ClientMessage>
```

Figure 5.17.: XML Event Document

Events arrive to the LAM server in the form of XML documents. Figure 5.17 shows a client message document which contains an Event sample originating from the target `"feptpcao10-charm"` running the Monitor `"CMOSStatus"` (see figure 5.9). This Event reports a faulty state of the CMOS composed of two values, the CMOS checksum status and the CMOS battery status. This kind of client message

can contain more than one Event, but the Event Management subsystem considers an Event as the smallest entity to be treated.
Therefore, the client messages are parsed on the LAM server, which generates single Event objects. These include data divided into three parts, the first part contains information about the originator, the second part about the transmitted data and the third part is the CurrentClientTID (i.e. Current Client Task Identifier) of the originator.
In the following is the description of the Event attributes used in all three parts. Furthermore, the table 5.7 shows their ranges, data type and default values.

- CurrentClientTID: This attribute is set on the target Events in order to inform the server which SysMES Task have been executed successfully. The detailed usage of this information is subject of the section 5.4.2.5.
- Target Information:
  - DeviceID: This identifier is used for the identification of the target, which generated the Event.
  - Hostname: The target name used for its network identification.
  - FirmID: The SysMES framework is able to manage targets with different affiliations (see section 5.2). For this purpose, each target must have a unique identifier composed of the DeviceID and the FirmID.
  - Type: The hardware type of the client, i.e. x86_64, armv4l.
  - System: Operating system release.
  - ClientVersion: The current installed SysMES client version.
- Event Information:
  - EventID: A unique identifier for the Events.
  - MonitorName: The name of the Monitor which is used to read out the resource information.
  - EventName: This attribute provides information about the Event Class, which has been used to trigger the generation of an Event. Note that Monitors are able to return multiple values and therefore it is possible to generate more than one Event using one Monitor result and a set of Event Classes.
  - Severity: This attribute describes the importance or urgency of the Events (introduced in section 5.3.2). Currently the SysMES framework utilizes the Severities 1-9.
  - Count: The number of occurrences of the same Event. This is important to recognize how many Events were cached on the targets during a offline phase and also how many Events have been taken into account by the Event Cache compression algorithm of section 5.3.2.
  - Status: Events have different status depending on the current location. At the client cache, Events can have the status `"open"`, `"cached"` and `"sent"`. On the server, Events can have the status `"open"`, `"in work"` and `"closed"`. The status attribute provides information about the momentary Event processing status.
  - Info: The info part contains the information about all measurements including their names, values and units

    - AttrName: The name of the system resource attribute read out by the Monitor.
    - Value: The measured value or the information to be delivered.
    - Unit: the measuring unit. At the moment the used units are C (centigrade), perc (percent), Byte, KB (kilo byte), MB (mega byte), RPM (revolutions per minute).
- CheckRule: This attribute informs the server about the necessity of further processing by the Rule system. This attribute reflects the configuration of the Event Class of Monitors so that it is possible to set it up individually in each Event Class. This is a part of the load reduction and reallocation strategy and consequently the scalability strategy.
- SaveOnServer: It reflects also a configuration of the Event Classes and is used on the server to define whether the Event has to be stored.
- FirstOccurrence: The time when the first of multiple equal Event was generated.
- LastOccurrence: The generation time of the last equal Event. Both time attributes and Count are used to recognize important information about the Events which have been compressed.
- ArrivalTime: It is a timestamp made on the server side immediately before finishing Event processing.
- Expiry: This attribute is used on the server and GUI to recognize if the Event is still valid. If the Expiry attribute value is missing, the client sets it up with the value of the Period attribute from Monitor.
- ProcessedRule: This attribute contains a list of Client Simple Rules, which have been executed due to the occurrence of this specific Event.

Table 5.7 visualizes the usage of the data type Long Token Sequence for the definition of several identifiers. This data type is defined as a vector of long and its textual representation is given as a dot-separated long value sequence i.e. "LongValue.LongValue. ... .LongValue". The first LongValue is a timestamp which describes the generation date and time. After that follows the value of the DeviceID attribute (which is also from type Long Token Sequence) where the Event has been generated. For example `"EventID=1233684185345.10.162.128.231"`, which means that this Event object was created at `"3 FEB 2009 19:03:05.345"` on the client with `"DeviceID=10.162.128.231"`.
In the description of the EventName attribute, the possibility is mentioned that a Monitor measurement can return more than one value. Consequently, it is possible to define a set of Event Classes which trigger the generation of multiple Events. All these Events have the same MonitorName and different EventName. On the other side, each Event can contain multiple values. One of these values is represented by a triple "[AttrName, Value, Unit]". On the server side, this information is stored in vectors so that in general each triple is built as follows:

$$[AttrName(i), Value(i), Unit(i)]$$

and conforming to the example of figure 5.17

$$[CMOSCheckSumStatus, bad, string], [CMOSBatteryStatus, bad, string]$$

The SysMES framework distinguishes between three types of Events. These are:

Event Attribute	Type	Default	Mandatory
Current Task ID	Long Token Sequence	0 (new installed target)	yes
Device ID	Long Token Sequence	ID of the target	yes
Host Name	String	host name	no
Affiliation ID	Long Token Sequence	-	yes
Type	String	Node Hardware Type	no
System	String	OS release	no
Client Version	String	-	no
Event ID	Long Token Sequence	<time>.<Device ID>	yes
Monitor Name	String	-	yes
Event Name	String	-	yes
Severity	Integer	-	yes
Count	Long	1	yes
Status	String	open	yes
Info	String[]	-	yes
Attribute Name	String[]	-	yes
Value	String	-	yes
Unit	String[]	-	yes
Check Rule	Boolean	true	yes
Save On Server	Boolean	true	yes
First Occurrence	Long	-	no
Last Occurrence	Long	-	no
Expiry	Long	Value of the Monitor Period attribute	no
Processed Rule	String	Name of executed Simple Rules	no

Table 5.7.: Event Attibutes

- Monitoring Events: This type was introduced in section 5.3.2 as Monitor results, which represent a special state or an error. This definition is extended on the server because Monitoring Events can also be generated on the server. This occurs for a similar purpose in order to report errors, failures or special states, as well as for the escalation and correlation of Events. More about this can be found in the following functionality part of this section. As a reminder, Monitoring Events are divided in four categories according to their severity and therefore their urgency and importance. These are Immediate Event `"Severity=1"`, Critical Event `"Severity=2"`, Service Event `"Severity=3"` and Information Event `"Severity=4"`.

- Application Events: These contain information about the state of external applications. This information is processed in the exact same way as the Monitoring Events and therefore it is possible to generate these Events using the Severities. Furthermore, it is possible to use the Log Event `"Severity=5"` for this purpose.

- Administrative Events: These Events are used for reporting of states inside the SysMES framework. There are four different types:

  – TaskReply Event `"Severity=6"`: The SysMES client utilizes this kind of Event to transmit to the server the result and return value of the Task execution. More information about this can be found in section 5.4.2.5

- Error Event "Severity=7": These Events are used in order to report errors and failures in the framework, as well as during the execution of Actions, i.e. to report a forced interruption after a time out.
- Alive Event "Severity=8": This kind of Event is used to test the correct functioning of the clients. Each client sends an Alive Event in a time interval defined in its configuration file; at the moment all clients are configured with a default value of "120000 milliseconds".
- Acknowledge Event "Severity=9": These Events are used for the acknowledgment algorithm used for the distribution and execution ofTasks to the clients. This algorithm is subject of the next section 5.4.2.5

Event Management Algorithm: After the Event definition, description and classification follows the introduction of the Event Management algorithms.
The sequence diagram of figure 5.18 visualizes several actuators and processing steps related to receiving, storing and processing of Events, these are:

- Event Receiving and Syntax Check: The SysMES server receives XML documents containing the Events. The server is in charge of checking the syntax of the documents and of parsing it in order to extract the Events. In case of syntactic or content errors, the document is discarded. Depending on some aspects, such as the utilization of targets with enough system resources as well as public networks, it is possible to use technologies for XML authentication and encryption. In this case, the server checks the credentials of the sender and decodes the information coded in the XML document.
- Event Creation: After the parsing of the XML documents, follows the Event object creation. Each "<Event> . . . </Event>" part of the XML document is used in order to generate an Event object. Such an object is composed of the Event data, the information about the Event originator and also its CurrentClientTID. From this point on there are only Event objects and the XML documents are discarded in order to avoid multiple parsing.
- Attribute Evaluation: The SysMES server evaluates some Event attributes in order to define the individual store and processing strategy. These are SaveOnServer to recognize whether the Events should be stored and where. The storage location depends on the value of the attribute Severity. Other important attributes are CheckRule, which describes the exigence to forward the Event to the Rule Management subsystem, and CurrentClientTID, which informs the Task Management subsystem about the execution state of the Event originator. More about the Rule Management subsystem follows in section 5.4.2.5.
- Event Storing: The SysMES server stores Events permanently in the database cluster. The database schema is designed for the separated storage of the Events in several tables. Each SysMES LAM has a database instance where the Events and all other relevant information are stored. The SysMES framework requires a clustered database infrastructure for the storage of Events (and also for the other management objects such as Rules and Tasks). This is due to the high requirements regarding scalability and dependability. The Events from the database are displayed on the GUI where the system administrator or operator interacts with them. These Events are used as a possibility to inform the system administrator and operator and to give them the possibility to react manually, as well as to change Event status.

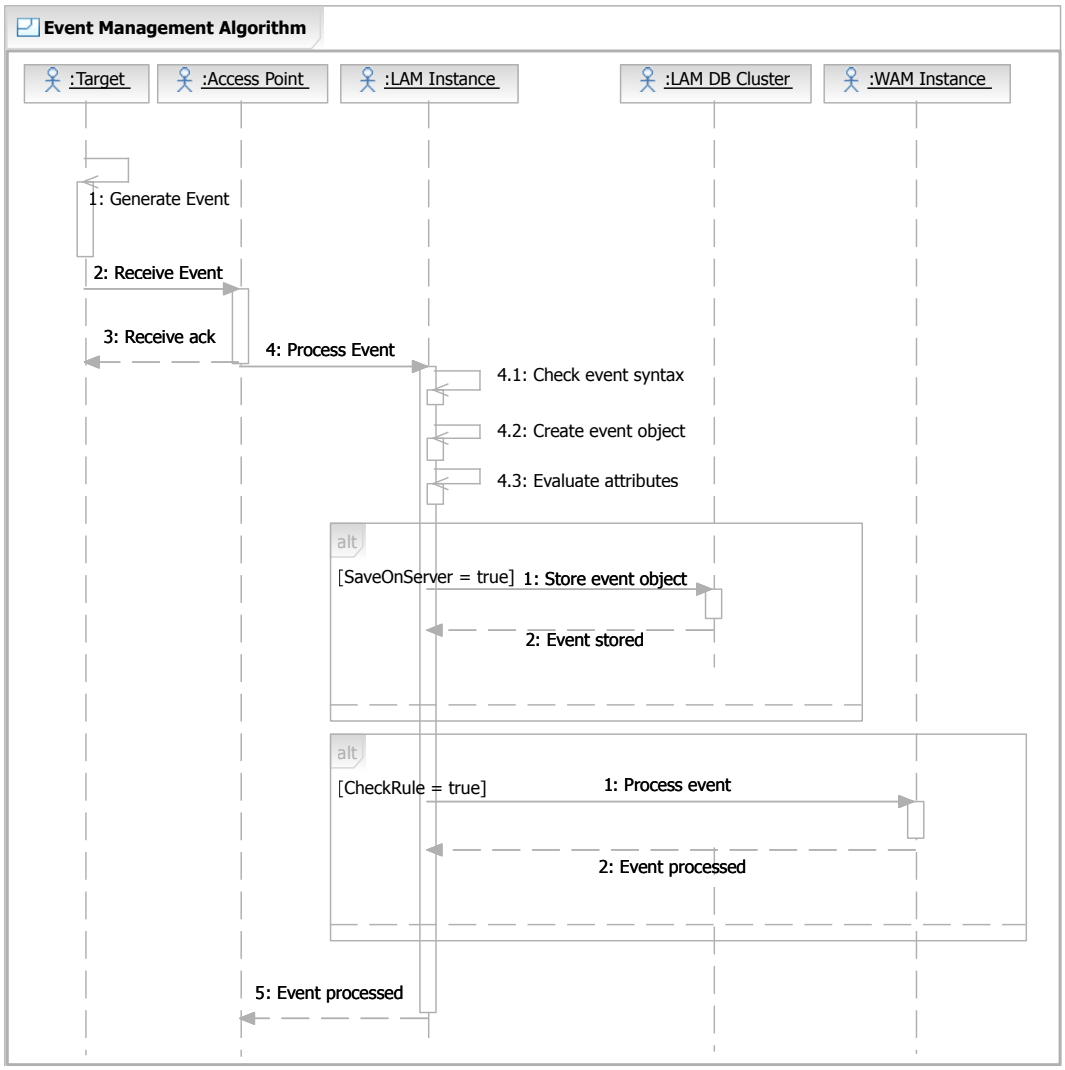


Figure 5.18.: Event Management Algorithm

- Information Forwarding: The decision about Event forwarding is made according to the value of the CheckRule attribute. In case of `"CheckRule=true"`, the Event Management subsystem sends the Event to the Rule Management subsystem in order to be analyzed and processed. The importance of this method is the reduction of the Rule Management complexity because only relevant Events will be checked.

  Other information to be forwarded concerns the value of the attribute CurrentClientTID. This value is sent to the Task Management subsystem, which utilizes it to recognize if there are pending Tasks to be executed on the targets. The detailed description of this functionality is the subject of section 5.4.2.5.

Event management clustering: The clustering of the Event Management functionality is done using the offered services from the JBoss AS introduced in section 4.5. In principle, the whole functionality is divided into three parts for:

1. Receiving of Events/interfacing the Access Point.
2. Event document parsing, Event building and forwarding.
3. Storage of Events.

The LAM server is composed of a set of LAM instances and each of these have the three functional parts described above. The clustering of the Event Management functionality means that each of these three single parts can be executed in any server. The execution of each part is a transaction and in case of a rollback, the initiator delegates the execution to another participant of the cluster. The decision where the execution location will be made by the JBoss container and only based on its load balancing strategy. Another conceivable scenario can be the clustering of the receiving capability and the local execution of the other two parts. In this case, the complete functionality is in a transaction and in case of a rollback, the Access Point delegates the Event to another LAM in the cluster.

Summary of the section:

XML documents containing one or multiple Events are sent by the SysMES targets and arrive at one server of the LAM cluster through an Access Point. The LAM server checks the syntax of the XML document, parses it and generates Event objects. These objects are composed of the Event data, information about the initiator and its CurrentClientTID. The Event Management subsystem utilizes the value of some attributes to decide if the Event has to be stored and has to be forwarded to the Rule Management subsystem. Another important issue is to update the Task Management subsystem with the CurrentClientTID in order to decide if the target has to execute other deployed Tasks. Events are stored in a database cluster and are displayed in the GUI in order to provide the system administrators and operators with facts about the state of the cluster. These facts are the input for the administrators and operators to decide if they have to intervene manually. The life cycle of an Event begins as XML document for transport, after that follows the Event object for internal processing in the SysMES framework and ends with database records where they are stored permanently. Events are the input of the Rule Management subsystem, whose principal function is processing the information contained in these. The last statement of this summary is that the Event Management component on the LAM layer was developed based on a complete location independence strategy.

5.4.2.4. Management of Rules and Reactions

The previous section described the server functionality for receiving and storing Events, as well as forwarding these to other SysMES subsystems. In section 5.4.2.3, it was mentioned that there are two kinds of Event Processing Units, a system administrator/operator and the Rule Management subsystem.
This section discusses one of these Processing Units, the Rule Management. Similar to previous sections, it is divided into three parts describing its purpose, characteristics and functionality.

Purpose: The general purpose of this subsystem is to offer an automatic capability to process the data contained in Events in order to recognize a specific state and to react by executing a desired action. Based on Events, the Rule Management subsystem is in charge of recognizing simple or complex states. While a simple state is described by one Event, a Complex State is defined by the occurrence and correlation of multiple Events. Another aim of the Rule Management is the execution of Actions as a reaction when a specific defined state occurs. There are two types of Actions, divided in those that contribute actively to the solution of an error state and those that are used for an administrative

purpose, i.e. informing administrative staff or forwarding Events to another Processing Unit, etc. The management of reactions is carried out automatically on the server side and the Actions are executed on the SysMES targets regardless of the fact whether those are servers or clients.

Characteristics: The SysMES Rule Management subsystem has the following characteristics.

- Multi-layered Rule Management: The SysMES Rule Management has been designed as a three-layer system. The first layer concerns the Client Simple Rule Management from section 5.3.3, the second layer is the Server Simple Rule Management located in the LAM and finally the Complex Rule Management in the WAM. The difference between the layers is related to the Event evaluation and reaction capabilities but, also the required reaction time. The Client Simple Rule Management is only able to evaluate one Event and to react by executing one action, the Server Simple Rule Management on the LAM side is also able to process one Event, but is able to execute a sequence of Actions, the Complex Rules Management is able to process multiple Events and also to execute multiple Actions. The reaction time depends on several factors, such as the transmission overhead, the number of Rules and also the number of needed Events to make a decision. In addition, the reaction time increases with the increased functionalities provided by the Management Layer. Therefore, clients are able to react faster than the LAM and consequently, also than the WAM, but only to very simple states.

  Two other reasons for the multi-layered design are the principle to offer the management capabilities as close as possible to the originator and the aim to build a scalable framework. The location of the Rules depends on the required functionality. The smaller the required management functionality, the closer it is located to the clients where the Events originate. That applies also for Events generated on the server side. A high scalability of the SysMES framework and especially of the Rule Management subsystem is achieved by the distribution of the Rules on the three layers. A distribution of the workload is the consequence of the distribution of the Rules.

- Dynamic Rule Management: Based on the introduced definition of scalability (see section 4.3), the relocation of management resources is one of the key points for a management system to become scalable. The Rule Management system is designed according to this requirement. It allows the relocation of Rules through several Management Layers when parts of the management framework are overloaded. Another aspect of dynamic Rule Management concerns the capability to change the characteristics of Rules dynamically. These changes concern the settings of the Trigger objects and also the attached Actions. Rule changes are deployed independently of the layer where they have to be hosted without downtime.

- High Availability: The Complex Rule Management algorithm on the WAM layer has been designed for the evaluation of state-full Rules. These Rules are able to process a set of Events and to store intermediate or provisional results until the completed Rule is fulfilled. In all other parts of the SysMES framework, high availability is a desired characteristic because of the uninterrupted deliverance of management services. For the Rule Management system, it is more important to ensure no states or processing results are lost in case of crashes. Therefore, the Complex Rule Management functionality has been designed as a master-slave architecture with replicated and synchronized data and processing capabilities.

- Object-Oriented Rule Representation: As described in the design consideration in chapter 4, the management environment is modeled in a object-oriented way. The definition of Rules follows

this principle so that all kinds of Rules are a set of objects and the associations between these. The syntax of the Rules are defined in the class diagrams and their semantics is defined by the interpretation of the object attribute values by the Rule evaluation algorithm. The specific representation of Rules is subject of the sections concerning Simple Rules (section 5.4.2.4.1) and Complex Rules (section 5.4.2.4.2).

Functionality: In section 5.3.3 the term Rules has been introduced as *"... If-Then clauses, which describe on the left side (LS) a system status and on the right side (RS) Actions to be executed when the left side - the Triggers - are fulfilled"*. This definition of a Rule is suitable for all the three Rule Management Layers, which differ in the complexity and number of involved Triggers and Actions in the Rule. Figure 5.19 shows the two different types of Rules located on the server side, the Server Simple Rule and the Complex Rule. The following two sections discuss the Simple Rule and Complex Rule functionality, as well as the method for the Rule representation and the corresponding detection algorithms.

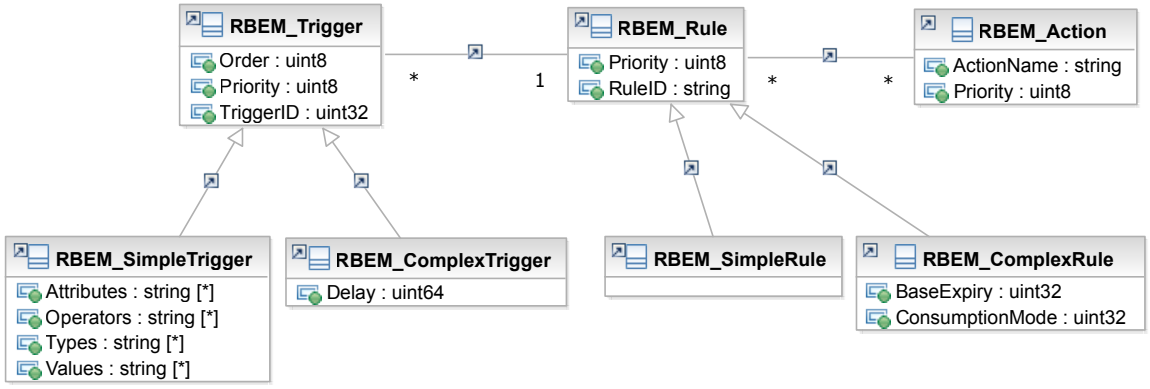


Figure 5.19.: SysMES - Server Rule Classes

5.4.2.4.1. Simple Rule Management The server Simple Rule functionality is located on the LAM. Server Simple Rules are responsible for the processing of Events if one of the upcoming situations occurs:

- Overloaded Target: In order to avoid target overload, a distributed Event storing and processing strategy has been developed. Storing of Events can be realized on the targets or remotely in the servers. The specific location is determined by the Monitors' configuration and can be changed dynamically at any time. Similar to the Event storage, it is possible to process the Events using the same Rules on the client or server side. The SysMES framework offers the capability to re-distribute the Rules to the servers in the LAM layer and consequently, to reduce the load on the targets.

- Lack of Target Functionality: The SysMES framework and especially its client is developed for the management of cluster nodes, but also for the administration of embedded systems with limited functionality and system resources. In some cases, these targets are not able to execute Actions which contribute to resolving or reporting problems, and therefore these Actions have

to be executed on the server side. The LAM server possesses the capabilities to execute almost any Actions, as well as to report the occurrence of Events. An example for this is the capability to send emails or SMS to one or multiple recipients.

- Events Routing Requirement: One of the most important usages of the Server Simple Rules is related to the forwarding of Events to the Complex Rule component in the WAM. The LAM servers are responsible for sending the Events to the specific instances of the WAM Layer where several Rules are located. For this purpose special Rules have been developed.

Similar to the Client Simple Rules introduced in section 5.3.3, this type of Rule is limited in the number of Events which can be processed. It is only possible to process one Event at a time. However, the most significant difference to the Client Simple Rule concerns the number and complexity of Actions to be executed. Multiple Actions can be attached to the Server Simple Rules and executed depending on their priority. The following is the description of the Server Simple Rule representation and its evaluation algorithm.

Simple Rule Representation: The Server Simple Rule representation is object-oriented. As described in figure 5.19 the Simple Rules are derived from the Rule class and associated with a set of Triggers and Actions. The formal definition of Simple Rules has been introduced in section 5.3.3 and therefore the focus of this section is set on the representation and characteristics of the right side, the Actions.
As mentioned above, the right side of the Server Simple Rules is composed of a set of Action objects to be executed on the LAM. The execution of more than one Action on the server side allows the definition of more complex reaction strategies. These can be defined as a composition of Actions which are used for reporting the Events and Actions which contribute actively to the solution of a problem.
Server Actions are always performed on the server side, but the effect of their execution can affect both the server and the client side and also external components, such as databases. For example, the execution of a Task Action on a server causes the generation of a Task object which is sent to a SysMES target.
Figure 5.20 visualizes the Action inheritance hierarchy. These Actions are used for the Simple Rule in the LAM Layer, as well as for the Complex Rules in the WAM Layer.
On the top of the hierarchy is the class Action which possesses the attributes ActionName and Priority. The ActionName is an identifier for the Action object and the Priority defines the execution order if more than one Action is attached. The next inheritance level is the class ServerActiveAction, which has the attributes DistinctAttr, ExecutionCount and ReEnableTime. These attributes are used in the Complex Rules and therefore will be introduced in section 5.4.2.4.2. The following Action classes inherit the previous attributes and extend their syntax and semantics by the definition of new attributes.

- SendSMS Action: This is used in order to inform a set of recipients via Short Message Service (SMS) about the occurrence of a specific Event. The information contained in the SMS corresponds to the Event originator, the Monitor and the measured value. Each object of this specific class contains an attribute Recipients to define the locations where the SMS is to be sent and an attribute Subject to give a hint about the importance or content of the message.

- SendMail Action: This is also used to send the data contained in the Event email. Similar to the previous one, it is possible to set the Recipients and Subject attributes in each SendMailAction object.

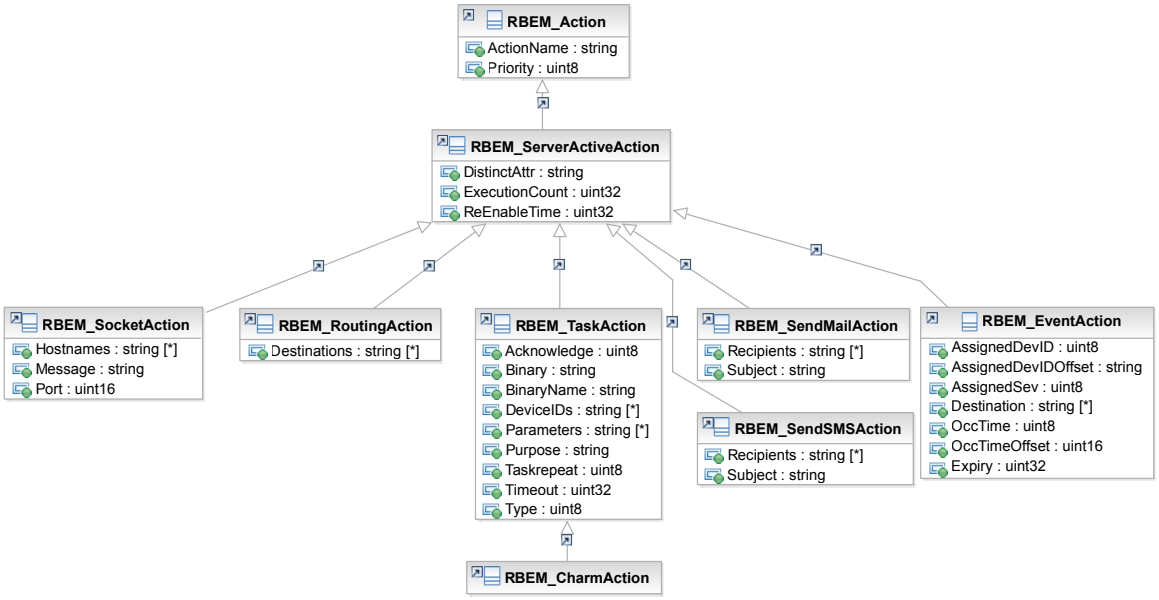


Figure 5.20.: SysMES - Server Action Classes

- Task Action: Objects of this class are used to execute binaries or commands on the clients automatically. The server utilizes the value of the object attributes to generate a Simple Task. In section 5.3.4 it was described that a Simple Task is associated to a Binary Action and to a Target Mask and therefore all attributes required for the object generation are included in the Task Action class.

  For the generation of the Simple Task object, the attributes used are: Acknowledge, Binary, BinaryName, Parameters, Purpose, TaskRepeat, Timeout and Type. For the generation of the Target Mask the attribute DeviceIDs is necessary. The detailed description of the Simple Tasks attributes follows in the section Task Management 5.4.2.5.

- Charm Action: This is a special kind of Task Action used for the creation and execution of Simple Tasks on the computer node where the CHARM card is embedded.

- Routing Action: This is used to redirect the incoming Events to specific locations using Java Message Service (JMS). The Simple Rule Management subsystem is responsible for forwarding the Events to other desired parts of the management framework, i.e. to be displayed or for further processing. The definition of the Event receivers is done by setting the attribute Destinations.

- Socket Action: This class is used for sending the Events to external applications using a TCP/IP connection. The connection settings are defined in the attributes Hostnames, Message and Port.

- Event Action: Objects of this class are used for generating a new Event and sending it to an arbitrary or a specific location. It is mainly used for the Complex Rules and therefore its specific explanation follows in further sections.

The object diagram of figure 5.21 visualizes a real-life example of a Server Simple Rule. In this

Figure 5.21.: SysMES - Server Simple Rule Sample

example, the Rule is activated by an Event, which has been generated by the CMOS Status Monitor (see 5.9).
In principle this Rule is responsible for reporting the recognized state `"CMOSCheckSumError=bad & CMOSBatteryStatusError=bad"` to a system administrator and also for redirecting the Event to another part of the SysMES framework. This other component is the Complex Rule subsystem. In order to simplify matters the following alternative non-formal Simple Rule representation is used to describe the object-oriented Simple Rule representation.

```
IF  (OR(AND(C1 ... Cn), AND(C1 ... Cn), ... AND(C1 ... Cn)))
THEN A1 ... An
```

In the case of the Server Simple Rule for the CMOS status, this representation looks like the following:

