**Construction Project**

# Detailed Design
## Of the
## Science Alert Generation of the ACADA System

| | | |
|---|---|---|
| Prepared by | *A. Bulgarelli*<br>*SAG Coordinator* | 2022-02-12<br>Date |
| Approved by | *ACADA System Engineer* | Date |
| Released by | *ACADA Coordinator* | Date |

# Change Log

| Issue | Revision | Date | Section/Page affected | Reason/ Remarks / Initiation Docu- |
|---|---|---|---|---|
| 1 | a | 2020-10-15 | All Sections | New document |
| 1 | b | 2022-01-14 | All section | Inclusion of comments received by the following reviewers: Igor Oya, Murach Thomas.<br><br>Feedback from discussions during the coordination meetings and during dedicated technical meetings.<br><br>Added more details on data quality and scientificpipelines. |
| 1 | c | 2022-02-12 | All sections | Internal review |
| 1 | d | 2022-08-26 | All sections | CR31, CR54, CR55<br><br>Answer to RIXs 8789 48791 48792 48854 48869 48906 48966 48980 49009 48904 48978 49236 49237 49010 49039 49058 49165<br>+ changes for |

| | | | | |
|---|---|---|---|---|
| 1 | | 2022-01-14 | 7.3.4 | Inclusion of DPPS statement for future releases |

## List of Contributors

| | | |
|---|---|---|
| ACADA SAG Team | INAF (A. Bulgarelli, N. Parmiggiani, L. Baroncelli, G. De Cesare, V. Fioretti, A. Di Piano)<br>LAPP (G. Maurin, S. Caroff, P. Aubert, E. Garcia, T. Vuillaume) | Detailed design document |
| I. Oya | CTA PO | Template and 1st version. |

**1 LIST OF ABBREVIATIONS ..... 7**

**2 EXECUTIVE SUMMARY ..... 8**

**3 REFERENCES ..... 8**

3.1 Applicable Documents (ADs) ..... 8
3.2 Reference Documents (RDs) ..... 8

**4 INTRODUCTION ..... 9**

4.1 Scope of the Document ..... 9

**5 ARCHITECTURE ..... 9**

5.1 Introduction ..... 9
5.2 Sub-system Context ..... 10
5.3 Logical Architecture ..... 11
*5.3.1 Components ..... 12*
5.4 Physical Architecture ..... 14

**6 DESIGN DECISIONS ..... 15**

6.1 General Design Principles ..... 15
6.2 Assumptions and Dependencies ..... 16
6.3 General Constraints ..... 17
6.4 Design Decisions ..... 17
*6.4.1 Design decisions applicable to all sub-components ..... 17*
*6.4.2 Reconstruction, Data Quality and High-Level analysis pipeline ..... 18*
*6.4.3 High-Level Analysis Pipeline ..... 19*
*6.4.4 SAG Pipeline Sub-array Supervisor and SAG Supervisor (SAG-SUP) ..... 19*

**7 STRUCTURE AND BEHAVIOR THE INTERNAL STRUCTURE IS SHOWN IN THE FOLLOWING SECTION. . 19**

7.1 Product tree breakdown ..... 19
7.2 Component: SAG-SUP ..... 20
*7.2.1 General description ..... 20*
*7.2.2 Structure ..... 21*
*Class Diagrams ..... 21*
*7.2.3 ..... 21*
*7.2.4 General behavior ..... 26*
*7.2.5 State machine ..... 26*
*7.2.6 Command Interface ..... 27*
*7.2.7 Activities ..... 27*
*7.2.8 Sequences ..... 28*
7.3 Component: Image Parameter Extractor and Low-Level Reconstruction Pipeline ..... 33
*7.3.1 General description ..... 33*
*7.3.2 Structure ..... 34*
*7.3.3 Class Diagrams ..... 35*
*7.3.4 General behavior ..... 35*
*7.3.5 State machine ..... 36*
*7.3.6 Command Interface ..... 36*
*7.3.7 Activities ..... 36*
*7.3.8 Sequences ..... 37*
7.4 Component: On-Line Data Quality Software ..... 39
*7.4.1 General description ..... 39*
*7.4.2 Structure ..... 39*
*7.4.3 Class Diagram: rta-dq-lib ..... 40*
*7.4.4 Class Diagram: rta-dq-pipe ..... 41*
*7.4.5 General behavior ..... 43*
*7.4.6 State machine ..... 45*