```
IF (OR(AND(Info.AttrName = CMOSCheckSumStatus, Info.Value = bad),
       AND(Info.AttrName = CMOSBatteryStatus, Info.Value = bad))
THEN CMOSStatus_Routing_A, CMOSStatus_SendSMS_A
```

Simple Rule Evaluation: After the introduction of both the object-oriented and the non-formal Simple Rule representation follows the internal algorithm for the evaluation of the Simple Rules.
The formal evaluation of Triggers, and consequently, of Simple Rules has been introduced in section 5.3.3. The following practical evaluation algorithm has been designed to be used for the evaluation of both Client Simple Rules and Server Simple Rules.
The evaluation algorithm works on a data structure named Rule Cache. This is a tree based structure, which contains all Simple Rules in an ordered manner. The order is given by the Priority attribute of the Rules. This is not an unique attribute, but rather an equivalence class for all Simple Rules with the same priority. The Rule Cache is built to regard increasing priorities, so that the most important Simple Rules `"Priority=1"` are located on the left side in order to be evaluated first. The evaluation order into an equivalence class is arbitrary. Figure 5.22 visualizes the general structure of the Rule Cache for a set of Rules $R_1, R_2, \ldots, R_n$.

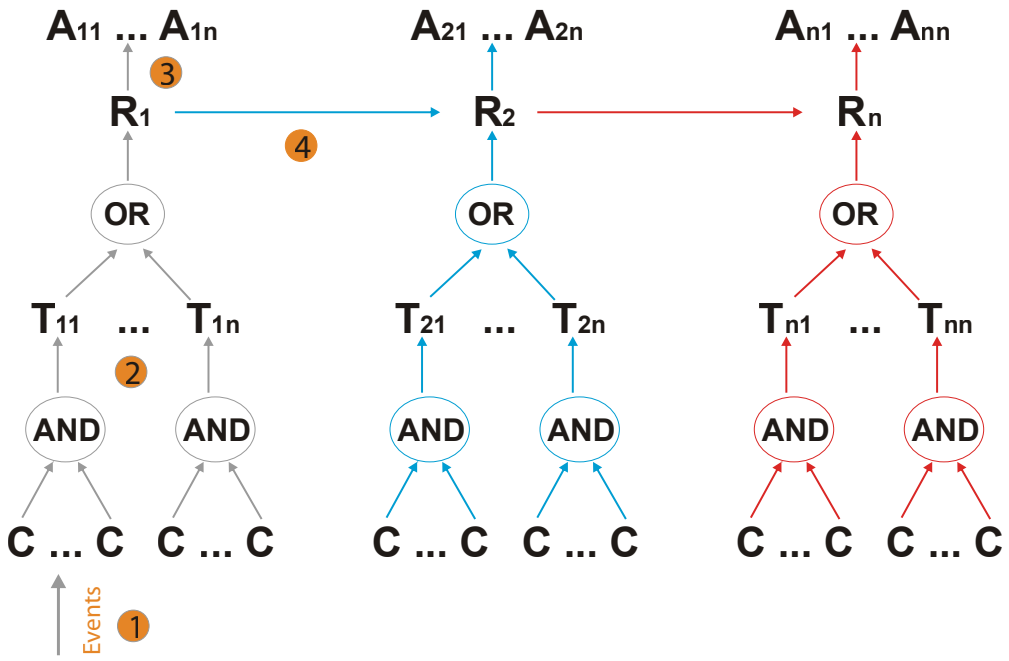


Figure 5.22.: SysMES - Server Simple Rule Cache

Triggers (more exactly Simple Triggers) and Actions also have a Priority attribute, which is related to the evaluation and execution order. High priority Triggers are located on the left part of the sub-tree and are evaluated first. High priority Actions are executed first. However, there is no order for the evaluation of Triggers or for the execution of Actions within a priority equivalence class.
As a reminder, a Simple Rule is composed of one or a set of Triggers joined by the "OR" operator on the left side. Each of these Triggers have one or a set of Conditions joined by the "AND" operator. The right side is composed of one or a set of Actions to be executed.

The tree-based and priority-based Rule Cache structure on one side offers the Rule developer the possibility to accelerate the evaluation giving high priorities to the Triggers, which are fulfilled by Events with the highest occurrence probability and on the other side to optimize the execution order of the Actions.
Simple Rules are deployed using Configuration Tasks (explained in section 5.4.2.5) and therefore are permanently stored in a database. The Rule Cache is built during the start process of each LAM server using the stored Rules - if any exist - and can be updated at any time by inserting and removing Simple Rules or changing their properties. This characteristic allows a highly dynamic Rule Management because the Simple Rule deployment strategy can be changed depending on the workload, which contributes to achieve the aims concerning scalability.
The evaluation algorithm checks if the incoming Event with the attribute `"CheckServerRule=true"` fulfills the Simple Rules in the Rule Cache. According to the sequence of figure 5.22, the algorithm starts ① testing all Conditions of the top priority Triggers of the top priority Simple Rule. If at least one Condition is not fulfilled then the evaluation of the remaining Conditions stops. The next step ② is to check the other Triggers ordered by priority. If one of the Triggers is fulfilled, then the Simple Rule is fulfilled and the evaluation of the remaining Triggers stops and ③ the execution of the Actions follows. The evaluation algorithm continues checking ④ the further Simple Rules using the same method.

Simple Rule Clustering: The clustering of the Server Simple Rule Management is tightly coupled to the Event Management clustering, for the Event forwarding and checking initiation, and to the Task Management 5.4.2.5 clustering, for the distribution of the Simple Rules as well as for the execution of Actions.
Each LAM server has the capabilities to check if an incoming Event needs to be taken into account for the Simple Rules and if so, then to check the Simple Rules locally. The implications of the local processing are firstly, the location independent Server Simple Rule Management because the SysMES targets sends the Events to an arbitrary server of the LAM layer and secondly, the requirement that each LAM keeps a copy of the Rule Cache locally and all copies are synchronized. The algorithm to distribute the Simple Rules - or any other management object - is subject of the forthcoming Task Management section 5.4.2.5.

Summary of the section:

The Server Simple Rule Management is the middle part of the three layered SysMES Rule Management subsystem. Server Simple Rules are similarly structured like those located on the client side, containing a left side (LS) as a set of Simple Trigger objects and a right side (RS) as a set of Action objects. Another common characteristic is that for the evaluation of these Rules only one Event is taken into account and therefore only simple states from only one originator. The most significant difference to the Client Simple Rules is the number and the complexity of Actions which can be used. The SysMES framework provides Actions to inform an administrator via email or SMS about the state represented by the Event. Further Actions are offered for executing any kind of binary that contributes to the resolution of an undesired state and for forwarding Events to other parts of the management environment. Simple Rules are distributed using the Task Management capabilities of the SysMES framework and are stored in a database. Each instance of the LAM Layer is able to use the stored Simple Rules to create a tree-based and priority-based structure - the so-called Rule Cache - which is used to evaluate incoming Events. The Rule Cache has the capability to include newly deployed Simple Rules, remove these or to change their characteristics in the running system without restart.

Each member of the LAM layer has the capabilities for Simple Rule Management and is therefore responsible for the local processing of Events. Furthermore, each LAM server also has the capability to forward Events to the forthcoming Complex Rule Management subsystem.

5.4.2.4.2. Complex Rules Management The Complex Rule Management subsystem builds the third and last layer of the SysMES Rule Management system. It is located on the WAM Layer and is responsible for the processing of Events in order to cope with the following cases:

- Recognition of Complex States: A Complex State is defined by the specific occurrence, combination and correlation of Events. Complex States can also be identified by calculations on the attribute value of Events. For this purpose set and boolean operators can be used. Typical Complex States are e.g. the correlation of the room temperature with the rack temperature `"room temp > 30 & rack temp > 30 & room temp > rack temp"`, the temperature average within a rack `"temp avg in rack 3 > 32"` or the value of a specific temperature sensor plus a specific value `"temp1 + 10 > room temp"`.

- Recognition of Global States: A Global State is a special type of a Complex State. In this case, Events of all available targets are taken into account in order to calculate a Global State. An example is the average temperature of all targets `"temp avg in all targets > 32"`.

For the detection of Complex States, a new type of Rule, the so-called Complex Rule, has been developed. This type of Rule is composed on the left side of one or multiple Complex Triggers, which are used for the definition of the Complex States to be detected, and on the right side one or multiple Actions.
The main difference between the two types of Rules in the SysMES framework is that Simple Rules are stateless and Complex Rules are stateful. That means that processing of a single Event by a Simple Rule does not change a state internal to the Simple Rule and thereby does not affect the evaluation of Events arriving later. In contrast to that, Complex Rules are fulfilled by the correlation of multiple Events. Therefore, Events or partial matches have to be stored and make up the state of a Complex Rule.
Until now, it was only possible to define and evaluate Rules for the occurrence of one Event. Therefore, it is necessary to extend the Rule representation and evaluation capabilities in order to define and recognize Complex States and Global States. The Subjects of the following sections are the representation and evaluation of Complex Rules and the definition of a fail tolerant for the Complex Rule Management subsystem.

Complex Rule Representation: Similar to the representation of all management objects in the SysMES framework, the Complex Rule representation is object-oriented and based on the class hierarchy and relationships shown in figure 5.23.
The object-oriented representation of Complex Rules is a tree based representation. The general Complex Rule syntax is described in figure 5.23. According to this, the ComplexRule class is derived from the Rule class and each of its objects has one association to a Complex Trigger object and one or many associations to Action objects.
The expressiveness of the Complex Rule representation is realized by the optional recursive association of Complex Trigger objects with other Complex Trigger objects, as well as with objects of the Simple Triggers and Operation classes. Before the specific composition of Complex Rule is covered, the respective syntax and semantics of the classes are introduced. In order to simplify matters, the

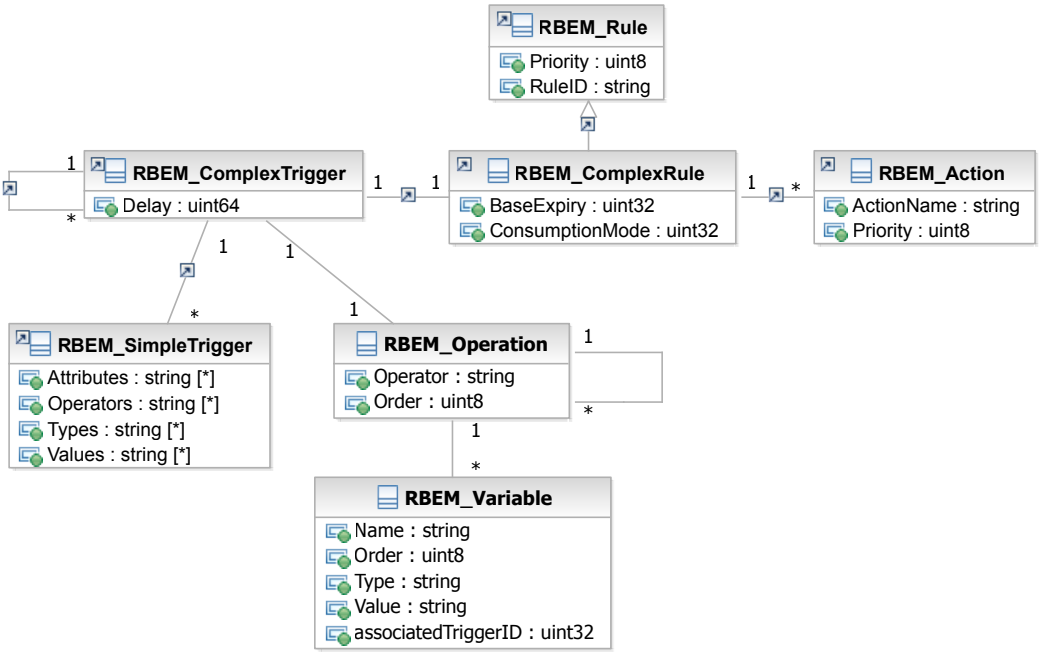


Figure 5.23.: SysMES - Complex Rule Class and Associations

data type of the attributes is only defined in the respective class diagram and a specific description follows in exceptional cases only.

ComplexRule (CR): Derived from its super class, the Complex Rule class has the attributes Priority and RuleID and additional the attributes BaseExpiry and ConsumptionMode.

RuleID is a unique runtime object identifier. The Priority attribute is used in order to define an evaluation sequence. The Complex Rule checking algorithm checks all Complex Rules according to this attribute beginning with the highest Priority `"Priority=1"`. In the case that more Complex Rules have the same Priority, these are evaluated in a random order.

BaseExpiry is a time attribute, which describes how long the evaluation of a Complex Rule can take from the occurrence of the first Event until all Triggers are fulfilled. This attribute is specified in milliseconds.

CosumptionMode is used in order to describe what happens with processed Events. The detailed description of its semantics follows separately in this section.

After this general description of Complex Rules follows the introduction of the Trigger classes and the Actions.

Trigger (T): Conforming to figure 5.24, this class is the top of the inheritance hierarchy. It has three attributes: TriggerID, Priority and Order. The first one is a unique runtime identifier for each type of Trigger object, the second one has the same semantics as the Priority attribute of the previous Complex Rule class and is used for an ordered evaluation of the Triggers and the third one defines an order for the associations between objects of the derived classes Simple Trigger and Complex Trigger. The necessity of this is justified by the potential usage of non-commutative operators for the Event processing.

The difference between Priority and Order is that Priority is designed to accelerate the evaluation of Rules by setting it according to the frequency of occurrences of Events, i.e. Triggers which are

fulfilled by Events with the most frequent occurrences should have the highest priority.
The Order attribute is used to define the order of operands for operators. Note that the semantics of a Trigger evaluation can change according to the order of the operands depending on whether the applied operator is commutative or not.
The Simple Trigger (ST) class has already been introduced in section 5.3.3 and therefore the explanation of the Complex Trigger class follows.
Complex Trigger (CT): This class is used to represent an operator to be applied to Event or Event correlation occurrences or to the attribute values contained into the Events. These Events can be from one or several targets as well as from different Monitors. The evaluation of Complex Triggers is always related to a time interval (ΔCT) defined by the occurrence of the first Event, [5] which fulfills parts of the Triggers and ends with the occurrence of the last relevant Event.
ΔCT = $E_{first}.lastoccurrence\ -\ E_{last}.lastoccurrence$.

Complex Trigger is an abstract class with the attribute Delay, which is used to delay the further processing of Events by another Trigger after ΔCT. A more specific description of this attribute can be found separately at the end of this section.

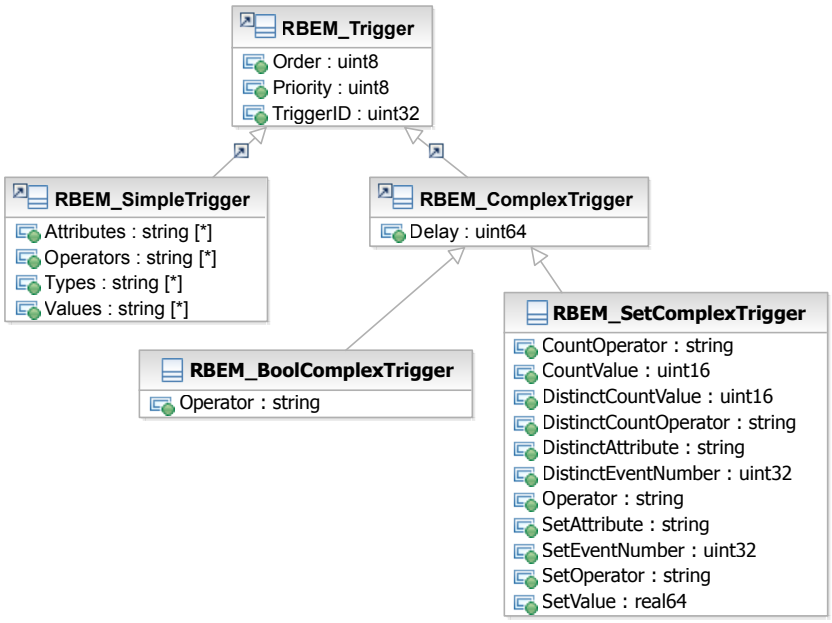


Figure 5.24.: SysMES - Complex Trigger Class

The classes Boolean Complex Trigger and Set Complex Trigger are derived from the Complex Trigger class and are used to define boolean or set operators.

- Boolean Complex Trigger (BCT): This special derived Trigger class is used in order to apply boolean operators to the occurrences of Events. Its attribute Operator defines a specific operator and can have the values `"AND"`, `"OR"` or `"NOT"`. The following table 5.8 contains their syntax and semantics.

[5] The value of the `"LastOccurrence"` Event attribute introduced in section 5.3.2.

Operator	Arity	Syntax	Semantics
AND	2	AND(T1, T2)	true when both triggers are fulfilled
OR	2	OR(T1, T2)	true when one of both triggers are fulfilled
NOT	2	NOT(T1, T2)	true if T2 is not fulfilled within ΔCT1

Table 5.8.: SysMES Complex Rules: Boolean Operators

- Set Complex Trigger (SCT): This class is used for the definition of operators which have to be applied to a set of Events or Event correlation occurrences. Typical usage of this is to count Event occurrences for calculations on the Event attribute values for all Events in an Event set (e.g. average). Objects of this class are stateful because they store Events or Event correlations which have fulfilled the associated Triggers. Set Complex Trigger objects offer the capability to calculate a single value out of stored Event attribute values.

  The attributes of this class and their respective data types are shown in figure 5.24 and can be divided into three parts, the count part, the set part and the distinct part. Figure 5.25 helps to visualize the setup of a Set Complex Trigger object. The sample contains only the attributes necessary for the explanation of this kind of Trigger. This Set Complex Trigger object is fulfilled if there are three different targets (identified by the DeviceID attribute of Events) where each has sent 10 Events reporting a high temperature of the hard disks HD1 and HD2 (in other words 10 correlated Events) and for each target, the temperature average of the HD1 (measured by the value of the 10 Events) is greater than 46° C.

  - Count Part: This attribute part is mandatory. It defines the size of the Event set which has to be taken into account and the operator to be applied to this set. Operator is the main operator of the Set Complex Trigger object and can have the values `"Count"`, `"DistinctCount"`, `"Average"`, `"DistinctAverage"` or `"Sum"`.

    In contrast to the boolean operators, these have an arity of one and therefore each object of this class has exactly one association to another Trigger object. CountOperator and CountValue define the set size. CountOperator has one of the values `"gt(greater than)"` or `"eq(equals)"` and together with the CountValue define the number of relevant Event occurrences which have to be stored for the evaluation of the Trigger.

    In case of figure 5.25, these attributes are set as follows `"Operator=DistinctAverage"`, `"CountOperator=eq"` and `"CountValue=10"`, which means that the Trigger correlates the data of exactly 10 Events.[6]

  - Set Part: This attribute part contains the attributes SetAttribute, SetOperator, SetValue and SetEventNumber. It is mandatory to set these if a single value (e.g. `"Average"`, `"Sum"` and `"DistincAverage"`) has to be calculated.

    Sample 5.25 shows that the Set Complex Trigger is associated to one Boolean Complex Trigger, which is fulfilled by the occurrences and correlation of two Events, which fulfill the Simple Triggers. In this case, the Set Complex Trigger object stores pairs of Events (E_i, E_j) where E_i fulfill the Simple Triggers with `"TriggerID=111"` and E_j the other Simple Triggers with `"TriggerID=222"`.

    The SetEventNumber attribute is used in order to define which Events are taken into account for the calculation. According to the introduced sample `"SetEventNumber=111"`,

[6] without consideration of the next explained distinct part.

Figure 5.25.: SysMES - Set Complex Trigger Sample

the Events which fulfill the left Trigger are relevant for the result calculation.

Furthermore, SetAttribute is used to identify one Event attribute to extract the values for the calculation. SetOperator and SetValue define a condition which has to be fulfilled by the calculated result value. SetOperator can have the values`"gt(greater than)"`, `"lt(less than)"` or `"eq(equals)"`.

In the case of the figure 5.25, the attributes have the values `"SetAttribute=Info.Value"`, `"SetOperator=gt"`, `"SetValue=46"`.

That means that the `"DistinctAverage"` Operator calculates one result using the data from the Event attribute Info.Value of the 10 Events and returns `"true"` if the result is `"gt 46"`.

– Distinct Part: The distinct part is applicable to the previous Count Part and to the Set Part and is used to specify that all relevant Events must have the same value of a specific Event attribute. More exactly, the Distinct Part is used to store Events in equivalence classes according to the value of one specific Event attribute of all Events, which fulfilled one specific Trigger. This specification is realized by setting up the attributes DistinctEvent-Number, DistinctAttribute, DistinctCountOperator, DistinctCountValue.

  Similar to the SetEventNumber, the DistinctEventNumber attribute is used to define which

Events from an Event correlation have to be taken into account. The current example defines the `"DistinctEventNumber=222"` and therefore the sorted storage of the Events considers only the attributes of Events which fulfill the right Trigger.

DistinctAttribute is used for the definition of a specific Event attribute. DistinctCount-Operator and DistinctCountValue define a condition, which specifies how many equivalence classes have to store how many Events (CountValue) and fulfill the Set Complex Trigger (SetAttribute, SetOperator, SetValue).

The attribute values in the previous example were: `"DistinctAttribue=DeviceID"`, `"DistinctCountOperator=eq"` and `"DistinctCountValue=3"` and mean that storage of Events in equivalent classes is dependent on the value of the DeviceID Event attribute, the Trigger is fulfilled if three equivalence classes store 10 Events and the average of the value of the Info.Value attribute of the Events is `"gt 46° C"`.

Operator	Data type	Semantics
and	Boolean	boolean and
or	Boolean	boolean or
not	Boolean	boolean not
eq	Numeric	arithmetic $=$
neq	Numeric	arithmetic $\neq$
gt	Numeric	arithmetic $>$
gte	Numeric	arithmetic $\geq$
lt	Numeric	arithmetic $<$
lte	Numeric	arithmetic $\leq$
streq	String	identical string
strueq	String	not identical string
plus	Numeric	arithmetic $+$
plus	String	string concatenation
minus	Numeric	arithmetic $-$
mult	Numeric	arithmetic $*$
div	Numeric	arithmetic $\div$

Table 5.9.: Operation Class - Operators

Operation (O) and Variable (V): While the SetComplexOperator enables calculations (e.g. average, sum) for attributes of Events of the same type, (i.e. that have fulfilled the same Trigger), comparisons between attributes of Events that have fulfilled different Triggers cannot be expressed yet with the given means. Therefore the classes Operation and Variable have been launched. These classes both have the common attribute Order, which is used for the introduction of a specific order of the operands The Operation class is used for the definition of an operator, which is applied to the associated objects. The operator attribute can assumes the values described in table 5.9.
An Operation object can only be associated to Boolean Complex Trigger object. This is because the Operation should be performed on Events or Event correlations of different Triggers and the Set Complex Trigger works on Events or Event correlations of the same Trigger.
The Variable class is furthermore used for the definition of operands. There are two ways to define the operands: by value (i.e. giving an value explicit) and by reference (i.e. giving an object where the values can be extracted). The definition by value is realized by setting the attributes Value and

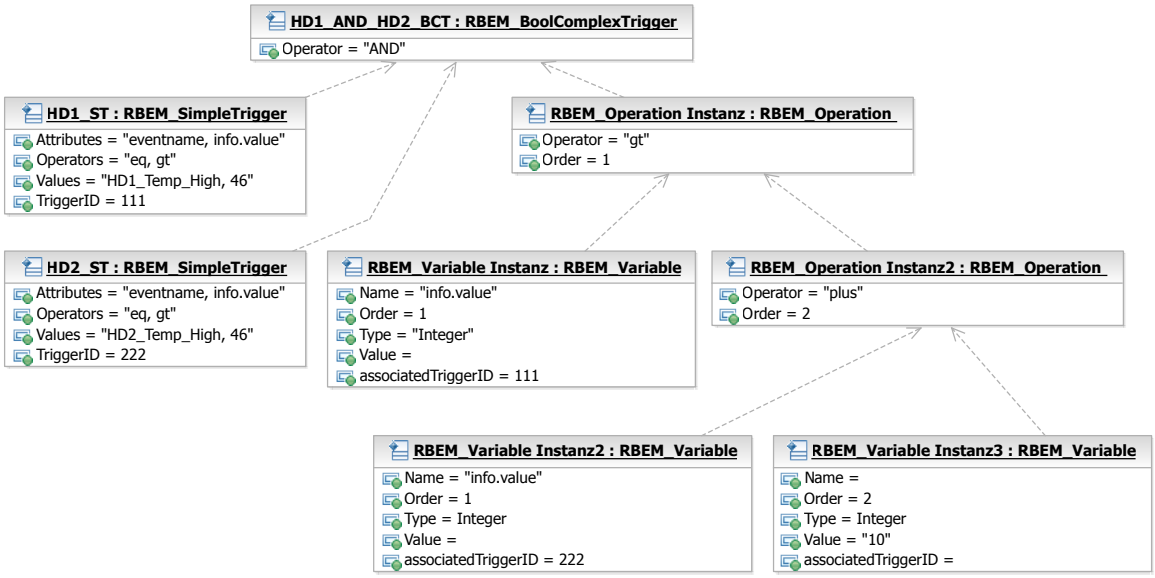


Figure 5.26.: SysMES - Operation and Variable Sample

Type. It is important to define the Type attribute because the semantics of operators depend on the data types.

An operand by reference is defined through the attributes AssociatedTriggerID, holding the unique identifier of a Simple Triggers, and Name, holding the name of one Event attribute relevant for the calculation. In Other words, Event occurrences which have fulfilled the given Trigger are taken into account as operands.

Figure 5.26 shows an example of how Operators and Variable objects can be used. It extends the Boolean Complex Trigger of figure 5.25 in order to express that the value of the Info.Value attribute of the `"HD1_Temp_High"` Events must be greater than the value of the Info.Value attribute of the `"HD2_Temp_High"` Events plus a value of `"10"`. (i.e. `"HD1_Temp_High.Info.Value > HD2_Temp_High.Info.Value + 10"`)

At the beginning of this section, a Complex Rule class diagram 5.23 was introduced. This class diagram shows the general design of Complex Rules, but the set of all syntactic and semantic correct Complex Rules is a subset of this. This is due to the inability to include syntactic characteristics in a class diagram such as that a Complex Trigger can only have as many associations to other Triggers as the arity of the used operator or that Operator objects can only be associated to Boolean Complex Trigger objects.

In order to formalize the set of syntactic and semantic correct Complex Rules, the following context free grammar has been developed:

Be $G = \{V,\ \Sigma,\ R,\ CR\}$ a context free grammar with V is a set of non terminal symbols

$$V = \{CT,\ SCT,\ BCT,\ T,\ Operation,\ ArithOperation,\ BoolOperator,\ RelOperator,\ MathOperator,\ Variable,\ CR\}$$

Σ is a set of terminal symbols

$\Sigma = \{AND,\ OR,\ NOT,\ AVG,\ COUNT,\ ST,\ and,\ or,\ not,\ =,\ >,\ \geq,\ <,\ \leq,\ +,\ -,\ *,\ \div,\ Value,\ Reference\}$

CR (Complex Rule) is the initial symbol and R the following set of production rules

$R_1:$ $CR \Rightarrow CT\ A^*$

$R_2:$ $CT \Rightarrow SCT \mid BCT$

$R_3:$ $SCT \Rightarrow AVG(T) \mid COUNT(T) \mid DISTCOUNT(T) \mid DISTAVG(T)$

$R_4:$ $BCT \Rightarrow AND(T,T)Operation \mid OR(T,T)Operation \mid NOT(AND(T,T),\ T)Operation$

$R_5:$ $T \Rightarrow SCT \mid BCT \mid ST$

$R_6:$ $Operation \Rightarrow BoolOperator(Operation, Operation) \mid RelOperator(ArithOperation, ArithOperation)$

$R_7:$ $ArithOperation \Rightarrow MathOperator(ArithOperation, ArithOperation) \mid Variable$

$R_8:$ $BoolOperator \Rightarrow and \mid or \mid not$

$R_9:$ $RelOperator \Rightarrow = \mid \neq \mid > \mid \geq \mid < \mid \leq \mid streq \mid strneq$

$R_{10}:$ $MathOperator \Rightarrow + \mid - \mid * \mid \div$

$R_{11}:$ $Variable \Rightarrow Value \mid Reference$

Figure 5.27 shows an example of a Complex Rule object. It contains objects of almost all classes introduced before and also values of all needed attributes. This Rule correlates the Event data from two different Monitors and makes sure that the correlated Events are from the same target.
This Complex Rule is generated by the introduced context free grammar through the application of the following derivation rules:

$R_1 : CR \Rightarrow CT\ A^*$

$R_2 :\Rightarrow BCT\ A^*$

$R_4 :\Rightarrow AND(T,T)Operation\ A^*$

$R_5 :\Rightarrow AND(SCT,T)Operation\ A^*$

$R_3 :\Rightarrow AND(DISTCOUNT(T),T)Operation\ A^*$

$R_5 :\Rightarrow AND(DISTCOUNT(ST),T)Operation\ A^*$

$R_5 :\Rightarrow AND(DISTCOUNT(ST),ST)Operation\ A^*$

$R_6 :\Rightarrow AND(DISTCOUNT(ST),ST)RelOperator(ArithOperation, ArithOperation)\ A^*$

$R_9 :\Rightarrow AND(DISTCOUNT(ST),ST)streq(ArithOperation, ArithOperation)\ A^*$

$R_7 :\Rightarrow AND(DISTCOUNT(ST),ST)streq(Variable, ArithOperation)\ A^*$

$R_7 :\Rightarrow AND(DISTCOUNT(ST),ST)streq(Variable, Variable)\ A^*$

$R_{11} :\Rightarrow AND(DISTCOUNT(ST),ST)streq(Reference, Variable)\ A^*$

$R_{11} :\Rightarrow AND(DISTCOUNT(ST),ST)streq(Reference, Reference)\ A^*$

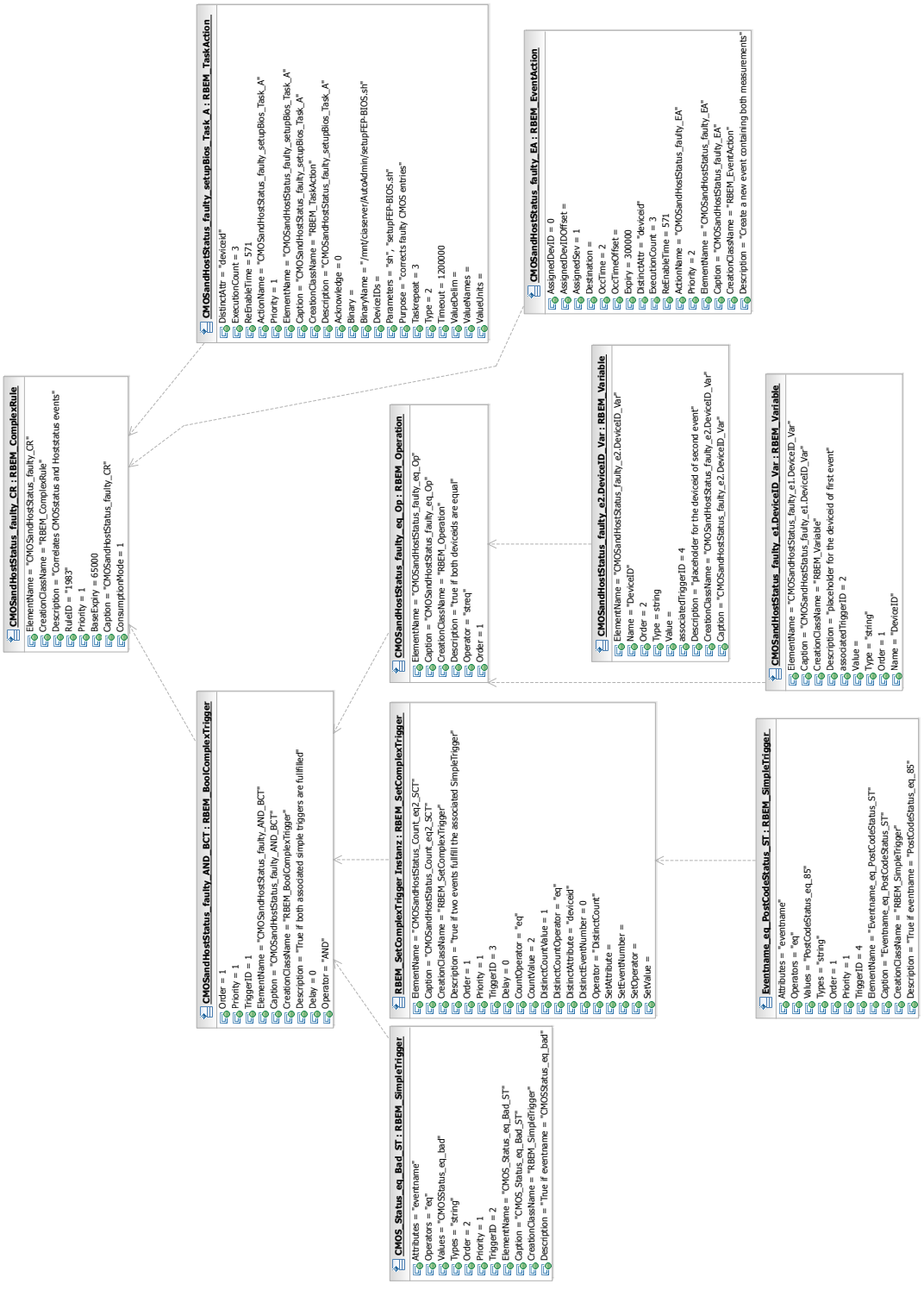


Figure 5.27.: SysMES - Server Complex Rule Sample

Action(A): Similar to the Simple Rules, Actions build the right side of the Complex Rules and are used in order to react automatically to a detected Complex State. The Complex Rule subsystem utilizes the same Actions as the Simple Rule Management subsystem described in the section above. However, the Action objects of the Complex Rules define three additional attributes DistinctAttr, ExecutionCount and ReEnableTime. These are used to control the execution behavior of the Action. These attributes describe how often the Action should be executed at the occurrence of a specific Event or Event correlation. ExecutionCount defines the number of executions for the same Event or Event correlations related to one Event attribute defined in DistinctAttr. Normally the execution of Actions for the same Event correlation is disabled for the period defined in the ReEnableTime attribute. After the first execution, the Action stores the values of the Event attribute of all involved Events as well as the next execution time: $(next\ execution\ time = last\ execution\ time + ReEnableTime)$, and decreases the stored value of ExecutionCount.
Before the next execution, the value of the stored data from the first execution is checked. The action execution follows only if the ExecutionCount is greater than zero for the specific stored value of the Event attributes and if the next execution time is less than the system current time. In the case of the attribute values used in the example 5.27 `"DistinctAttr=DeviceID"`, `"ExecutionCount=3"` and `"ReEnableTime=571"`, the Action is executed `"3"` times in a time interval of `"571"` seconds for each Event correlation with different values of the `"DeviceID"` attribute of Events.
Event Action is mainly, but not exclusively, used for the Complex Rules. It extends the Event generation capabilities to the servers so that it is possible to generate Events on the server side. For this purpose, the following attributes are used: AssignedDevID, AssignedDevIDOffset, OccTime, OccTimeOffset, AssignedSev, Destination and Expiry.
As described above, a Complex Rule is normally fulfilled by the correlation of multiple Events such as E_0, E_1, E_2 ... E_n. The attributes of the new Event $\mathrm{E}_{\mathrm{new}}$ are set up from the values of the attributes of the single Events. AssignedDevID and OccTime are used to identify one Event of the correlation in order to extract its value of the DeviceID, FirstOcccurrence and LastOcccurrence attributes. Each of these attributes can be obtained from different single Events, which participate in the Event correlation. For example if `"AssignedDevID=0"` and `"OccTime=2"` then

$$E_{new}.DeviceID = E_0.DeviceID$$
$$E_{new}.LastOccurrence = E_2.LastOccurrence$$
$$E_{new}.FirstOccurrence = E_2.FirstOccurrence$$

Furthermore, AssignedDevIDOffset and OccTimeOffset are used to define an offset, which can be added to the extracted value of the indexed Event. Destination is used in order to define a server to send the Event to and Expiry for the definition of a time related validity of the new Event. The AssignedSev attribute is used to set up the Severity value of the new Event.

The semantics of the Rule language are defined in the semantics of the class attributes described above. However, there are two other relevant aspects which have to be taken into account. The first one concerns the usage of time related attributes and the second one the usage of Consumption Modes.

Time Related Attributes: One relevant aspect for the processing of Events by the Rule system is that the Events are generated at a specific point in time in distributed targets and arrive at a later point in time at distributed servers. Depending on the network load, the target and server utilization, this latency may cause errors in the Rule matching. Rules may keep waiting for delayed Events or expected Events may not be taken into account because they arrive too late. In order to deal with this synchronization problem, two attributes have been introduced. These attributes are BaseExpiry of the Complex Rules and Delay of the Complex Trigger.

- BaseExpiry: The BaseExpiry attribute is used in order to identify old Events and to exclude them from the Complex Rule evaluation. It is a value in milliseconds. For a single Event, the value of its LastOccurrence attribute is used as the reference value. This is due to the capability of the SysMES clients to compress Events if needed (see section 5.3.2) and to reset the value of the FirstOccurrence and LastOccurrence so that the occurrence of the last Event is only reflected in the LastOccurrence attribute. A single Event (E) is only taken into account if:

  $$E.LastOccurrence + BaseExpiry > CurrentTime$$

  Granted that there are Event correlations $\{\mathrm{E}_0, \mathrm{E}_1 \ldots \mathrm{E}_\mathrm{n}\}$ which have fulfilled Complex Triggers and then have to be evaluated against another Trigger, then the time validity check occurs as follows:

  $$max\{E_i.LastOccurrence\} - min\{E_i.LastOccurrence\} < BaseExpiry \, \forall \, i \in 0 \ldots n$$

- Delay: As mentioned above, this is an attribute of the Complex Trigger class. It represents a value in milliseconds for the insertion of an artificial waiting period in the propagation of an Event correlation, which fulfills the Trigger. The reason for this is due to possible transmission delays on the Event arrival.

  One of the most important use cases of this attribute is when used by the `"NOT"` operator, i.e. in $(\mathrm{E}_1 \text{ AND } \mathrm{E}_2) \text{ NOT } \mathrm{E}_3$.

  Conforming to the description of the operator this is fulfilled by the occurrence of the Events E_1 and E_2 and no E_3 exists with $\mathrm{E}_3.\mathrm{LastOccurrence}$ in the time interval:

  $$[min(E_1.LastOccurrence, E_2.LastOccurrence\ ),$$
  $$max(E_1.LastOccurrence, E_2.LastOccurrence)]$$

  The Delay value forces the Rule engine to wait for a while in case a delayed Event E_3 arrives.

  Setting the value of this attribute requires experience with the network delays in the environment where the SysMES framework operates.

Consumption Modes: The Consumption Mode determines how often single Events and Event correlations can be taken into account in the evaluation of Complex Rules. It is defined in each Complex Rule object and is therefore flexible because each Complex Rule can be treated independently.
In the actual development state of the SysMES framework, the Unrestricted Consumption Mode and Unrestricted Detection Mode are implemented with the following meanings:

- Unrestricted Consumption Mode `"ConsumptionMode=1"`: In this mode, an Event or an Event correlation can only participate in one firing of a specific Rule. The Events are removed after the firing has been carried out
- Unrestricted Detection Mode `"ConsumptionMode=2"`: In this mode, Events can be used for multiple Rule evaluations as long as they satisfy the BaseExpiry requirements of the Complex Rule. .

Other Consumption Modes such as Recent, Chronicle, Continuous and Cumulative can be found in [3], [5] and [4].

The definition of a Consumption Mode of a Rule has to be chosen carefully because the Rule evaluation behavior changes depending on this. For example, the Complex Rule

$$\texttt{IF AND}(\texttt{E}_1\texttt{.EventName} = \texttt{CMOSStatus\_eq\_bad},\ \texttt{E}_2\texttt{.EventName} = \texttt{PostCodeStatus\_eq\_85})$$
$$\texttt{THEN Shutdown}(\texttt{E}_2\texttt{.DeviceID})$$

This Rule is fulfilled by the occurrence of two Events with "EventName=CMOSStatus_eq_bad" and "EventName=PostCodeStatus_eq_85" occur. The Events can be from the same or different targets. If the Rule is fulfilled, then the target which sent the Event "EventName=PostCodeStatus_eq_85" has to be shutdown.
Now imagine the following chronological occurrence of four Events $\mathrm{E}_1,\ \mathrm{E}_2,\ \mathrm{E}_3,\ \mathrm{E}_4$ with:

$$\texttt{E}_1\texttt{.EventName} = \texttt{CMOSStatus\_eq\_bad}$$
$$\texttt{E}_2\texttt{.EventName} = \texttt{PostCodeStatus\_eq\_85},\ \texttt{DeviceID} = 10.162.128.231$$
$$\texttt{E}_3\texttt{.EventName} = \texttt{PostCodeStatus\_eq\_85},\ \texttt{DeviceID} = 10.162.128.232$$
$$\texttt{E}_4\texttt{.EventName} = \texttt{PostCodeStatus\_eq\_85},\ \texttt{DeviceID} = 10.162.128.233$$

In the Unrestricted Detection Mode, the Rule fires sequentially three times for $(\mathrm{E}_1,\ \mathrm{E}_2)$, $(\mathrm{E}_1,\ \mathrm{E}_3)$ and $(\mathrm{E}_1,\ \mathrm{E}_4)$ and this causes all three targets to be shutdown. In contrast to this, in the Unrestricted Consumption Mode, the Rule fires only one time for $(\mathrm{E}_1,\ \mathrm{E}_2)$.
The chronological occurrence of the Events is now changed to $\mathrm{E}_2,\ \mathrm{E}_3,\ \mathrm{E}_4\ \mathrm{E}_1$ with:

$$\texttt{E}_2\texttt{.EventName} = \texttt{PostCodeStatus\_eq\_85},\ \texttt{DeviceID} = 10.162.128.231$$
$$\texttt{E}_3\texttt{.EventName} = \texttt{PostCodeStatus\_eq\_85},\ \texttt{DeviceID} = 10.162.128.232$$
$$\texttt{E}_4\texttt{.EventName} = \texttt{PostCodeStatus\_eq\_85},\ \texttt{DeviceID} = 10.162.128.233$$
$$\texttt{E}_1\texttt{.EventName} = \texttt{CMOSStatus\_eq\_bad}$$

When the Event E_1 arrives, the processing in Unrestricted Detection Mode is able to fire three times in a random order. However, in the Unrestricted Consumption Mode all three pairs of Events $(\mathrm{E}_2,\ \mathrm{E}_1)$, $(\mathrm{E}_3,\ \mathrm{E}_1)$ and $(\mathrm{E}_4,\ \mathrm{E}_1)$ may make the Rule fire. In this case, it is not clear which pair of Events causes the Rule to fire. One of the firing options is arbitrarily chosen so that it is impossible to predict which target will be shutdown.

Complex Rule Evaluation: This section introduces the Rete algorithm as the basis for the Complex Rule evaluation algorithm. The second part of the section addresses the extensions of the Rete algorithm realized in order to cope with the SysMES Complex Rules functionality requirements. Afterwards follows the SysMES Complex Rule evaluation algorithm and the fault tolerance strategy for this Rule Management subsystem.

The Rete Algorithm: It is a tree-based algorithm designed for the evaluation of statements (facts) using a rule-based decision logic in the area of expert systems. This algorithm was developed by Charles L. Forgy and published first in [48] and then in his doctoral thesis [49] and in a publication [50].
This algorithms was developed in order to accelerate the evaluation of Rules by having the following characteristics:

- Structured Evaluation: The first part of the algorithm concerns the transformation of all relevant rules [7] left sides to a Rete network. This network can be understood as a set of trees where each

[7] The usage of small letters in the word "rule(s)" symbolizes that in this case this word represents a general rule concept and not necessarily the SysMES specific Rules.

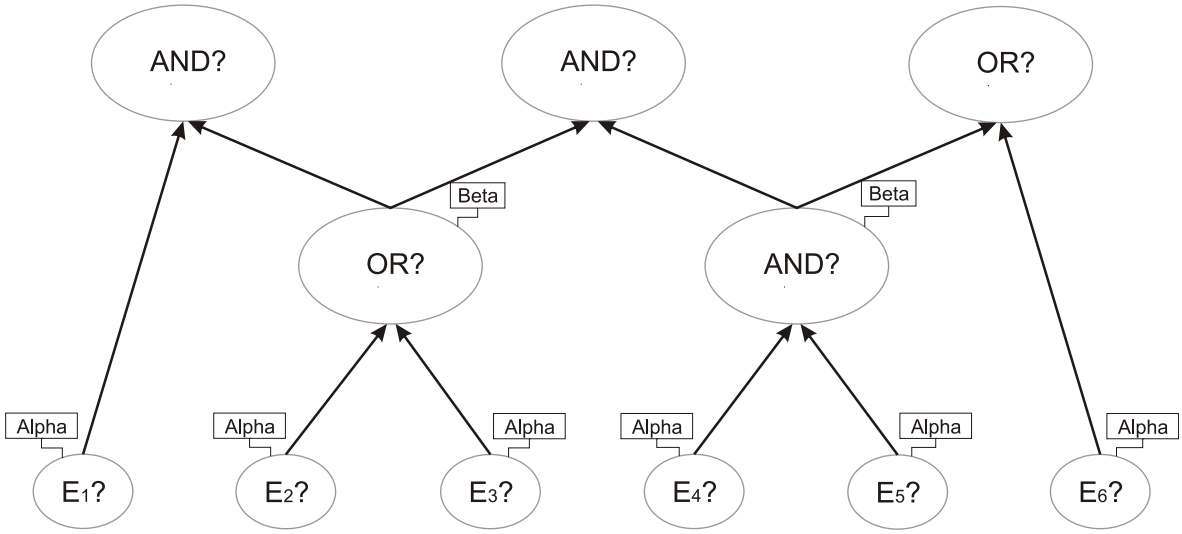


Figure 5.28.: Rete Network - Structure

tree represents the left side of a rule. Important is that several trees share common nodes so that there are no redundant nodes with the same semantics.

The Rete network is a data-flow network and represents data dependencies between conditions specified in the left side of rules.

The figure 5.28 visualizes the Rete network for the following rules. The $\mathrm{E_i}$ represent the occurrences of facts which in the case of the SysMES framework are coded into Events.

$\mathtt{R_1}$: $\mathtt{IF}$ ($\mathtt{AND}$($\mathtt{E_1}$, $\mathtt{OR}$($\mathtt{E_2}$, $\mathtt{E_3}$))) $\mathtt{THEN}$ $\mathtt{A_1}$

$\mathtt{R_2}$: $\mathtt{IF}$ ($\mathtt{AND}$($\mathtt{OR}$($\mathtt{E_2}$, $\mathtt{E_3}$), $\mathtt{AND}$($\mathtt{E_4}$, $\mathtt{E_5}$))) $\mathtt{THEN}$ $\mathtt{A_2}$

$\mathtt{R_3}$: $\mathtt{IF}$ ($\mathtt{OR}$($\mathtt{AND}$($\mathtt{E_4}$, $\mathtt{E_5}$), $\mathtt{E_6}$)) $\mathtt{THEN}$ $\mathtt{A_3}$

In this sample, the rules $\mathrm{R_1}$ and $\mathrm{R_2}$ share the node $\mathrm{OR(E_2,\ E_3)}$, $\mathrm{R_2}$ and $\mathrm{R_3}$ share the node $\mathrm{AND(E_4,\ E_5)}$ which demonstrates that there are no redundant nodes in the Rete network.

- Single Evaluation: Due to the optimal node distribution without redundancies, the evaluation of a condition will be executed once per fact. When a fact (or in our case an Event) fulfills a condition, then this partial match is stored and can be reused for the evaluation of other rules, i.e. in case of shared nodes.

  The Rete algorithm distinguishes between two different kinds of nodes: Leaves which are fulfilled by single facts, and Join Nodes, which are fulfilled by multiple facts. Tokens are facts which have passed their evaluation against a node. If the evaluation node is a Leaf then the token is stored in an alpha memory, which is a container for tokens representing a single fact.

  When moving up in the network, tokens are evaluated against other tokens at the Join Nodes. If such an evaluation is successful, a new token can be set up consisting of the compared tokens. Such a token refers to multiple facts. This new token can be stored in the beta memory related to that Join Node. These beta memories hold tokens which represent a concatenation of facts. Stored tokens are valid as long as these are invalided by another token.

Facts are inserted to the Rete network from the Leaves. The first step is to check if this single fact fulfills the conditions of the leaves. In that case, a match a token is created containing the information of the fact. The second step consists of the propagation of the token to the parent node. The parent node stores the token and matches it to stored tokens from another child. This method proceeds until the top node of the rule is reached. Tokens successfully matching this top node cause the rule represented by the tree to fire.

More information about the Rete Algorithm can be found in [48], [49] and [50]. Another example of the rule evaluation through Rete can be found in [23].
The Rule evaluation engine of the SysMES Complex Rule Management subsystem is based on the Rete algorithm. It is tree-based and it allows the storage of partial matches. It avoids multiple computation of facts. The current development state of the SysMES framework considers facts which are coded into Events, extensions of this are proposed in chapter 8.
Another important aspect concerning the tree-based Rule evaluation is that the top nodes know all the facts involved in a Rule firing. This information is relevant for executing the Actions depending on the firing context, i.e. if and how the value of the Event attribute has an impact on the Action execution behavior.
The functionality of the classical Rete algorithm is not sufficient for the evaluation of SysMES Complex Rules. This is due to the lifelong validity of tokens as well as to the incapability to calculate matches based on multiple occurrences of the same Events.

Extensions of the Rete Algorithm:

The SysMES Complex Rule evaluation algorithm extended the functionality of the Rete algorithm with respect to the following aspects:

- Limited Token Lifetime: Tokens in the SysMES Complex Rule Management subsystem are built from Monitoring Events which arrive at the framework in a frequency related to the time attribute of the Monitors, i.e. Period and Repeat. The values contained in the Events reflect the state of the monitored resources and therefore differ for distinct Events if the measured value changes. In the classical Rete algorithm, a Token is a fact with a lifelong validity. In order to process actual Events, the Expiry attribute of the Complex Rules has been introduced. As already described, it is used in order to define if incoming Events are used for Token building, as well as if Tokens have to be taken into account for further processing.

- Introduction of Set Nodes: The SysMES Complex Rule evaluation algorithm adopts the Rete node types introduced before. These are Leafs for single Events and Join Nodes for the correlation of Events arriving from different Leafs or another Join Nodes. According to the semantics of these nodes, it is not possible to make calculations on multiple occurrences of Events which have been inserted at the same Leaf. For this purpose a special Join Node type, the Set Node is introduced. This kind of node extends the Rete functionality in order to cope with sets of Events. It allows the usage of set operators (such as `"Count"` or `"Average"`) which are able to create a new Token of multiple Events passing the same Leaf. Another important issue concerns the capability to compare the values of the attributes of multiple Events of the same kind having arrived earlier, i.e. define a set of two Events of the same type and match if the value of the first Event is greater than that of the second Event.

- Delayed Token Propagation: The Rete algorithm processes Events instantaneously. That means if the conditions for a node are fulfilled, then a Token is created and propagated immediately.

Delays have to be considered in a Complex Rule Management subsystem for the processing of Events from distributed and decentralized targets. The extended Rete algorithm includes the capability to delay the Token creation and propagation of Join Nodes and Set Nodes in order to cope with Event delivering delays resulting of network or system latency.

- Introduction of Consumption Modes: The Rete algorithm is designed to operate in Unrestricted Detection Mode. This Consumption Mode is not enough for the SysMES management because it is not possible to restrict to one firing of Rules per Event occurrence. Therefore the Unrestricted Consumption Mode has been developed.

- Time-Related Calculations: The SysMES Events have time-related attributes such as FirstOccurrence and LastOccurrence which describe the Event generation time. Using the value of these attributes it is possible to make calculations concerning the occurrence time of the Events. This is especially important if the arrival order of Events has an impact on the Event evaluation, e.g when the `"NOT"` operator is used.

- No Token-to-Token Invalidation: The classic Rete algorithm knows only one kind of fact or Token invalidation. This is when another fact or Token invalidates it. The SysMES Rule evaluation engine does not use this Token invalidation method. This is due to the nature of the SysMES Events which represent states of monitored resources. The extended Rete algorithm introduces a new method for the invalidation of facts and Tokens based on the values of the BaseExpiry attribute of the Complex Rules. Every Token which does not fulfill the time related restrictions is deleted from the Rete network.

- Extension of the Matching Capabilities: According to the Complex Rule language described in the previous section, it is possible to define Triggers which make calculations with the Event attribute. For this purpose the classes Operation and Variable are introduced. Therefore the Rete algorithm has been extended in order to achieve the capability to evaluate such language constructs. The syntax of the Complex Rule definition language describes that Operation and Variable objects build a sub-tree which can only be associated to Boolean Complex Trigger objects. The evaluation of this sub-tree follows the fulfillment of the Operator described by the Boolean Complex Trigger. As described before, the Variable object stores either a value or a reference to a Simple Trigger. If the Boolean Complex Trigger has an association to a Set Complex Trigger, then the Variable can be used to reference the Simple Triggers used to inject the Events into the Set Complex Trigger (see example 5.27). In this case, the last single Event involved in the Set Complex Trigger fulfillment is used for the Operation-Variables calculation.

SysMES Rule Set and Evaluation Network:

On the SysMES server side there is a Rule Set which is a subset of all modeled Complex Rules. It is composed of all these Rules, which are deployed from the SysMES Operator Layer 5.5 to the SysMES Management Layer. The Rule Set is stored persistently in the database in the WAM layer. The SysMES server initialization routine reads the stored Complex Rules from the database and sets up the Evaluation Network.
Due to the desired characteristics of dynamic system management, the Complex Rule Management subsystem is able to accept and deploy changes in the deployed Complex Rules. These changes can be modifications of Triggers, of time related attributes or the insertion or deletion of parts of the Complex Rule. The insertion and deletion of entire Complex Rules is also supported. Although the Complex Rule evaluation is realized in-memory, it is possible to deploy those changes without downtime. The

only restriction is that changes for a deployed Complex Rule cause state loss. After the performance of a change, the part of the Evaluation Network concerning this Complex Rule is empty.
Similar to the classic Rete network, the SysMES Evaluation Network is in-memory tree based. The Complex Rule Management subsystem is responsible for building the Evaluation Network for all Rules stored in the Rule Set. The resulting Evaluation Network is ordered by the Priority attribute of the Rule objects, as well as by the Order attribute of the Trigger objects. The top-level node of each Complex Rule sub-tree is the Root Node

SysMES Complex Rule Evaluation Algorithm:

The detailed Complex Rule evaluation procedure is described in the flow diagram 5.29. It is an iterative algorithm which always starts with the injection of one single Event into the left Leaf of the Evaluation Network. In order to visualize this, the left Leaf corresponds to `i = 1`. The index "i" illustrates the value of the attribute Order utilized for the generation of the Evaluation Network.

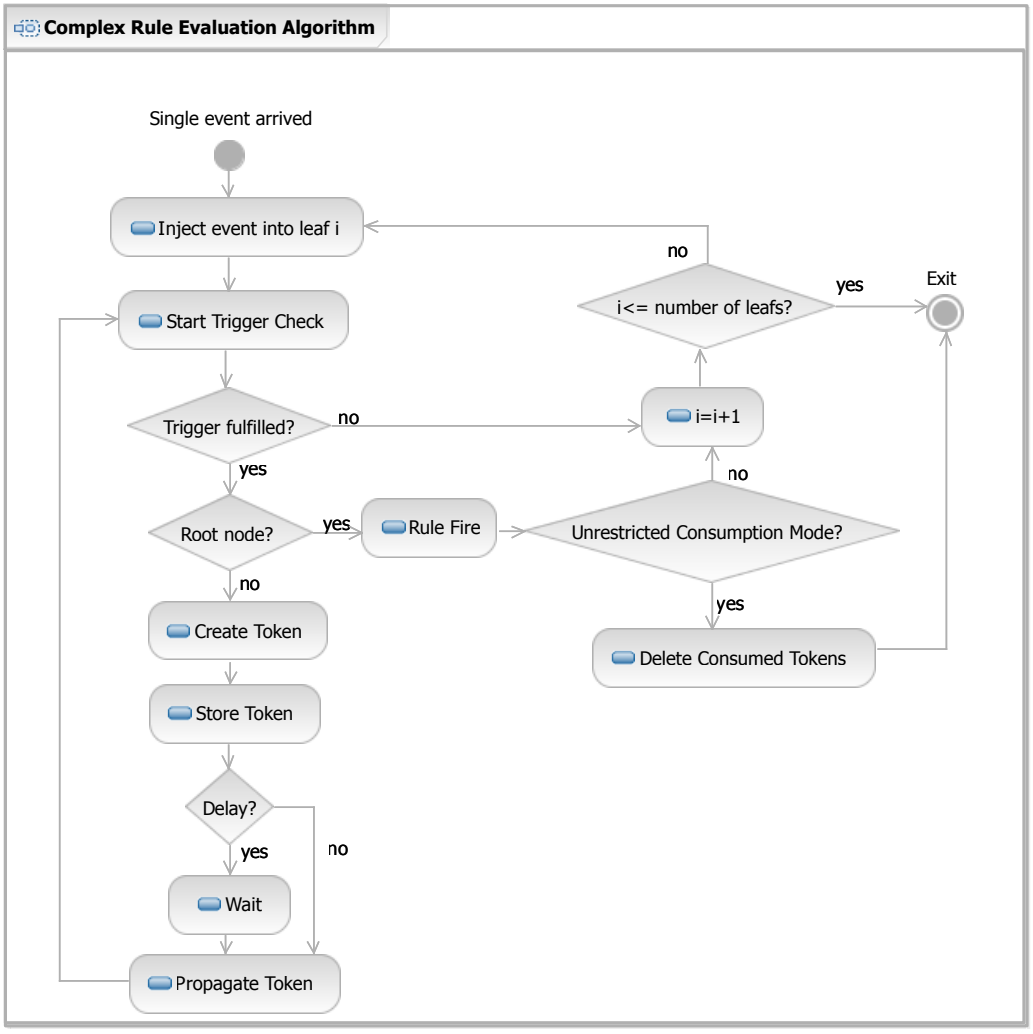


Figure 5.29.: SysMES Complex Rule Evaluation Algorithm

The algorithm is composed of two loops. The first one is the vertical loop, which is responsible for testing the single Event in the Leaf and then recursively the Tokens stored at the parents. The second loop is responsible for injecting the single Event to the next Leaf ($\texttt{i} = \texttt{i} + 1$) if there is nothing more to do in the previous sub-tree.
The first loop consists of the following actions and tests to be done. The algorithm starts after the injection of the single Event with the evaluation of the Trigger conditions of the first Leaf, i.e. the most left Leaf in the Evaluation Network. In the case the Event fulfills this Leaf it is necessary to test if the current checked node is the Root Node. Assuming that, it is the Root Node, then the Rule fires. In the case that it is not the Root Node, then a new Token is created, stored and propagated to the parent node. The Token propagation follows immediately or delayed depending on the tested node and its characteristics. This procedure continues recursively until a Root Node is reached or the Trigger condition is not matched.
In the case the Trigger conditions test fails at one node, then the second loop is initialized. It is in charge of injecting the original single Event to the next Leaf node. The algorithm repeats this operation until all tests are performed for the last Leaf, i.e. the most right Leaf.
The evaluation of one Event ends either with the firing of the Complex Rule or the modification of the Evaluation Network, e.g. if Tokens are created and stored. Another possible exit condition is if the Event does not fulfill any Leaf.
In the case the Rule fires, then the further processing differs according to the Consumption Mode. In the case of Unrestricted Consumption Mode, the algorithm traverses the evaluation tree and deletes all consumed Tokens and finalizes the evaluation. In the case of Unrestricted Detection Mode, the algorithm injects the original single Event into the next Leaf and continues checking.
The Trigger condition tests can have different complexities depending on the node type where the test has to be performed, i.e. Leaf or Join Nodes. In the case the evaluation algorithm operates in a Leaf, then testing the Trigger condition is very simple because only the information of a single Event is used for the evaluation. In the case of a Join Node, the test of a propagated Token (i.e. a generated Token because the Event fulfills the Leaf) is initiated in one child.
The evaluation algorithm tests the initialization Token iteratively against all Tokens stored at the other child of the Join Node. In the case the opposite side has multiple Tokens stored, there is no defined order for the test. It occurs in a randomized order. Each iteration tests the arbitrarily selected Token of the opposite side regarding the expiration of its validity on the basis of the BaseExpiry attribute of the Complex Rule. If the Token does not fulfill the BaseExpiry requirements, it will be deleted from the Evaluation Network.
The evaluation of the associated Operation part starts after the Join Node fulfillment. This test is a part of the complete Trigger test and necessary in order to create and propagate a new Token.
Figure 5.30 visualizes the states of the Evaluation Network for the Complex Rule introduced earlier in this section. The following Events arrive at the Evaluation Network and their index defines the injection order.

$\texttt{E}_1.\texttt{EventName} = \texttt{PostCodeStatus\_eq\_85}, \texttt{DeviceID} = \texttt{10.162.128.231}$

$\texttt{E}_2.\texttt{EventName} = \texttt{CMOSStatus\_eq\_bad}, \texttt{DeviceID} = \texttt{10.162.128.232}$

$\texttt{E}_3.\texttt{EventName} = \texttt{CMOSStatus\_eq\_bad}, \texttt{DeviceID} = \texttt{10.162.128.231}$

$\texttt{E}_4.\texttt{EventName} = \texttt{PostCodeStatus\_eq\_85}, \texttt{DeviceID} = \texttt{10.162.128.231}$

The first Event injected into the Evaluation Network is $\texttt{E}_1$. It passes the ST($\texttt{id} = 4$) and is therefore

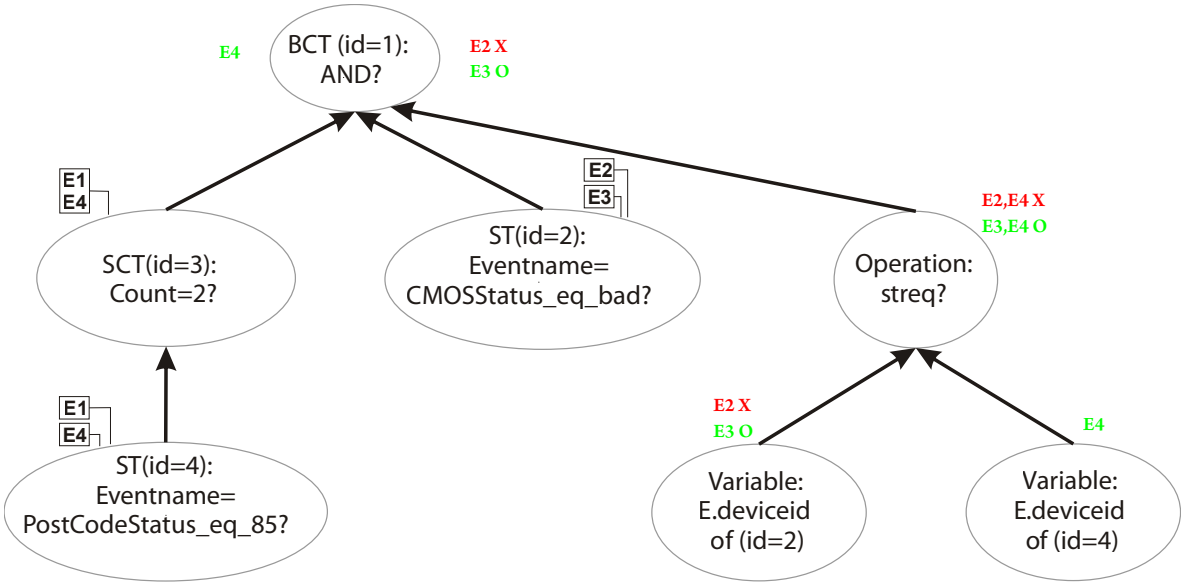


Figure 5.30.: SysMES Evaluation Network - Complex Rule Sample

stored in its alpha memory [8]. E_1 is propagated to the SCT(id = 3) which stores it into its beta memory. Afterwards the algorithm tests E_1 against the Trigger ST(id = 2) without success and therefore E_1 is not stored into its alpha memory. The next Event E_2 does not pass the test on ST(id = 4) but the test on ST(id = 2).

The following test is to check if BCT(id = 1) is fulfilled with the Tokens stored in their child node memories, i.e. SCT(id = 3) and ST(id = 2). This test fails because the node SCT(id = 3) is not fulfilled. The third Event E_3 has the same impact as the second Event and gets stored as a new Token in the alpha memory of the node ST(id = 2). The last Event E_4 is injected and tested successfully against ST(id = 4) and propagated to SCT(id = 3).

As a result of the Token propagation SCT(id = 3) is also fulfilled. In consequence of the SCT(id=3) fulfillment it is necessary to check if BCT(id = 1) is fulfilled with the child Tokens. At this point in time, there is one Token $\{E_4\}$ stored in the right child side of BCT(id=1) and two Tokens stored in the right side $\{E_2, E_3\}$. As a result, there are two possible pairs of Tokens for the Operation-Variable sub-tree evaluation on the BCT(id = 1) node. These are $\{E_4, E_2\}$ and $\{E_4, E_3\}$

However, the additional test of the Operation-Variables sub-tree fails for $\{E_4, E_2\}$ because both involved Events are not from the same target, i.e. $E_4.\text{DeviceID} \neq E_2.\text{DeviceID}$ (displayed red). The next and last iteration concerns the check of $\{E_4, E_3\}$. This test fulfills the Operation-Variables sub-tree because $E_4.\text{DeviceID} = E_3.\text{DeviceID}$ (displayed green) and the Rule fires. The evaluation algorithm deletes all Tokens where E_1, E_2, and E_4 participate because this Rule works in Unrestricted Consumption Mode.

Note that after the fulfillment of the SCT(id = 3) only the last Event relevant (in this case E_4) is taken into account for the processing of the Operation-Variables sub-tree. This is a characteristic of the evaluation of Set Nodes.

Figure 5.30 shows the serial evaluation of one single Event and one Rule. The evaluation algorithm is

[8] As a reminder the alpha memory is used for the storage of tokens which have passed Leaves and the beta memory for the storage of tokens which have passed Join Nodes.

designed for the one-event-one-rule parallel evaluation. That means that during the evaluation of one single Event against a Rule, this Rule is blocked for another incoming Event. However, the evaluation of following Events is performed parallelly at other Rules, which are not blocked at this time. Within the scope of this project, other kinds of parallel evaluation such as per-leaf and per cross-product of Tokens are reviewed. More about the other parallel execution paradigms can be found in [23].

Complex Rule Clustering: One important design decision taken into account during the conceptual task concerns the "location independent management". This is especially important due to the intended high degree of scalability and dependability. The basic idea is that every member of a Management Layer (Access Point, LAM and WAM) can replace any other because there is not a fixed connection or load distribution. All members of a specific management sublayer belong to a functionality cluster, which makes the layer dependable. Additionally to this idea, it is possible to add further members into the management sublayer on the fly in order to deal with additional load and therefore to make this Management Layer scalable. This capability is developed and implemented for Event Management, Simple Rule Management and Task Management.
In the case of the server-side Simple Rule management, each member of the LAM has exactly the same Rule Set. Therefore it is possible to evaluate incoming Events in every LAM server instance and also independent of the target which sent the Event.
In the case of the Complex Rule Management subsystem, there are three reasons why it is not suitable to emulate the clustering method of the other SysMES management functionalities, i.e. Simple Rules or Tasks. These are:

- Communication Overload: As described below, Complex Rules are designed for the correlation of data contained in multiple Events. These Events arrive at the SysMES framework in an non-deterministic order. It is also not possible to predict which Events from which targets will arrive at which LAM server. If Complex Rules were clustered in a similar way to the Simple Rules, then it would not be possible to determine where Events are routed into the WAM layer. This behavior makes the evaluation of multiple Events impossible because the single evaluation engines of the WAM servers wait constantly for Events which are routed to other server instances. One possible solution of this problem is that all relevant Events are routed from the LAM layer to all members of the WAM layer. This method increases the communication load and also the Rule evaluation time due to the transaction based Event interlayer communication.

- Rule Firing Conflict: Another problem resulting from the Event routing to all WAM server instances is that each of them has the same Evaluation Network state. This causes multiple firing of Rules and also undesired multiple execution of Actions. A plausible solution of this problem is the development of a firing decision algorithm, which leads to Rule evaluation delays and more communication overhead.

- Complex Rule Subsystem Scalability: One of the most critical points concerns the scalability of the Complex Rule Management subsystem. It is assumed that all WAM instances are clustered, have the same Rule Set and the same state of the Evaluation Network. In this case, the extension of the number of members of the WAM layer does not contribute to deal with increased load and only causes more communication load. This is due to the synchronization and firing decision effort where the new instance also has to participate.

In order to deal with the problems identified before, a master-slave architecture has been developed. This architecture features the following characteristics:

- Master-Slave Clustering: Each Complex Rule evaluation engine (master) has a mirrored evaluation engine (slave) with exactly the same Rule Set. Events are evaluated on the master instance and changes in the Evaluation Network are propagated to the slave instance. In order to ensure data consistency between the Evaluation Network of master and slave, a two-phase commit protocol has been developed. In case of a master failure, the slave instance assumes the Complex Rule evaluation. At this point in time of the SysMES development, it is only possible to define a single slave instance to each master instance. There are no restrictions for the extension to a multi-slave architecture for further development steps. A master-slave architecture has been developed in order to achieve the dependability requirements of all components of the SysMES framework.

- Disjunct Rule Sets: In order to make the Complex Rule Management subsystem scalable, disjunct master Rule Sets are introduced. The processing and evaluation load depends on the number of Complex Rules in a Rule Set. In the case of overloaded evaluation units, a new instance is added to the WAM layer. The next step is the relocation of Complex Rules from the overloaded evaluation instances to the new one.

- Targeted Event Routing: Another important aspect is the routing of Events from the LAM layer to the specific Complex Rule evaluations engines. Events are only sent to evaluation engines if there are Complex Rules in their Rule Set, which are responsible for the evaluation of these Events. For this purpose there is a special kind of LAM Simple Rule, the Routing Rule (see section 5.4.2.4.1). These Rules are created in combination with the Complex Rules and are responsible for routing the Events to the WAM instances where they are needed. The deployment of a Complex Rule is always related to the deployment of the respective Routing Rule. The direct Event routing capability also contributes to making the SysMES framework scalable. This is due to the reduction of transmitted data and communication overhead.

- Multiple WAM Instance Setup: The SysMES framework supports two different setups. These are master-slave and master-only. The first has already been mentioned. The second one is a non dependable setup. This setup can be used if there is no critical data to be analyzed by the Complex Rule Management subsystem. The reason for this configuration is to avoid the master-slave synchronization overhead.

Summary of the section:

The Complex Rule Management subsystem is the third part of the SysMES Rule management. It is responsible for the correlation of Events from distributed targets in order to recognize Global States or Complex States. Complex Rules are stateful because they store partial matches. Complex Rules are defined in an object-oriented Rule definition language. The semantics of this language are related to the object attributes and their values. The syntax is defined by a context free grammar. The evaluation of Complex Rules is based on the Rete algorithm. This standard algorithm is not sufficient for the evaluation of all Rules generated by the context free grammar and therefore the SysMES Complex Rule evaluation algorithm extends it. The additional functionality comprises the integration of set operators, e.g. AVG, COUNT, the invalidation of partial matches based on time attributes and the introduction of delays in order to deal with network latencies. Another important extension concerns the introduction of Operation and Variable classes, which are used for comparisons and calculations on the attribute values of Events. The evaluation algorithm operates on an in-memory and tree-based structure, the Evaluation Network. Each deployed Complex Rule is represented by a sub-graph of the

Evaluation Network. Events are injected into the Leaves of the Evaluation Network. Matches at the Leaves are propagated to the inner nodes where a new evaluation takes place. When the Root Node is reached then the Complex Rule fires. After a firing, there are multiple strategies for dealing with Events involved in the evaluation. These depend on the value of the ConsumptionMode attribute. One strategy, the so-called Unrestricted Consumption Mode deletes all Events involved in a Complex Rule match. The other one, the so-called Unrestricted Detection Mode keep Events and partial matches stored in the Evaluation Network. In order to make the Complex Rule Management subsystem scalable and highly available, a master-slave architecture has been chosen. All Complex Rules are divided into disjunct Rule Sets. Each master has a Rule Set with the Rules to be evaluated. A mirror of this Rule Set is located on the slave. The evaluation occurs in the master node and changes in the Evaluation Network are immediately and transactionaly-based propagated to the slave. In case of master failures, the slave takes over the Complex Rule evaluation.

5.4.2.5. Task Management

The Task Management section introduces the functionality for the configuration of the management environment and for the active interaction between different members of the SysMES framework.

Purpose: The general purpose of the Task Management subsystem is the automatically or manually initiated transmission of management objects from the top of the SysMES architecture to the targets. Targets are the SysMES servers located in the middle layers and the SysMES clients located at the bottom of the architecture.
The system management objects to be transmitted are Monitors, Rules and also several kinds of Actions. Furthermore, the Task Management subsystem is responsible for the distribution of Task objects, which can be used in order to set up the management environment and to execute any other Action.
The transmission of management objects is initiated manually by a system administrator, e.g. to distribute a new Monitor, or automatically by another SysMES subsystem, e.g. the Rule Management subsystem when a Rule fires and an Action has to be performed.
According to the basic Task definition and classification in section 5.3.4, there are two specific purposes for the Task Management subsystem, one configurational and one administrative.

- Configurational Purpose: The Configuration Tasks are used on the one side to set up the system management environment and on the other side for the performing of changes. These are concretely used for the deployment of Monitors, Rules and Actions to the SysMES targets (clients or servers), as well as for changing the attributes and features of the deployed objects and finally, for the elimination of undesired management objects.
- Administrative Purpose: The Administrative Tasks are used by the system administrator for the execution of Actions (binaries, scripts, etc) on the targets. In principle, this kind of Task is an instrument to interact actively with the SysMES targets and to be informed about the result of the execution.

Characteristics:

- Distributed Functionality: The Task Management subsystem has been designed in a distributed manner so that the functionality for the creation and storage of Task objects is located in the WAM layer and the distribution and propagation of those objects at the side of the Task receiver.

If t Task should be deployed to a SysMES client, e.g. the distribution of Monitors, then the distribution and propagation occurs at the LAM layer, otherwise at the WAM layer, e.g. by the distribution of Complex Rules to the WAM servers. The division of the functionality into different layers has been made in order to achieve better scalability due to the distribution of load. Another aspect for the distributed functionality is to achieve the requirements of management close to the originator.

- Top-Down Communication Method: One of the design characteristics of the SysMES architecture described in section 5.2 concerns the usage of XML documents as message data format for the communication between several SysMES layers. The top-down communication is realized by the usage of messages containing the XML representation of the Task objects. These are generated at the Operator Layer at the top of the SysMES architecture (see figure 5.4) and distributed to the targets across all other layers, i.e WAM, LAM, clients. The targets receive the Task documents, parse the contained management object and deploy it.

- Dynamic Distribution: One of the most useful characteristics of the Task Management is the dynamic deployment of management objects, on the fly and without downtime. Each target is able to receive new Tasks and to deploy these, as well as to change the attributes and semantics of deployed objects. For example, the Task for the distribution of the Monitor in figure 5.9 was shown in section 5.3.4. Now it is possible to reconfigure the attributes of the Monitor as well as any other associated object, such as Binary Actions or Event Classes. In order to modify the semantics of the Monitor, it is enough to change the value of the attribute Binary or BinaryName in the Binary Action. In case of timing discrepancies, it is possible to re-set the value of the Period or Repeat and using this method it is possible to reconfigure any kind of management object. The transfer of the changes is done during the re-transfer of the management object by Tasks.

- Guaranteed Task Distribution and Deployment: The distribution of management objects by Tasks is a critical issue because this is the method used for the solution of problems recognized by the Rule Management subsystem and also the way to deploy changes in the management environment and targets. The Task Management subsystem is thus designed to guarantee the distribution and deployment of management objects. As described above, the Task Management functionality is divided into several management sublayers. The transmission of Tasks from the originator to the targets is carried out in a transactional way using the capabilities of the JBoss AS. Another characteristic is the persistent storage of Tasks to avoid information loss. Parallel to the usage of transactions it is necessary to guarantee the successful deployment of Tasks, as well as the execution of the attached Actions. For this purpose a two-phase acknowledgment algorithm has been developed. In the first part, the target acknowledges the reception of the XML Task documents and in the second part it acknowledges the successful execution of the Task. The decision about the usage of the acknowledgment capabilities is made by setting up the Acknowledge attribute and depends on the importance of the Task, and respectively, on the importance of the included management object.

- Automatic Target Recovering and Upgrading: The last important characteristic concerns the capability to distribute Tasks to SysMES targets which are offline at deployment time. The Task Management subsystem utilizes the TaskID attribute of the Tasks to recognize the deployment and execution state of the targets. The SysMES targets store the TaskID of the last successfully executed Task and report this to the servers using the Event attribute CurrenClientTaskID. This

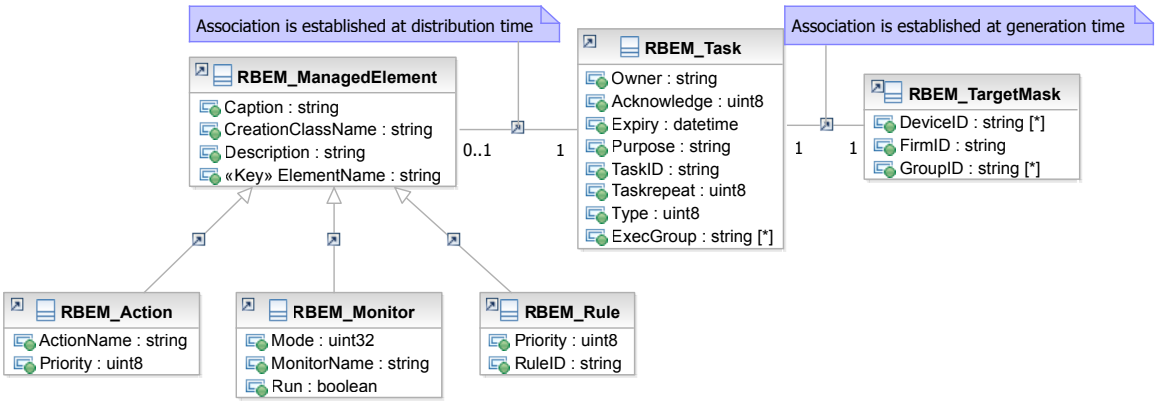


Figure 5.31.: SysMES Task Class

identifier is used on the SysMES server side to decide which Tasks have to be sent. This method allows the automatic upgrade of SysMES targets, as well as to recover their state after a crash or a undesired offline phase.

Functionality: This section begins with the definition and classification of Tasks and the description of the Target Mask class. Following that is the description of the Task Management algorithm used for the deployment, storage and distribution of Tasks, and finally, the description of the Task Management cluster capabilities.

Task Definition and Classification: The term Task describes, in the SysMES framework a container used for the transmission of any other management object to a target with the intention to execute any kind of action, such as the deployment of Monitors and Rules or the execution of binaries and commands.
Tasks are defined in the exact same object-oriented method as the other SysMES management objects. The general structure of Tasks is described in figure 5.31. According to this figure a Task object has two associations to an object of ManagedElement and one to a Target Mask object. The ManagedElement class is an abstract class and therefore the real association is realized with an object of a derived class, i.e. Monitor, Rule or an Action. The second association is realized in order to identify one or a group of targets where the Task has to be transmitted and deployed. Each Target Mask object is described by the following attributes:

- DeviceID: A list of device identifiers. This attribute describes SysMES internal identifiers for several targets. It is possible to define group target masks which contain multiple DeviceID entries. One condition for the usage of multiple DeviceIDs is that all targets in a group have the same GroupID and FirmID.
- FirmID: An identifier for a company or institution that targets belong to.
- GroupID: A list of group identifiers that targets belong to.

In principle, for each target there is one Target Mask object used for its explicit identification, but there are also group target masks which classify multiple targets for a specific purpose or affiliation.

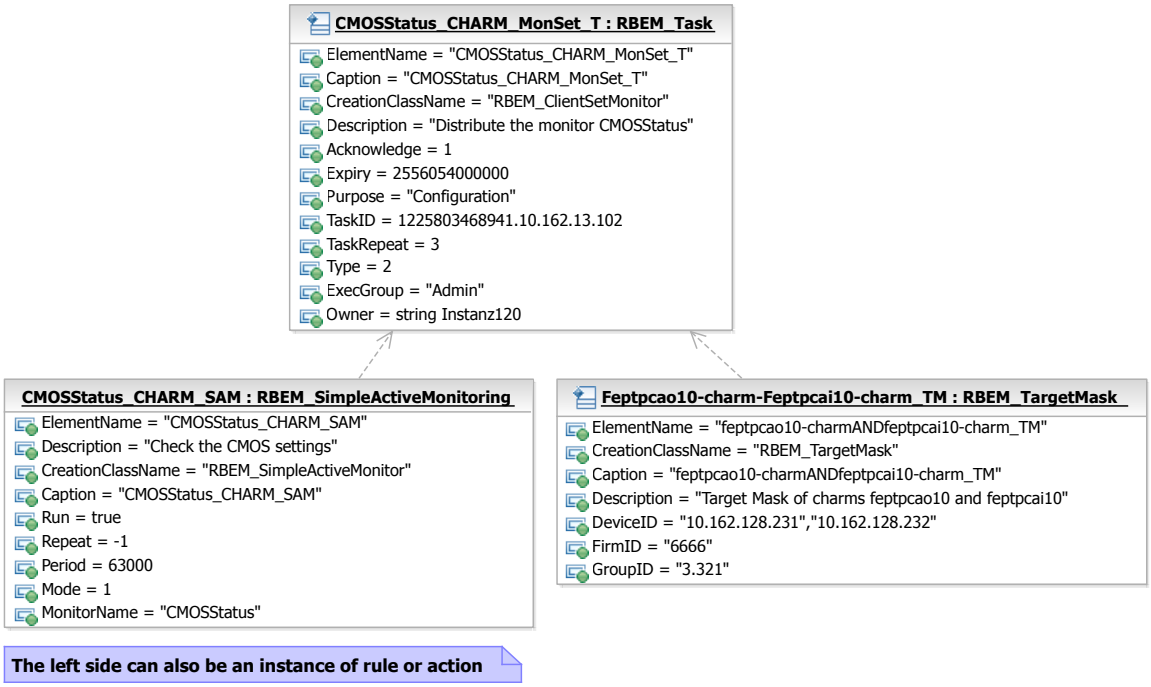


Figure 5.32.: SysMES Task Object

Figure 5.32 visualizes an example of a Task object associated with a Target Mask and a Monitor object. In this example, the Task is responsible for the distribution and configuration of a new Monitor. It will be deployed to the targets feptpcao10-charm with the `"DeviceID=10.162.128.231"` and feptpcai10-charm with the `"DeviceID=10.162.128.232"`.
Figure 5.37 shows the Task hierarchy used in the SysMES framework. Each of these Tasks is derived from the class Task, which contains the following attributes:

- TaskID: This attribute is a unique identifier of the Task object in the SysMES framework and is set up on the server where the Task object is created.
- Acknowledge: This describes the expectation of the SysMES servers to get client Events, which confirm the reception and execution of the Task on the target side. More about the Tasks acknowledgment algorithm can be found in the subsequent sections. `"default value=1 i.e. true"`.
- Expiry: This describes how long the Task is valid. `"default value=Task creation time + 600 sec"`.
- Type: In the SysMES framework there are two type of Tasks, the server `"type=1"` and the client `"type=2"` Tasks.
- TaskRepeat: This attribute describes how often a server tries to deploy the Task to a target `"default value=3"`. It is mainly used for dealing with network connectivity failures.
- Owner and ExecGroup: These two attributes are used in order to define users who are allowed to execute the Tasks. The first attribute contains the creator of the Task who always own the

execution rights. Using the ExecGroup attribute, it is possible to define a user group whose members are permitted to execute the Task. The values of this attribute should match with the user groups of the used authentication mechanism such as LDAP, OS authentication, etc.

- Purpose: This describes for what the Task is used. It can have the values `"Configuration"` and `"Administration"`.

According to their purpose, the SysMES Tasks are classified into Administration Tasks and Configuration Tasks and according to their structure into Dynamic Tasks and Static Tasks.

- Administrative Tasks: These are used by the system administrator or operator in order to interact manually with the SysMES targets. This kind of Task has an association with an Action object, which includes the code to be executed and parameters if needed. The SysMES targets execute the Action and send the return value to the server using TaskReply Events. In case of errors during the execution, the targets send the error code in Error Events. Administrative Tasks expire automatically after their deployment and therefore are ignored for the automatic update and recovery capabilities.
- Configuration Tasks: These are used for the configuration of both the management capabilities of the SysMES targets (i.e. Rules, Monitors) and the characteristics of the hosts (i.e. daemon configuration, network settings, etc.). Another usage of Configuration Tasks is for the reconfiguration and deletion of management resources. These Tasks are taken into account for the automatic update of targets and do not expire after their distribution.
- Dynamic and Static Tasks: The distinction between Dynamic Tasks and Static Tasks is related to the capabilities to change the attributes or associations of the management objects to be deployed and consequently to change their semantics. The SysMES framework has a collection of Static Tasks with static semantics, which are mainly used in order to get information about the targets. The behavior of Static Tasks are specified in the Task objects and therefore they do not associate another management objects. Examples of Static Tasks are `"ClientGetAllMonitors"` and `"ClientGetCache"` to retrieve the Monitors configuration or Events stored in the Event Cache. In principle, the Static Tasks are a collection of basic Tasks which are included in each installation of the SysMES framework and the Dynamic Tasks are created according to the requirements of the specific environment to be managed. Dynamic Tasks can be changed, removed, extended and reconfigured at any time.

Task Management Algorithm: The Task Management algorithm was designed based on the general SysMES design considerations of chapter 4. Concerning decentralization and location independent management it is important to consider that each SysMES target is able to request system management support from an arbitrary server. Each of these servers should have the capabilities for recognizing which Tasks have to be sent to the connected clients. Concerning the principle of management close to the initiator, it is important to offer different possibilities to initiate the deployment of Tasks for different initiators, such as operators on the GUI or a Rule subsystem. In order to achieve the requirement of scalability and dependability, it is necessary to consider the clustering of Task Management resources, as well as the reduction of data to be exchanged between the Management Layer and clients.
The SysMES Task Management algorithm is divided into four parts concerning the Task creation, deployment, storage and distribution. The sequence diagram 5.33 is used to visualize these and their

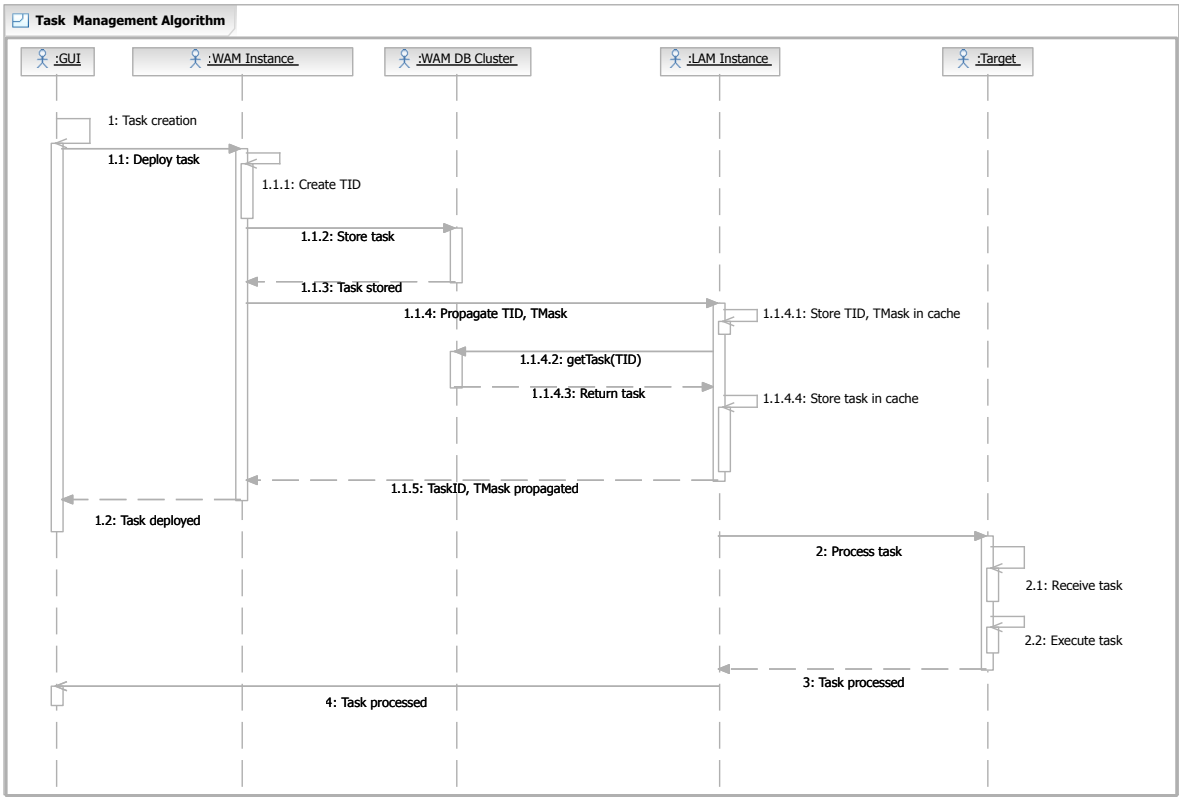


Figure 5.33.: Task Management Algorithm

chronological execution order as well as the involved actuators. This diagram describes the creation and deployment of a Task in the GUI which needs to be acknowledged and sent to a connected target.

Task Creation:

The SysMES framework offers two methods for the creation of Tasks. The first method is the instantiation of the management model in the Operator Layer 5.5.1.1. The object model stores the Task objects as well as the associated objects, i.e. Monitors, Rules and Actions. Other stored objects are the Target Masks, which describe where Tasks can be deployed. This method is used by the GUI when a system administrator deploys Tasks manually.
The second method is the usage of an Application Programming Interface (API). The user of this method is in charge of the creation of a Task XML document according to the syntax of the management model. These Tasks are not stored into the management model and therefore cannot be reused. This method is used by the SysMES Rule Management subsystem in order to react to a recognized state.

Task Deployment:

Independently from its creation method and according to the location independent system management, the generated Task is deployed to one arbitrary WAM server instance which is member of the WAM layer.

In the case of a model managed Task, the association between Tasks and Target Masks is realized at deployment time by the deployment initiator. After that, the Task object is deployed to the WAM layer for its further processing.
The next step concerns the generation of a unique Task identifier (TaskID). The Task Management algorithm requires the generation of strictly monotonic increasing TaskIDs related to the Task deployment order in a decentralized environment. For this purpose the WAM servers have the capability to assign TaskIDs composed of a time component (deployment current time) and a server identifier (i.e. server IP address). An example of this is `"TaskID=1233684185345.10.162.13.102"` which means that this Task was created at `"3 FEB 2009 19:03:05.345"` on the server `"10.162.13.102"`. The server identifier is defined in a server configuration file and therefore its IP address is not necessary for the value of its DeviceID attribute. Using this method, it is possible to deploy several Tasks at the exact same time, but in different server instances. The Task deployment within a server instance is realized sequentially and according to the TaskID order. This is important in order to define dependencies between the execution of Tasks as well as to determine the Task execution state of a target for the Task distribution introduced below. The following two steps - storage and propagation - are also initiated in the WAM instance during the deployment phase.

Task Storage:

All generated Tasks are stored in the WAM database cluster. For this purpose the WAM server instance which has deployed the Task sends the storage request to the WAM cluster so that the WAM instance with the lowest load starts the storing process. This way has been chosen in order to redistribute the load generated during the deployment and storage activities to more than one server if necessary.

Task Propagation

Conforming to the connection algorithm introduced in section 5.4.1, there is a one-to-one connection between a client and a server. The distribution of client connections is only related to the network and server load and in case of a connectivity failure, the client opens a new one to the server with the lowest load. It is not possible to define a fixed relation between clients and servers and therefore it is necessary to propagate a minimum of information about deployed Tasks to all instances of the LAM layer. In order to publish the deployment of a new Task, the Task Management algorithm propagates the TaskID and the Target Mask to all LAM instances. This information allows the LAM server to identify if a new deployed Task has to be sent to connected clients. Each LAM server stores this information in an ordered list called Task Cache and uses it to compare the Target Mask with the connected targets. With the first match, the server accesses the database and retrieves the Task object associated with the TaskID, stores it in the Task Cache and sends it to the target. In case of further matches, i.e. when the Task should be sent to several targets, the LAM server instance utilizes the Task object from the Task Cache and sends it without accessing the database. The size of the Task Cache is a parameter defined in the SysMES configuration file `"default=100"`. The Task Cache has the attribute LowestTaskID, which represents the TaskID of its oldest Task.
This part of the algorithm is a consequence of the expected location independent system management, but also as a procedure to reduce the amount of accesses to the database cluster, and consequently, to reduce the amount of information to be sent to all LAM instances.

Task Distribution:

The Task distribution process requires information about the Task execution state of the targets. This is stored locally in each SysMES client in an attribute named CurrentClientTID. It is set to the last TaskID executed successfully.

Targets report their CurrentClientTID to the servers in each Event which is sent to the server so that LAM server instances always have the information about the execution state of the connected targets.

When a new SysMES client establishes a connection to an arbitrary server, it sends an Alive Event containing its actual CurrentClientTID. The Task Management subsystem compares it with the LowestTaskID of the Task Cache. If the CurrentClientTID is greater or equal the LowestTaskID of the Task Cache then only the Tasks of the Task Cache are analyzed in order to find out which of these have to be sent to the target. Otherwise the Task Management subsystem retrieves all Tasks from the database cluster with a TaskID greater than CurrentClientTID and less or equal than the Task Cache's LowestTaskID, and processes these.

This method has been developed in order to distribute Tasks to targets which were offline at the Task deployment time. It is the basic method to keep the SysMES clients always updated.

The distribution of selected Tasks for a specific client is carried out sequentially and considering the TaskID order. The LAM server instance checks the validity of the value from the attribute Expiry for each Task and only the Tasks with an Expiry value greater than the current system time are taken into account for a distribution. Another relevant attribute for the Task distribution is TaskRepeat. This is used for an individual Task configuration concerning how often SysMES servers try to send the Task in case of failures or timeouts.

The acknowledgment algorithm for Tasks has been developed as a two-phase process. The SysMES client sends an Ack Event with the message `"received"` if the Task was transmitted correctly and has passed a syntax check. In the second phase the client sends an Ack Event with the message `"executed"` and sets its CurrentClientTID to the TaskID of this Task. In case of errors during the Task execution, the SysMES client sends an Ack Event with the message `"error"` and the CurrentClientTID remains unchanged.

If the Acknowledge attribute is set to `"1"`, the server sends the Task and waits until a Ack Event (received) arrives. The default value to wait is `"15 seconds"`. However, if the Task is associated to a Binary Action object, then the LAM server instance waits as long as the value of the Timeout attribute. The justification of a longer wait interval is because the execution of a binary can take longer than the default value. If no Ack Event (received) arrives within the time interval, then the server tries to send it again as often as declared in the Task attribute TaskRepeat. If the send process still fails, the server generates an Error Event in order to inform an administrator about the problem.

After the arrival of the Ack Event (received), the server blocks the transmission of other Tasks to the same client until the second Ack Event (executed) arrives. Note that the distribution of Tasks works parallel for different clients and it is sequential only for the distribution of several Tasks to one client and therefore the other clients are not affected if the Task submission fails. Similar to transmission errors, the targets generate an Error Event in case of a failure during the Task execution.

If the Acknowledge attribute is set to `"0"`, the server sends Tasks asynchronously without waiting for a successful Task execution.

The acknowledge algorithm was designed in order to allow the definition of Task dependencies. The implementation of the algorithm ensures the ordered execution of Tasks according to the value of their attributes.

The decision about the usage of the Acknowledge attribute should be made according to the importance of Tasks because of the overhead for the two-phase acknowledgment. All automatically generated Tasks, i.e. for the execution of Rule Actions, are classified as very important and therefore expect acknowledgment.

Task Management Clustering: The clustering of the Task capabilities requires the usage of a clustered database for the storage of Tasks. The algorithm for distributed generation of TaskIDs requires time synchronization for the system clock of the servers because Tasks are distributed according to the TaskIDs. Each member of the WAM and LAM layer is equipped with the whole functionality for working in a stand alone mode as well as clustered one. The clustering of the Task Management subsystem is carried out by the utilization of the clustering capabilities of the JBoss AS where these are running on. In the case of a very small environment to be managed, the SysMES framework allows the unification of the WAM and LAM layer to a single layer which contains the functionality of both.

Summary of the section:

Tasks are management objects used for the distribution, configuration and reconfiguration of other management objects, i.e. Monitors and Rules. Another usage of Tasks is the active interaction with the SysMES targets. There are two forms of active interaction: manual interaction initiated by a system administrator and automatic interaction initiated by the Rule Management subsystem. Each Task has an association with an object of the type Target Mask, which describes where it has to be sent. The SysMES framework has different types of Tasks. According to its purpose, there are Configuration Tasks and Administration Tasks and according to its structure, there are Static Tasks and Dynamic Tasks. Configuration Tasks are used to set or change the characteristics of management objects, as well as the characteristics of the host operating system. The Administration Tasks are mostly used by the administrators in order to execute any kind of Action on the targets and to get the execution result in form of Events, i.e. the execution of the `"ps -ef"` command. The most used Administrative Task class is the Simple Task. Each Simple Task object has an association with a Binary Action object, which is used to distribute any binary (coded in base64 [18]). The distinction between Dynamic Tasks and Static Tasks reflect the capability to change the Task attributes and consequently the semantics of the Task at runtime. The Task Management algorithm is developed in a distributed way. The Task creation occurs in the Operator Layer, the storage, deployment and propagation in the WAM layer and the distribution in the LAM layer. In order to ensure the transmission and correct execution of Tasks, a two-phase acknowledgment algorithm was developed. The Task Management algorithm reduces the amount of generated traffic because only the absolutely necessary data is transmitted to all server instances in a cluster and contributes therefore to the compliance of the requirements in terms of scalability, dependability and location independent management.

5.5. Operator Layer

The Operator Layer builds the top of the SysMES framework architecture. It is divided into two sublayers: the Modeling Layer and the Graphical User Interface (GUI) Layer. This section introduces first the modeling technologies used and the SysMES management model. The second part is dedicated to the GUI where operators can interact with the SysMES framework in a manual way.

5.5.1. Modeling Layer

The Modeling Layer (commonly referred to as the CIM Layer) is one of the top layers of the Operator Layer and is composed of the services and technologies needed for the development, hosting and management of the system management model. Due to the requirements of dependability, the Modeling Layer has been designed using the Distributed Management Task Force (DMTF) technologies, which allow the usage of distributed and redundant models. These technologies are the CIM, CIMOM and the WBEM which are introduced here.

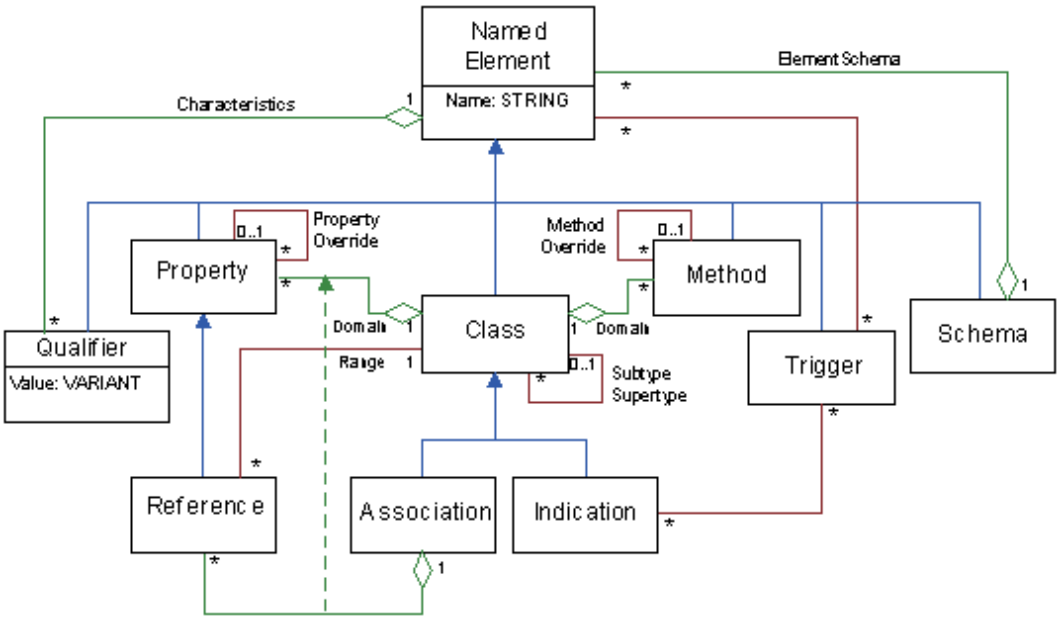


Figure 5.34.: Common Information Model - Meta Schema Diagram

Common Information Model (CIM): The CIM is an object-oriented standard for the description of management data, based and realized using UML class diagrams. It has been developed by the DMTF in order to achieve interoperability and information sharing between the management systems and other system management providers. The CIM is a conceptual model which allow the definition of relationships between several objects of the management model independent of its nature, e.g. the relationship between applications to be managed and devices where these are running on.
The CIM consists of an infrastructure specification and a schema. The specification language and the method to describe the management data are subject of the infrastructure specification. The schema is used for the definition of CIM based models.

1. CIM Infrastructure Specification: It defines the syntax, semantics and rules of the model and also a language and a method to describe the management data. Furthermore, the specification describes the details for the integration with other management models [37].

   The CIM specification comprehends the CIM meta schema [35] and the schema elements displayed in figure 5.34 [9], Managed Object Format (MOF) as the specification language and the CIM naming method.

   The CIM Meta Schema: It describes the elementary elements and relationships used by the model development. These elements are Schema, Class, Property, Method and Qualifier. Furthermore, the CIM meta schema includes Association and Indication as two special elements of Class and Reference as a special Property. More information about the CIM Meta Schema can be found at [35]

   The CIM MOF: The MOF is a textual language based on the Interface Definition Language (IDL) [60] and has been primarily developed for the representation of classes and instances of a model or the customer model extensions.

[9] Picture is copyright of the DMTF.