*7.4.7 Command Interface ..... 46*
*7.4.8 Activities ..... 47*
*7.4.9 Sequences ..... 48*
7.5 COMPONENT: HIGH-LEVEL ANALYSIS PIPELINE ..... 51
*7.5.1 General description ..... 51*
*7.5.2 Structure ..... 52*
*7.5.3 Class Diagrams ..... 54*
*7.5.4 General behavior ..... 58*
*7.5.5 State machine ..... 59*
*7.5.6 Command Interface ..... 59*
*7.5.7 Activities ..... 60*
*7.5.8 Sequences ..... 61*

**8 DATA MODEL ..... 63**

8.1 OVERALL INFORMATION FLOW DIAGRAM ..... 63
*8.1.1 DL0 ..... 63*
*8.1.2 DL1 ..... 65*
*8.1.3 DL2 ..... 66*
*8.1.4 DL3 ..... 68*
*8.1.5 Monitoring point ..... 71*
COMPONENT: SAG-SUP ..... 72
8.2 ..... 72
*8.2.1 Configuration ..... 72*
*8.2.2 Run-Time Setup ..... 72*
*8.2.3 Operations Logging ..... 72*
*8.2.4 Alarms Triggered ..... 73*
*8.2.5 Monitoring Points ..... 73*
*8.2.6 Input Data ..... 73*
*8.2.7 Output Data ..... 73*
*8.2.8 Internal Data ..... 74*
COMPONENT: IMAGE PARAMETER EXTRACTOR AND LOW-LEVEL RECONSTRUCTION PIPELINE ..... 74
8.3 ..... 74
*8.3.1 Configuration ..... 74*
*8.3.2 Run-Time Setup ..... 77*
*8.3.3 Operations Logging ..... 77*
*8.3.4 Alarms Triggered ..... 78*
*8.3.5 Monitoring Points ..... 78*
*8.3.6 Input Data ..... 79*
*8.3.7 Output Data ..... 79*
*8.3.8 Internal Data ..... 79*
8.4 COMPONENT: ON LINE DATA QUALITY ..... 79
*8.4.1 Configuration ..... 79*
*8.4.2 Run-Time Setup ..... 81*
*8.4.3 Operations Logging ..... 81*
*8.4.4 Alarms Triggered ..... 82*
*8.4.5 Monitoring Points ..... 82*
*8.4.6 Input Data ..... 83*
*8.4.7 Output Data ..... 83*
*8.4.8 Internal Data ..... 83*
8.5 COMPONENT: HIGH-LEVEL ANALYSIS PIPELINE ..... 83
*8.5.1 Configuration ..... 83*
*8.5.2 Run-Time Setup ..... 83*
*8.5.3 Operations Logging ..... 84*
*8.5.4 Alarms Triggered ..... 84*
*8.5.5 Monitoring Points ..... 85*

*8.5.6 Input Data ..... 85*
*8.5.7 Output Data ..... 85*
*8.5.8 Internal Data ..... 86*

**9 HUMAN MACHINE INTERFACES ..... 86**

**10 PERFORMANCE ..... 86**

10.1 System Performance ..... 86
10.2 Data Rates ..... 86
10.3 Synchonization ..... 86
10.4 Control Loops ..... 87
10.5 Start-up and Shut-down ..... 87
10.6 State Transitions ..... 87
10.7 Command Handling ..... 88

**11 FRAMEWORKS & LIBRARIES ..... 88**

11.1 External dependencies: ..... 88
*11.1.1 SAG Pipeline Sub-Array Supervisor and SAG Supervisor (SAG-SUP) ..... 89*
*11.1.2 Image Parameters Extractor and Low-Level Reconstruction Pipeline ..... 89*
*11.1.3 On-Line Data Quality Software ..... 90*
*11.1.4 High-Level Analysis Pipeline ..... 90*
11.2 Framework/Library: Munge ..... 91
*11.2.1 Justification ..... 91*
*11.2.2 Licensing ..... 91*
*11.2.3 Version Control ..... 91*
*11.2.4 Distribution/Installation ..... 91*
*11.2.5 Obsolescence Handling ..... 91*
11.3 Framework/Library: ACS ..... 91
*11.3.1 Justification ..... 91*
*11.3.2 Licensing ..... 91*
*11.3.3 Version Control ..... 92*
*11.3.4 Distribution/Installation ..... 92*
*11.3.5 Obsolescence Handling ..... 92*
11.4 Framework/Library: ZeroMQ ..... 92
*11.4.1 Justification ..... 92*
*11.4.2 Features ..... 92*
*11.4.3 Licensing ..... 92*
*11.4.4 Version Control ..... 92*
*11.4.5 Distribution/Installation ..... 92*
*11.4.6 Obsolescence Handling ..... 92*
11.5 Framework/Library: Slurm ..... 92
*11.5.1 Justification ..... 92*
*11.5.2 Features ..... 93*
*11.5.3 Licensing ..... 93*
*11.5.4 Version Control ..... 93*
*11.5.5 Distribution/Installation ..... 93*
*11.5.6 Obsolescence Handling ..... 93*