```
// ==================================
// RBEM_TargetMask
// ==================================

[Description ("no description available")]
class RBEM_TargetMask : RBEM_ManagedElement {

   [Description ("no description available")]
   string DeviceID[];

   [Description ("no description available")]
   string FirmID[];

   [Description ("no description available")]
   string GroupID[];
   };
```

```
instance of RBEM_TargetMask {
   DeviceID = {"10.162.141.96"};

   FirmID = {"6666"};

   GroupID = {"3.321"};

   Caption = "cndev0-charm_TM";

   CreationClassName = "RBEM_TargetMask";

   Description = "Target Mask for the Node cndev0-charm";

   ElementName = "cndev0-charm_TM";

};
```

Figure 5.35.: The CIM Managed Object - Syntax and Example

The main components of a MOF specification are textual descriptions of element qualifiers (meta-data about classes, properties, methods, etc.), comments and compiler directives, and the specific class and instance definitions that convey the semantics of the CIM Schema [78].

Figure 5.35 shows a sample how classes and instances can be defined in MOF. The left side of the figure shows the syntax used for the definition of classes and the right side for its instances. The classes and instances defined in MOF are compiled using the MOF compiler in order to submit these to a CIM Model in a CIM server.

2. CIM Schema: The CIM Schema is a well-structured collection of different models and basically a set of classes, associations, methods and properties by which it is possible to build a specific object model of the environment to be managed. The most important profit of using the CIM schema and its models is that the managed data are described in a standard form which contributes to a better interoperability between different subsystems or vendors.

   The CIM schema is basically composed of the Core Model and the Common Model. It also contains the Extension Schema for a customer specific extension of the common model.

   Core Model: The Core Model defines a basic model, which is used in all areas of system management and environment modeling. Furthermore, the core model is used for the definition of complex or specific modeling elements, e.g. the different sub models of the common model.

   Common Model: The Common Models are information models that capture notions that are common to particular management areas, but independent of any particular technology or implementation. The specific common models include applications, database, device, event, interop, metrics, networks, physical, policy, support, system and user. The classes, associations, properties and methods in the Common Models are intended to provide a view of the area that is detailed enough to be used as a basis for program design and, in some cases, implementation [37].

   The complete specification of the common model can be found at [36].

   Extension Schema: The Extension schema is an interface for the developer which can be used in order to extend or customize the common model. Normally the extension will be used in order to add a proprietary schema, which defines the own managements needs.

In the case of the SysMES Framework and in order to fulfill the modeling requirements for a distributed system management framework, a new schema has been developed. This schema is called the Rule Based Event Management (RBEM) Model Schema and will be introduced in chapter 5.5.1.1.

CIM Server The CIM server is primarily responsible for hosting the CIM Model and it extensions, the instantiation, storage and management of the model object and the interaction local or remote with CIM model and objects. The CIM server is concerned with the reliability and availability of the CIM model and its objects.
The CIM server is based on the next three DMTF technologies. A short description of these technologies is given as follows and further information can be found in [42] and [103].

- Common Information Model (CIM) Repository: This repository is a database for the storage of models. These models are the meta information for the creation of objects. In principle, it stores a MOF based representation of the introduced models.
- CIM Object Manager (CIMOM): It is composed of an object database, where class instances of the models are stored, and a set of tools for accessing and managing objects, i.e. object creation, changing and deletion.
- Web Based Enterprise Management (WBEM): This is a collection of standards which describes methods for accessing a CIMOM. Examples of these are CIM-XML as a data format, CIM Query Language, etc.

5.5.1.1. Rule Based Event Management

The RBEM is the system management model developed for the SysMES framework. It is based and built on the extension schema of the CIM model and consists of classes and associations, which describe the management resources such as Monitors, Tasks and Rules.
All classes of the RBEM model are derived from the super class RBEM_ManagedElement. This class has four basic attributes, ElementName, which represents the primary and unique key of each object, CreationClassName, which contains the name of the class which the objects belongs to, Caption and Description, which contains object information for the graphical user interface.
The RBEM model knows one specific type of association class, the RBEM_ConcreteDependency derived from CIM_Association. The CIM model does not provide classes to build directional associations. The RBEM_ConcreteDependency extends the association classes of the CIM model in order to define a directional one-to-one association between objects of the management model. This class has two attributes: Antecedent and Dependent which have to be set with the primary key (in this case the ElementName) of the objects to be associated. The Antecedent attribute identifies the object at the origin of the association and Dependent the end of the association. Directional associations are needed in order to model a tree-based and cycle-free object hierarchy, which represent the SysMES management objects.
Figures 5.36 and 5.37 visualize the inheritance hierarchy containing the super class and all SysMES specific classes. The second level in the inheritance hierarchy reflects the main SysMES management objects such as Monitors, Rules and Tasks as well as others used to build these, such as Event Class, Trigger, Action, etc.
The lower sublayers are used in order to build an organizational structure related to the nature of the functionality behind the tree branches.

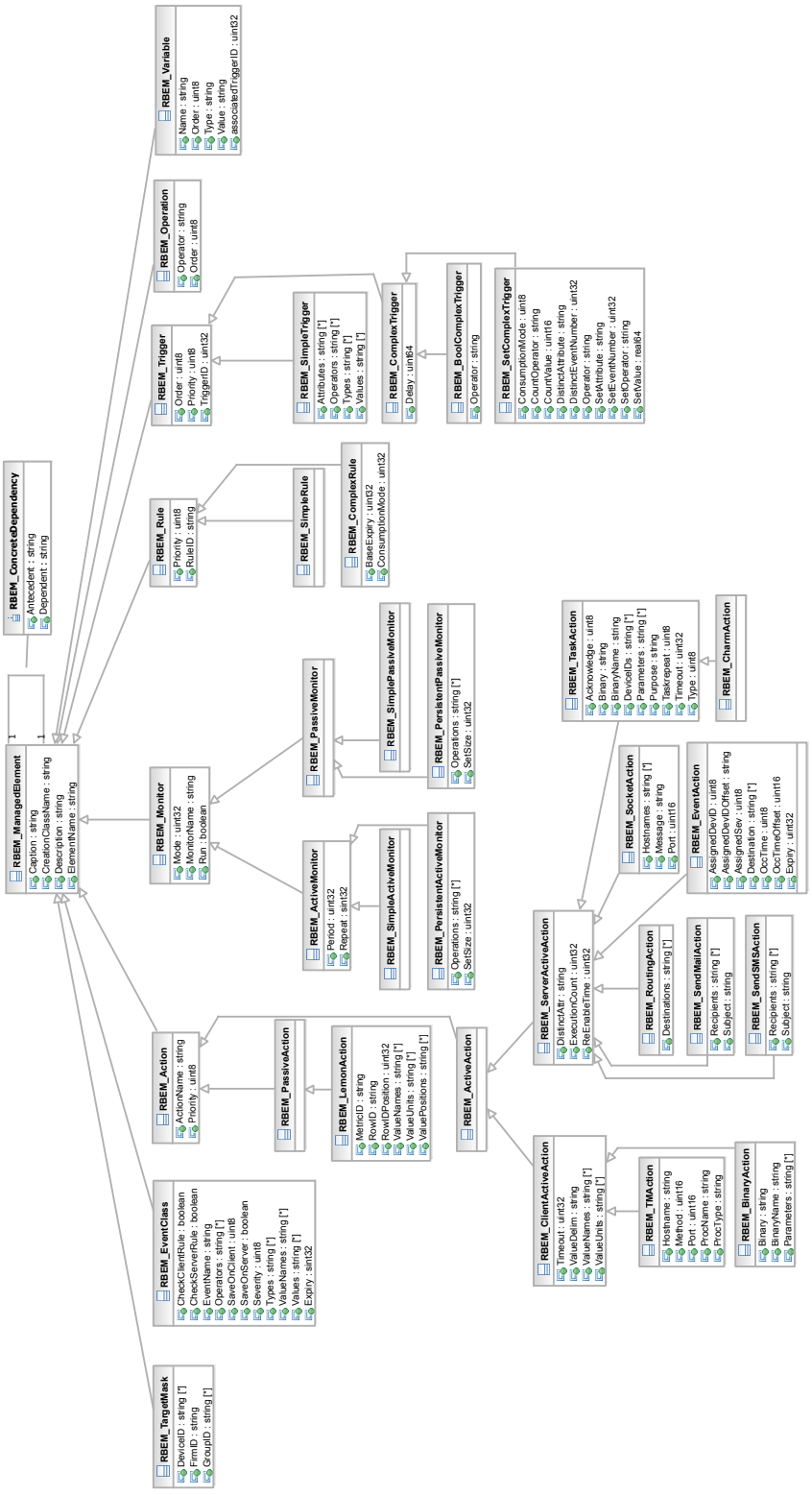


Figure 5.36.: RBEM Model First Part

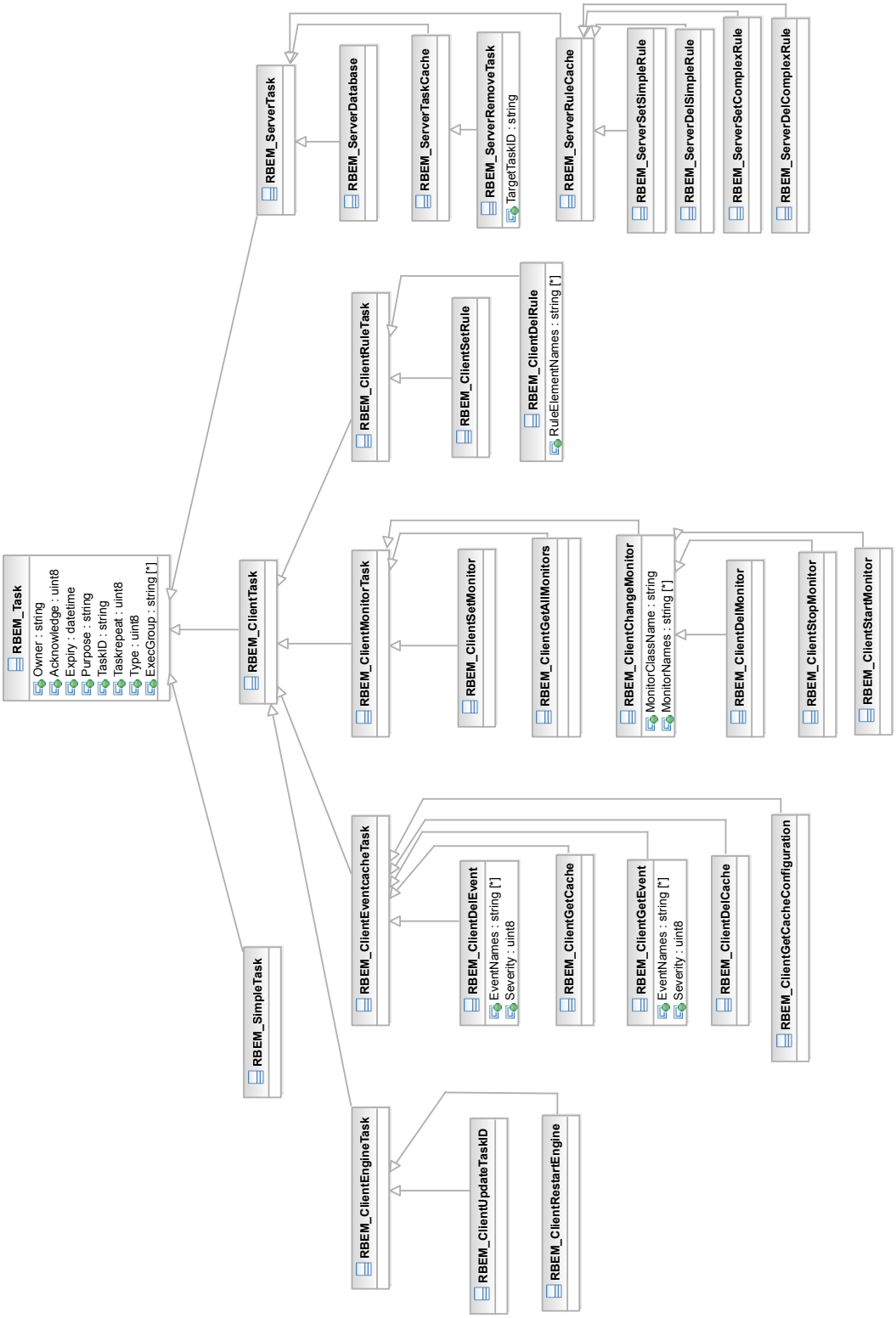


Figure 5.37.: RBEM Model Second Part

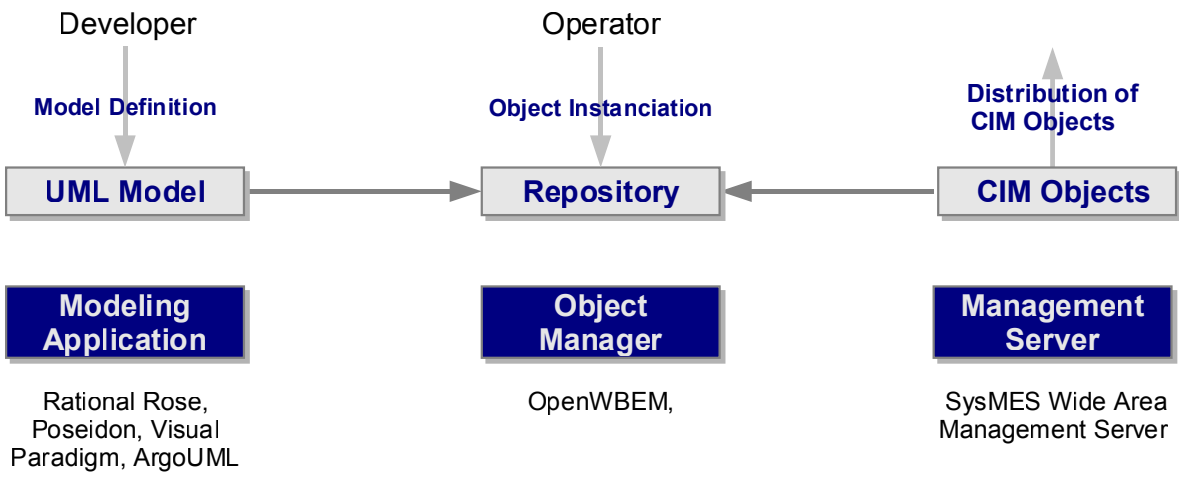


Figure 5.38.: RBEM Model Creation, Instantiation and Distribution

For example, Task objects are used to transmit management objects to targets. The RBEM_Task branch is divided into RBEM_ClientTask, RBEM_ServerTask and the general purpose RBEM_SimpleTask in order to organize the Tasks according to the targets where these have to be deployed.
The Rule Management subsystem is able to process single or correlated Events and therefore the RBEM_Rule branch is divided into RBEM_SimpleRule and RBEM_ComplexRule.
In previous sections and figures 5.7, 5.23, 5.31, the relationship between several objects of the RBEM classes and the semantics of the class attributes have been introduced and therefore the emphasis of this section is the management of the RBEM model.
The management of the RBEM model is visualized in figure 5.38 [111] and is composed of the following steps:

- Model Definition: The developer of system management services is in charge of the definition of the system management management model. In the case of the SysMES framework, this concerns the definition of the RBEM UML class diagram. The model definition can be done using a common UML tool. The newly developed model has to be stored in the manufacturer-independent XML Metadata Interchange format (XMI) [107].

  The next step concerns the exportation of the UML model to MOF format used by the CIM model and server. In context of the development of the SysMES framework, the tool XMI2MOF [112] was developed. This tool allows the generation of MOF from UML XMI format. It has been accepted by the Open Group and integrated into the offered WBEM tools [108].

- Model Instantiation: The instantiation of the RBEM model is realized using the capabilities offered by the CIM Object Manager. In the case of the SysMES framework, the openWBEM [104] implementation of the CIM server and object manager is used.

  In principle, a SysMES administrator or operator can build new management objects in MOF format and export these to the CIM server. OpenWBEM offers a command-line tool (called "owmofc") to compile and import the MOF objects into the CIM object manager. The right side of figure 5.35 visualizes how the objects can be built. In this case, the operator defines a new object of type RBEM_TargetMask. Associations between objects are realized by the definition of instances of RBEM_ConcreteDependency. An association object utilizes the value of the

attributes Antecedent and Dependent in order to identify the associated objects. Changes to the attribute values and object deletion can also be done by the OpenWBEM command-line tool.

- Object Distribution: The access to the management objects is realized by the OpenWBEM API. Depending on the tasks to be done, the SysMES WAM server or the GUI has access to the required object as well as to all associated objects. The XML representation of these objects is transfered to the SysMES framework and distributed to the desired locations. For example, if an operator decides to send a new Monitor object to a SysMES target, the GUI accesses the CIM, reads out the Task for this purpose (RBEM_ClientSetMonitor) and deploys it to any WAM server. The SysMES GUI is also able to perform minor changes in the stored CIM objects and to propagate the changes to the CIM server. At this time in the development, it is not possible to build new objects in the SysMES GUI. These capabilities are subject of the new development works described in the outlook chapter.

More information about the used standards and technologies as well as about the RBEM Model can be found in [111].

Summary of the section:

SysMES management objects are modeled in an object-oriented way. The RBEM is the specific model that SysMES management objects are built from. This model has been developed based on the extension schema of the CIM. RBEM is a class model containing both the classes for the creation of management objects, i.e. RBEM_Rule, RBEM_Monitor, RBEM_Tasks, and classes for the definition of relationships between management objects, i.e. RBEM_ConcreteDependency. All classes of the RBEM class model are derived from a super class RBEM_ManagedElement, which contains four basic attributes: ElementName, CreationClassName, Caption and Description. RBEM_ConcreteDependency is a special kind of association which extends the association capabilities of CIM in order to allow users to create directional associations. This kind of association is used for the creation of a tree-based, ordered and cycle-free object hierarchy. The RBEM_ConcreteDependency class has the attributes Antecedent and Dependent for the definition of the source object and target object. The RBEM class model defines a class hierarchy in order to specify management objects according to their functionality. The definition and instantiation of the RBEM model as well as the distribution of created objects are task of the system administrator. The first step for building a management model concerns the definition of an UML class diagram, which represents the needed management functionality. The second step is the export of this UML class diagram into the CIM MOF. For this purpose a tool named XMI2MOF has been developed. The MOF version of the RBEM model is exported to the CIM server where it is hosted and managed. The instantiation of the model is realized by the definition of further MOF documents which describe the desired objects and associations and the insertion of those into the CIM server. The CIM server provides an interface to interact with created objects, i.e changing their attributes and associations and an interface for the distribution of these objects to the SysMES framework.

5.5.2. Graphical User Interface

The SysMES GUI builds the last sublayer in the Operator Layer. It is the interface for operators for interacting manually with the SysMES framework. The SysMES GUI has been developed in order to visualize information, undesirable states and errors and for the deployment of configurations to SysMES targets. It is divided into the following parts:

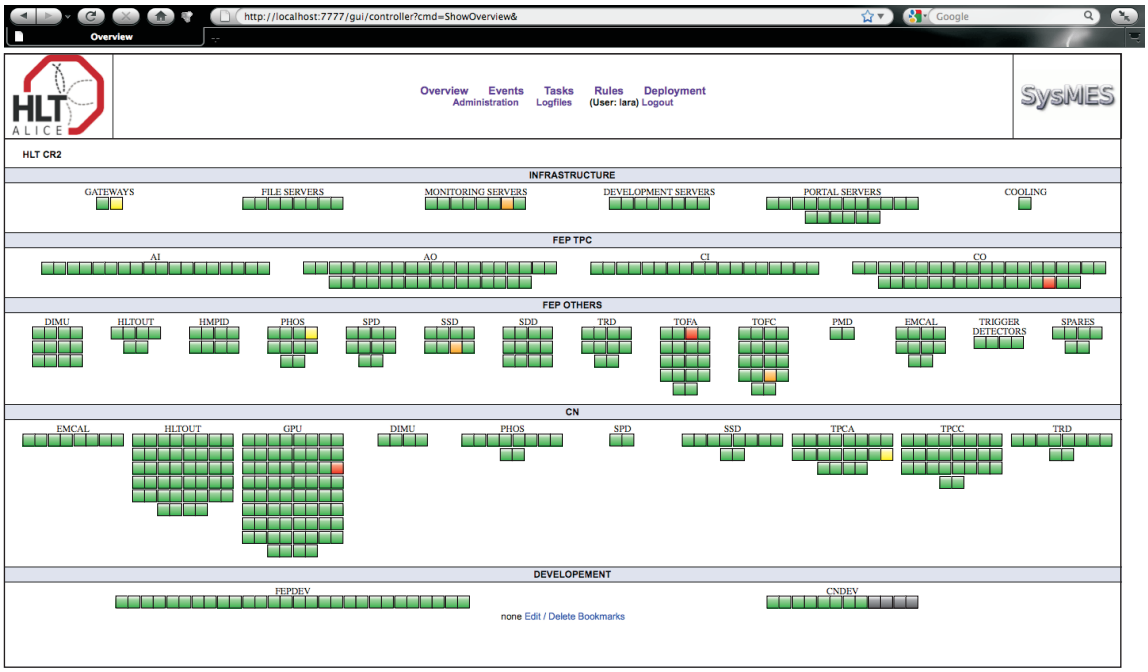


Figure 5.39.: SysMES Graphical User Interface - Overview

- Overview: The overview page displays the current state of the SysMES targets. Figure 5.39 illustrates the overview page for the ALICE HLT Cluster of section 7.1.1.

  Each square represents one SysMES target. The overview configuration and the division of the targets in groups is realized using a XML configuration file. The color of the squares represents the current state of the targets as follows:

  - Green: Target is running properly or only Information Events are available.
  - Black: Target not running or not available.
  - Grey: Target in maintenance mode or it has been removed temporarily from the cluster.
  - Red: Immediate Events available. Target is in a fatal state.
  - Orange: Critical Events available. Target is in a critical state.
  - Yellow: Service Events available. Target is in an abnormal but non-critical state.

  The SysMES GUI utilizes received Events to diagnose the state of the targets. The target availability state is defined by the Alive Events. Each target sends such Events periodically according to a configured time interval `"default value=60 sec"`. In case the SysMES server does not receive Alive Events from a target, then its square is displayed with black color.

  The square of a target which has sent Monitoring Events is displayed in one of the four colors which represent Event occurrences. The current color of a target square is related to the Event with the highest severity `"Immediate=red"`, `"Critical=orange"`, `"Service=yellow"` and `"Information=green"`.

The overview window is used for immediately recognizing the state of the targets using the color codes. By clicking on the respective squares a new window appears. It contains a full description of current Events and also links to other offered services, i.e. vncviewer, Lemon page, SSH, etc.

At the bottom of the overview site, there are operator "Bookmarks". These are shortcuts for often used Tasks. The definition of these bookmarks can be done in the forthcoming deployment section.

- Events: This page is used for displaying Events of all targets and to perform changes on Events, i.e. changing the Event status.

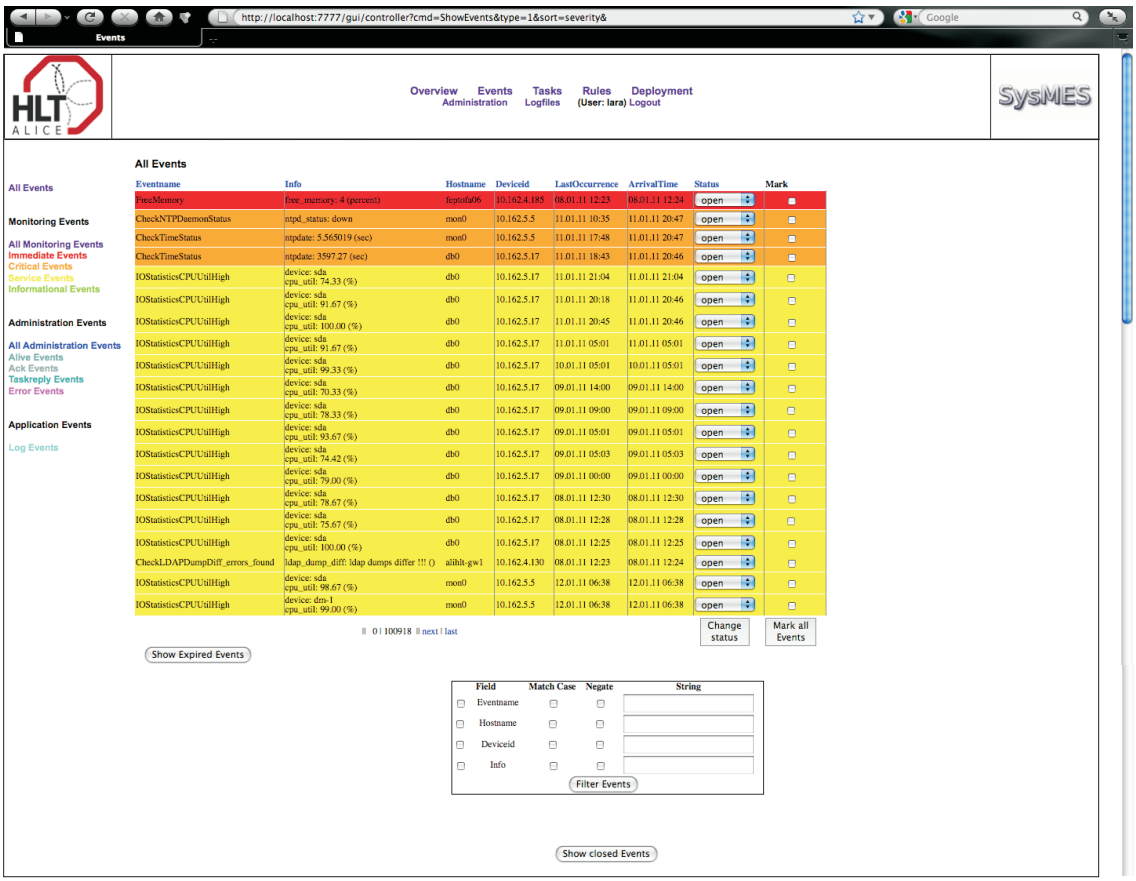


Figure 5.40.: SysMES Graphical User Interface - Events

Figure 5.40 illustrates the layout of several Event viewers. The left side of the window is divided into three categories: Monitoring Events, Administrative Events and Application Events. Monitoring Events are grouped according to their Severity in the four introduced classes: Immediate, Critical, Service and Information. Administration Events are used in order to inform the operator about the availability of targets (Alive Events), the delivering and execution state of Tasks (Acknowledge Events), its return value (TaskReply Events) and also about possible errors during the Task or monitoring execution (Error Events). Application Events is a ge-

neral category utilized for the integration of data from external applications into the SysMES framework.

The Event viewer displays in columns the same information for all kinds of Events and provides the same actions to be performed. This information is: Eventname, Info, Hostname, DeviceID, LastOccurrence, ArrivalTime, Status. A click on a column causes sorting the content of the Event viewer window by the chosen Event attribute.

The operator has the possibility to select Events on the right side of the viewer by ticking a box. It is also possible to perform changes in the Event status setting it to {`"open"`, `"closed"` or `"in work"`}. Another feature is a search mask located on the bottom of the site, which allows the operator to filter relevant Events.

- Tasks: The Tasks window is used for displaying which Tasks are deployed to targets. The most important feature of this site is the indication of the acknowledgment state of Tasks. SysMES targets send Ack Events with the content {`"received"`, `"executed"` or `"error"`} in case the Task acknowledgment attribute is set to the value `"1"` (which means true). The Task window of the SysMES GUI parses these Events and displays three lists; one for each of these states. The sum of all entries in the list represent the number of targets in the Target Mask object of the Task. The operator can recognize immediately if there were problems in the distribution and execution of the Tasks.

  Besides the Task Name and identifier, other information is also displayed. This is the user name of the operator who deployed the Task, the deployment time and the Task expiration time.

- Rules: This site visualizes all deployed Rules. It is divided into four categories: All Rules, Client Rules, Server Simple Rules and Server Complex Rules. This table displays the Rule Name, the associated Action, as well as the target where the Rule has been deployed.

- Deployment: The deployment window builds the operator interface in order to interact manually with the SysMES targets and servers. The figure 5.41 visualizes the layout of this window. It is divided into two parts.

  The left side of the windows is used in order to choose resources to be distributed and the right side is used for the selection of targets for the resources.

  The left side contains the SysMES resources which can be distributed to the targets. These are: Tasks (more exactly the Simple Tasks), Monitors, Simple Rules and Complex Rules. Remember that the distribution of objects in the SysMES framework is based on the Tasks and the Task Management capability and therefore the distribution of Monitor or Rule objects means the deployment of a Task for this purpose.

  In figure 5.41, the menu list ① is used in order to display the object of the respective category in the box located below ②.

  Multiple objects can be selected (ctrl + mouse left or shift + up) and added to the deployment list by pressing the "Add to list" button ③. It is also possible to select multiple objects from different categories.

  In the case of Monitors and Rules, there are other actions to be performed. This appears in the respective selection window. The Monitor window offers options for `"setting"`, `"deleting"`, `"starting"` and `"stopping"` of Monitors. Similar for the Rules, it is possible to perform `"set"` and `"delete"` actions.

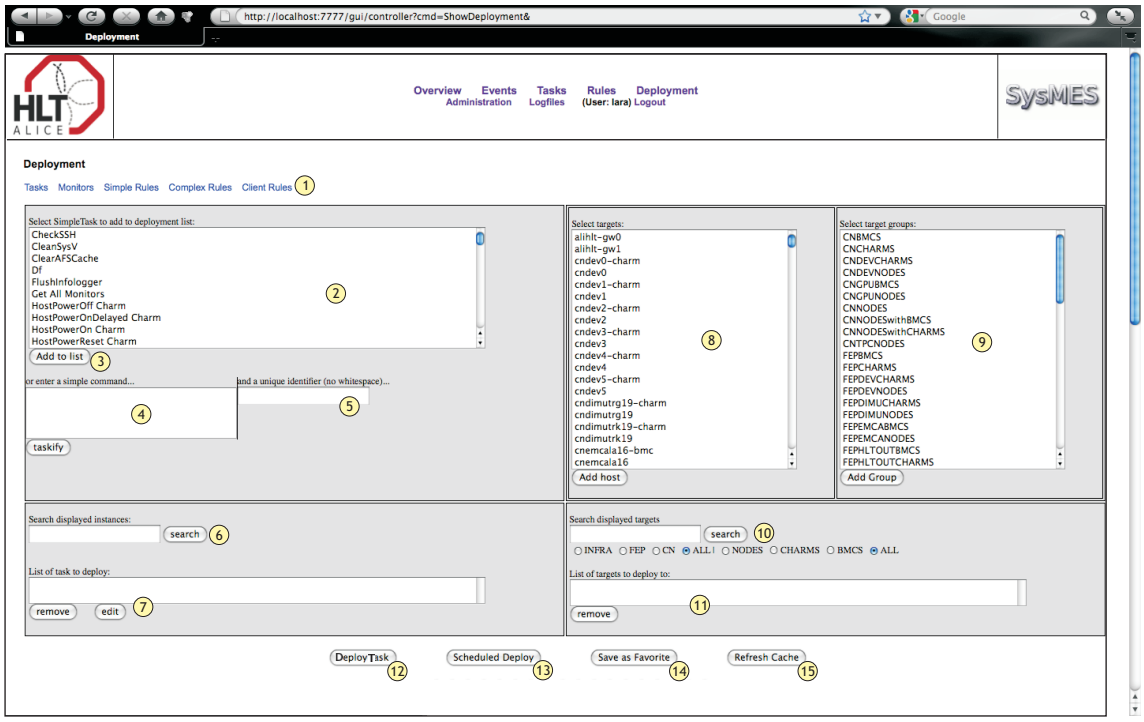


Figure 5.41.: SysMES Graphical User Interface - Deployment

The SysMES GUI offers the capability for a quick creation of Simple Tasks by giving a command in the box ④ and an unique identifier for the Task in the box ⑤

The "search" box ⑥ is used in order to find objects in the current selected object category. The box ⑦ displays all objects to be deployed. There it is possible to set or change the attributes Acknowledge and Expiry. The default value of acknowledge is set to `"acknowledge=1"`. The default value of Expiry depends on the kind of Tasks to be distributed and varies between `"60 sec"` and a couple of `"months"` depending on the Task to be deployed.

The right side of the deployment window is used in order to select the targets where the selected management objects have to be deployed. There it is possible to select single targets ⑧ or groups of targets ⑨. It is also possible to select multiple targets as well as to mix single targets and groups. The search box and the radio buttons ⑩ are introduced in order to simplify the selection of targets in large environments. The box ⑪ displays selected targets.

After the selection of management objects and targets follows their deployment. It will be initiated by pressing the "Deploy Task" button ⑫. The GUI accesses the RBEM model using the WBEM interfaces. It fetches the selected management objects from the right side and associates these with the Target Mask objects from the right side and injects these into the Task Management subsystem on the WAM server. The storage and distribution of these objects are carried out by the algorithms described in section 5.4.2.5.

Besides the manual and immediate deployment, it is possible to define a future deployment time by pressing the "Scheduled Deploy" button ⑬. The operator is in charge of defining the scheduled execution time and the GUI is responsible for the execution.

The button "Save as Favorite" ⑭ offers the feature to save the actual deployment configuration. The operator is asked for the definition of a unique label for the storage of the favorites. These will appear in the overview site of each operator.

The last feature in the deployment window is the "Refresh Cache" button ⑮. The SysMES framework caches the actual configuration of displayed management objects. In case of changes in these objects, it is necessary to update the cache by pressing this button. In further versions of the GUI, the update of the object cache may occur automatically.

- Administration: The Administration site is developed for the installation of SysMES clients. The method for the interaction with this feature is similar to the deployment. The operator is in charge of selecting targets for the execution of an action and executes one of these on the selected targets. The actions are installation, starting and stopping of SysMES clients.

Summary of the section:

The SysMES Graphical User Interface (GUI) is used for the manual interaction with the SysMES framework. It is designed for the visualization of the state of targets and for the distribution of management objects. The state of SysMES targets is displayed in different colors depending on their Severity `"( red = Immediate, orange = Critical, yellow = Service and green = Information)"`. Furthermore, all Events are displayed in an Event viewer. The viewer includes information about the Event originator, measured values, occurrences and Event state. Another feature of the GUI is the deployment window. There, it is possible to choose management resources from the RBEM model like Monitors, Simple Tasks and Rules and to distribute these to selected targets. Some object attributes can be set in the GUI. The GUI also offers capabilities for the installation of the SysMES clients and for starting and stopping these.

6. Realization and Implementation

In the previous chapter 5, the SysMES functionality has been introduced. Each layer was described in detail, but isolated. The focus of this chapter is to give a global overview about the interaction of the SysMES components and to show the principles of the implementation. In order to simplify matters, classes used for this purpose contain no attributes and data types and also only the necessary methods. These methods are introduced in a short manner without parameters and return values.
The first part of the chapter contains a description of used technologies. Following this are the sections which describe the realization and implementation of the SysMES layers.

6.1. General Information

The server side of the SysMES framework has been developed using the Java programming language and services offered by the Java Enterprise Edition (J2EE) [63]. J2EE is a specification for a software platform which allows the development of component-based distributed applications. One of the core elements of J2EE is the Enterprise Java Beans (EJB) container. The SysMES framework utilizes the JBoss AS [65] version 5.0.1.GA, which is an implementation of J2EE.
JBoss AS integrates features for clustering and high-availability of Java application and also load balancing strategies for the repartition of load in distributed environments. Another useful feature concerns the transactional based execution of Java applications.
Figure 6.1 illustrates one instance of each layer, such as one WAM server instance or one LAM server instance. The SysMES framework is designed for the usage of more instances of these layers in order to achieve the requirements of failure tolerance and scalability. All instances of one specific layer form a single logical entity. The usage of this single logical entity is transparent, i.e. the instantiation of EJB and execution of their methods. The JBoss AS receives a request for the execution of a clustered method and determines which instance of the cluster should handle the request. The decision for a specific instance is made according to a load balancing strategy. This strategy is defined in the configuration of the JBoss AS.
The SysMES framework utilizes those features for the implementation of the algorithms and functionality introduced in the previous chapter 5. As described before, the SysMES framework utilizes databases for the persistent storage of management objects. The part of SysMES that initiates the storage of management objects is realized by the usage of Hibernate [55], which provides the capabilities for mapping the object-oriented SysMES RBEM model to a relational database schema.
The SysMES client functionality is implemented in C++. The reason for this is because most of the devices to be managed are not able to run Java binaries. A full Java client is planned for the near future.
In section "Inter Layer Communications and Data Format" 5.2, it was stated that the SysMES top-down communication is realized by exchanging Task objects and the bottom-up communication by exchanging Events. The component diagram of figure 6.1 describes the basics of the implementation of the SysMES management algorithms. The red colored path represents the top-down communication concerning the distribution of Tasks from the GUI to the targets and the blue path, the bottom-up

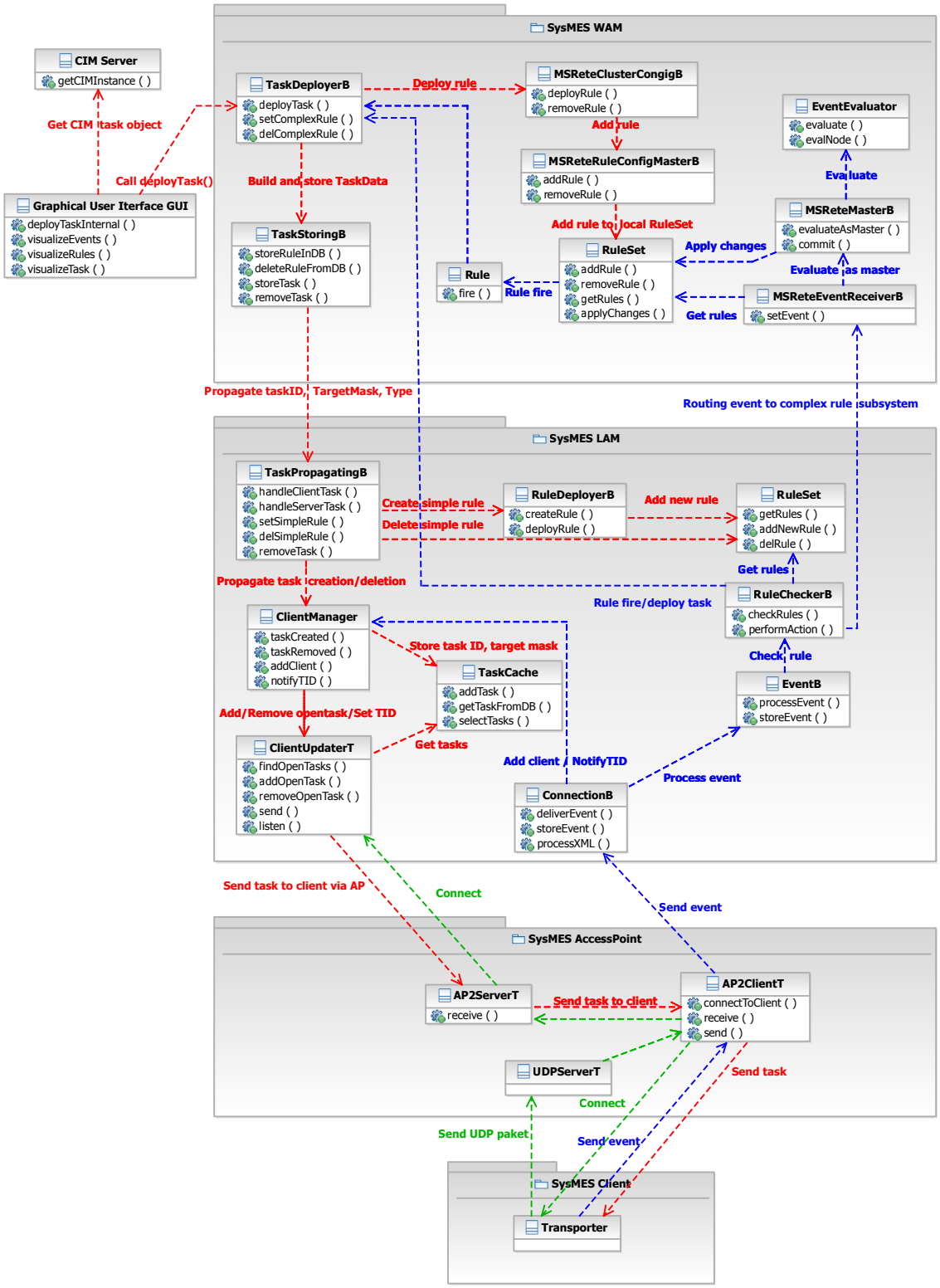


Figure 6.1.: SysMES Framework - Implementation Basics

concerning the receiving and processing of Events. This diagram is a simplified representation of the classes and methods used in both communications paths. Classes with the suffix "B" in the class name are Enterprise Java Beans for example the TaskDeployerBean. The suffix "T" represents Threads, such as ClientUpdaterThread. All other classes are plain Java classes.
The following sections introduce the implementation details of both of the communications paths.

6.2. Top-Down Communication Path

As previously mentioned, the top-down communication path is used for the deployment of Tasks from the modeling server to the targets. Those targets can be servers, cluster nodes or other types of devices where a SysMES client is running. Tasks are used for setting up the management environment. In order to cover the distribution of Tasks in all SysMES layers, three Tasks are considered exemplary: A Tasks for the distribution of a Complex Rule, a Task for the distribution of a Server Simple Rule and a Task for the distribution of a Monitor.

6.2.1. Top-Down Communication - Modeling Server and GUI

The SysMES Graphical User Interface (GUI) is the interface for administrators and operators in order to set up the management environment by the usage of Tasks. The steps for Task deployment were described in the GUI section 5.5.2 . In principle a system administrator has to choose management objects to be deployed and targets objects as destination. Although in the GUI management objects like Monitors and Rules are displayed, the deployment process is always related to the distribution of Task objects, which contain the Monitors, Rules or Actions. Therefore, from now on only Task objects will be taken into account.
The CIM server offers an interface which is used by the SysMES GUI for getting the actual configuration of the Task objects chosen in the deployment act.
The GUI calls the getCIMInstance() method of the CIM interface iteratively for each Task object chosen on the left side of the deployment window of the SysMES GUI and also for all target objects chosen on the right side. Getting objects from the CIM server is the first part of the deployTaskInternal() method of the GUI. The second part consists of the association of a Task object with a Target Mask object and the invocation of the deployTask() method of the TaskDeployerBean. The GUI repeats this procedure for all chosen Tasks and chosen Target Masks.

6.2.2. Top-Down Communication - WAM Layer

The first class of the WAM Layer used for the deployment of a Task is the TaskDeployerBean. The GUI requests a TaskDeployerBean object from the WAM cluster and receives the instance with the least load. The GUI invokes the deployTask() method on the received object, which initiates the Task deployment on the WAM Layer.
In figure 6.1, the red arrows going out from the TaskDeployerBean represent the processing flow for two different kinds of Tasks. The right arrow is a Task for a target in the WAM Layer (such as ServerSetComplexRule). The arrow going down to the bottom represents the standard processing path for all kinds of Tasks with targets in the LAM Layer or in the Target Layer (such as ClientSetMonitor).
As already mentioned in section 5.4.2.4.2, the Complex Rule Management subsystem is located in the WAM Layer. The configuration of this subsystem is realized by the distribution of specific Task objects of the RBEM model. These Tasks are objects of the class ServerSetComplexRule, as well as objects of the class ServerDelComplexRule.

The TaskDeployerBean analyzes the Task objects to check if the current Task object belongs to one of these classes. If the Task object is destined for the Complex Rule Management subsystem, then the TaskDeployerBean calls the deployRule() or removeRule() method of the MSReteClusterConfigBean. This method call initiates the internal configuration of the Complex Rules in the master-slave cluster. The MSReteClusterConfigBean has knowledge about the current master-slave configuration and is in charge of initiating the desired Actions in all involved instances. In the figure 6.1, the MSReteClusterConfigBean invokes the addRule() or removeRule() of the MSReteRuleConfigMasterBean which is used to set up the master and therefore to set up the Complex Rule Management subsystem in master-only mode.
For a desired master-slave mode the MSReteClusterConfigBean iteratively executes the Task on the master and also on the slave using the same procedure. For both modes - master-only and master-slave - the respective bean induces the inclusion or deletion of a Complex Rule into the local Rule Set by calling the addRule() or removeRule() method of the RuleSet class.
Regardless of the Task type, all Tasks have to pass the red path going out from the TaskDeployerBean to the bottom. The next steps in the processing of Tasks are their persistent storage and the propagation of necessary information to the LAM Layer for their further handling.
The execution of both processing steps should be done in a transactional way because the involved beans do not have to be in the same WAM instance. Furthermore, regardless of the number of members in a WAM cluster, the Task storage and deletion procedures should be executed once and some information about the Task has to be propagated to all instances of the LAM Layer. These functionalities are implemented using Java Message Service (JMS) [66].
JMS defines two kinds of message destinations called queues and topics. Messages which are sent to a queue can be received by only one consumer (point-to-point communication) and messages which are sent to a topic are received by all consumers (pub/sub communication). Messages are stored in a persistent and transactional way until they have been delivered.
Both the TaskStoringBean and the TaskPropagationBean are Java Message Driven Beans, which are used for processing Java messages. These kinds of beans act as a JMS listener for receiving messages from queues or topics. Figure 6.2 visualizes its usage for exchanging messages within the SysMES Layers and also in between these.
All instances of the TaskStoringBean in the WAM Layer are listeners for the StoringQueue and instances of the TaskPropagatingBean are subscribers of the PropagatingTopic.
Conforming with figure 6.2, one of the TaskDeployerBean instances receives the Task to be deployed and sends a message to the StoringQueue containing the Task to be stored. In this example it is the instance on `"WAM1"`, but the deployment of Task can be done using any instance of the TaskDeployerBean.
All instances of the TaskStoringBean are listening to the StoringQueue and JBoss AS decides which of these instances receives the message (i.e. the Task). The StoringQueue holds the Tasks until the delivering transaction to a TaskStoringBean is committed. Afterwards, the Task is consumed and can be removed from the queue. JBoss AS is able to deliver the message to another listener in case of an aborted transaction.
The designated instance of the TaskStoringBean is in charge of decoding the message and of storing the contained Task into the database. In the case of Tasks for setting up or removing Complex Rules and Simple Rules, it is also responsible for performing these Actions by the execution of storeRuleInDB() or deleteRuleFromDB().
The last responsibility of the TaskStoringBean concerns the preparation of a message to be propagated to all instances of the LAM Layer and to send this to the PropagatingTopic. The message contains the TaskID, Target Mask and Type of the Task to be propagated.

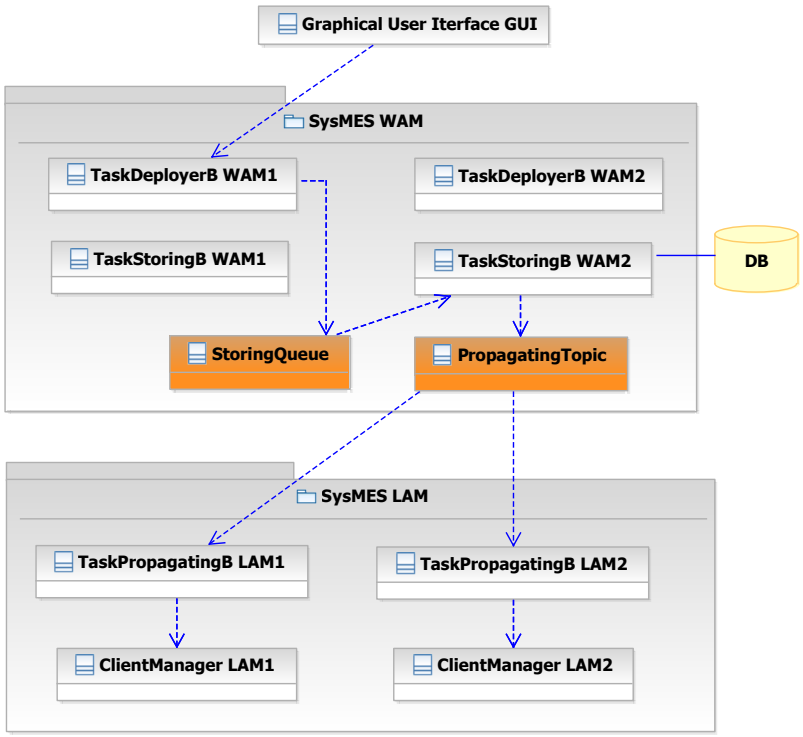


Figure 6.2.: SysMES Framework - Server Layers Messaging

6.2.3. Top-Down Communication - LAM Layer

All server instances that are members of the Server Layer require information about deployed Tasks in order to recognize if the Task is relevant for the SysMES clients which they are connected to. There is no static distribution of clients to LAM servers (via Access Points) and therefore all instances of the LAM have to be informed about newly deployed Tasks. The following processing path corresponds to the red path in the SysMES LAM component of the component diagram in figure 6.1.
The TaskStoringBean is a Message Driven Bean used in the LAM Layer for listening to the PropagatingTopic. Each server member of the LAM Layer receive the message containing the TaskID, Target Mask and Type of the Task in processing.
The TaskStoringBean parses the message and acts depending on the value of the Type attribute (i.e. Server Task refers to `"Type=1"` and Client Task refers to `"Type=2"`).
In the case of a Server Task, the TaskStoringBean calls handleServerTask() which compares the local server identifier with those stored in the Target Mask in order to find out if the Task has to be processed by the LAM server instance. In case of a match, the entire Task object is retrieved from the database and depending on the class that the Task objects belongs to, one of the following four methods are called:

- removeTask(): This method is used for deleting a deployed Task from the database, from TaskCache and from internal structures of the ClientUpdaterThread.
- setSimpleRule(): This method is responsible for sending the Server Simple Rule contained in the Task to the Server Simple Rule subsystem. It requests a local instance of the TaskDeployer-

Bean from the JBoss AS and executes the methods createRule() and deployRule() sequentially. The first one is used for creating a Simple Rule object and the second one to add this object to the RuleSet.

The RuleSet class is displayed in figure 6.1 as a plain Java class. JBoss AS extends the J2EE specification to a new kind of EJB the so-called Service Beans. Typical Enterprise Java Beans are classes managed by the application server. This server - in this case JBoss AS - manage a pool of bean instances, which can increase or decrease in the number on demand. Contrary to this, there is only one instance of Service Beans. Access to this object is managed by the respective application server. In other words, a Service Bean is an applications server managed singleton.

- delSimpleRule(): This method is used for removing a Server Simple Rule from the RuleSet. It interacts directly with the RuleSet object, executing its method delRule().

In case of a Client Task, the TaskStoringBean calls handleClientTask() which initiates the Task handling from the LAM Layer to the SysMES clients by calling the taskCreated() method of the ClientManager Service Bean.
The ClientManager Service Bean is in charge of managing SysMES clients connected to a specific LAM instance. When a SysMES client starts, a ClientUpdaterThread is generated and a reference to this thread is stored in a list ordered by the target "DeviceIDs".
The ClientManager Service Bean stores the TaskID and Target Mask into the TaskCache structure calling its addTask method. Such a cache structure is used in order to avoid unnecessary access to the database. The ClientManager compares the identifier of the SysMES clients connected to the LAM server with the Task recipients and in case of a match, it calls the addOpenTask() method in the respective ClientUpdaterThread.
The ClientUpdaterThread is directly connected to the SysMES clients through the Access Point. In the case of a new Task, the ClientUpdaterThread is in charge of receiving the TaskID and searching within the TaskCache for a Task object associated with this TaskID. In case of a miss, the ClientUpdaterThread fetches the Task object from the database and stores it in the TaskCache structure. Otherwise, the Task objects from the TaskCache are used without interaction with the database. This Method is useful when one Task has to be sent to multiple SysMES clients connected to the same LAM instance. In this case, the LAM server accesses the database only once for fetching the Task object.
Finally, the ClientUpdaterThread calls the send() method to distribute the Task to the target.

6.3. Access Point

The Access Point implementation is described separately because there is no difference in its functionality for both communication paths. As already described in section 5.4.1, Access Point instances are communication components. These are needed for the transaction-based transmission of data between clients implemented in C++ and servers running in a JBoss AS.
The green path in figure 6.1 describes the initialization path for establishing a connection between a SysMES client and a server using an Access Point. In the SysMES framework, the process for establishing a connection is always initiated by the SysMES clients. Without a network connection to the server side, the clients are able to execute Monitors and to check if the measured values are relevant. The Transporter class of the client requests a connection to the Server Layer if it has Events to send.

The SysMES Access Point is composed of three threads. The first thread is the UDPServerThread, which is responsible for getting UDP packets from the SysMES clients. The UDPServerThread implements a UDP listener. The client UDP packet contains information about a port where the client starts a TCP listener process.
The UDPServerThread starts an instance of the AP2ClientThread for each client and transfers the information from the UDP packet to it. The AP2ClientThread establishes a TCP/IP connection to the client at the desired port. Using this connection, the SysMES client is able to send Events to the Access Point. With the first Event received at the AP2ClientThread, this thread calls the deliverEvent() method of the ConnectionBean of one arbitrary LAM instance and returns a message containing the IP address and port where the LAM instance starts a server listener process. Afterwards, the AP2ClientThread starts an AP2ServerThread, which establishes a connection to the server.
For each client the Access Point stores references to the AP2ClientThread and AP2ServerThread threads in order to find the clients in both directions. Using this method, an Access Point is able to manage connections from clients to different instances of the LAM Layer. In case of transmission or connectivity errors and also LAM server failures, the Access Point is in charge of destroying the current connections and threads and the client is forced to initiate a new connection to another server instance.
The AP2ClientThread implements the transaction-based Event delivering through the Access Point. The SysMES client includes an Event packet identifier (4 bytes) in each packet independent from the number of Events contained in the packet. This identifier is used by the AP2ClientThread in order to acknowledge the reception and processing of all Events contained in the packet. The SysMES client keeps the Event packet stored until it is acknowledged or a predefined timer expires. In case of an Access Point failure which causes the packet not to be acknowledged, the SysMES client closes the connection to the Access Point and sends a UDP packet again in order to connect to another Access Point.

6.4. Bottom-Up Communication Path

The bottom-up communication path represents the way to send data from the targets to the servers. This data can be monitoring data, reports about the execution of Tasks, errors and in general all data which has to be sent to the SysMES server side. The SysMES framework utilizes Events for this purpose. The Bottom-Up communication path is the blue path in figure 6.1. This path begins at the client side, followed by the storage of Events and their processing in the LAM Layer. Afterwards, the Events are processed by the Complex Rules in the WAM Layer. The last step in the blue path is the visualization of the Events in the GUI.

6.4.1. Bottom-Up Communication - LAM Layer

The first steps of the bottom-up communication, i.e. from client to the Access Point, were introduced in the previous section concerning the Access Point(see section 6.3). Therefore this section starts with the step where the Access Point intends to send Events to the LAM Layer.
Whenever the AP2ClientThread tries to send Events to the server, it tests if an AP2ServerThread already exists and also if it is running and available. If there is no instance of this thread (e.g. sending the first Event from a client), the AP2ClientThread calls the deliverEvent() method of the ConnectionBean, which induces the execution of the addClient() method of the ClientManager Service Bean. This method is responsible for starting a new instance of the ClientUpdaterThread, which starts a

listener for establishing a connection between itself and the Access Point.

After the successful execution of the deliverEvent() method, the AP2ClientThread starts a new instance of the AP2ServerThread which connects to the listener of the ClientUpdaterThread started before. The AP2ClientThread internally holds a reference to the AP2ServerThread, which allows it to find the SysMES client in both communication paths.

From now on, the AP2ClientThread has a valid reference to an instance of the AP2ServerThread and therefore it is not necessary to execute the instantiation and connection procedure again. For the next Events, the AP2ClientThread calls the processXML() method directly.

The next steps for processing Events in the LAM Layer consist of their storage and checking against Server Simple Rule. For this purpose the ConnectionBean for each Event calls the processEvent() method of the EventBean. Each Event object has two attributes which describe its specific storing and processing strategy. In principle, it is possible to configure each Monitor (and more exactly the Event Classes associated to the Monitor) for storing the Event, for forwarding it to the Server Simple Rule subsystem or to skip one or both of these processing steps. The decision concerning this configuration has to be taken during the development of the Monitors and this can be changed at any time using Tasks.

In the case of figure 6.1 both storing and forwarding activities should be performed for Events. The storage of Events is realized by calling the storeEvent() method of the EventBean, which utilizes the Hibernate Persistence Service. Event forwarding to the Server Simple Rule subsystem is realized by calling the checkRules() method of the RuleCheckerBean. The implementation of the Rule checking algorithm works parallely because instances of both the EventBean and the RuleCheckerBean are managed by the JBoss container which, increases and reduces the number of these dynamically and according to the current requirements.

Each instance of the RuleCheckerBean stores a local copy of the Server Simple Rules objects of the RuleSet Service Bean. This copy is obtained by the execution of the getRules() method of the RuleSet Service Bean and is used in order to check if the current Event fulfills one or multiple Server Simple Rules.

In case of a match while checking the Rules, the RuleCheckerBean executes the performAction() method, which is in charge of executing the perform() method of the specific Action objects attached to the Server Simple Rule. It is possible to attach multiple Actions. The blue processing path of figure 6.1 describes the execution of two Action types: the Routing Action and the Task Action. The description of these Action types can be found in the section "Server Simple Rules" 5.4.2.4.1.

By the execution of a Task Action, its perform() method generates a Task object and it calls the deployTask() method of the TaskDeployerBean. This causes the distribution of the Task to the desired targets using the method described in the previous section about the top-down communication path. Such Tasks are used in order to sort out problems, which are recognized through the monitoring capabilities of SysMES.

Routing Action objects are used in order to forward Events from the Server Simple Rule subsystem in the LAM Layer to a desired instance of the Complex Rule Management subsystem in the WAM Layer. Its perform() method calls the setEvent() method of the MSReteEventReceiverBean (on the WAM Layer).

The development of Routing Rules and Routing Actions are coupled closely to the development of Complex Rules because these Rules/Actions are necessary to forward Events from the LAM Layer to the desired location where the Complex Rules are deployed.

6.4.2. Bottom-Up Communication - WAM Layer

The process of checking Events against Complex Rules, firing and executing Actions is similar to the case of Server Simple Rules. The major difference is that Complex Rules are stateful and therefore Event occurrences that match cause changes in the current evaluation state of Complex Rules.

The implementation of the Complex Rule checking algorithm begins with the MSReteEventReceiver-Bean, which is the interface between the LAM Layer and the WAM Layer. The setEvent() method of this bean called by the Routing Action is in charge of initiating Event-Rule evaluation. Whenever an Event arrives to the Complex Rule Management subsystem, the setEvent() method is first responsible for recognizing in which modus the Complex Rule Management subsystem is running (i.e. master-slave or master-only mode) and for instantiating concerned beans. Second, the setEvent() method preselects all Complex Rules from the RuleSet whose Leaves are fulfilled by the specific Event.

In general, the main task of the MSReteEventReceiverBean is the orchestration of the Event evaluation depending on the Complex Rule Management subsystem mode. Figure 6.1 shows a Complex Rule Management subsystem running in a master-only mode and therefore it calls the evaluateAsMaster() method of the MSReteMasterBean. Afterwards, the commit() method is invoked if the evaluation ends without errors, otherwise the rollback() method (not displayed in the figure) is called.

In the case of a master-slave mode, this bean calls in parallel evaluateAsMaster() and evaluate-AsSlave(). The commit() method on the master instance is only called after the slave instance committed the evaluation, otherwise both instances execute the rollback() method. This method is chosen in order to evaluate simultaneously in both the master and the slave instances. Consequently, it utilizes the processing power of two different server instances instead of waiting for a a sequential blocking master and slave evaluation and the synchronization of the processing status. It is assumed that master and slave are located in different physical servers due to the expected high availability of the Complex Rule Management subsystem.

The real evaluation of an Event against preselected Rules occurs in the evaluate() method of the EventEvaluator class, which is called by the MSReteMasterBean. evaluate() is in charge of checking all preselected Complex Rules, i.e. all Rules of the RuleSet where at least one leaf is fulfilled by the Event. The evalNode() method is called for each node in a Rule starting with the lowest node up to the root node on the top of the evaluation network. evalNode() checks if the Event fulfills the trigger conditions and stores tokens. The Rule checking algorithms finalize at last when the root node is reached and checked. In case of a match in the root node, then the consumption mode strategy has to be performed. In the Unrestricted Consumption Mode, all consumed tokens are deleted.

In the case of a successful finalized check, i.e. without errors, the MSReteMasterBean calls the previous mentioned commit() method, which causes the execution of the applyChanges() method of the RuleSet. Until now, the evaluation of an Event is processed in a working copy of the preselected Rules and therefore changes in the state of these Complex Rules have to be propagated to the evaluation network. applyChanges() synchronizes the state of the Complex Rules and call their fire() method, which checks if there is a valid token to fire on the root node. If so, the perform() method of all attached Action objects is called.

The execution of Actions follows according to their priority. In figure 6.1, the blue arrow going out of the Rule class indicates the execution of a Task Action object and its injection in the WAM Layer by calling the deployTask() method of the TaskDeployerBean.

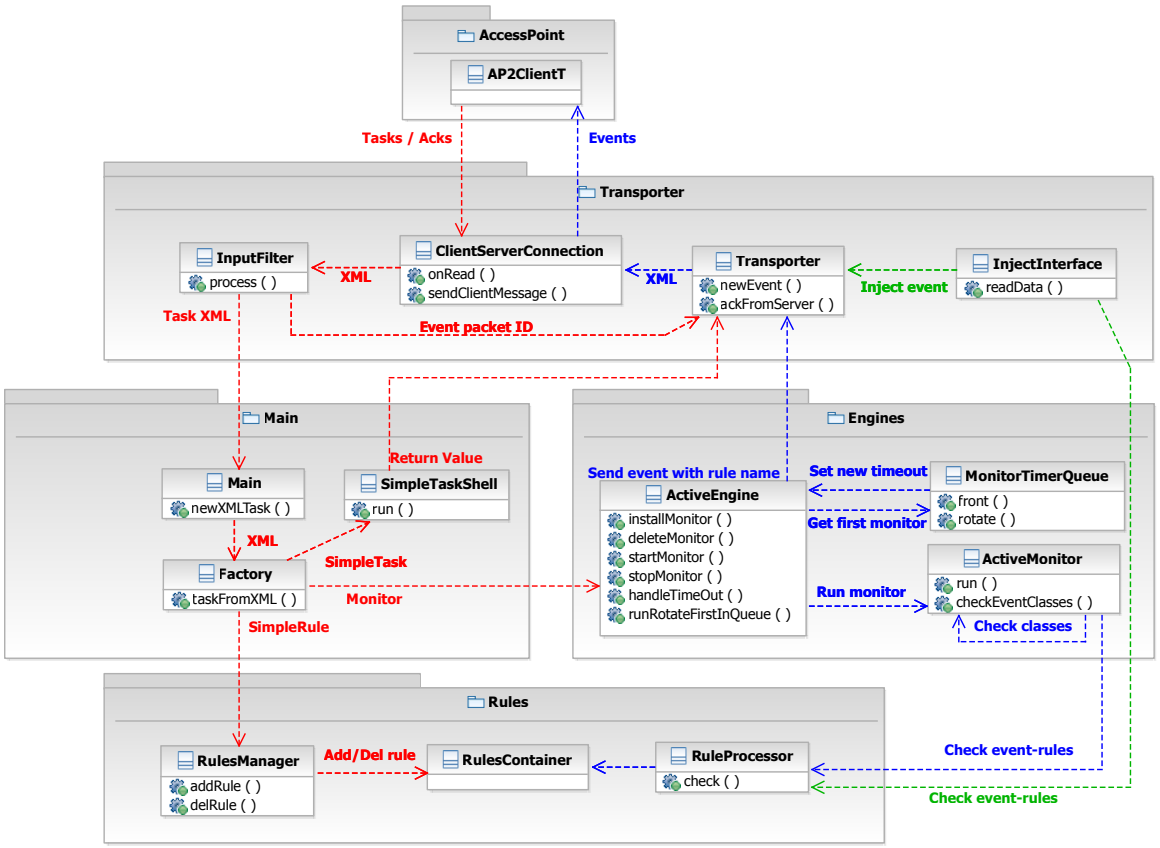


Figure 6.3.: SysMES Client - Implementation Basics

6.5. Client Implementation

SysMES clients are used as targets for the distribution deployment and execution of management objects such as Monitors, Rules and Actions.
The SysMES client, as a part of the SysMES framework, is designed and developed conform to the design considerations of chapter 4. It is a crucial part for the realization of the design considerations which concern a decentralized, scalable and dependable system management and, of course, for the realization of management services close to the target.
The SysMES client is in charge of assuming the processing of monitoring data and Events and also to react to undesirable state. It is also in charge of performing these activities in a stand alone mode, i.e. without communication to the Server Layer. This is very important for the detection and solution of problems in case of network or SysMES server failures.
This section describes the basics of the SysMES client realization based on the top-down communication path for the distribution and execution of Tasks and the bottom-up communication path for the monitoring of system resources and reporting errors and undesirable states via Events.
Figure 6.3 illustrates the components involved for the realization of both communication paths. Analogous to the previous sections and the figure 6.1, the red path corresponds to the top-down communication and blue path to the bottom-up.
This section is a continuation of the previous sections where the communication paths end at the

Access Point. A comparison of the figures "SysMES Framework - Implementation Basics" 6.1 and "SysMES Client - Implementation Basics" 6.3 shows that the red and blue arrow between the AP2ClientThread and the Transporter component of the SysMES client are the same. This indicates the continuation of the communication to the SysMES clients.
Figure 6.3 visualizes the main components of the SysMES client implementation and how these are involved in the respective communication paths.
In principle, the top-down communication was designed by the transmission of Task objects from the Server Layers to targets. There is one exception for this design. It concerns the transmission of Events acknowledgments packets from Access Point to the targets. As already introduced in section 6.3, the transmission of Events from client to Access Point is implemented in a transactional way and therefore the Access Point sends acknowledgment packets to the clients.
In the other direction, the SysMES client sends packets to the Access Point containing one or multiple Events and a unique packet identifier for the realization of transactions.
Outgoing packets from a SysMES client to an Access Point have the following format:

Size [Byte]	1	4	Message_Size	4
Context	0x01	Message_Size	Payload	ID

- Message_Size describe the payload length.
- ID is a 4 bytes unique signature for the Event packet.

Incoming packets to the SysMES client have the following format:

Size [Byte]	1	4	Message_Size-1	1
Context	0x01	Message_Size	Payload	Flag

- Message_Size describes the payload length plus one byte for the flag.
- Flag is 0x00 for a packet containing Task objects and 0x06 for an Event acknowledgment.
- In case of an Event acknowledgment, packet payload contains the 4 byte ID of the previously sent Event packet.

6.5.1. SysMES Client Top-Down Communication Path

The top-down communication path on the client side starts when the Access Point sends a packet to the SysMES client. The ClientServerConnection class of the Transporter component is responsible for managing a connection to the Access Point and provides methods for getting Tasks and acknowledgment packets, i.e. onRead() from the server side and to send Events packet to the LAM Layer, i.e. sendClientMessage().
If a new packet arrives at the client, the onRead() method calls the process() method of the InputFilter class which checks the value of the flag byte in order to decide if the current packet contains a Task or a Event packet acknowledgment identifier.
In case of a acknowledgment packet, i.e. `"flag = 0x06"` the process() method reports the value of the Transporter class by calling its ackFromServer() method. Further processing is described in the next section.

In case of a Task packet, i.e. `"flag = 0x00"`, the process() method calls newXMLTask() method of the Main class, which belongs to the same-named Main component. The called method acts as a XML pre-processor for the validation and parsing of the XML document and to generate a traversable tree. The Factory class operates in this tree. It analyzes it in order to recognize the type of management object that is contained in the Task XML. According to the object type, the taskFromXML() method generates new internal objects such as Simple Task, Simple Rule and Monitor and it calls the respective methods for further processing of these objects as described next.

- Simple Task: In case of a Simple Task,, its execution is induced by calling the run() method of the SimpleTaskShell class. This method captures the return value and sends it as a TaskReply Event. In case of an error during the execution, an Error Event is generated.
- Monitor: As described in section 5.3.1, there are basically two different kinds of Monitors according to the measuring strategy, these are Active Monitors and Passive Monitors. For both there is an Engine component that is responsible for the management of the respective Monitors (i.e. installing, deleting, stopping and starting), for their maintenance (i.e. reconfiguration) and also for their execution (i.e. timing). In the case of figure 6.3, the Active Engine is displayed exemplarily. In case of a Task containing a new Monitor to be deployed, the request to delete or stop or start a deployed Active Monitor, the ActiveEngine class is in charge to execute the respective method, i.e. installMonitor(), deleteMonitor(), startMonitor() or stopMonitor().
- Simple Rule: In case of a Client Simple Rule to be deployed or deleted, one the addRule() or the delRule() method is called. Both of these methods access the RuleContainer class where the Client Simple Rules are stored persistently and perform the desired action.

6.5.2. SysMES Client Bottom-Up Communication Path

The client side bottom-up communication path is initiated by the execution of Monitors. In the case of Active Monitors, these are executed according to the value of the Period and Repeat attributes. Passive Monitors get the measured values from third party monitoring systems and therefore it is not necessary to take these into account for the explanation of the timing management procedures.
The management of Active Monitors in terms of their timing is realized in the MonitorTimerQueue class. For this purpose this class holds a indexed data structure - the MonitorTimer queue - to store tuples of `"(ΔT, Monitor object)"`. Elements of the queue are ordered according to the execution sequence of the Monitor objects. The first element of the MonitorTimer queue contains the next Monitor to be executed. ΔT describes the relative execution time of the Monitor in relation to previous Monitors in the queue. The value of ΔT for the n-th element is calculated as

$$\Delta T(n) = Monitor_n.Period - \sum_{i=0}^{n-1} \Delta T(i)$$

For the given Monitors $M_1.\text{Period} = 17$, $M_2.\text{Period} = 9$, $M_3.\text{Period} = 13$, $M_4.\text{Period} = 7$ and $M_5.\text{Period} = 19$, the table 6.1 shows the original state of the MonitorTimer queue.
The ActiveEngine class is in charge of controlling the execution of the Monitors. It has a timer which is set up to the ΔT of the first element of the MonitorTimer queue. Whenever the timer expires handleTimeOut() is executed and it performs the following actions:

- Executes the front() method of the MonitorTimer queue class which is used in order to remove the first element of the queue and to declare it as the CurrentMonitor.

Index	Monitor	ΔT
0	M_4	7
1	M_2	2
2	M_3	4
3	M_1	4
4	M_5	2

Table 6.1.: MonitorTimer Queue Original

Index	Monitor	ΔT
0	M_2	2
1	M_3	4
2	M_4	1
3	M_1	3
4	M_5	2

Table 6.2.: MonitorTimer Queue Rotated

- Executing the CurrentMonitor by calling its run() method.
- Decreases the value of the Repeat attribute if it is not equal to `"0"` or `"-1"`, which means that the Monitor runs indefinitely. In case of `"0"` the Monitor will be uninstalled.
- The next step concerns the reinsertion of the CurrentMonitor in the MonitorTimer queue. This is realized by the execution of the rotate() method of the MonitorTimerQueue class. For this purpose it is necessary to find an element n in the queue with

$$\sum_{i=0}^{n} \Delta T(i) \leq CurrentMonitor.Period < \sum_{i=0}^{n+1} \Delta T(i)$$

  In the case of the previous example, the n-th Monitor is M_3.
- The currentMonitor is reinserted as the n+1-th element and its ΔT is calculated as

$$\Delta T(n+1) = CurrentMonitor.Period - \sum_{i=0}^{n} \Delta T(i)$$

- Decrease the ΔT of the n+2-th Monitor

$$\Delta T(n+2) = \Delta T(n+2) - \Delta T(n+1)$$

- The ActiveEngine class resets the timer to the ΔT value of the new first Monitor in the queue.

The tables 6.1 and 6.2 respectively show the state of the MonitorTimer queue before and after the execution of the first Monitor.
The previous described method is also used in order to insert a new Monitor into the MonitorTimer queue, i.e. by calling the installMonitor() method of the ActiveEngine class.
After the Monitor execution through the ActiveEngine, the Monitor calls the checkEventClasses() method in order to define if the measured value represents a undesired state or error and to generate an Event.
In case of a match and if the value of the Event Class attribute `"CheckClientRule = true"`, then the Event-Rules checking is initiated by calling the check() method of the RuleProzessor class. This method checks iteratively if the Event fulfills one or multiple Rules and executes the attached Action. If one or multiple Rules have matched then the Event attribute ProcessedRule is set to the name of all Rules that have been fulfilled by the Event. This is important in order to inform the SysMES server side about the execution of Actions which should contribute to solve a recognized problem.

The run() method returns the complete Event to the ActiveEngine, which calls the newEvent() method of the Transporter class in order to send the Event to the Access Point.
In figure 6.3, a green path is also plotted. This paths belongs to the Inject Interface which is used to insert Events into the SysMES client without measuring and also without triggering in the Event Classes. The inserted value will be handled as an Event, and according to the given attributes, it is sent to the RuleProcessor class and afterwards to the Transporter class. The Inject Interface has been designed and added to the SysMES client in order to offer the possibility to generate a simulated client state through the injection of Events and, therefore, the possibility to test the Rule systems. Another usage of the interface is the inclusion of Monitor data from devices which are not able to run an own SysMES client, for example network switches or rack monitoring devices because these are only able to send errors as SNMP [97] or IPMI [61] traps. In this case, a dedicated target receives the traps and injects these into the SysMES framework using the Inject Interface.

7. System Tests and Evaluation

In the previous chapter 6, SysMES implementation details has been introduced. This chapter introduces several tests which are used to demonstrate that design and implementation of the framework achieve the goals defined in chapter 2.
This chapter is divided into four parts. The first part describes how the SysMES framework is used for the management of a production environment. The reference environment is the ALICE HLT Cluster in CERN (abbreviated as HLT Cluster).
The second part includes series of tests for the demonstration of the scalability grade of the SysMES framework. The third part demonstrates the high availability properties of the SysMES framework. The last part is a resources utilization test of the SysMES client.

7.1. Functionality Evaluation - HLT Cluster Management Using the SysMES Framework

The SysMES framework has been used since the beginning of the data taking period in 2009 and it is in charge of managing the ALICE HLT Cluster. This cluster is used for processing detector data from the ALICE experiment [2]. More exactly, the HLT Cluster is the physical infrastructure required for the execution of the High Level Trigger (HLT) application [10].
During the data taking period of the experiment, the cluster is used 24/7 for online analysis and between the run periods for developing and testing the HLT application.

7.1.1. Alice HLT Cluster

The HLT Cluster is a heterogeneous computer farm located in two rooms (CR2: 40 Racks and CR3: 12 Racks) in the computation facilities of the ALICE experiment in CERN - Geneva. In the actual commissioning state, the production cluster consists of 25 servers and about 200 nodes and the test cluster consist of 20 nodes. In the final planned commissioning state, the cluster will grow up to 1000 nodes. At the moment, there is one system administrator responsible for cluster management, maintenance and further development. The further description of the cluster refers to the production cluster. The nodes in the test cluster have the same basic configuration as those in the production cluster.

7.1.1.1. Physical and Network Infrastructure

The following figure 7.1 [1] visualizes both the physical and the network infrastructure of the HLT Cluster.
The production cluster is fully installed in the room CR2, which is divided in 3 rack lines: The X rack line (10 racks), the Y rack line (17 racks) and the Z rack line (13 racks).

[1] Picture design and copyright by Sebastian Kalcher.

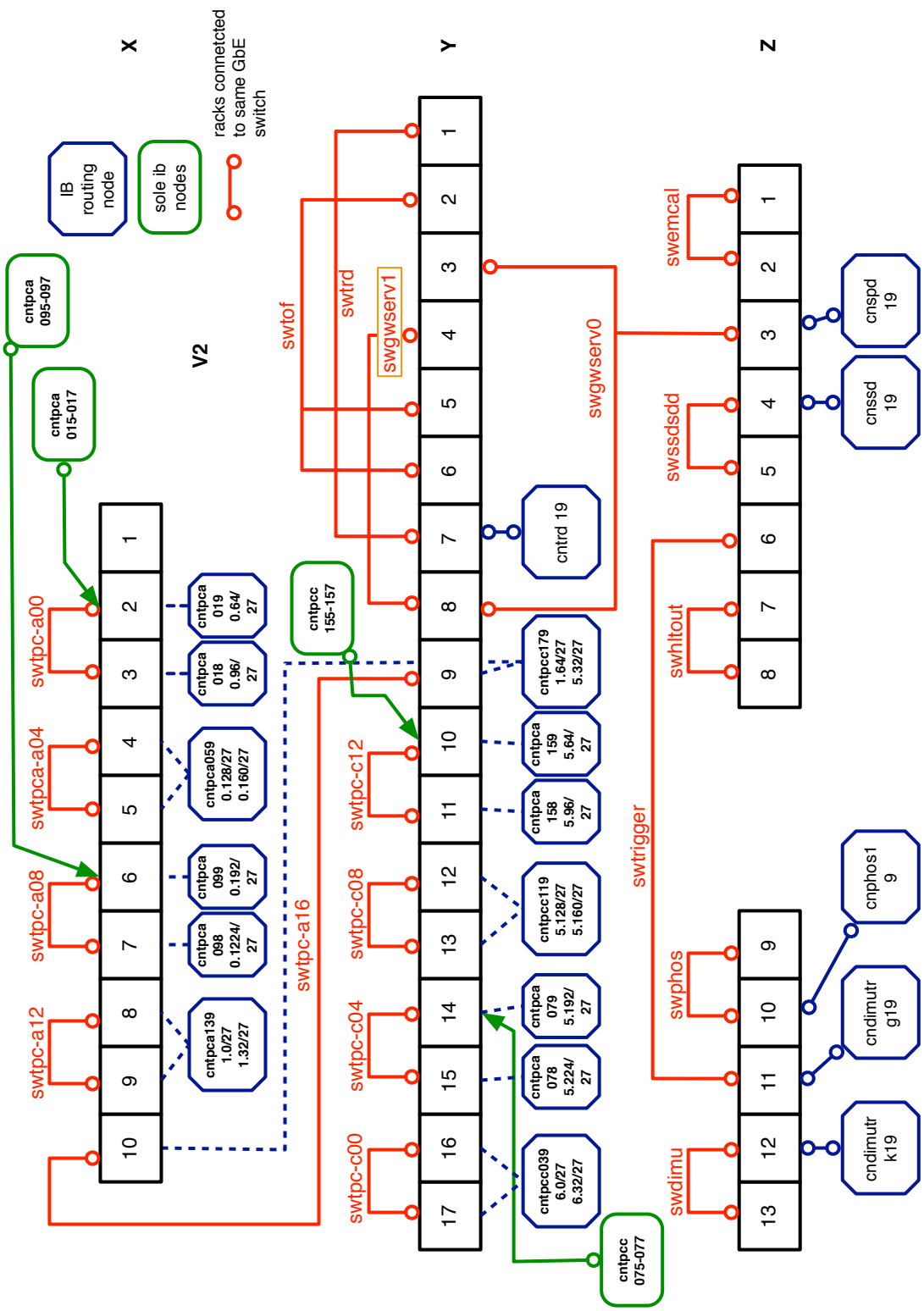


Figure 7.1.: HLT Cluster - Network Infrastructure

The HLT network has been designed as a hierarchical network. It is composed of one Gigabit network and an Infiniband (IB) backbone.
In general, there is one dedicated Gigabit Ethernet switch for all nodes of two racks (e.g. the switch swtpc-a12 on the top of the figure 7.1 is used for all nodes of the racks X8 and X9). On the top level of the network is the switch "swbackbone", to which all other switches are connected to (this switch is not included in figure 7.1).
The usage of the IB backbone is realized by IB nodes, which are responsible for routing data between racks using the high-speed backbone.
Besides these networks, there is a Fast Ethernet management and maintenance network dedicated to the remote control cards used in the cluster nodes. A short description of these cards follows in the next section.

7.1.1.2. HLT Cluster Nodes

The nodes of the HLT Cluster are divided into the following three categories:

- **Front-End Processor (FEP) Nodes:** This kind of nodes is used for getting raw detector data from the ALICE experiment and for running parts of the HLT application.

  At the current commissioning state there are 135 FEP nodes equipped with the motherboard Tyan [2] Thunder h2000M (S3992), two Quad Core AMD Opteron [3] 2378, 2.4 GHz processors and 12 GByte of RAM.

  For transferring the detector data into the HLT Cluster each FEP node has two High Level Trigger Read Out Receiver Cards (H-RORCs) [11]. Each H-RORC is equipped with two optical links for receiving a maximum input data rate of 400 MByte/sec (i.e. 200 MByte/sec is the maximum expected data rate per link). The H-RORC has a maximum output data rate of 900 MByte/sec to the PCI Bus of the FEP node where it is plugged in. Furthermore, the H-RORC acts as a FPGA based hardware preprocessor for the HLT application performing the first analysis step.

  Almost all FEP nodes have a Computer Health and Remote Management (CHARM) card [87], which is used for remote and automatic installation, testing, maintenance, monitoring and control of the nodes. The SysMES framework uses it as an interface for accessing sensor data, e.g. temperature data and other useful information, such as CMOS settings and status of the host operating system and for the execution of actions which impact the host, e.g. computer power off . The CHARM card is based on an embedded system running a Linux operating system on an ARM [4] 922T CPU. Some FEP nodes use the Tyan M3291 SMDC Add-On card for remote control.

- **Computing Node (CN):** They are used for hosting another part of the HLT application. These nodes have more computation power than the FEP nodes and are therefore more suitable for the placement of processes with a high demand of resources.

  At the moment, there are 59 CN nodes divided into two different versions. 21 CN nodes are equipped with the motherboard Tyan Tempest i5400PW(S5397), two Intel Xeon [5] E5420, 2.5

[2] Tyan is a registered name of the Tyan corporation.
[3] Opteron is a trademark of the AMD corporation
[4] ARM is a registered name of the ARM corporation.
[5] Intel and Xeon are registered names of the Intel corporation.

GHz processors and 16 GByte of RAM and 38 CN nodes are equipped with the motherboard Supermicro X8DTT-IBX, two Intel Xeon E5520, 2.27 GHz processors and 24 GByte of RAM.

The first version of the CN nodes utilize the CHARM card for remote control and the second version the on-board chip Winbond [6] WPCM450 BMC.

- **Infrastructure Nodes (Infra):** These kind of nodes are used for hosting required central services and are divided into the following types:
  - **Gateways:** These nodes are used for the access to the cluster. The gw0 and gw1 nodes are used for the user access from the CERN and Kirchhoff-Institute for Physics (KIP) network. The dcs0 and dcs1 machines are used for internal access to the cluster from the Data Acquisition (DAQ) [7] [83] network. The interface for the Experiment Control System (ECS) [8] [32] is realized by the ecs0 and ecs1 nodes. The vobox0 and vobox1 nodes are used as an interface for other applications such as Taxi [9] [14]. The following table 7.1 shows the basic hardware configuration of the gateway nodes.

Node	Motherboard Type	Processor Type (2 CPUs inside)	Memory/GByte
gw0	Tyan Thunder K8S Pro	AMD Opteron 242, 1.6 GHz	3
gw1	Tyan Thunder K8S Pro	AMD Opteron 242, 1.6 GHz	3
dcs0	Supermicro H8DCi	Dual Core AMD Opteron 265, 1.8 GHz	3
dcs1	Supermicro H8DCi	Dual Core AMD Opteron 265, 1.8 GHz	4
ecs0	Tyan Thunder K8S Pro	AMD Opteron 242, 1.8 GHz	2
ecs1	Tyan Thunder K8S Pro	AMD Opteron 242, 1.8 GHz	2
vobox0	Supermicro H8DCi	Dual Core AMD Opteron 265, 1.8 GHz	4
vobox1	Supermicro H8DCi	Dual Core AMD Opteron 265, 1.8 GHz	4

Table 7.1.: HLT Cluster: Gateways

  - **Mass Storage:** These nodes are used for running Andrew File System (AFS) [8]. The file servers are in charge of hosting the user home directories and the home directory for the HLT application, i.e. for the distribution of the HLT software. The mass storage nodes are configured according to table 7.2

Node	Motherboard Type	Processor Type (2 CPUs inside)	Memory/GByte
ms0	Supermicro X8DTT	Intel Xeon E5520, 2.27 GHz	24
ms1	Supermicro X8DTT	Intel Xeon E5520, 2.27 GHz	24
ms2	Tyan Thunder K8WE	Dual Core AMD Opteron 265, 1.8 GHz	2
ms3	Tyan Thunder K8WE	Dual Core AMD Opteron 265, 1.8 GHz	2

Table 7.2.: HLT Cluster: Mass Storage

[6] Winbond is a registered name of the Winbond corporation.
[7] DAQ is a part of the ALICE experiment responsible for handling firstly the storage of detector data to the tape drives and secondly data forwarding to the HLT.
[8] ECS is the control system for the ALICE experiment.
[9] Taxi is an application used by the HLT for getting calibration data.

– **Monitoring and System Management:** These nodes are used for running the server side of the SysMES framework (i.e. the Operator and Management layers of figure 5.4). Parallel to this they are also used for hosting LHC Era Monitoring (Lemon) [71], which is used for collecting data from the nodes and offers statistical visualization of this. The basic configuration of the monitoring and system management nodes is described in the table 7.3

Node	Motherboard Type	Processor Type (2 CPUs inside)	Memory/GByte
mon0	Supermicro X8DTT	Intel Xeon E5520, 2.27 GHz	24
mon1	Supermicro X8DTT	Intel Xeon E5520, 2.27 GHz	24

Table 7.3.: HLT Cluster: Monitoring and System Management

– **Database:** These nodes are used for hosting Oracle [10] 11 database servers. Both nodes are configured for building a hot-failover instance based on the Oracle Active Data Guard [85] technology. This technology allows the definition of one or more standby databases in order to protect productive database instances against failures. The standby instances are synchronized and can be used for the distribution of read-only transactions and, consequently, to distribute load.

  In the case of the HLT Cluster there is a master database instance located on db0 and a standby instance on db1. Table 7.4 shows the basic hardware configuration of the database nodes.

Node	Motherboard Type	Processor Type (2 CPUs inside)	Memory/GByte
db0	Supermicro X8DTT	Intel Xeon E5520, 2.27 GHz	24
db1	Supermicro X8DTT	Intel Xeon E5520, 2.27 GHz	24

Table 7.4.: HLT Cluster: Databases

– **Development:** These nodes are used for the development of the HLT application, for the creation of simulated data and for testing the HLT application. Another usage of the development nodes is for running the NX server [84] for sessions and virtual desktop management for the users of the cluster. Table 7.5 shows the basic hardware configuration of the development nodes.

– **Graphical User Interface:** The GUI nodes are in charge of hosting the ESMP [47] application. This application has been developed for the visualization and controlling of the HLT. Table 7.6 shows the basic hardware configuration of the GUI nodes.

The previously presented facts about the cluster nodes make clear that the HLT Cluster is a heterogeneous cluster. Two types of heterogeneity can be identified. Functional heterogeneity is related to the different dedicated node types, e.g. CN, FEP, etc. and a consequence of this is that nodes from different types normally are not interchangeable. Physical heterogeneity is related to the different hardware and software configuration of the nodes.
The following section introduces the SysMES setup for managing the HLT cluster.

[10] Oracle is a registered name of the Oracle corporation.

Node	Motherboard Type	Processor Type (2 CPUs inside)	Memory/GByte
dev0	Supermicro X8DTT	Intel Xeon E5520, 2.27 GHz	24
dev1	Supermicro X8DTT	Intel Xeon E5520, 2.27 GHz	24
dev2	Tyan Thunder H2000M	Quad Core AMD Opteron 2346 HE, 1.8 GHz	8
dev3	Tyan Thunder H2000M	Quad Core AMD Opteron 2346 HE, 1.8 GHz	8
mondev0	Tyan Tempest i5400PW	Intel Xeon E5420, 2.50 GHz	16
mondev1	Tyan Tempest i5400PW	Intel Xeon E5420, 2.50 GHz	16

Table 7.5.: HLT Cluster: Development

Node	Motherboard Type	Processor Type (2 CPUs inside)	Memory/GByte
gui0	Supermicro H8DCi	Dual Core AMD Opteron 265, 1.8 GHz	4
gui1	Supermicro H8DCi	Dual Core AMD Opteron 265, 1.8 GHz	4
gui2	Tyan Thunder K8WE	Dual Core AMD Opteron 265, 1.8 GHz	8

Table 7.6.: HLT Cluster: GUI

7.1.2. SysMES@HLT Configuration

The SysMES framework installation for the HLT Cluster is composed of the two clustered servers mon0 and mon1. Figure 7.2 visualizes the functionalities running on each server.
Both servers are configured for running the functionalities located in all layers of the SysMES architecture (see section 5.2) in a fault-tolerant manner. In case of a system crash, the other server is able to handle the management of the cluster.
The functionalities located on the servers are Event Management, Task Management, Rule Management, GUI and modeling. The only difference between the setup of both servers concerns the Complex Rule Management subsystem. The mon0 server is configured as a master and the mon1 as a slave.
Each of these servers has an access point for managing the connections to the SysMES clients.
The JBoss AS [65] version 5.0.1.GA is installed on the SysMES servers. JBoss AS is an open-source implementation of Java Enterprise Edition [63] specification and is used as middleware for running the SysMES server side functionality.
There are two database servers as a back-end for the persistent storage of the management objects. As already mentioned in the previous section 7.1.1.2, the db0 and db1 servers build a fault-tolerant database instance using Oracle Active Data Guard.

7.1.3. SysMES@HLT Management Strategy

SysMES management objects have been designed to be configured individually and according to a specific management strategy.
A system administrator is in charge of defining the management strategy for the cluster. In order to do this, the system administrator first has to find answers to the following questions:

1. Who should use the SysMES framework?

   The SysMES framework can be used by system administrators (experts) or operators. Experts use SysMES for an active management of the cluster. They are able to change the manage-

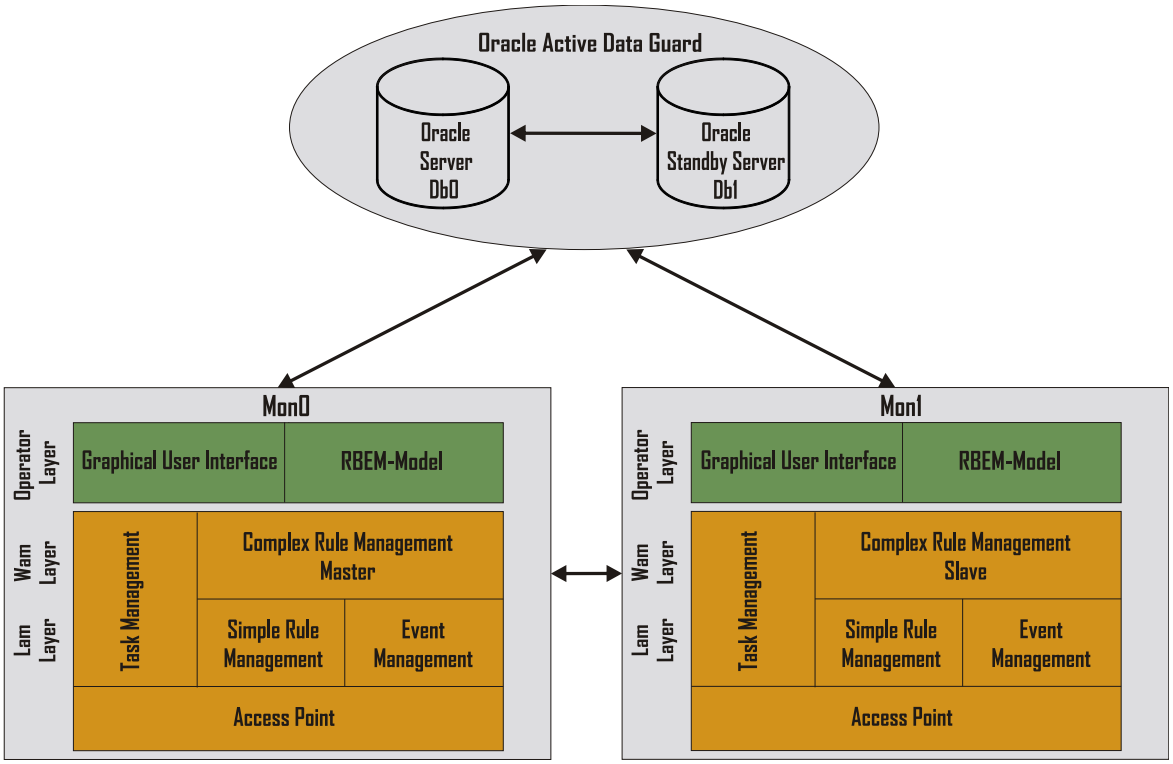


Figure 7.2.: SysMES@HLT Management Configuration

ment environment by developing new management resources or just by redefining the object attributes of existing objects. Operators have restricted rights and responsibilities and therefore they also should use SysMES in a restricted way.

2. What should be managed?

   SysMES can be used for monitoring and managing any kind of cluster resource, such as hardware, applications, network, racks, etc. It is possible to define Monitors which run locally on the SysMES client (e.g. for monitoring the hardware), but it is also possible to access sensor data remotely, e.g. using SNMP [97] and IPMI [61].

3. What information should be offered, displayed?

   SysMES offers the possibility to inform system administrators and operators via SMS, email or just with an Event on the SysMES GUI. For each Monitor and Event Class it is possible to define the desired recipients and the communication medium.

4. How should be reacted in case of errors or failures?