**12 MISCELLANEOUS ASPECTS ..... 94**

12.1 Deployment ..... 94
12.2 Exception and Error Handling ..... 94
12.3 Logging and Tracing ..... 94
12.4 Concurrency and Threading ..... 94
12.5 Debugging and Troubleshooting ..... 94
12.6 Open Points and Issues ..... 94

**13 TRACEABILITY MATRIX ..... 95**

13.1 FROM REQUIREMENTS ..... 95
13.2 FROM USE CASES ..... 97
13.3 FROM INTERFACES OF THE SUB-SYSTEM ..... 98

# 1 List of Abbreviations

| | |
|---|---|
| ACADA | Array Control and Data Acquisition (corresponds to the previous OES and ACTL systems, old naming may still be used in some diagrams in this document) |
| ACS | Alma Common Software |
| ADH | Array Data Handler, corresponds to what previously was called Data Handling System (DHS) |
| CC | Central Control System |
| CDB | Configuration database / Array Configuration System |
| CORBA | Common Object Request Broker Architecture |
| CTA | Cherenkov Telescope Array |
| CTA-N | CTA Northern Array |
| CTA-S | CTA Southern Array |
| CTAO | Cherenkov Telescope Array Observatory |
| DBMS | DataBase Management System |
| DL0 | Data Level 0 |
| DL1 | Data Level 1 |
| DL2 | Data Level 2 |
| DL3 | Data Level 3 |
| DL4 | Data Level 4 |
| DL5 | Data Level 5 |
| DQ | Data Quality |
| FoV | Field of View |
| IMPAR | Image Parameter Extraction |
| IRF | Instrument Response Function |
| HMI | Human Machine Interface |
| HPC | High-Performance Computing |
| RECO | Reconstruction (Low Level Reconstruction) |
| REL1 | ACADA Release 1 |
| RTA | Real-Time Analysis |
| SAG | Science Alert Generation |
| SCI | Scientific High-Level Analysis |
| SW | Software |
| SUP | Supervisor |
| UC | Use Case |

# 2 Executive Summary

The scope of this document is to provide a detailed design description of the SAG Sub-system of the Array Control and Data Acquisition (ACADA) System that will be operating the instruments at both CTA-N and CTA-S installations.

The main audience of this document is the development team that is expected to implement the sub-system. Motivations for the main design decisions taken are provided in connection with requirements and standards.

# 3 References

## 3.1 Applicable Documents (ADs)

AD1: Array Control and Data Acquisition Architecture Design Document, CTA- Doc. No. TRE-COM-303000-0001 Issue 2, rev g, 2021-05-28.
AD2: Requirement Specification for Science Alert Generation Pipeline of the ACADA System, CTA-SPE-COM-303000-0006 Issue 1, Rev. f, 14.01.2022
AD3: Array Control & Data Acquisition Use Cases, Doc. No. CTA-TRE-COM-003000-0001 ACADA Use Cases Issue 3, Rev. d, 2022-01-14.
AD4: Top-level Data Model, Doc. No. CTA-SPE-OSO-000000-0001, Issue 1, Rev. b, 2020-04-30.
AD5: ACADA-SUSS ICD, Doc. No. CTA-ICD-SEI-000000-0020, issue 1, revision a (draft)
AD6: CTA-SPE-COM-303000-0029, 1b, ACADA Interface Specification
AD7: Scheduling Block Data Model , CTA-SPE-COM-000000-0003, Issue 1, Rev c, 2021-10-19

## 3.2 Reference Documents (RDs)

RD1: ADH Design Document, CTA- Doc. No. CTA-TRE-COM-303000-0004 Issue 1, Rev. f, 2022-01-13
RD2: L. Baroncelli, et al, “rta-dq-lib: a software library to perform online data quality analysis of scientific data”, ADASS conference, 2020, https://arxiv.org/abs/2105.08648
RD3: Development / Release Schedule for Array Control and Data Acquisition System – Doc. No. CTA-PLA-ACA-303000-0005, Issue 1, Rev. c, 2022-01-14

# 4 Introduction

This document describes writing SW design for the SAG sub-system in the context of the ACADA system.