   It is possible to react to errors/failures manually by deploying Tasks or automatically by the execution of Rule triggered Actions. It is important to define a clear strategy, which defines how much autonomy is desired and allowed.

5. Are manual interactions required?

A system administrator should define a set of Tasks which allows him and the operators to execute actions manually. This is important especially for unprivileged operators because for them this is the only way to perform some actions on the cluster nodes.

6. Are multiple management strategies required?

   This question concerns the option of developing different management strategies depending on the state of the cluster (open for all users, in maintenance, etc) or the mode (production, test).

In principle, the management strategy includes the definition of how cluster resources have to be monitored (i.e. timing, mode, associated Event Classes, etc) and the definition of desired reactions in case of errors or failures. A system administrator develops a set of Monitors for observing the cluster resources, a set of Rules for the recognition of (un)desired states, a set of Actions (attached to the Rules) for reporting these states and solution of errors and a set of Tasks for the manual intervention in the cluster.
In the case of the HLT Cluster the answers to these questions are:

1. Who should use the SysMES framework?

   SysMES should be used by both experts, who are able to develop new management objects and have full access to the cluster nodes, and operators who use it for getting an actual state of the cluster nodes. HLT operators normally are in charge of configuring and taking care of the HLT application and use SysMES for recognizing whether cluster nodes have failures. They do not have the rights to access the cluster nodes and to perform administrative tasks on the cluster.

2. What should be managed?

   SysMES manages all parts of the HLT Cluster, i.e. the hardware, applications including the HLT application and the network.

3. How should be reacted in case of errors or failures?

   In the case of the HLT Cluster, it is important to try to solve errors automatically. This is necessary in order to reduce the number of human interventions in the cluster. Specific reactions to errors are introduced in the following section.

4. Are manual interactions required?

   Task-based manual interactions are required. A collection of Simple Tasks will be offered. This includes Tasks for reading out system information (e.g. execute "ps -ef") or for performing administrative tasks (e.g. for shutting down single nodes).

5. Are multiple management strategies required?

   Two different management strategies have been developed for the HLT Cluster, the "production" and the "testing" strategies.

   The production strategy is used for the period when the ALICE experiment is running. In consideration of the fact that the major part of the SysMES users of the HLT Cluster are unprivileged operators, it is important to configure SysMES for running in an unattended mode.

   The unattended mode distinguishes between those Events which can be treated automatically, i.e. the problem can be solved by the automatic execution of Tasks, and those which can only be solved manually by a system administrator.

Automatic treatable Events are generated with a Severity of `"Critical = 2"` (appears orange in the SysMES GUI). Furthermore, two Rules have been developed for its processing. The first one is a Simple Rule which tries to solve the problem actively by the execution of Tasks. This Rule fires upon occurrence of one of these Events.

The second one is a Complex Rule (A so-called Escalation Rule). It fires if a predefined number of instances of the same Critical Event type occurs, i.e. the problem can not be solved by the Simple Rule. The Complex Rule has two associated Actions. One informs the system administrators that the Simple Rule was not able to solve the problem and the other one escalates the problem by generating a new Event with a Severity of `"Immediate = 1"` (appears red in the SysMES GUI). Red Events in the SysMES GUI indicate to the operator that a major failure has occurred. The operator is also in charge of calling an expert and excluding the node (i.e. the Event originator) from being used by the HLT application.

Further Rules are used for informing system administrators about Event occurrences which can not be treated automatically.

The testing strategy is a simplified version of the production strategy, which does not include some Monitors, e.g. those for monitoring the HLT application, and also uses no problem escalation by Rules. Another specialty of this strategy is that several Monitors have an Event Class with a Severity of `"Information = 4"` (appears green in the SysMES GUI). This Event Class utilizes an `"always operator (a)"` and therefore all Monitor measurements are sent to the server. This is a flexible manner to collect real time data using the Monitors.

The following section is an overview about the management objects developed for the production management strategy.

7.1.3.1. HLT Cluster Monitoring

The first step for managing a cluster is the definition and development of Monitors. In the case of the HLT cluster, the developed Monitors can be divided into the following categories:

- Basic monitoring: This is a collection for monitoring the basic hardware and software resources. At the moment there are Monitors for checking the availability of server services (e.g. AFS, DHCP, DNS, LDAP, etc.) but also for checking the respective client services (e.g. daemons for AFS, NTP, etc.). There are Monitors for checking the status, utilization and temperature of hard disks, for checking the status of the raid controller, for checking several temperature values (CPUs, main board, etc.) and the speed of the system fans, for checking the state and usage of the memory and for checking the system time status.

  There are some special Monitors for checking the validity and age of an AFS backup, the availability of AFS volumes and LDAP servers and for comparing the state of both LDAP server databases.

- Rack monitoring: In the HLT Cluster, there is a Rack Monitoring System (RMS) installed. This RMS is used for monitoring the voltage and room temperature. Outside of the RMS (normally in a server) a SysMES client is installed and responsible for accessing this information via SNMP. Currently, there are Monitors for checking the voltage value of the three-phase electricity and for checking the temperature in the counting room.

- Network monitoring: There are two strategies for the usage of network related Monitors. Monitors which read out statistics of the cluster nodes, and Monitors, which access the network switches via SNMP. There is a local Monitor for the cluster nodes, used for reading out the configuration of the network interface related to the actual network speed, i.e. 10 M/Bit, 100 M/Bit or 1000 M/Bit. Another important local Monitor is the network statistics Monitor, which is used for getting statistics about the network interfaces. The following metrics are monitored: collisions, rx_crc_errors, rx_dropped, rx_errors, rx_fifo_errors, rx_frame_errors, rx_length_errors, rx_missed_errors, rx_over_errors, tx_aborted_errors, tx_carrier_errors, tx_dropped, tx_errors, tx_fifo_errors, tx_heartbeat_errors, tx_window_errors.

  There are also Monitors for the measurement of the actual network bandwidth usage and for the measurement of packet loss.

  Every port of every switch is monitored by accessing their statistical information via SNMP. This information concerns the numbers of collisions, CR alignment errors, fragments, jabbers and drop packets.

- H-RORC monitoring: These Monitors are deployed to all FEP nodes. There are two Monitors for checking the status of the H-RORC (i.e. online or offline) and the internal status of their buffers. Another Monitor is used for providing the actual Event counter of the H-RORC. This value should increase during a run of the HLT application. The correlation of this information is used in order to recognize if a H-RORC has a problem or if it has stopped working.

- CHARM monitoring: The CHARM card is equipped with one internal and 6 external temperature sensors, which can be placed on different parts of the computer nodes. These sensors are used for monitoring the temperature of the H-RORC, chassis, power supply, CPUs and hard disk. There is also a stand-alone CHARM card, which is used for monitoring the cooling plant temperature (in and out) and the rack cooling temperature (in and out) in order to recognize failures in the provided cooling plant.

  As described in the referenced document [87, section 6.3.1.], the CHARM card is able to capture the actual screen content. This capability is used in a SysMES Monitor in order to recognize bugs, which are reported to the computer console.

  Another feature of the CHARM card is its capability to read out the Power On Self Test (POST) codes [11] of the cluster nodes. This feature is used for checking the state of a node after it has been rebooted. In this regard, there are other Monitors for checking the CMOS values of the nodes. More exactly, there are Monitors for checking the status of the CMOS battery and the checksum.

- HLT monitoring: The High-Level-Trigger offers an API, which is used for accessing and monitoring its relevant components and internal resources. These are: Output Buffer Usage, Current Receive Rate, Current Process Rate, Received Event Count, Current Received Event Count, Processed Event Count, Current Processed Event Count, Current Processed Input Data Size, Current Processed Output Data Size, Current Processed Input Data Rate, Current Processed Output Data Rate, Total Input2Output Data Size Ratio, Current Input2Output Data Size Ratio, Announced Event Count, Current Announced Event Count, Current Announce Rate, and Total Output Buffer.

[11] A POST code represents the status of a test during the boot process.

The data gained by the SysMES Monitors is used in order to recognize bottlenecks in the HLT, as well as for collecting statistics.

- Other applications: There are other applications such as "Taxi" used for calibration of the HLT against other components of the ALICE experiment. Another monitored application is "squid", which is a proxy for the communication to the ECS.

7.1.3.2. Rules and Automatic Reactions Strategy

The strategy for automatic problem recognition, solution and reporting is realized by Rules. According to the required recognition complexity and reaction speed, it is possible to allocate the Rules in the SysMES clients, LAM or WAM.
The management strategy 7.1.3 introduced before describes the common and standard way to handle (un)desired states. It is composed of the next four parts.

- Default Management Strategy: Conforming to the previously discussed management strategy 7.1.3, there are Server Simple Rules for the solution of problems and Complex Rules for reporting and the escalation of the Events to a higher Severity. As an example of this, there are pairs of a Server Simple Rule and an escalation Complex Rule for handling crashes of the AFS daemons (AFSDaemonDown, AFSDaemonEscalate). Problems with the availability of the NTP daemon or with the system clock synchronization are treated by the Rule pair (NTPDaemonDown, NTPDaemonEscalate). Another example concerns the recognition of a very high usage of the "/tmp" partition. The log files of the HLT application are located in this partition and therefore it is strictly necessary to guarantee free hard disk space. This is realized by the definition of a Rules pair (CleanTmp, TmpUsageEscalate) for removing old log files in a desired manner and to inform an system administrator if this procedure can not be performed with a successful result (i.e. usage of /tmp <= 95%).

  As a reminder, if Events with a critical Severity occur, then the recovery method starts (Simple Rules). If this method is successful and the problem is solved, then there will be no more Events of this Severity, otherwise the escalation Complex Rule fires, generating a new Event with a higher Severity (i.e. immediate) and sending a SMS or email to the experts.

  The last part of the default management strategy consists of the definition of Complex Rules for the recognition of complex states. Examples of this are used in the next section 7.1.4.

- Emergency Strategy: In general, this strategy consists of Client Simple Rules. It is used if the SysMES clients have to react standalone without server interaction or for the realization of a immediate reaction in order to avoid damages.

  There are Simple Rules for the recognition of failures of the DHCP server, DNS server, AFS server and LDAP server. These Simple Rules act locally on the SysMES client and induce a restart of the services which have crashed. This method is a local emergency procedure, which also works in case of network disconnection or unavailability. The emergency strategy was developed as a part of the production management strategy and therefore escalation Complex Rules extended it.

  Another emergency procedure concerns a locally triggered node shutdown in case of a very high temperature value of the CPU or the hard disk. For this case, there are Simple Rules for Immediate Events originated by the CHARM cards or the node itself. In this case there are no

escalation Rules because the Events which trigger those Rules have been generated with the highest Severity `"Immediate = 1"`.

- Simplified Management Strategy: This management strategy includes the definition of Rules for problem solution without problem escalation, for example for Events which occur definitely only once. Examples for this case are the Rules for reacting to a critical temperature of the CPU and hard disk. The reaction to such an Event is a server initiated shutdown of the Event originator. There is no escalation Complex Rule for this Rule because the node goes down due to the Simple Rule execution and therefore no more Events follow.

- Reporting Strategy: The next step in the realization of the management strategy concerns the definition of Rules (Simple Rules or Complex Rules) for Events which can not be treated automatically. These Rules are used in order to inform an system administrator about the recognition of an (un)desirable state. As an example, Rules have been developed for reporting errors in the Redundant Array of Independent Disks (RAID) controllers, for informing about recognized kernel panics, for reporting a power outage, etc.

7.1.3.3. Tasks Collections

As mentioned in section 5.4.2.5, Tasks have either a configurational or an administrative purpose. There are Configuration Tasks for the (re)distribution and (re)configuration of all other management objects introduced before, i.e. Monitors, Simple Rules and Complex Rules.
Besides these, there is a collection of Tasks, i.e. Simple Tasks used for the manual interaction with the cluster nodes. In principle, these Simple Tasks are used for the execution of an action in one or multiple cluster nodes and for getting the result of the execution by Events (more exactly by Task Reply Events).
Very useful examples of Simple Tasks are:

- HostPowerOn and HostPowerOff: As the name implies, these Simple Tasks are used for starting and shutting down the cluster nodes, and consequently, the entire cluster. The HostPowerOn Task has to be executed in the remote management solution of each node (i.e. CHARM, BMC, etc.) in order to power it on. The execution of this Task in multiple cluster nodes at the same time can cause a voltage drop with unpredictable consequences. In order to avoid this, a waiting interval is calculated for each node. It is possible to define a minimal interval as a parameter for the Task.

  There are two realizations of the HostPowerOff Simple Task. The first one is executed directly on the cluster node and causes a soft shutdown of the node (i.e. the execution of "shutdown -h now"). The second one has to be executed in the respective remote management card and it causes a interruption of the power supply of the node. This Task should be used carefully and only in cases when a node has crashed and cannot react anymore.

- CheckSSH: This Task is used for testing passwordless access from any cluster node to any other. A successful execution of this Task is required for running the HLT application.

- OS Commands: There is a set of OS commands which can be executed using Simple Tasks. Examples of those are "ps" for listing the current processes, "df" for listing the actual disk space usage, "who" for listing logged-in users and "date" for checking the actual system date and time, etc.

- RORC related Simple Tasks: This set of Tasks is used for interacting with the H-RORC. This method allows a remote update of the firmware, the (de)activation of several configurations and the execution of other actions.
- TMGetStatus: This Task is used for getting information about the current state of the HLT Task Manager (TM), which runs on each node. The possible returned states are "running", "stopped" or "error".

The SysMES framework also offers an interface for the creation of Simple Tasks in a quick and dynamic manner. The "taskify" part of the Task deployment window (see figure 5.41) can be used by the administrators in order to execute any command. Tasks which have been generated using this, way are stored in the CIM model (more exactly in the database where CIM management objects are stored) and can be reused at any time.
The following section gives an overview about the interaction of different management objects.

7.1.4. SysMES@HLT Management Scenarios

The previous section described management objects in an isolated manner. This section illustrates how management objects can be used in an integrated manner in order to manage a specific scenario. Four typical scenarios for the HLT have been chosen.

7.1.4.1. Event Rate Monitoring:

As already mentioned in section 1, the HLT is responsible for the online analysis of data from the ALICE experiment. The HLT analyzes the data and generates a decision concerning its physical relevance. Relevant data is sent back to the DAQ including the additional decision. In the normal running status of the HLT, the incoming data rate is equal to the outgoing data rate.
Monitoring the data rate is necessary in order to recognize discrepancies between the input data rate and the output rate in the HLT. For this purpose several SysMES Monitors have been developed. The Monitor for the incoming data rate is deployed to a group of FEP nodes, e.g. the `"fepspd0-4"` nodes and the Monitor for the outgoing data rate is deployed to the `"fephltout0-2"` nodes. On the incoming side it is not necessary to Monitor the value of all FEP nodes because this value is equal everywhere. On the outgoing side it is necessary to Monitor all fephltout nodes because the going out data rate is the sum of the value of all fephltout nodes.
The recognition of data congestion in the HLT is realized by a Complex Rule. The Complex Rule correlates on the one side the information contained in the SysMES Events, which reports the incoming data rate and calculates the maximum value of all monitored FEP nodes, and on the other side adds the outgoing data rate values to a single value. The Complex Rule fires if the input rate minus the output rate is greater than a predefined threshold.
In the case of a match, the Complex Rule fires and sends an email and a SMS to the experts, who react manually to this problem. In the future, it is planned to extend this functionality to including monitoring of the local buffers of all nodes in the HLT analysis chain and the correlation of this information with detected data congestions and reacting with a analysis chain reconfiguration.

7.1.4.2. Power Supply Failure:

As already mentioned in 7.1.1, the HLT cluster utilizes 52 racks, which are located in the computation facilities of CERN - Point 2. Four of those racks (i.e. those where the servers and network infrastruc-

ture is located) are connected to an Uninterruptible Power Supply/Source (UPS), which guarantees power availability and stability in case of failures in the main power supply.
The current installed UPS is able to supply these four racks for a period of about 15 Minutes. All computers in the other racks will crash if the power supply is interrupted.
The strategy for avoiding damages to the server and network infrastructure consists of monitoring the voltage values of the three-phase electricity, detecting a power supply interruption, waiting a period and shutting down the servers in a staged way.
As already introduced, there is a SysMES Monitor which utilizes the RMS for getting the actual voltage value of the three-phase electricity with a `"Period=39 seconds"`. This Monitor has been developed for running on any cluster node. It accesses the values measured by the RMS (3 values one for each phase) via SNMP. At the moment, this Monitor has been deployed to the SysMES clients on the monitoring (mon0, mon1) and the database (db0, db1) servers. Each Monitor generates an immediate Event `"EventName = RMS_Voltage_Low, Severity = 1"` if the lowest measured voltage value of the three-phase electricity is less than 200 Volt.
The second part of the error detection is realized by a Complex Rule. The Complex Rule is activated by the first occurrence of one RMS_Voltage_Low Event. The Complex Rule fires upon occurrence of three RMS_Voltage_Low Events from two different servers within a time interval of 150 seconds. Afterwards, the staged shutdown of the servers is initialized by the execution of a Task Action. The execution behavior of this action is defined by their attributes `"ReEnableTime=3600"` and `"ExecutionCount=1"`. Conforming to this setup the action will be executed once and disabled for the next 3600 seconds in order to avoid multiple executions.
The number of needed Events is a part of the Complex Rule configuration and it depends on the desired grace period before the shutdown procedure has to be performed (i.e. for avoiding a shutdown by a glitch).
The server shutdown action is performed on one of the gateway servers. The servers are divided into three groups in order to shutdown these in a staged manner. Group1 is composed of the servers gui0, gui1, gui2, ecs0, ecs1, dcs0, dcs1, vobox0, vobox1, dev0, dev1, dev2, dev3, mon0 and mon1, group2 is composed of the servers ms0, ms1, ms2, ms3, db0, db1, and finally, the group3 is composed of the servers gw0 and gw1.
This division has been made because the home directories of the cluster users are stored on AFS servers and therefore cached data in running sessions on the servers of the group1 should be stored persistently to the AFS servers (i.e. ms0-3) before these go down.
The shutdown action executes the shutdown in parallel on all servers of a group, afterwards it waits 2 minutes before executing the action on the next group.

7.1.4.3. Kernel Panics of the Hosts:

Kernel panics are very serious errors of the cluster nodes, which affect the smooth usage of the cluster resources running the HLT. Normally if a kernel panic occurs, then the trace information is dumped to the console. This behavior is problematic for monitoring and managing the nodes remotely because first, it is not possible to observe the consoles of all nodes by administrators and second, there is no network connectivity to the crashed node.
In order to monitor the console of the nodes for recognizing kernel panics a Monitor for the CHARM card has been developed. This Monitor uses the capabilities of the CHARM card to access the host console and searches for kernel panics in the text. If the Monitor detects a kernel panic then the CHARM card generates an Event `"EventName = KernelPanicScan, Severity = 1"`.

For the treatment of this Event there is a Simple Rule which is responsible for restarting the node and for informing the system administrators about this issue by SMS and email.
Similar to this is the management procedure in case of software locks, bugs and other errors.

7.1.4.4. CMOS Errors:

The error state to be recognized is when a computer freezes during the start up procedure and waits for a manual intervention of a user pressing F1 because the CMOS battery is low or the CMOS check sum is bad.
In order to Monitor these values the CMOS Status Monitor has been developed. This Monitor reads out the CMOS battery status and the CMOS check sum status of the computer node via the CHARM card. In case of a low battery value or a bad check sum value an Event is generated.
The primary solution of the problem consists of informing a system administrator about this state (i.e. for replacing the battery) and pressing F1 in an automatic way.
As mentioned in section 5.3.1, there are Monitors for checking the status of the actual CMOS values. The CMOSStatus Monitor is deployed to the CHARM card and it generates several Events if one of the measured values is equal to `"bad"`. Furthermore, a Monitor has been developed for the observation of the POST code of the node. This Monitor is also deployed to the CHARM card and it generates an Event if the actual POST code is equal to `"0x85"`, which means that an error has occurred during the booting procedure.
A Complex Rule (see figure 5.27) correlates the Events from the CMOS Status Monitor with those of the POST code Monitor. In principle, the Complex Rule is fulfilled by the occurrence of one CMOS Status Event and two POST code Events from the same node within a time `"Period=65000 ms"`. In case of a match, the SysMES server executes an action on the CHARM card, which is able to make a user interaction pressing F1 in order to guarantee that the node comes up. The second action sends an email and a SMS to the system administrators in order to solve this problem, i.e. by changing the system battery.

7.2. Scalability Tests

One desired characteristic of a system management solution is a high grade of scalability. According to the scalability definition in 4.3, the SysMES framework is scalable because it can deal with increasing load by the extension of its system resources (i.e. addition of new server instances) and also by the relocation of the system management services (e.g. Event processing in the SysMES client instead of the SysMES server).
The scalability grade of SysMES has been tested by five test series. Each of those test series is composed of a number of single tests, which have been performed using the SysMES functionality (Events, Rules, Tasks) in the production environment of the HLT cluster (see section 7.1.2). As previously mentioned, there are two SysMES servers (mon0 and mon1) and a Oracle Active Data Guard installation as a back-end running on the db0 and db1 servers.
All test series are designed to ascertain the maximum work which can be processed by a SysMES server. The same test series is repeated with two servers in order to compare the measured values with doubled management resources. The different configurations of the test series are used to demonstrate the scalability capabilities through relocation of management capabilities.
Each test series has a specific configuration which, depends on the used SysMES functionalities, i.e. the capabilities for receiving and processing of Events (e.g. by Simple Rules or Complex Rules) and

the execution of Actions (locally on the server or remotely on the clients).
A single test of a series is defined by the configuration of the test parameters "Number of Clients" (**# SysMES Clients**) and "Number of Events per Second and Client" (**Evt / (Sec x Client)**).
These two parameters are used in the following manner:

- # SysMES Clients: There were 150 cluster nodes available for testing. The number of used SysMES clients depends on the respective test and it varies between 10 and 1000 clients.
- Evt / (Sec x Client): This parameter is defined by the number of Monitors running on one client and the Period of the Monitors.

Each test (i.e. the combination of those two variables) has a duration of 300 seconds.
The evaluation of a test is divided into two steps:

- Calculate the processing time for each Event: This procedure considers two time measurements for each Event. The first time is the generation time in the clients and the second time is the end of the Event processing on the server side. Note that Event processing is different according to the running configuration.
- Calculate the processing time for the test: The test result is the average of all Event processing times.

Due to the time related distributed measurements, (i.e. first measurement in the client, second in the server) it is necessary to synchronize the system clocks of the servers and nodes before starting the test. In order to have the same test conditions, the database servers are emptied before starting a test.
A test framework has been developed. This framework is used for configuring the test series setup i.e. client installation, starting, stopping, deployment of Monitors and Rules, as well as for the realization of the time related activities (e.g. compliance of a defined test duration). Another task for this framework concerns the automatic test evaluation and logging.
The value of ranges for the test parameters (i.e. "# SysMES Clients" and "Evt/(Sec x Client)") for each test series has been defined based on experience by the management of the HLT cluster using the SysMES framework. This experience is also used for the definition of a maximum average processing time that defines a passed test.
The standard deviation of the following test has been measured in a range of [0.2 - 2.23], which makes it impossible to display these values in the figures.

7.2.1. Test Series 1: Server Simple Rules & Server Actions

As an initial test series, this test has been designed for testing most of the functionality located on the LAM server side. These are Event Management and Simple Rule Management.

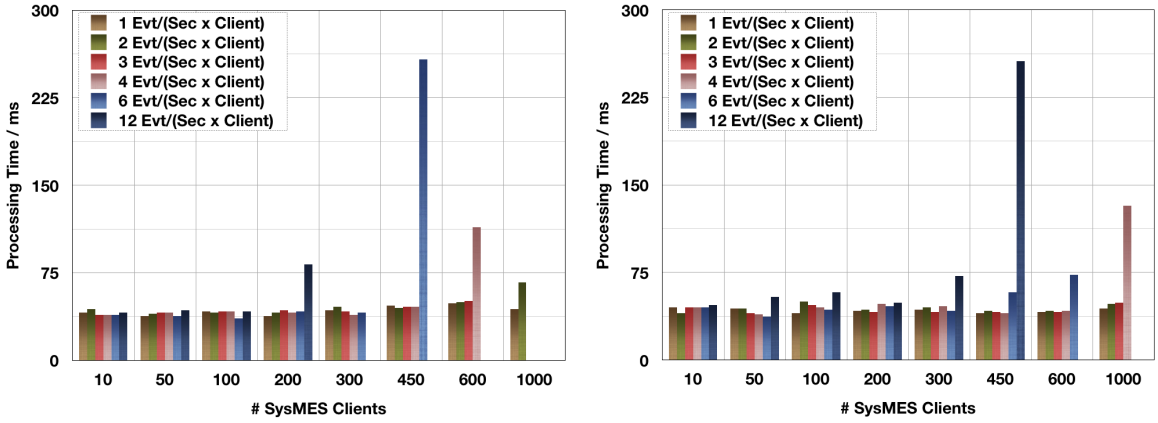
Configuration: Both SysMES servers are configured with 48 Server Simple Rules with server Actions. Each client runs 12 Monitors and generates 12 different kinds of Events. The Events of those Monitors match to the 48 Simple Rules. The Simple Rules have a server Action, i.e. an Action which will be executed on the SysMES servers.
The single tests of this test series have been set up as follows:

- # SysMES Clients: 10, 50, 100, 200 (100 nodes x 2 SysMES clients), 300 (150 x 2), 450 (150 x 3), 600 (150 x 4) and 1000 (100x10).
- Evt/(Sec x Client): 1, 2, 3, 4, 6, 12 (i.e. 12 Monitors / 12 sec, 6 sec, 4 sec, 3 sec, 2 sec, 1 sec).

Test Procedure: Each test of the series starts with deployment of the 48 Simple Rules to the SysMES servers and the 12 Monitors to the clients. After that, the clients send Events to the servers. Each Event checks all 48 Simple Rules and causes firing of four Simple Rules. As a consequence of this, four Actions per Event are executed. The server Action is configured for writing a control message to a log file.
To sum up, the test procedure is composed of the following steps:

- Execution of a Monitor.
- Event generation with the first time measurement point.
- Event sending to the server.
- Simple Rule matching and Action execution on the server.
- Event storing.

Evaluation: The first time measuring point is the Event generation time which is coded in an Event attribute. The second time measurement point for the evaluation of this test is set after the execution of the server Action. The test series is composed of all tests with an average processing time of less than 300 ms (test passed if average processing time < 300 ms).



(a) One Server

(b) Two Clustered Servers

Figure 7.3.: Test1: Processing time of the Events which trigger Server Simple Rules and react by Server Actions

The following bar charts 7.3(a) and 7.3(b) visualize the test series results for one server and for two clustered servers. The X axis represents the value of the parameter "# SysMES Clients". The bars (6 with different colors) are related to the value of the second test parameter "Evt/(Sec x Client)" for a specific value on the X axis . The Y axis represents the measured test results, i.e. average processing time for the current test setup and is displayed as the height of the bars.
In other words, the measured values describe how long it takes for generating an Event, sending it to the server, storing it in the database, checking 48 Simple Rules and firing four of these with a server Action.
The behavior of SysMES for a single server and two clustered servers is similar. The processing time for Events is between 40 ms and 70 ms until a maximum number of Evt/Sec has been reached. After

this the processing time increases. In the case of the test series for one server (see diagram 7.3(a)), the first results with increased processing times are for 2400 Evt/Sec (12 Evts/(Sec x Client) x 200 Clients and 4 Evts/(Sec x Client) x 600 Clients) and 2700 Evt/Sec (6 Evts/(Sec x Client) x 450 Clients). Those three test are the last tests which had a measured processing time less than 300 ms. All missing tests in the bar diagram (e.g. 12 Evts/(Sec x Client) x 300 Clients) were also performed, but with a processing time greater than 300 ms and are omitted in the charts for a a better readability of the diagrams.
In the case of the test series with two clustered SysMES servers, the last tests with the desired processing time are performed for 3600 Evt/Sec (12 Evts/(Sec x Client) x 300 Clients and 6 Evts/(Sec x Client) x 600 Clients). The next higher test was performed for 4000 Evts/sec (4 Evt/(Sec x Client) x 1000 Clients) with an increased result of 132 ms.
In conclusion, both test series demonstrate the scalability grade of the SysMES server from 2000 processed Evt/Sec to 3600 processed Evt/Sec (i.e. an improvement factor of 1.8)

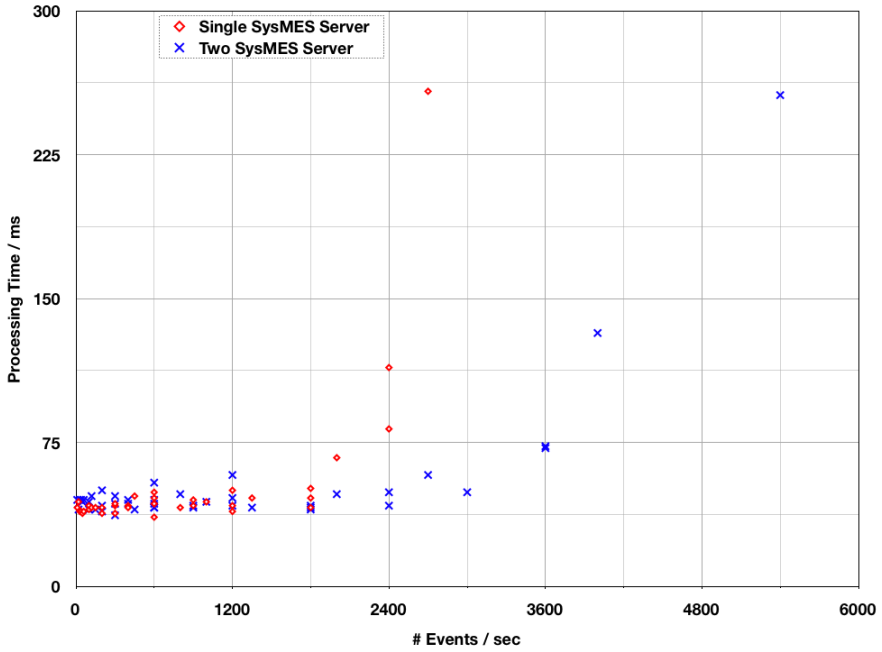


Figure 7.4.: Comparison of the Events processing time for one server and two clustered servers. Event processing by Server Simple Rules and reactions by server Actions

Figure 7.4 visualizes a comparison of the results of both previous test series. The X axis represents the absolute number of Evts/Sec independent from the configuration of the test parameters, e.g there are four results of each color for the X value 1200 Evt/Sec for the parameter configuration: 12 Evts/(Sec x Client) x 100 Clients, 6 Evts/(Sec x Client) x 200 Clients, 4 Evts/(Sec x Client) x 300 Clients and 3 Evts/(Sec x Client) x 400 Clients. The Y axis represents the average processing time for a single test. The colors of the points depict the number of SysMES servers used for the test (red = one server and blue = two servers).
The comparison of the results for both setups shows the behavior of the SysMES framework before and after a maximum of work has been reached. In the case of a single SysMES server, the red result points stay constantly under 70 ms until the limit of 2000 Evts/Sec is reached and they increase continuously if the number of Evts/Sec increases (114 ms for 2400 Evts/Sec, 258 ms for 2700 Evts/Sec, etc). For the setup with 2 servers it is possible to recognize the same behavior with a significantly

higher number of Evts/Sec. The Events processing times start increasing after 3600 Evts/Sec (132 ms for 4000 Evts/Sec, 256 ms for 5400 Evts/Sec, etc).

7.2.2. Test Series 2: Client Simple Rules & Client Actions

This test series is similar to the previous one and the most important change is the location of the Simple Rules, which is changed from the server side to the client side.

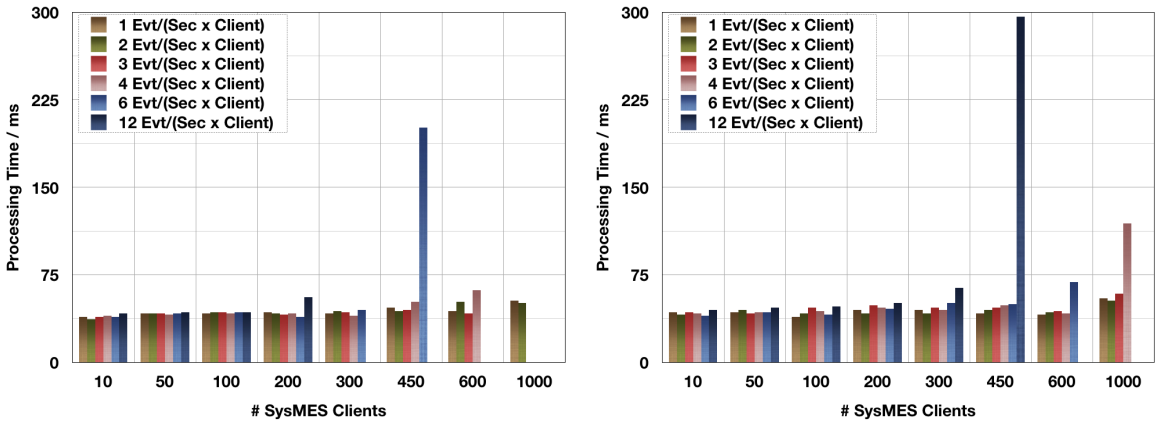
Configuration: The SysMES servers are configured for receiving and storing of Events. The Event processing functionality has been transferred to the SysMES clients. For this purpose the clients are configured with 48 Client Simple Rules, which process the Events generated by 12 Monitors. The Simple Rules have a Binary Action, which is executed decentralized on each client.
The configurations of the single tests of this series are:

- # SysMES Clients: 10, 50, 100, 200 (100 nodes x 2 SysMES clients), 300 (150 x 2), 450 (150 x 3), 600 (150 x 4) and 1000 (10x10).
- Evt/(Sec x Client): 1, 2, 3, 4, 6, 12 (i.e. 12 Monitors / 12 sec, 6 sec, 4 sec, 3 sec, 2 sec, 1 sec).

Test Procedure: The first part of each test consists of the configuration of the clients with the 48 Simple Rules and the 12 Monitors. After their deployment, the clients generate 12 different kinds of Events which have to be checked against the 48 Simple Rules. Those Rules are configured for matching and firing four times per Event. At each firing a Binary Action will be executed. This Action writes a control message to a log file on the client side.
The detailed sequence of activities is:

- Execution of a Monitor.
- Event generation with the first time measurement point.
- Simple Rule matching and Action execution local on the client.
- Event sending to the server.
- Event receiving and storing on the server.

Evaluation: As mentioned before, the first time measuring point is related to the Event generation time. The second time measurement will be made after the Event storing in the database. All single tests with an average processing time less than 300 ms have been taken into account for the generation of the evaluation plots.
The bar charts 7.5(a) and 7.5(b) visualize the results for the single tests for a configuration with one SysMES server and two clustered SysMES servers. In both cases, the X axis represents the values of the parameter "# SysMES Clients". The Y axis is the average processing time of each test and the different bars represent the values of the parameter "Evt/(Sec x Client)".
In the case of figure 7.5(a), it is possible to recognize that the test results stay constant until a work of 2700 Evts/Sec (6 Evts/(Sec x Client) x 450 Clients) with a processing time of 201 ms. The behavior for the test series with two clustered servers is similar (see figure 7.5(b)), where the processing time starts increasing at a load of 4000 Evts/Sec (4 Evts/(Sec x Client) x 1000 Clients) and 5400 Evts/Sec (12 Evts/(Sec x Client) x 450 Clients).



(a) One Server

(b) Two Clustered Servers

Figure 7.5.: Test2: Processing time of the Events which trigger Client Simple Rules and react by Client Actions

The interpretation of the results in figure 7.6 is clearer. This figure shows the direct comparison of the results of both test series. The red points are related to the results with one server and the blue points are the results with two servers. The processing time increases for the red points at a load of 2700 Evts/Sec and for the blue point at 4000 Evts/Sec.

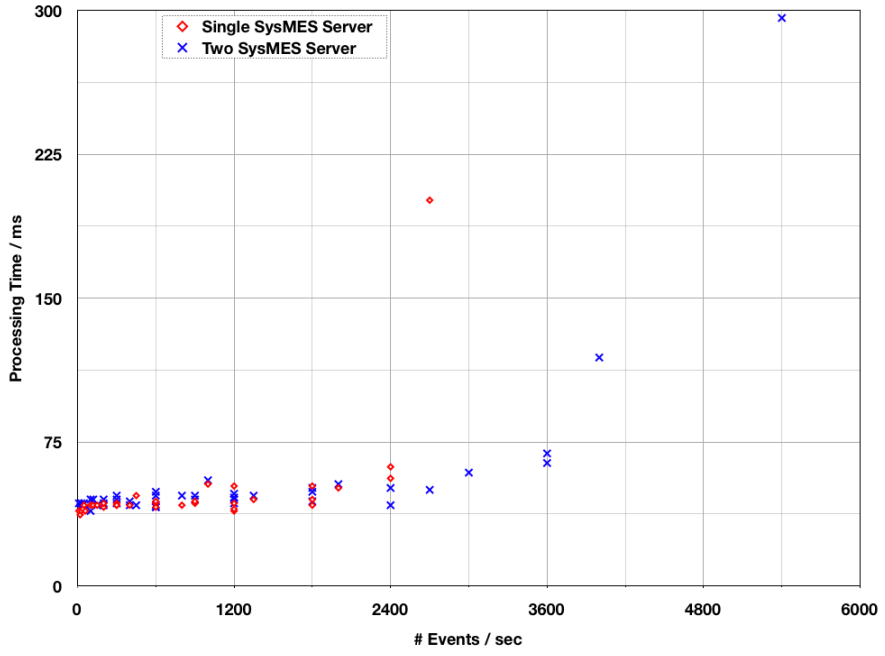


Figure 7.6.: Comparison of the Events processing time for one server and two clustered servers. Event processing by Client Simple Rules and reactions by client Actions

In comparison, the current test series differs from the previous one only in the location of the Simple Rules. In the first test series, the Simple Rules were located on the server side and in the current test

series on the client side.
The comparison of the results of both test series shows that it is possible to increase the performance by the relocation of functionality. For example, the maximum work one server can handle has increased from 2000 Evts/Sec in the first test series to 2400 Evts/Sec in the second test series. For the configuration with two servers the processing time has been reduced so that almost all tests in the second series have a processing time which is about 10 ms lower than in the first test series.

7.2.3. Test Series 3: Server Simple Rules & Client Actions (Task Actions)

This test series has been designed for testing the behavior of the SysMES framework in case that the Simple Rules have an Action attached which causes the execution of a Task on the clients. The location of the Simple Rules is irrelevant and therefore the server side has been chosen in order to minimize the number of Simple Rules to be deployed.

Configuration: The SysMES clients are configured with 12 Monitors which send 12 different kind of Events. For processing these Events the SysMES servers are configured with 48 Simple Rules with a Task Action. Actually, the Task Action is executed on the server, but it causes the generation of a Simple Task which is sent to the client. It is possible to send the Task to the Event originator which caused the Simple Rule firing or to a different predefined target. In the current test scenario the Simple Tasks are always executed on the target which sent the Event.
Based on the experience working with the SysMES framework and the expected effort in the test series, the test parameters have been chosen as following:

- # SysMES Clients: 10, 50, 100.
- Evt/(Sec x Client): 1, 2, 3, 4, 6, 12 (i.e. 12 Monitors / 12 sec, 6 sec, 4 sec, 3 sec, 2 sec, 1 sec).

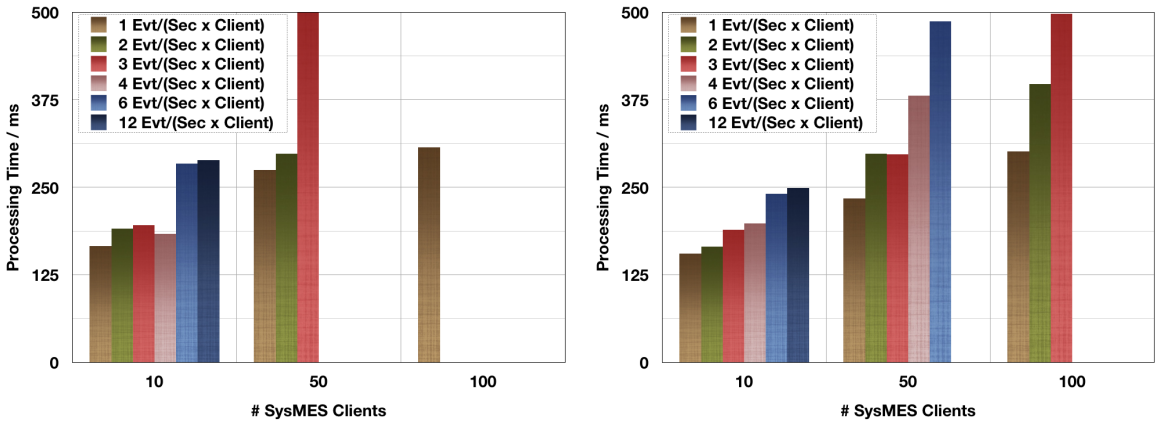
As mentioned in section 5.4.2.5 there are multiple ways to send a Task to a client. The first one is without acknowledgment, i.e. the Task is sent faster, but there is no guarantee for its successful deployment. The second way is with acknowledgment, which is more time consuming because two serially generated Ack Events have to be sent to the server in order to report the status of the Task execution. This test series has been configured for using Task acknowledgment as the safe manner for Task deployment.

Test Procedure: The first part of each test consists of the configuration of the servers with the Simple Rules and the clients with the Monitors. Afterwards, the clients send Events to the servers. Each incoming Event has to be checked against all 48 Simple Rules and it causes one Simple Rule to fire. The server then sends the Simple Task to the client and waits until this Action has been executed successfully. The Action reads the content of a file on the clients and sends it to the servers encapsulated in a TaskReply Event.
Processing one single Event with the selected functionality is composed of the following steps:

- Execution of a Monitor.
- Event generation with the first time measurement point.
- Event sending to the server.
- Event receiving and storing on the server.

- Simple Rule matching.
- Action execution and generation of a Simple Task on the server.
- Simple Task sending to the client.
- Client receives Simple Task, parses it and sends an Ack Event `"Info.Value = received"` to the server.
- Client executes Simple Task and sends an Ack Event `"Info.Value = executed"` to the server.
- Client sends a TaskReply Event with the result of the Task execution.
- Events are received and stored on the server.

Evaluation: The first time measuring point is, as in previous tests, defined after the Event generation. The second time measuring point is taken after the storing of the TaskReply Event, i.e. one measurement is composed of the processing time for the monitoring Event, the Simple Rule, the Simple Task, the two Ack Events and the Task Reply Event.



(a) One Server

(b) Two Clustered Servers

Figure 7.7.: Test3: Processing time of the Events which trigger Server Simple Rules and react on the client using Task Actions

Similar to the previous test series, this one is designed for using one single SysMES server first (see figure 7.7(a)) and two clustered SysMES servers as second (see figure 7.7(b)).
For both figures the X axis represents the value of the test parameter "# SysMES Clients". The Y axis represents the measured processing time and the different bars show the value of the second test parameter, "Evt/(Sec x Client)".
Both figures contain the single tests which passed with an average processing time of less than 500 ms [12]. All other tests were also performed, but these are not included in the charts for better readability of the results.
Figure 7.7(a) visualizes an average processing time between 150 ms and 300 ms until the maximal value of 120 Evts/Sec (12 Evts/(Sec x Client) x 10 Clients) is reached. This range describes the

[12]The maximal measured value is 499 ms

number of Events which can be processed continuously with this processing time. The first test that exceeds this value tries to process a load of 150 Evts/Sec (3 Evts/(Sec x Client) x 50 Clients) with a measured average processing time of 499 ms. This last value represents the amount of Events which can be processed temporarily, for example if there is a peak in the Event incoming rate.
The second figure 7.7(b) shows the behavior when two clustered SysMES servers are used. The first remarkable difference is the maximum load that can be managed in the normal value range of 150 - 300 ms. It is possible to process 150 Evts/Sec (3 Evts/Sec x50 Client). The second difference is that it is also possible to process more Events for a middle long time, i.e. 200 Evts/Sec (4 Evts/Sec x 50 Clients, 2 Evts/Sec x 100 Clients) and for a short time or a peak, i.e. 300 Evts/Sec (6 Evts/Sec x 50 Clients, 3 Evts/Sec x 100 Clients). An extra test shows that the middle long time is about 30 minutes and the short time is about 5 minutes.

7.2.4. Test Series 4: Server Simple Rules & Client Actions (Task Actions) / Client Simple Rules & Client Actions

The previous test shows how time-consuming the processing of Events can be. This test demonstrates what happens when changing the location of the major time-consuming functionality. Specifically, half of the time-consuming processing (i.e., the execution of Actions on the clients by Simple Tasks) is relocated to the client side (i.e. execution of actions local on the client).

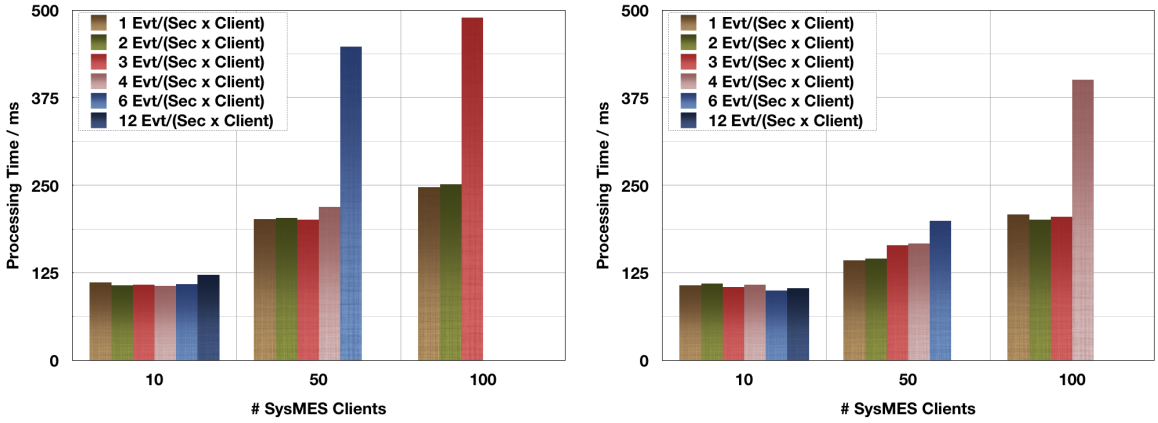
Configuration: The SysMES clients are configured with 12 Monitors, which send 12 different kinds of Events. The required Rules for processing these Events are located in two different locations. There are 24 Simple Rules on the server side and 24 Simple Rules local on the client side. The trigger condition of these Rules are disjunct so that either one Server Simple Rule or one Client Simple Rule fires. The Server Simple Rules are configured for the execution of Task Actions and the Client Simple Rules have a Binary Action, which is executed locally.
In order to compare the test results with those of the previous test series 7.2.3, the test parameters have been chosen as follows:

- # SysMES Clients: 10, 50, 100.
- Evt/(Sec x Client): 1, 2, 3, 4, 6, 12 (i.e. 12 Monitors / 12 sec, 6 sec, 4 sec, 3 sec, 2 sec, 1 sec).

Test Procedure: Similar to all previous test series this one starts with the configuration of the SysMES clients and servers. The clients are configured with 12 Monitors for the generation of 12 different kinds of Events and 24 different Simple Rules for processing half of these Events locally on the client and the other half on the server side. Half of the Monitors (i.e. those which generate the relevant Events for the Client Simple Rules) are configured with the attributes `"CheckClientRule=true"`, `"CheckServerRule=false"` for the generation of Events which match the Client Simple Rules and the other half with the attributes `"CheckClientRule=false"` and `"CheckServerRule=true"` for the Events which match the Server Simple Rule. The SysMES servers are configured with 24 Server Simple Rules.
After the configuration procedure, the Monitors generate Events. Each Event has to be checked against either the 24 Client Simple Rules or the 24 Server Simple Rules. Each Event matches one time and causes one Rule firing and consequently the execution of one Action. Besides the Rule checking and matching all Events are sent to the server in order to be stored in the database.

The specific test procedure for a single Event which triggers the Client Simple Rule is the same as the one described in the test series 7.2.2, and the one for the Events which trigger the Server Simple Rules is the same as the one described in the test series 7.2.3.

Evaluation: The first time measuring point has been set to the Event generation time. In this test there are two different second measuring points depending on where the Action attached to the Simple Rule is executed. For the processing of Events by Client Simple Rules, the time measurement follows immediately after storing the Event in the database. For the Events which are processed by Server Simple Rules with Task Action the time measurement follows after storing the TaskReply Event (i.e. similar to the previous test series 7.2.3).



(a) One Server

(b) Two Clustered Servers

Figure 7.8.: Test4: Processing time of the Events. One half of the Events are processed by Server Simple Rules and react on the clients by Task Actions the other half of the Events are processed by Client Simple Rules with local client Actions

The figures 7.8(a) and 7.8(b) contain the average processing time for Events with a single SysMES server and two clustered servers.
As usual, the several bars represent the configuration of the test parameter "Evt/(Sec x Client)", the X axis is related to the changing test parameter "# SysMES Clients" and the Y axis represents the measured average processing time for the Events. In the case of this test series, the figures visualize the results of all tests with a result less than 500 ms. All other tests were performed too, but are not displayed for better readability of the figures.
With the gained experience managing the HLT by the usage of SysMES, it is possible to define a normal range for processing an Event with the given configuration between 150 ms and 300 ms. As expected, the comparison of the results in both figures demonstrates the same behavior of the SysMES framework as in the test series before. Figure 7.8(a) shows that the maximum number of Events which can be processed by SysMES with the selected configuration is 200 Evts/Sec (i.e. 219 ms for 4 Evts/Sec x 50 Clients and 251 ms for 2 Evts/Sec x 100 Clients).
Independent of the combination of the test parameters, the processing time increases if the total number of Events exceeds 300. For example, the processing time was measured as being 448 ms for 6 Evts/(Sec x Client) x 50 Clients and 489 ms for 3 Evts/(Sec x Client) x 100 Clients. Comparable results for the tests with the configuration 6 Evts/(Sec x Client) x 50 Clients (448 ms) and 3 Evts/(Sec x Client) x 100 Clients(489 ms) demonstrate how much work can be sustained for a middle short time.

In the case of figure 7.8(b), the maximum number of Events within the expected range of 150 ms to 300 ms increases from 200 Evts/Sec for the previous test with one server to 300 Evts/Sec for the test with two servers. An Events processing time of 199 ms was measured for the setup 6 Evts/(Sec x Client) x 50 Clients and 205 ms for 3 Evts/(Sec x Client) x 100 Clients. In this test series there is also a test result which is not in the normal range but within the 500 ms limit. This is the test with 4 Evts/(Sec x Client) x 100 Clients and an average processing time of 401 ms.
For the demonstration that relocation of functionality also contributes to increasing the scalability grade of the SysMES framework, there is another important comparison to be discussed. It concerns the results from this test series and the one of the previous series 7.2.3. Both test series were performed with the same functionality except that half of the Server Simple Rules with a Task Action were relocated to the client side.
In the previous test series, a maximum Event number of 120 Evts/Sec was found. In this test series, a maximum value of 200 Evts/Sec was observed, which represents an increase of 66% . For two clustered servers, the values are 150 Evts/Sec for the test series 7.2.3 and 300 Evts/Sec for the current test series, which means an increase of 100%.

7.2.5. Test Series 5: Complex Rules

This test series was developed in order to test the behavior of the Complex Rule Management subsystem in terms of scalability.

Configuration: The SysMES clients are configured with 12 Monitors which generate 12 different kinds of Events. There is also one special Monitor which runs only once and generates a special Event, i.e. an activation Event for the Complex Rules.
The SysMES servers are configured with 48 Complex Rules. The Complex Rules have the following syntax conforming to the context free grammar of section 5.4.2.4.2:

$$AND(ST_1, ST_2)RelOperator(ST_1.Severity, ST_2.Severity) \Rightarrow EventAction.$$

By applying another production rule of the grammar, the Complex Rule reads as follows:

$$AND(EventName = inittest, EventName = test) <^{13}(ST_1.Severity, ST_2.Severity) \Rightarrow EventAction$$

The Complex Rule fires if there are two Events which fulfill the Simple Triggers ST_1 and ST_2. The Operation part of the Complex Rule is a comparison of the Severities of the Events which have fulfilled the AND concatenated Simple Triggers .
More exactly, the Complex Rule fires if there is a pair of Events with `"EventName=inittest"` and `"EventName=test"` and the Severity of the first Event is less than this of the second Event.
In addition to the Complex Rules, there is a special Simple Rule, the so-called Routing Rule which is used for forwarding the Events from the LAM layer to the WAM layer, where the Complex Rules are located.
The test series parameters have the values:

- # SysMES Clients: 10, 50.

[13]The relational operator "<" is applied to the operands "$(ST_1.Severity, ST_2.Severity)$" and not to the boolean operator "AND".

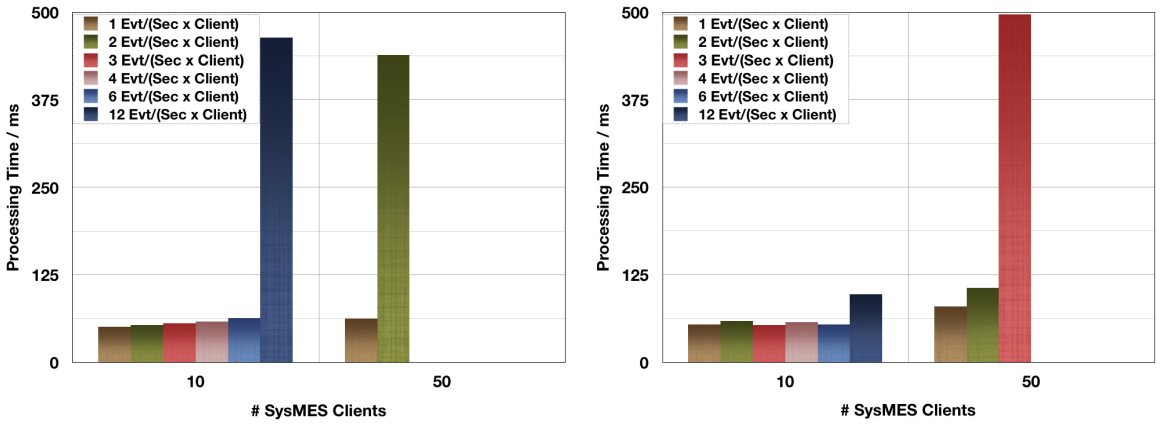
- Evt/(Sec x Client): 1, 2, 3, 4, 6, 12 (i.e. 12 Monitors / 12 sec, 6 sec, 4 sec, 3 sec, 2 sec, 1 sec).

As mentioned in the section 5.4.2.4.2, these Complex Rules are stateful and partial matches are stored in an evaluation network. When ever a Complex Rule fires, there are changes in the evaluation network because the partial matches are consumed according to the value of the attribute ConsumptionMode. In order to avoid changes in the evaluation network which would alter the test conditions, the Monitors have been designed for sending Events which do not fulfill the Complex Rules. All those Events will be checked against all Complex Rules and after this they will be discarded.
Another important part of the Complex Rule configuration concerns the time related behavior of the Rules, which is also defined by setting up attributes such as BaseExpiry. This attribute defines a time interval within which all required Events must have passed the Complex Rule check. In the case of the Complex Rules for this test series, the value of this attribute has been chosen as greater than the test runtime.

Test Procedure: This test starts with the configuration of the servers with the 48 Complex Rules and one Routing Rule. The next step consists of the deployment of a special Monitor, which sends an initialization Event once.
This Event is used for the activation of the Complex Rules. Afterwards, the deployment of the remaining Monitors follows as usual. The Monitors generate 12 different kinds of Events which are first processed by the Routing Rule (Simple Rule on the LAM side) and afterwards stored in the database. The Events are forwarded to the Complex Rule Management subsystem as a consequence of the executed Routing Action. All Events are passed through all 48 Complex Rules before its evaluation stops.
Processing one single Event with the selected functionality is composed of the following steps:

- Execution of a Monitor.
- Event generation with the first time measurement point.
- Event sending to the server.
- Events receiving on the server.
- Simple Rule matching.
- Action execution and routing the Event to the Complex Rule subsystem.
- Complex Rule matching.
- Event storing.

Evaluation: The first time measuring point was set on the Event generation on the SysMES client. The second measurement point is taken after storing the Event in the database, that means at the end of the Event processing chain. Figures 7.9(a) and 7.9(b) show the results for the determination of the average processing time for Events using Complex Rules. The X axis is related to the configuration of the test parameters "# SysMES Clients" and the Y axis corresponds to the measured average processing time for the test and the different bars represent the configuration of the test parameter "Evt/(Sec x Client)". Both figures show only passed single tests with an average processing time of less than 500 ms.



(a) One Server

(b) Two Clustered Servers

Figure 7.9.: Test5: Processing time of the Events which trigger Complex Rules

Figure 7.9(a) contains the results for one single SysMES server. As can be seen in the figure, the maximum number of Events which can be tested parallelly against the Complex Rule subsystem are about 60 (6 Evts/(Sec x Client) x 10 Clients). All single tests under this limit were performed with very similar results between 51 ms and 63 ms. The processing time increases immediately for tests with a higher load than this limit. For example the test for 120 Events/Sec (12 Evts/(Sec x Client) x 10 Clients) has a processing time of 464 ms and the one with 100 Events/Sec (2 Evts/(Sec x Client) x 50 Clients) has an processing time of 439 ms.
Figure 7.9(b) visualizes the results of the same tests, assuming that there are two clustered SysMES servers which share the load to be processed. As shown in the figure, it is possible to process about 100 Evts/Sec - 120 Evts/Sec (2 Evts/(Sec x Client) x 50 Clients and 12 Evts/(Sec x Client) x 10 Clients) with an acceptable processing time of about 80 ms - 97 ms. The next higher test with 150 Evts/Sec (3 Evts/(Sec x Client) x 50 Clients) has an increased processing time of 497 ms.
More information and tests about the scalability of the Complex Rule Management subsystem can be found in [23].

7.3. Fault Tolerance Test

Fault tolerance describes the characteristic of a system to manage system crashes or failures in any of its components. Concerning the SysMES framework the following tests were developed for testing the fault tolerance of the Event Management subsystem, the Task Management subsystem and the Rule Management subsystem.
The tests are done using the SysMES installation of the HLT Cluster (see 7.1.2). As a reminder, this environment is composed of two SysMES servers (mon0 and mon1) and a Oracle DB Cluster (db0 and db1). 100 cluster nodes are used for running the SysMES clients. The distribution of the clients to both SysMES servers is realized by the usage of Access Points, one on each server. The connection method between clients and Access Points is described in 5.4.1.
Each test is designed to simulate a crash of one of the SysMES servers. This crash is realized by terminating the JBoss AS by a shell script. This script finds out the process identifier of the JBoss AS and sends a kill signal (i.e. `"kill -9 $PID"`). The SysMES server is immediately interrupted and all connected clients have to establish a new connection.

Each test has a predefined amount of work, which is related to the tested functionality (e.g. generating and storing 18000 Events). While this work is being performed one server goes down and the other server has to take over the work for all nodes involved in the test. At the end of the test, the actual performed work is accounted for and the processing time for several system management resources is recorded in order to detect delays in their processing caused by the server crash.
All tests are performed using the same test framework already used for the scalability test. It is in charge of installing starting and stopping of the SysMES clients, deploying the desired management resources using Tasks and for evaluating the test results.

7.3.1. Test Series 1: Events - Fault Tolerance

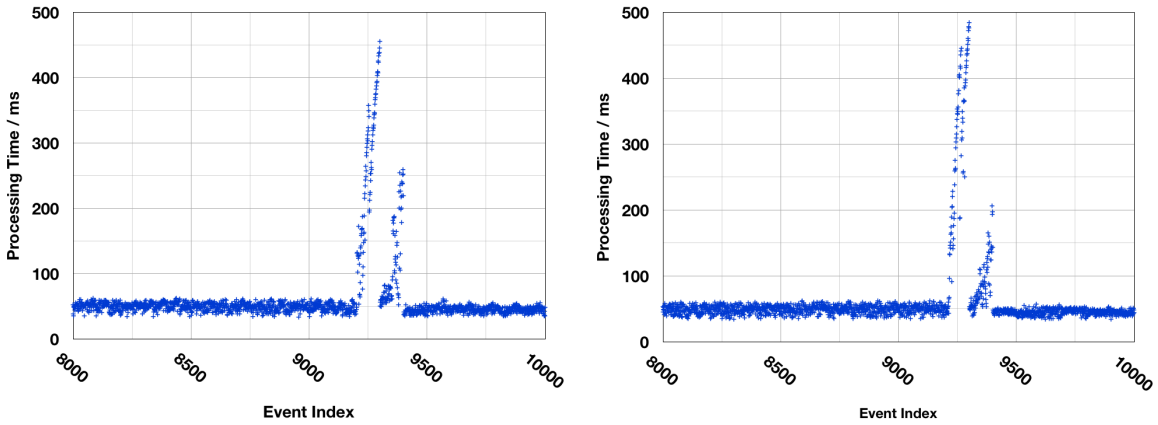
This test is designed for testing the fault tolerance of the Event Management subsystem.

Configuration: Both SysMES servers are started as a clustered pair. Each client runs 10 Monitors for generating 10 different types of Events. These Monitors are configured with `"Period=1 sec"` and `"Repeat=180"` and an attached Event Class is configured with an `"always operator (a)"` (i.e. each Monitor sample generates an Event) and a `"Severity=4"`. This configuration causes each client to run 10 Monitors per second, 180 times, generating a total of 1800 Information Events during the test. As already mentioned, 100 SysMES clients are used and therefore the expected total number of Events is 18000.

Test Procedure: After the start of both clustered SysMES servers, the start procedure of the clients follows. Some clients are connected to the mon0 server and the rest to the mon1 node compliant with the connection management algorithm described in the Access Point section 5.4.1.
The test starts with the deployment of the 10 Monitors to the respective clients. Afterwards, the clients send Events to the servers. One server is stopped after 90 seconds, which forces connected clients to initiate a new connection to another SysMES server. These clients send cached Events generated during the time without connectivity, as well as all other expected Events. The consumed processing time of each Event is measured in order to calculate delays as result of the server crash.
Processing one single Event is composed of the following steps:

- Execution of a Monitor.
- Event generation with the first time measurement point.
- Event sending to the server.
- Event receiving and storing on the server.

Evaluation: The processing time of an Event is given by two time measurements. The first one is set by the Event generation on the client side and the second one is set on the server side after Event storing.
The following bar charts visualize the results of two tests concerning changes in the Event processing time when the mon0 server (see figure 7.10(a)) and when the mon1 (see figure 7.10(b)) are stopped by a crash.



(a) Mon0 Down

(b) Mon1 Down

Figure 7.10.: Events Fault Tolerance Test

The X axis represents an index corresponding to the order of the attribute ArrivalTime [14] of the Events. The server crash occurs after 90 seconds, i.e. approximately shortly after 9000 Events have arrived at the servers. The range of the X axis has been chosen as [8000 - 10000] in order to focus on the detail of the figure when the server crash occurs. The Y axis represents the measured Event processing time in milliseconds.

Figure 7.10(a) describes the behavior of SysMES during a server crash. The processing time of Events is constant between 40 and 60 ms, until the mon0 server crashes. The effect of the crash can be seen at an Event index of about 9200 (i.e. two seconds after server crashed) with increased processing time of Events. Within these two seconds, several Events are processed from the clients which were not affected by the crash (i.e. clients which were connected to the mon1 server).

The following two peaks represent increased Event processing time for all Events generated on the clients affected by the crash. These processing times are higher because for the time interval without server connectivity the SysMES clients generate Events and cache these until the new connection has been established. In the case that Events were processing on the crashed server, these Events are sent again to the new server because of missing acknowledge packages (see section 6.5).

Another observation is a slight increase of the processing time for the Events between the both peaks. These Events were sent by the clients which were not disconnected and their processing time is higher due to the additional load generated by the management of connections for the new clients and the processing of their cached Events.

This test was repeated with a crash of the other SysMES server (i.e. the mon1) and similar results can be found in 7.10(a).

Both tests show that the clients lose the connection to the crashed server, find another server as described in section 5.4.1 and are able to continue working by the usage of the connection to the new server. Events are not lost because of the transactional-based Event Management strategy, which allows the recovery of failed transactions and performs these again. All desired Events (i.e. 18000) were treated and stored in the database.

[14] As a reminder the ArrivalTime attribute describes a timestamp created on the server side after Event processing and immediately before Event storing on the database.

7.3.2. Test Series 2: Tasks - Fault Tolerance

This test is designed for testing the fault tolerance of the Task Management subsystem.

Configuration: Both SysMES servers are started as a clustered pair. 100 clients are started and connected to the servers. 360 Simple Tasks are deployed as fast as possible to the clients (i.e. 360 Task/Client and a total of 36.000 Tasks). These Tasks are configured with `"Acknowledge=1"` which induces the generation of two Ack Events for reporting Task reception and execution.
The execution of each Simple Task causes an entry in a file (i.e. in TASK_FT_Test.log) on the client for logging the successful execution of each Task. This entry is composed of the TaskID of the Simple Task and a timestamp. Furthermore, the client generates a TaskReply Event per Simple Task, which also contains the TaskID of the Simple Task. The TaskReply Event, the log entry as well as the Ack Event are used for demonstrating whether the test has been performed successfully.

Test Procedure: Servers and clients are started and interconnected at the beginning of the test. It follows the deployment of the 360 Simple Tasks. The servers receive two Ack Events, one with the information that the Simple Task has been successfully received and a syntax check has been passed (i.e `"Info.Value=received"`) and another one which confirms the Simple Task execution (i.e `"Info.Value=executed"`).
Furthermore, one TaskReply Event per Simple Task is generated. This Event contains the TaskID from the respective Simple Task in order to check that all clients executed all Tasks successfully.
The server crash is triggered after 180 Simple Tasks have been deployed. At the end of the test another Simple Task is deployed for reading out the number of entries of the local log file TASK_FT_Test.log. The number of entries is returned to the server encoded in TaskReply Events.
Processing one Simple Task is composed of the following steps:

- Simple Task deployment (TaskID as first time measurement).
- Simple Task receiving and parsing on the clients.
- First Ack Event sent to the server (`"received"`).
- First Ack Event received and stored on the server.
- Simple Task execution on the clients.
- Second Ack Event sent to the server (`"executed"`).
- Second Ack Event received and stored on the server.
- TaskReply Event sent to the server (`"Info.Value=TaskID"`).
- TaskReply Event received on the server and stored (ArrivalTime as second time measurement).
- Additional Simple Task execution.
- Additional TaskReply Event sent to the server (`"Info.Val=<# of executed Tasks>"`).
- TaskReply Event received and stored on the server.

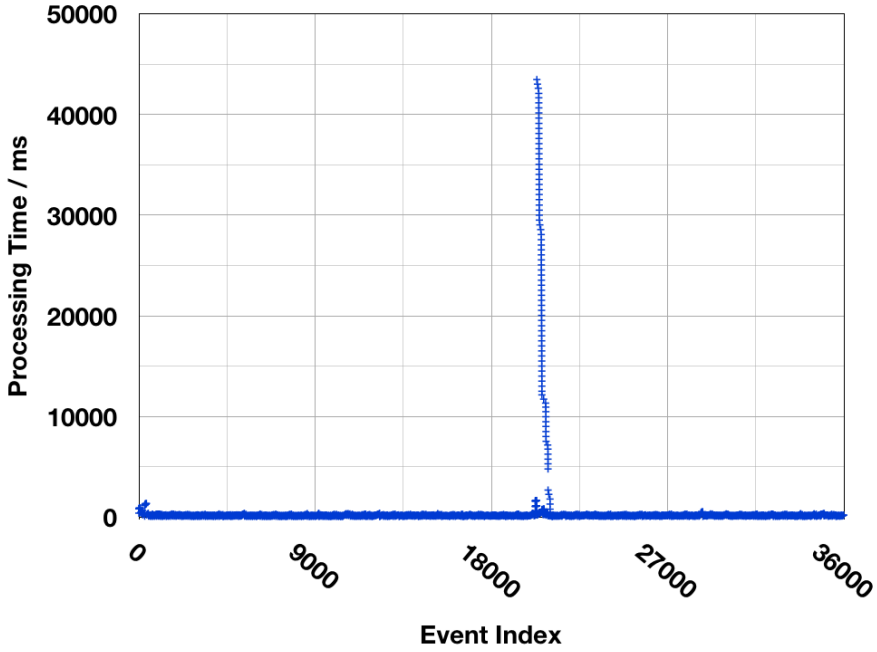


Figure 7.11.: Tasks Fault Tolerance Test

Evaluation: The measured time for processing a Task is described by two time measurement points. The first of these is the Task generation time on the server and the second point is the value of the ArrivalTime attribute of the respective TaskReply Event.

Figure 7.11 describes how the SysMES framework reacts when a server crashes. The X axis is an index, which represents an ascending order of the ArrivalTime attribute of the TaskReply Events. More exactly, the values of the X axis are related to the processing termination times of the TaskReply Events and consequently the processing termination times of the Simple Tasks.

The Y axis represents the processing time for a Simple Task, which is calculated as the difference between the ArrivalTime of the TaskReply Event, and the first part of the TaskID of the Simple Task, which represents the time when it was generated.

As already mentioned, the server crashes (Server mon1 in this case) after the deployment of 180 Tasks. As a consequence of this, the SysMES clients are disconnected and open a connection only when a new Alive Event has to be reported. While the clients are offline, the servers deploy the rest of the Simple Tasks. As per design (see figure 5.13) servers are not able to open a connection to the clients themselves and have to wait until the clients reconnect for sending the remaining Simple Tasks. This offline period is the reason for the peak, which increases the processing time up to about 43 seconds (Processing Time = 42091).

Figure 7.12 is a detailed view of the previous test, which shows the X axis range of values close before and after the server crash i.e. [20000 - 22000].

The first remarkable behavior related to the value of the processing time can be observed at an Event Index between 20250 and 20300. The reason for the increased processing times is the additional work in the remaining server (mon0 in this case) because new client connections have to be established.

The next behavior to be figured out concerns the Event Indexes between 20300 and 21000. It is observable that the processing times are very high and decrease until reaching a normal value, like one

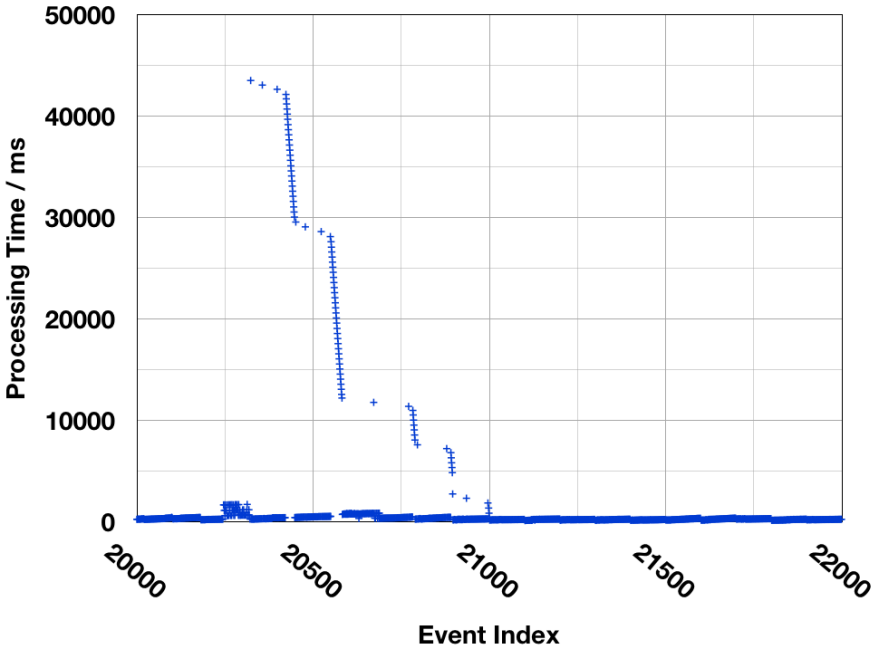


Figure 7.12.: Tasks Fault Tolerance Test - Detailed View

observed before the crash. There are two reasons for that: the first one is that the time where clients were offline is also taken into account for the processing time calculation. Tasks for the same client are deployed sequentially and therefore the first time measurement (the TaskID) is also sequential, i.e. the first deployed Task after the crash has a longer processing time than next deployed one and the processing time decreases with decreasing number of cached Tasks. The second reason is that by recovering the connectivity, the involved server experiences an increased load to be dealt with because there is the load caused by the clients which stayed connected and the additional load caused by the deployment of the cached Tasks from the new connected clients.
Important is the message that a server crash does not has a considerable impact, except a momentary load peak. All 36000 Tasks were processed successfully and all Ack Events and TaskReply Events were stored in the database.

7.3.3. Test Series 3: Rules - Fault Tolerance

The next fault tolerance test is dedicated to the Complex Rule Management subsystem as an important part of the Rule Management system because of its characteristics of stateful Rule checking.
Failures or crashes in the servers where Complex Rules are evaluated can cause information loss about the previous evaluated Events and partial matches, i.e. the state of the Evaluation Network gets lost.
As already mentioned in section 5.4.2.4, the Complex Rule Management subsystem relies on a master-slave strategy for avoiding information loss and for dealing with server crashes.
Tests for the Simple Rule Management subsystem are skipped because this functionality is closely tied to the fault tolerance strategy for Events and Tasks. As a reminder, the Simple Rules are stateless and the contents of the Rule Set are equal for all server members of the LAM layer. Therefore the fault tolerance of Simple Rules depends on the fault tolerance of Events for error detection and Tasks for error solution.

Configuration: The Complex Rule Management is configured as master-slave with the server mon0 as master and mon1 as slave. In order to test the fault tolerance of the Complex Rule Management subsystem, a Complex Rule has been developed and deployed to the master 10 times (i.e there are 10 Complex Rules with the same Trigger and Action configuration). The master distributes the Complex Rules to the slave so that both have the same initial configuration.
The Complex Rules are equals to those used in the scalability tests of section 7.2.5 and are as follows:

$$AND(ST_1, ST_2)RelOperator(ST_1.Severity, ST_2.Severity) \Rightarrow EventAction.$$

more exactly:

$$AND(EventName = inittest, EventName = test) < (ST_1.Severity, ST_2.Severity) \Rightarrow EventAction$$

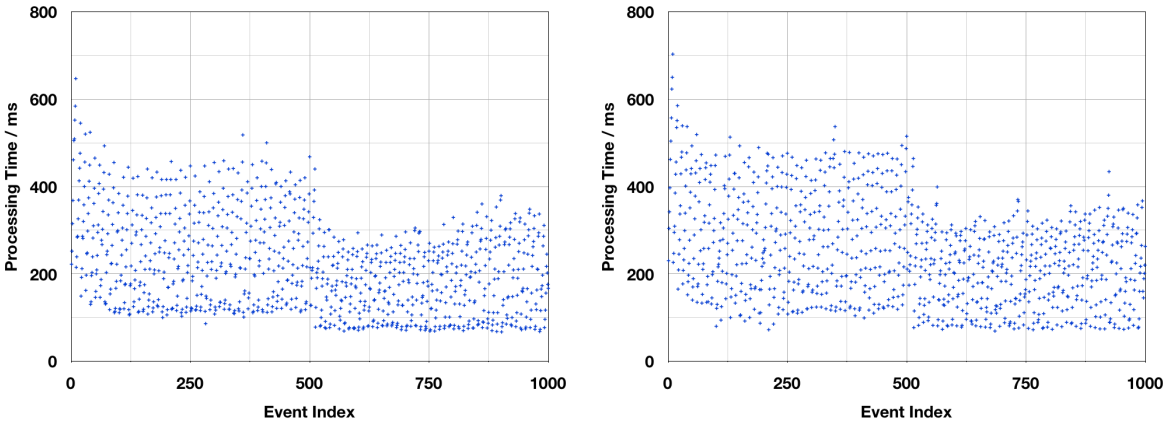
This Complex Rule is configured to fire at the occurrence of two different Events, the first one with `"EventName=inittest"` and the second one with `"EventName=test"` and the Severity of the first Event is less than that of the second Event.
An Event Action is attached to the Rule. This Action is in charge of generating a new Event with `"EventName=AutoGenEvent"` and `"Severity=4"` whenever it fires. Furthermore, it is configured with a `"BaseExpiry=300000"` (i.e. 30 seconds) and `"ConsumptionMode=2"` (i.e. Unrestricted Detection Mode for using Events for multiple evaluation).
A Routing Rule for Event forwarding from the LAM Layer to the Complex Rule Management subsystem on the WAM Layer has been deployed. This Rule redirect Events with a `"Severity=1"` or `"Severity=2"` to one arbitrarily chosen WAM server.
An activation Event is required for the Complex Rule activation. This is realized by a Monitor, which generates one Event with `"EventName=inittest"` and `"Severity=1"`.
10 SysMES clients are used for generating the required Events. Each of these clients runs a Monitor which generates an Event with `"EventName=test"` and `"Severity=2"`. The Monitor is also configured to run 100 times (`"Repeat=100"`) every two seconds (`"Period=2"`).
Conforming to this configuration 10 clients send 100 Events, which are routed to WAM servers (i.e. a total of 1000 Events). 10 Complex Rules are located on the server side, which are activated by one Event. Each of the 1000 Monitoring Events causes that each of the 10 Complex Rule fires and therefore 10000 new Events are generated and stored in the database. These new Events are generated with `"Severity=4"` to avoid the Routing Rule firing for them and sends them to the WAM servers.

Test Procedure: Servers and clients are started at the beginning of the test. The 10 Complex Rules are deployed to the master (mon0) and the Routing Rule is deployed to both servers (mon0 and mon1). After that both Monitors are deployed to the desired targets.
The SysMES clients start sending Events to the LAM servers which are in charge of redirecting these Events to the WAM servers.
One SysMES client is configured with the required Monitor for the Rule activation. The deployment of the Monitor which generates the Events follows. Each Event occurrence causes each Complex Rule to fire and generate a new Event. This is due to the configuration of the ConsumptionMode in Unrestricted Detection Mode which allows for the activation Event to be re-used.
The server crash is done (i.e. `"kill -9 $PID"`) on the master (mon0) after half of the Events per client are processed, and in a second test on the slave (mon1) likewise.
The test configuration part and processing of Events is composed of the following steps:

- Complex Rule deployment on the master.

- Routing Rule deployment.
- Deploy activation Monitor to one client.
- Activation Event received on the LAM server.
- Activation Event routed to the WAM server.
- Activation Event evaluated by the 10 Complex Rules.
- Storing of these partial matches.
- Activation Event stored.
- Deploy Monitor on all clients.
- Monitoring Event received on the LAM server and stored.
- Monitoring Event routed to the WAM server.
- Monitoring Event evaluated by the 10 Complex Rules.
- Firing of the Complex Rules.
- Generation of a new Event per Complex Rules.
- Monitoring Event stored.
- New Events received and stored on the LAM server.

Evaluation: The behavior of the Complex Rule Management is explained based on the processing time of the Events which cause that the Complex Rules fire. Similar to previous tests there are two time measurement points and the processing time is the difference of these. The first time measurement is done at Event generation time on the SysMES client and corresponds to the FirstOccurrence Event attribute and the second one is done at the end of the Event processing and corresponds to the value of the ArrivalTime Event attribute.
Figure 7.13 visualizes the processing time of the Events before and after a server crash. The X axis is an index which describes in ascending order the Events ArrivalTime and the Y axis represents the processing time in milliseconds.
Figure 7.13(a) shows changes in the processing time when the master goes down and figure 7.13(b) shows the case when the slave fails.