Throughout the document, UML is the preferred notation for depicting the component to be designed. The modelling tool used is Sparx Enterprise Architect.

Throughout this document, the term “component” is used to refer to anything beyond the level of the system to be designed. A component usually consists of several sub-components. The main components of the SAG sub-system are:

1) SAG-SUP: Supervisors
2) SAG-RECO: Image Parameter Extractor and Low Level Reconstruction Pipeline
3) SAG-DQ: Data Quality
4) SAG-SCI: High-Level Analysis

## 4.1 Scope of the Document

This document describes the design of the SAG sub-system of the ACADA software system.

The document describes both the software elements and the run-time environment where these software elements run. The goals of this document are to:
specify the design of the whole sub-system down to the level of details where the software developers can work in implementing the software;
justify the specifics and design decisions according to their drivers, such as ACADA Architecture, requirements, use cases and interfaces;.

- provide input for the product acceptance process (verification and validation);
- provide input for product effort estimates for realizing the ACADA components.

In this document REL1 means is "ACADA Release 1", see [RD3].

# 5 Architecture

## 5.1 Introduction

The Science Alert Generation (SAG) system, part of the Array Control and Data Acquisition (ACADA) system, analyses the telescope data online to detect transient gamma-ray events with a foreseen data rate of tens of kHz.

The SAG-SUP is compounded by two main components, and is the interface with the rest of the ACADA system: (i) SAG Pipeline Sub-Array Supervisor, a supervised component that supervises the operation of the SAG pipeline component associated to a sub-array; (ii) the SAG Supervisor, a supervised component with the same lifetime of the nightly operations that manages the generation of scientific monitoring results to HMI, and proves input to the ACADA Reporting Subsystems for the generation of reports to HMI, DPPS, and SUSS.

The SAG-RECO includes the Image Parameter Extractor and the Low Level Reconstruction components. The SAG-RECO implements a pipeline for fast reconstruction to process

individual DL0 events and produces the corresponding DL3 data with a latency of less than 15 s. SAG-RECO shall provide a library or software component to integrate data-cubes (DL0) into images, then clean images and extract image parameters from Cherenkov camera images. The parameterized images (DL1 data) are then delivered to other SAG pipelines. This library must be able to extract the image parameters at a rate of 1000 events/s/CPU/telescope.
The SAG system also performs an online Data Quality analysis (SAG-DQ) to assess the instruments' health during the data acquisition.
Two python libraries called rta-dq-lib and rta-dq-pipe have been developed to satisfy the data quality analysis requirements. The first package implements the core logic of data quality analysis, while the second manages the parallel execution of multiple data quality analysis pipelines.
The scientific pipelines (SAG-SCI) are a sub-system of the SAG based on a framework designed to simplify the development of real-time scientific analysis pipelines. Using this framework, developers and researchers can focus more on the scientific aspects of the pipelines and integrate existing science tools, providing a common pipeline architecture and automatisms. The framework can be easily configured with new or existing science tools implemented in different programming languages. The pipelines can execute all the analysis automatically after the initial setup. The scientific analyses are performed in parallel and can be prioritized. The pipelines distinguish between High-Level Analysis pipelines for Science Monitoring to produce science quick look (DL4 and DL5 data levels, generation of sky maps and light curves) for the Support Astronomer, and Science Alert Generation pipelines to produce science alert candidates.

Slurm Workload Manager manages the workload of all the pipelines allowing to scale them on a cluster of machines.

## 5.2 Sub-system Context

In this section, the context diagram is shown, including all relevant boundary systems and relevant stakeholders.