The first noticeable effect is the wide range of values of the Y axis (i.e. the Events processing time). This is related to the one-event-one-rule parallel evaluation strategy of the Complex Rules. Conforming to this strategy during the evaluation of an Event, this Rule is blocked for the evaluation of further Events. This means that in such a test scenario (i.e. any Event causes each Complex Rule to fire) the processing time for an Event depends on their waiting time and therefore there are different values for the processing time.
Another remarkable behavior in both figures is the reduction of the Event processing time after the server crashed (`"Event Index = 500"`). In a master-slave configured Complex Rule Management subsystem each change in the Evaluation Network has to be propagated. The reduction of the processing times is related to the fact that changes do not have to be propagated in a master-only configuration and therefore this particular communication overhead does not exist anymore.



(a) Master Down (b) Slave Down

Figure 7.13.: Complex Rules Fault Tolerance Test

The most significant result of this test is that although a server crashes, the predefined workload (i.e. 1000 + 1 Events cause 10 Complex Rules to fire generating 10000 new Events) is performed without information loss. This is verifyed by checking the number of Monitoring Events and new Events stored in the database after the test.

7.4. SysMES Client - Resources Utilization Test

One important goal from those described in the Goals Chapter 2 (Item 1e) concerns the usage of system resources of the SysMES clients. System management methods and activities should not influence the performance of the main applications running on the nodes.
The reference system is a FEP node. As a reminder, this kind of node is equipped with two Quad Core AMD Opteron 3 2378, 2.4 GHz processors and 12 GByte of RAM.
The following test has been performed in order to demonstrate the low resource usage of the SysMES client.

Configuration: This test involves one SysMES client for performing Monitoring and Rule matching and one SysMES server for deploying the required system management resources and storing Events.
There are two test parameters:

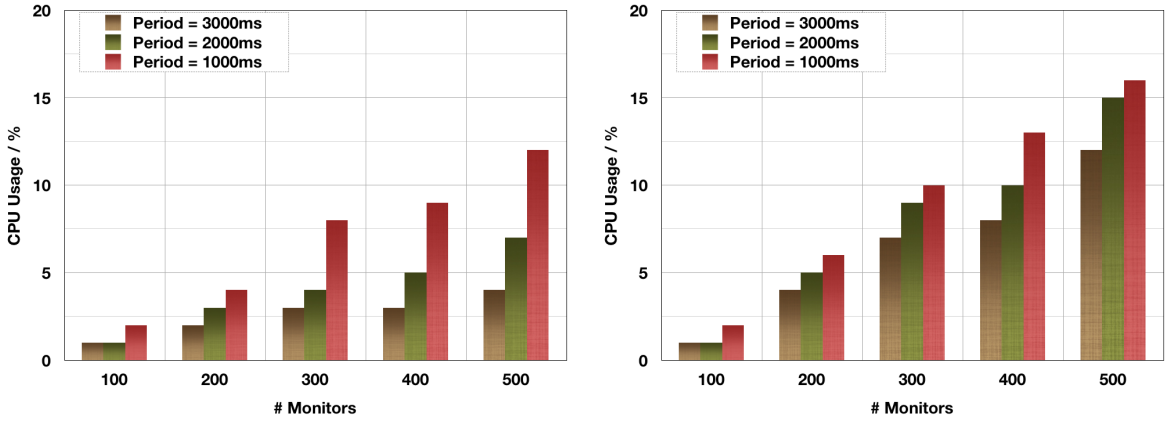
- # Monitors: 100, 200, 300, 400, 500.
- Period (ms): 3000, 2000, 1000.

The clients are configured with a specific number of Monitors and Rules for a test series and each single test is related to the variation of the test parameter "Period".

Test Procedure: Before a test starts, the Simple Task for measuring the used resources is deployed. Every second the corresponding Binary Action collects information about the memory and CPU usage of the SysMES client and stores it locally in a file. Afterwards, the desired number of

Monitors and Rules are deployed and consequently the client starts sending Events to the server. The test has a duration of 300 seconds.
In the case of tests with Simple Rules, these are deployed in the same number as the Monitors so that each Event causes one Simple Rule fire. The Rules have a dummy Action, which only prints a message on the standard output of the node. Dummy Actions have been chosen on purpose in order to measure the resource usage for Monitoring, Event generation and Rule checking and not the resources used by performing an Action.

Evaluation: Figure 7.14 visualizes the CPU usage of a SysMES client during the test. The left figure 7.14(a) contains the measurements of a test when a SysMES client performs Monitoring and sends one Event per measurement to the server. In the right figure 7.14(b) the client additionally performs Rule checking and execution of a dummy Action.

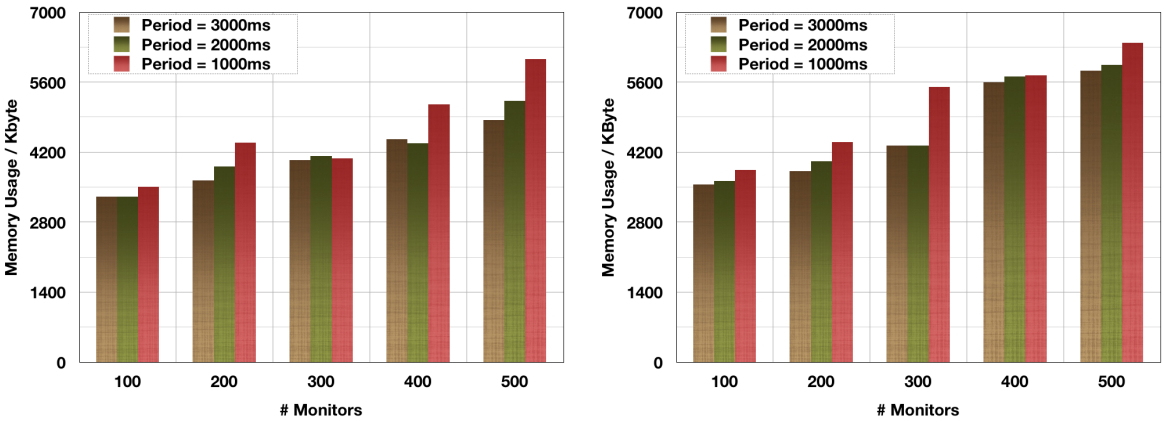


(a) CPU Usage - Events

(b) CPU Usage - Events, Rules and Actions

Figure 7.14.: SysMES Client - CPU Usage

The CPU usage of the node that was measured ranges from 1% of one CPU core (e.g. for 100 Monitors running every two or three seconds) to 12% of one CPU core for running 500 Monitors per second (see figure 7.14(a)). In the case of Simple Rule checking for the generated Events and Action execution, the CPU usage increases to 16% of one CPU core. In the case of the FEP node which has eight cores, the measured results mean that the SysMES client uses no more than 2% of the CPU resources under a very high load.
In the case of the memory tests, the SysMES client uses between 3313 KByte and 6063 KByte running Monitoring and Event handling (see figure 7.15(a)) and about 10% more memory resources for additional Rule checking and Action execution. The maximal used memory measured was 6288 KByte, which represents about 0.052% of the total memory resources of the node.
It is necessary to clarify that both CPU and memory usage depend on the binaries required for Monitoring and the Rule Actions. The measured loads in these tests are related to the required resources of the SysMES client and do not include the required resources for these binaries.



(a) Memory Usage Events

(b) Memory Usage Events and Rules

Figure 7.15.: SysMES Client - Memory Usage

8. Conclusions and Outlook

As described in chapter 2 (Goals), the general challenge of this thesis consists of *"...the design and development of a system management framework for managing a large environment of networked cluster nodes, embedded micro controllers and applications"*. This general goal has been reached and the proof-of-concept "Management of the ALICE HLT cluster" (see section 7.1) shows a management strategy which includes several types of objects to be managed such as nodes, networks, applications, embedded systems, etc.
The usage of SysMES in the HLT cluster allows the coverage of a 24/7 operating time with only one system administrator because the most common problems are detected and solved unattendedly. Furthermore, the additional effort and cost for managing a bigger cluster (e.g. the HLT cluster in the final state with 1000 nodes instead of the current with 200 nodes) is negligible due to the demonstrated capabilities to manage increased load and complexity.

The following parts of the present chapter show that all specific goals have been reached.

Concerning the design goals, the SysMES framework has been designed as a decentralized and distributed framework, which is the first requisite for being scalable and fault tolerant. Other aspects which contribute to improving scalability and fault tolerance are the design decision of clustering all members of the Management Layer and the Operator Layer, the design decision that members of a sub-layer are interchangeable (e.g. several LAM servers), the possibility to add or remove functional replicas and the avoidance of single-point-of-failures.

The test results presented in section 7.2 show that the framework scales by extension of the server resources. Furthermore, it is possible to achieve better scalability by relocation of functionality for example from the Target Layer to the LAM and vice versa. These relocations are possible because the management resources can be changed and reconfigured dynamically and on the fly without downtime.
The test results discussed in 7.3 show that the SysMES framework is able to manage a system crash without service unavailability and data loss. This refers to the goals 1.a, 1.b and 1.c.

The information about both the environment to be managed and the management resources is stored in an object-oriented model. This model is a well organized information source and it is used for managing increased complexity as a result of the heterogeneity of the reference environment, the ALICE HLT Cluster. This refers to the goal 1.a.

The requirement for interoperability with other management systems or information sources has also been included in the design of the SysMES framework. The clients offer an "Inject Interface" used for handling Events from other systems or applications on the client side. Furthermore, the "Passive Monitors" are used for the treatment of measurements from third party monitoring systems and to process these as if they were measured by the SysMES clients. On the server side the, GUI offers an interaction interface and it is also planned to implement a Hypertext Transfer Protocol (HTTP) interface for an external automated deployment of SysMES management resources and for the reception of collected information (e.g. reception of Events). This refers to the goal 1.d.

A conclusion taken from the SysMES client resources utilization test 7.4 is that few client resources are required to carry out the client part of a management strategy. Therefore, the usage of the SysMES clients on computer nodes (i.e. in a cluster) as well as for embedded systems (such as the CHARM cards) does not impair their intended purpose. This refers to the goal 1.e.

SysMES utilizes XML documents for exchanging data between clients and servers. These documents contain the required information about a system state or about an action to be executed. The basic SysMES functionalities are Distributed Monitoring, Event Management, Rule Management and Task Management.

Distributed Monitoring is used for retrieving data (e.g. sensor values) from the SysMES clients. It is possible to get this data locally on the client side or remotely from a server using Monitors. A Monitor is a management object containing the logic for accessing a device and making measurements and the Triggers for deciding if and how this information has to be treated (i.e. stored and analyzed). This refers to the goal 2.a.

On the client side, the Event Management is responsible for checking if the measured values have to be taken into account for further processing and for sending these to the servers in form of XML documents. An advantage of this method is the reduction of data to be sent to the servers because only measurement results which fulfill Triggers will be processed further and therefore contributes to a better scalability of the framework. The clients are also able to store Events persistently, for example in case of disconnection to the SysMES servers. This capability can be also used to relieve overloaded databases. The parsing of the XML document, the generation of Event objects and their storage in the database follows on the server side. This refers to the goal 2.a.

The recognition of an (un)desired state is realized by Rules. These are in charge of recognizing if the information contained in Events represents an (un)desired state and of initiating an automatic problem reporting and solution strategy. Rule Checking can be performed on the clients using the so-called Client Simple Rules or on the servers using the so-called Server Simple Rules and Complex Rules. Both kinds of Simple Rules are able to recognize a state which is coded in one Event. The Complex Rules are used for the correlation of Event occurrences in order to recognize a complex or global state. The execution of Actions (i.e. execution of SysMES Action objects) which contribute to the problem solution follows in case of a Rule match. This refers to the goal 2.b.

The automated execution of actions is related to the SysMES Task Management functionality. Tasks are management objects which contain information about the Action to be executed, the required acknowledgment level and the target(s) where the Action has to be executed.
As already mentioned, Actions are used for problem solution. However, there is another kind of Actions used for the distribution and configuration of the other management objects (e.g. the deployment of Monitors to the clients). This refers to the goal 2.e.
Tasks can also be used for manual interaction with the targets. System administrators and operators use Tasks to perform Actions on the clients and the return values of these Actions are sent to the servers in form of Events. Furthermore, there are several Actions to be used for reporting status via email or SMS. This refers to the goals 2.c and 2.d.

The organization of all management resources is realized in an object-oriented model, which is suitable for managing large heterogeneous environments due to the possibility to build inheritance hierarchies and to reuse objects. This model is based on the Common Information Model (CIM) which is a common standard for describing IT environments. The usage of other common technologies such as XML or EJB is chosen for the achievement of platform and vendor independence. This refers to the goal 2.e.

The SysMES framework can firstly, be extended by developing and deploying new management objects (such as Monitors, Tasks and Rules) and secondly, be reconfigured at runtime without downtime by changing the value of attributes and redeploying the objects to the desired targets. This method is useful for the calibration of the management strategy according to the current system characteristics and usage. This refers to the goal 2.f.

The interaction with the SysMES framework has been realized by the usage of a Web-based GUI, which hides the decentralized and distributed structure of the framework and offers a centralized view to the system administrators and operators. The overview part visualizes the most important Events, displaying involved nodes in colors according to the Event Severity. Furthermore, there are options for the deployment of management resources as well as for the manual execution of Actions. Using this GUI it is also possible to perform some changes on the attributes of the management objects (e.g. changing the Period value of a Monitor). This refers to the goals 1.f and 2.c.

All presented characteristics and functionalities of the SysMES framework enable its usage for managing large distributed and heterogeneous environments. The main task is to support system administrators and operators through the automatic and faster recognition of errors and problems and the automatic solution of these.
Beside the reference installation presented here, the SysMES framework is also in use for managing a test cluster of the computer engineering group at the Kirchhoff Institute for Physics, University of Heidelberg. Further installations in the data center of the University of Frankfurt are planned.

Although the SysMES framework is installed and in use new development, redevelopment, extension and optimization works are planned. The outlook part of this chapter gives an overview of these tasks.

It is planned to extend the Monitoring capability to a multi-row Monitor type, which is also able to retrieve the information from the monitored devices presented in form of a matrix. This feature is useful for reading out values of similar devices by a single measurement.

The implementation of the Task Management subsystem requires optimizations. The scalability tests have shown that concurrent access to resources of the ClientManager Service Bean cause the serialization of several method invocations and therefore the required time for Task deployment increases.

The Rule Management subsystem requires extensions. One important research topic for the future concerns an automatic Rule generation according to the current Event occurrences. This module analyzes incoming Events to recognize repetitive patterns. The module should generate a Rule for the recognition of this pattern and present it to the system administrator, who decides if this Rule should be deployed. Furthermore, the automatic calibration of Trigger objects and Conditions according to the current Monitor values is also desired.

In the case of the Complex Rules, some extended functionality is planned. One extension concerns the possibility to create Triggers which include external information that is not encoded in Event objects. At the current development state, it is only possible to process Events and although there are interfaces for injecting Events or including other monitoring systems, it is often required to define external information sources.
To increase the Rule checking flexibility new Consumption Modes should be developed. Furthermore, it should be possible to define a Consumption Mode for each Trigger instead of one per Rule.
Automated Rule validation is also desired. It is necessary to develop a module which analyzes the Rules in a Rule Set to recognize undesired behaviors such as contradictory Rule semantics (e.g. one Rule shutting down a node and another one powering it on and both Rules are fulfilled by the same

Event correlation), Rule checking circles (e.g. one Rule fires and generates a new Event which causes the same Rule to fire again), etc.

The GUI functionality has to be extended offering the already planned HTTP interface for accessing the stored data such as the management resources. Furthermore, the GUI should offer an interface for developing new management objects and to reconfigure existing ones.

Another task for the future is the development of SysMES-based modules, which utilize the basic SysMES functionality (such as Monitors, Events, Tasks and Rules). At this development stage, modules for inventory and configuration management are planned.

Appendix A.
Abbreviations

ACMS Autonomic Cluster Management System

AFS Andrew File System

ALICE A Large Ion Collider Experiment

API Application Programming Interface

ATLAS A Toroidal LHC Apparatus

BIOS Basic Input Output System

CA Channel Access

CERN Conseil Europeen pour la Recherche Nucleaire

CHARM Computer Health and Remote Management

CIM Common Information Model

CIMOM CIM Object Manager

CMDaemon Cluster Management Daemon

CMOS Complementary Metal-Oxide Semiconductor

CMS Compact Muon Spectrometer

CN Computing Node

CORBA Common Object Request Broker Architecture

DAQ Data Acquisition

DCS Detector Control System

DMTF Distributed Management Task Force

ECS Experiment Control System

EDMS Engineering & Equipment Data Management Service

EJB Enterprise Java Beans

EPICS Experimental Physics and Industrial Control System

FEP Front-End Processor

FTV FactoryTalk View

GSI GSI Helmholtzzentrum fuer Schwerionenforschung GmbH

GUI Graphical User Interface

HADES High Acceptance DiElectron Spectrometer

HLT High Level Trigger

HP Hewlett Packard

HPOM Hewlett Packard Operations Manager

H-RORC High Level Trigger Read Out Receiver Card

HTTP Hypertext Transfer Protocol

IOC Input/Output Controller

IB Infiniband

IP Internet Protocol

IPMI Intelligent Platform Management Interface

ITM IBM Tivoli Monitoring

J2EE Java Enterprise Edition

JBoss AS JBoss Application Server

JMS Java Message Service

KIP Kirchhoff-Institute for Physics

LAM Local Area Management

Lemon LHC Era Monitoring

LHC Large Hadron Collider

LHCb Large Hadron Collider Beauty

MPI Message Passing Interface

MOF Managed Object Format

RAID Redundant Array of Independent Disks

UDP User Datagram Protocol

UML Unified Modeling Language

UPS Uninterruptible Power Supply/Source

OS Operating System

POST Power On Self Test

PV Private Variable

PVSS Prozessvisualisierungs und Steuerungs System

RBEM Rule Based Event Management

RMI Remote Method Invocation

RMS Rack Monitoring System

RRD Round Robin Database

SAN Storage Area Network

SMS Short Message Service

SNMP Simple Network Management Protocol

SNL State Notation Language

SLO Service Level Objective

TANGO TACO New Generation Objects

TCM Tivoli Configuration Manager

TCP Transmission Control Protocol

TDAQ Trigger and Data Acquisition

TEC Tivoli Enterprise Console

TM HLT Task Manager

TMF Tivoli Management Framework

TMR Tivoli Management Region

WAM Wide Area Management

WBEM Web Based Enterprise Management

WLCG Worldwide LHC Computing Grid

XMI XML Metadata Interchange format

XML eXtensible Markup Language

Bibliography

[1] AAD, G. et al.: The ATLAS Experiment at the CERN Large Hadron Collider. In: *JINST* 3 (2008), p. S08003. – DOI 10.1088/1748–0221/3/08/S08003

[2] AAMODT, K. et al.: The ALICE experiment at the CERN LHC. In: *JINST* 3 (2008), p. S08002. – DOI 10.1088/1748–0221/3/08/S08002

[3] ADAIKKALAVAN, R.: *SNOOP Event Specification: Formalization Algorithms, and Implementation Using Interval-Based Semantics*, University of Texas at Arlington, Master Thesis, 2002

[4] ADAIKKALAVAN, R.: *Generalization and Enforcement of Role-Based Access Control Using a Novel Event-Based Approach*, University of Texas at Arlington, Ph.D. Thesis, 2006

[5] ADAIKKALAVAN, R. ; CHAKRAVARTHY, S.: SnoopIB: interval-based event specification and detection for active databases. In: *Data Knowl. Eng.* 59 (2006), October, 139–165. `http://portal.acm.org/citation.cfm?id=1176530.1176536`. – DOI 10.1016/j.datak.2005.07.009. – ISSN 0169–023X

[6] ADEEL-UR-REHMAN, A. et al.: System administration of ATLAS TDAQ computing environment. In: *Journal of Physics: Conference Series* 219 (2010), No. 2, 022048. `http://stacks.iop.org/1742-6596/219/i=2/a=022048`

[7] ADOLPHI, R. et al.: The CMS experiment at the CERN LHC. In: *JINST* 3 (2008), p. S08004. – DOI 10.1088/1748–0221/3/08/S08004

[8] AFS: Andrew File System. `www.openafs.org`

[9] AGARWAL, M. ; BHAT, V. ; LIU, H. ; MATOSSIAN, V. ; PUTTY, V. ; SCHMIDT, C. ; ZHANG, G. ; ZHEN, L. ; PARASHAR, M.: AutoMate: Enabling Autonomic Grid Applications. In: *The Autonomic Computing Workshop, 5th Annual International Active Middleware Services Workshop (AMS2003)*, 2003

[10] ALT, T. et al.: The ALICE High Level Trigger. In: *J. Phys.* G30 (2004), p. S1097–S1100. – DOI 10.1088/0954–3899/30/8/066

[11] ALT, T. ; LINDENSTRUTH, V.: *High Level Trigger ReadOut Receiver Card (H-RORC)* / GSI. 2005. `http://www.gsi.de/informationen/wti/library/scientificreport2005`. – Scientific Report

[12] ALVES, A. et al.: The LHCb Detector at the LHC. In: *JINST* 3 (2008), p. S08005. – DOI 10.1088/1748–0221/3/08/S08005

[13] AVIZIENIS, A. ; LAPRIE, J.-C. ; RANDELL, B.: Fundamental Concepts of Dependability. In: *TECHNICAL REPORT SERIESUNIVERSITY OF NEWCASTLE UPON TYNE COMPUTING SCIENCE* 1145 (2001), No. 010028, 7–12. `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.63.120&rep=rep1&type=pdf`

[14] BABLOK, S.: *Heterogeneous Distributed Calibration Framework for the High Level Trigger in ALICE*, University of Bergen, Ph.D. Thesis, 2008

[15] BALDASSARI, J.D. ; KOPEC, C.L. ; LESHAY, E.S.: *Autonomic Systems*, Worcester Polytechnic Institute, Master Thesis, October 2004

[16] BALDASSARI, J.D. ; KOPEC, C.L. ; LESHAY, E.S. ; TRUSZKOWSKI, W. ; FINKEL, D.: Autonomic Cluster Management System (ACMS): A Demonstration of Autonomic Principles at Work. In: *Engineering of Computer-Based Systems, IEEE International Conference on the* 0 (2005), p. 512–518. – DOI http://doi.ieeecomputersociety.org/10.1109/ECBS.2005.21. ISBN 0–7695–2308–0

[17] BARANA, O. ; BARBATO, P. ; BREDA, M. ; CAPOBIANCO, R. ; LUCHETTA, A. ; MOLON, F. ; MORESSA, M. ; SIMIONATO, P. ; TALIERCIO, C. ; ZAMPIVA, E.: Comparison between commercial and open-source SCADA packages–A case study. In: *Fusion Engineering and Design* 85 (2010), No. 3-4, 491 - 495. `http://www.sciencedirect.com/science/article/B6V3C-4YGHKDT-1/2/43eb8692b947ff0676eebff26bde5a27`. – DOI DOI: 10.1016/j.fusengdes.2010.02.004. – ISSN 0920–3796. – Proceedings of the 7th IAEA Technical Meeting on Control, Data Acquisition, and Remote Participation for Fusion Research

[18] Base16, Base32, Base64 Data Encodings. `http://tools.ietf.org/html/rfc4648`

[19] BAUER, G. et al.: The CMS online cluster: IT for a large data acquisition and control cluster. In: *Journal of Physics: Conference Series* 219 (2010), No. 2, 022002. `http://stacks.iop.org/1742-6596/219/i=2/a=022002`

[20] BEZROUKOV, N.: *Tivoli Alternatives*. `http://www.softpanorama.org/Admin/Tivoli/tivoli_alternatives.shtml`

[21] BMC Event and Impact Management. `http://www.bmc.com/products/offering/Event-and-Impact-Management.html`

[22] BODIK, P. ; GOLDSZMIDT, M. ; FOX, A. ; WOODARD, D. ; ANDERSEN, H.: Fingerprinting the datacenter: automated classification of performance crises. In: *Proceedings of the 5th European conference on Computer systems*. New York, NY, USA : ACM, 2010 (EuroSys '10). – ISBN 978–1–60558–577–2, 111–124

[23] BOETTGER, S.: *Distributed Composite Event Monitoring*, University of Leipzig, Master Thesis, 2006

[24] BOUCHENAK, S. ; BOYER, F. ; HAGIMONT, D. ; KRAKOWIAK, S. ; MOS, A. ; PALMA, N.D. ; QUEMA, V. ; STEFANI, J.: Architecture-Based Autonomous Repair Management: An Application to J2EE Clusters. In: *In 24th IEEE Symposium on Reliable Distributed Systems (SRDS-2005*, 2005, p. 13–24

[25] BOUCHENAK, S. ; BOYER, F. ; HAGIMONT, D. ; SICARD, S. ; TATON, C. ; PALMA, N.D.: *JADE: A Framework for Autonomic Management of Legacy Systems*. 2006. – `http://wiki.jasmine.ow2.org/xwiki/bin/download/Community/Resources/Middleware2006.pdf`

[26] BOUCHENAK, S. ; PALMA, N.D. ; HAGIMONT, D. ; TATON, C.: Autonomic Management of Clustered Applications. In: *IEEE International Conference on Cluster Computing (Cluster 2006)*, 2006

[27] Bright Computing. http://www.brightcomputing.com/

[28] *Bright Cluster Manager 5.0, Administrator Manual Revision 293*. February 2010

[29] Bright Cluster Manager. http://www.brightcomputing.com/Bright-Cluster-Manager.php

[30] *Bright Cluster Manager 5.0, User Manual Revision 281*. February 2010

[31] CA Network and System Management. http://www.ca.com/us/system-management.aspx

[32] CARENA, F. ; CARENA, W. ; CHAPELAND, S. ; DIVIA, R. ; MARIN, J. ; SOOS, K. Schossmaierand C. ; VYVRE, P. V. ; VASCOTTO, A.: *The ALICE Experiment Control System*. 2005. – http://icalepcs2005.web.cern.ch/Icalepcs2005/

[33] CARON, J.: *Overall view of LHC experiments. Vue d'ensemble des experiences du LHC*. May 1998. – AC Collection. Legacy of AC. Pictures from 1992 to 2002.

[34] Common Information Model. http://www.dmtf.org/standards/cim/

[35] CIM Meta Schema. http://www.wbemsolutions.com/tutorials/DMTF/metaschema.html

[36] Common Information Model Schema. http://www.dmtf.org/standards/cim/cim_schema_v216

[37] CIM Tutorial. http://www.wbemsolutions.com/tutorials/DMTF/index.html

[38] ClusterVision. http://www.clustervision.com

[39] Common Object Request Broker Architecture: Core Specification. http://www.omg.org/cgi-bin/doc?formal/04-03-12.pdf

[40] Dawning 4000A Supercomputer. http://www.top500.org/system/7036

[41] Dell Management Console. http://content.dell.com/us/en/enterprise/dcsm-dell-consoles.aspx

[42] Distributed Management Task Force. http://www.dmtf.org

[43] DOBSON, M. ; MALIK., U.A ; ELEJABARRIETA, H.G.: Management of Online Processing Farms in the ATLAS Experiment. In: *Nuclear Science, IEEE Transactions on* 55 (2008), February, No. 1, p. 411 –416. – DOI 10.1109/TNS.2007.913489

[44] Engineering & Equipment Data Management Service. https://edms.cern.ch/cedar/plsql/cedarw.site_home

[45] Enterprise JavaBeans Technology. http://java.sun.com/products/ejb/

[46] EPICS: Experimental Physics and Industrial Control System. http://www.aps.anl.gov/epics/

[47] ESMP. http://www.esmp.com

[48] FORGY, C.: *A Network Match Routine For Production Systems*. – Working Paper

[49] FORGY, C.: *On the efficient implementation of production systems*. Pittsburgh, PA, USA, Carnegie Mellon University, Ph.D. Thesis, 1979. – AAI7919143

[50] FORGY, C.: Rete: a fast algorithm for the many pattern/many object pattern match problem. 1990. http://portal.acm.org/citation.cfm?id=115710.115736. In: RAETH, Peter G. (Hrsg.): *Expert Systems*. Los Alamitos, CA, USA : IEEE Computer Society Press, 1990. – ISBN 0–8186–8904–8, 324–341

[51] The Fractal Project. http://fractal.ow2.org/

[52] Ganglia Monitoring System. http://ganglia.info/

[53] HADES: High Acceptance DiElectron Spectrometer. http://www-hades.gsi.de/gsi/

[54] HARIRI, S. ; KHARGHARIA, B. ; CHEN, H. ; YANG, J. ; ZHANG, Y. ; PARASHAR, M. ; LIU, H.: The Autonomic Computing Paradigm. In: *Cluster Computing* 9 (2006), No. 1, p. 5–17. – DOI http://dx.doi.org/10.1007/s10586–006–4893–0. – ISSN 1386–7857

[55] Hibernate. https://www.hibernate.org

[56] HOKE, E. ; JIMENG, S. ; STRUNK, J. D. ; GANGER, G. R. ; FALOUTSOS, C.: InteMon: continuous mining of sensor data in large-scale self-infrastructures. In: *SIGOPS Oper. Syst. Rev.* 40 (2006), July, 38–44. http://doi.acm.org/10.1145/1151374.1151384. – DOI http://doi.acm.org/10.1145/1151374.1151384. – ISSN 0163–5980

[57] HOKE, E. ; SUN, J. ; FALOUTSOS, C.: InteMon: Intelligent system Monitoring on large clusters. In: *Proceedings of the 32nd international conference on Very large data bases*, VLDB Endowment, 2006 (VLDB '06), 1239–1242

[58] HP Operations Manager Software. http://www.hp.com

[59] HP Operations Manager Software - Topology Based Event Correlation. www.hp.com/go/omi

[60] Interface Definition Language. http://www.omg.org/gettingstarted/omg_idl.htm

[61] IPMI: Intelligent Platform Management Interface. http://www.intel.com/design/servers/ipmi/

[62] JALOTE, P.: *Fault tolerance in distributed systems*. Upper Saddle River, NJ, USA : Prentice-Hall, Inc., 1994. – ISBN 0–13–301367–7

[63] Java Platform Enterprise Edition. http://java.sun.com/javaee/

[64] JBoss Enterprise Application Plattform. – Document: RT#364475 - 04/07

[65] JBoss Application Server. http://www.jboss.org/jbossas/

[66] Java Message Service. http://java.sun.com/products/jms/

[67] Juelich Research on Petaflop Architectures. http://www.fz-juelich.de/portal/forschung/information/supercomputer/juropa

[68] KAGARMANOV, A.: *devSNMP: SNMP as device support in EPICS*. http://www-mks2.desy.de/content/e4/e40/e41/e12212/index_ger.html

[69] KEPHART, J.O.: Research Challenges of Autonomic Computing. In: *ICSE '05: Proceedings of the 27th international conference on Software engineering*. New York, NY, USA : ACM, 2005. – ISBN 1–59593–963–2, p. 15–22

[70] KEPHART, J.O. ; CHESS, D.M.: The Vision of Autonomic Computing. In: *Computer* 36 (2003), January, No. 1, p. 41 – 50. – DOI 10.1109/MC.2003.1160055. – ISSN 0018–9162

[71] Lemon: LHC Era Monitoring. http://lemon.web.cern.ch/lemon/index.shtml

[72] LIU, H. ; BHAT, V. ; PARASHAR, M. ; KLASKY, S.: An Autonomic Service Architecture for Self-Managing Grid Applications. In: *GRID '05: Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*. Washington, DC, USA : IEEE Computer Society, 2005. – ISBN 0–7803–9492–5, p. 132–139

[73] LOEWE-CSC Cluster - University of Frankfurt. http://www.top500.org/system/10591

[74] LYNDON(ED.), E. ; B.PHILIP(ED.): *The CERN Large Hadron Collider: Accelerator and Experiments*. CERN, 2009. – ISBN 978–92–9083–336–9

[75] LYNDON(ED.), E. ; PHILIP(ED.), B.: LHC Machine. In: *JINST* 3 (2008), p. S08001. – DOI 10.1088/1748–0221/3/08/S08001

[76] MATEEN, A. ; RAZA, B. ; SHER, M. ; AWAIS, M.M. ; HUSSAIN, T.: Evolution of autonomic Database Management Systems. In: *Computer and Automation Engineering (ICCAE), 2010 The 2nd International Conference on* Vol. 1, 2010, p. 33 –37

[77] Microsoft System Center. http://www.microsoft.com/systemcenter/en/us/default.aspx

[78] Management Object Format. http://www.dmtf.org/education/mof/

[79] MonALISA: MONitoring Agents using a Large Integrated Services Architecture. http://monalisa.caltech.edu/monalisa.htm

[80] MPI: Message Passing Interface Standard. http://www.mcs.anl.gov/research/projects/mpi/

[81] MySQL Database. http://www.mysql.com/

[82] Nagios Monitoring System. http://www.nagios.org

[83] NELSON, J. ; BAILLIE, O. V. ; DAONES, E. ; HOLBA, A. ; RUBIN, G. ; SZENDREI, L. ; KISS, T. ; MEGGYESI, Z. ; BOZZOLI, W. ; DIVIA, R. ; HARANGOZO, G. ; MCLAREN, R.A. ; RITTER, H.G. ; BIJ, E. V. ; VYVRE, P. V. ; VASCOTTO, A. ; BROCKMANN, R. ; KOLB, B.W. ; PURSCHKE, M.L. ; BELDISHEVSKI, M. ; BELLATO, M.A. ; MARON, G. ; KVAMME, B. ; SKAALI, B. ; WU, B. ; BEKER, H.: *The ALICE Data-Acquisition System*. 1996. – CERN-ALI-95-01. CERN-ALICE-PUB-95-01

[84] NX Server. http://www.nomachine.com/

[85] Oracle Active Data Guard. http://www.oracle.com/us/products/database/options/active-data-guard/index.html

[86] PandoraFMS. http://pandorafms.org/

[87] PANSE, R.: *CHARM-Card: Hardware Based Cluster Control And Management System*, University of Heidelberg, Ph.D. Thesis, 2009

[88] PARASHAR, M. ; LI, Z. ; LIU, H. ; MATOSSIAN, V. ; SCHMIDT, C.: Enabling autonomic grid applications: Requirements, models and infrastructures. In: *in Self-Star Properties in Complex Information Systems, Lecture Notes in Computer Science*, Springer Verlag. Editors, 2005, p. 2005

[89] ParaStation V5. http://www.par-tec.com/products/parastation-v5.html

[90] ParaStation V5 Users Guide, Release 5.0.5. April 2010. http://www.par-tec.com/fileadmin/Daten/products/ParaStation/userguide.pdf

[91] POMALES, W. T.: *Software Fault Tolerance: A Tutorial* / NASA. – Technical Report

[92] Quattor Toolkit. http://quattor.org

[93] RODRIGUES, J. ; MONTEIRO, P. ; SAMPAIO, J. De O. ; SOUZA, J. D. ; ZIMBRAO, G.: Autonomic business processes scalable architecture: position paper. In: *BPM'07: Proceedings of the 2007 international conference on Business process management*. Berlin, Heidelberg : Springer-Verlag, 2008. – ISBN 3–540–78237–0, 978–3–540–78237–7, p. 78–83

[94] SAHNER, R.A. ; TRIVEDI, K.S. ; PULIAFITO, A.: *Performance and reliability analysis of computer systems: an example-based approach using the SHARPE software package*. Norwell, MA, USA : Kluwer Academic Publishers, 1996. – ISBN 0–7923–9650–2

[95] Wikipedia: Definition of Scalability. http://en.wikipedia.org/wiki/Scalability

[96] SIKET, M. ; BABIK, M. ; LOPIENSKI, S. ; MANANA, F.D. B.: CluMan - Cluster management toolsuit. In: *Journal of Physics: Conference Series* 219 (2010), No. 5, 052025. http://stacks.iop.org/1742-6596/219/i=5/a=052025

[97] SNMP: Simple Network Management Protocol. https://datatracker.ietf.org/wg/snmpv3/

[98] *The Socrates Project: Self-Optimisation and self-ConfiguRATion in wirelEss networkS*. http://www.fp7-socrates.org/,

[99] IBM Tivoli Software. www.ibm.com/software/tivoli/

[100] Slightly Skeptical View on Tivoli. http://www.softpanorama.org/Admin/Tivoli/index.shtml

[101] TRUSZKOWSKIL, W. ; .HINCHEY, M ; STERRITT, R.: Towards an Autonomic Cluster Management System (ACMS) with Reflex Autonomicity. In: *Parallel and Distributed Systems, 2005. Proceedings. 11th International Conference on* Vol. 2, 2005. – ISSN 1521–9097, p. 478 –482

[102] Unifed Model Language. http://www.uml.org/

[103] Web Based Enterprise Management. http://www.dmtf.org/standards/wbem/

[104] OpenWBEM. http://openwbem.org/

[105] WEINSTOCK, C.B. ; GOODENOUGH, J.B.: *On System Scalability - Performance-Critical Systems* / Software Engineering Institute, Carnegie Mellon University. March 2006. http://www.sei.cmu.edu/pub/documents/06.reports/pdf/06tn012.pdf. – Technical Note

[106] Worldwide LHC Computing Grid. http://lcg.web.cern.ch/LCG/

[107] XML Metadata Interchange. http://www.omg.org/technology/documents/formal/xmi.htm

[108] XMI2MOF, WBEM Tools. http://www.wbemsource.org/wbem-tools/news.tpl?gnid=372

[109] XML: Extensible Markup Language. http://www.w3.org/XML/

[110] Zabbix Monitoring System. http://www.zabbix.com/

[111] ZELNICEK, P.: *Environment Modelling for Rule Based Event Management*, University of Leipzig, Master Thesis, 2005

[112] ZELNICEK, P. ; KEBSCHULL, U. ; LARA, C. ; ALCOCER, M.: *Converting from XML Metadata Interchange to Managed Object Format*. August 2007. – DMTF, Academic Alliance Conference: Systems and Virtualization Management 2007

[113] Zenoss. http://www.zenoss.com/

[114] ZHEN, L. ; PARASHAR, M.: Rudder: a rule-based multi-agent infrastructure for supporting autonomic Grid applications. In: *Autonomic Computing, 2004. Proceedings. International Conference on*, 2004, p. 278 – 279

[115] ZHI-HONG, Z. ; DAN, M. ; JIAN-FENG, Z. ; LEI, W. ; LIN-PING, W. ; WEI, H.: Easy and reliable cluster management: The self-management experience of Fire Phoenix. In: *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, 2006, p. 8 pp.

Erklärung zur selbständigen Verfassung

Ich versichere, dass ich die vorliegende Doktorarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg, im Mai 2010

Camilo Ernesto Lara Martinez

Errata

Page 18, Line 1: replace "led-ions" with "lead-ions".

Page 19, Line 8: replace "led-led-collision" with "lead-lead-collision".

Page 19, Line 11: change sentence order.

Page 19, Line 21: replace "price-performance ratio" with "value for money".

Page 20, Line 23: replace "sinks" with "lowers".

Page 21, Line 11: replace "Those are normally unprivileged for performing" with "They normally do not have the privileges to perform".

Page 25, Line 2: replace "attain" with "achieve".

Page 25, Line 32: replace "be reacted" with "the system react".

Page 26, Line 35: replace "According to own statements" with "According to Bright Computing own statements".

Page 26, Line 39: remove "performing".

Page 27, Line 20: replace "way" with "Shell".

Page 29, Line 11: replace "charge" with "fee".

Page 29, Line 21: add "proficiency".

Page 31, Line 37: change sentence order.

Page 33, Line 18: replace "is set first to the classification of crises" with "is to first classify crises".

Page 33, Line 29: replace "proceeds" with "occurs".

Page 39, Line 9: change sentence order.

Page 39, Line 28: replace "become" with "cause".

Page 41, Line 24: change sentence order.

Page 42, Line 19: replace "entire" with "complete".

Page 49, Line 7: add "account".

Page 50, Line 39: replace "independent" with "regardless".

Page 52, Line 3: change sentence order.

Page 61, Line 8: replace " the functioning of it " with "its functionality".

Page 61, Line 43: replace "independently" with "regardless".

Page 67, Line 5: replace "is strictly monotonic increasing" with "increases strictly monotonically".

Page 68, Line 5: replace "it is needed to handle the Monitor results" with "the Monitor results need to be handled".

Page 70, Line 3: replace "fewest" with "smallest".

Page 95, Line 22: add "CR".

Page 101, Line 16: replace "Leafes" with "Leaves".

Page 115, Line 10: replace "is not necessary its IP address" with "its IP address is not necessary".

Page 128, Line 16: remove "build".

Page 131, Line 16: replace "taken" with "made".

Page 134, Line 18: replace "independently" with "regardless".

Page 137, Line 37: replace "when" with "where".

Page 139, Line 30: replace "finalizes at the latest" with "finalize at last".

Page 143, Line 6: change sentence order.

Page 156, Line 29: change sentence order.

Page 157, Line 32: remove "the subtraction of".

Page 159, Line 5: replace "hangs" with "freezes".

Page 162, Line 18: replace "are related to" with "depict".

Page 166, Line 16: replace "are related to" with "show".

Page 169, Line 3: change sentence order.

Page 173, Line 6: replace "for" with "during".

Page 173, Line 16: remove "which can be found".

Page 180, Line 17: change sentence order.

Page 185, Line 17: change sentence order.