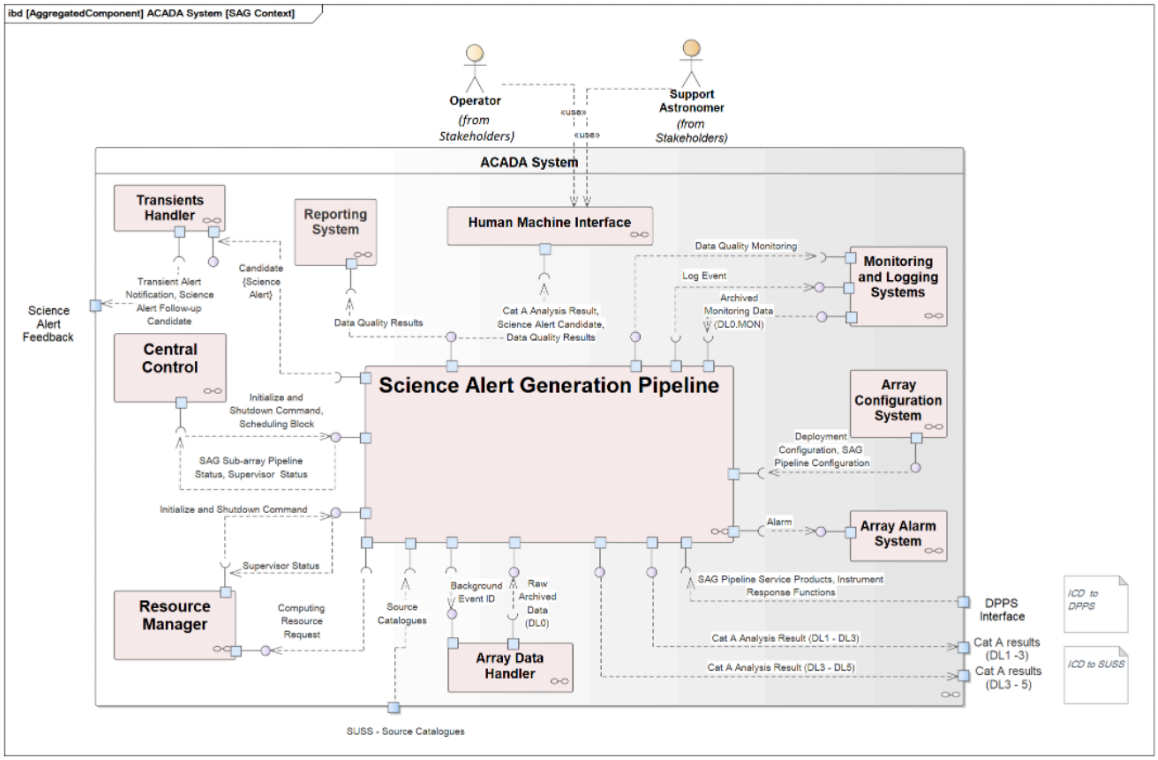


**Figure 1: Context diagram for the ACADA/SAG sub-system**

## 5.3 Logical Architecture

The logical architecture of the SAG sub-system is shown in the component diagram of the next FigureFigure 2.

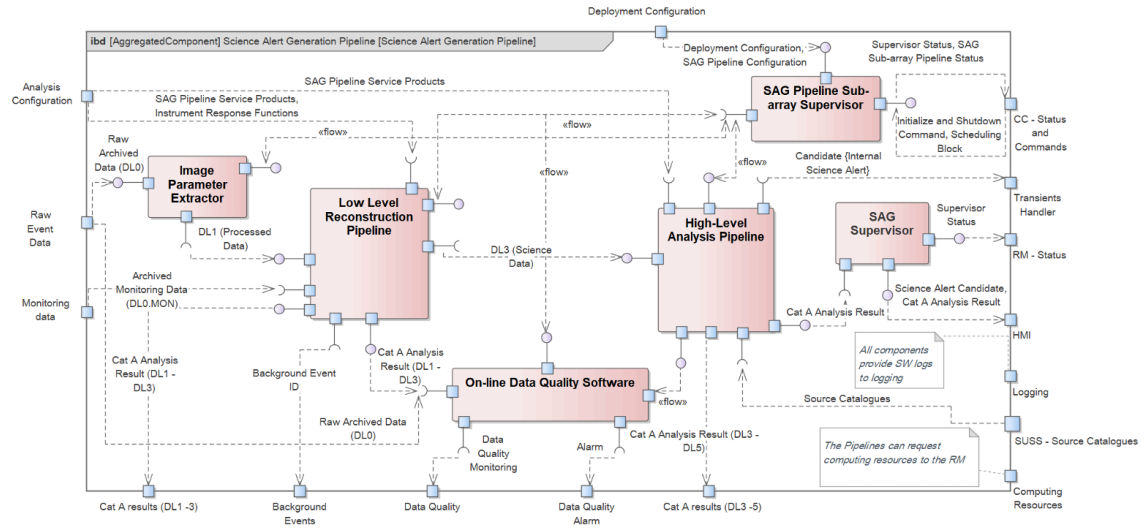


**Figure 2: The components diagram illustrating the internal structure of the ACADA SAG sub-system.**

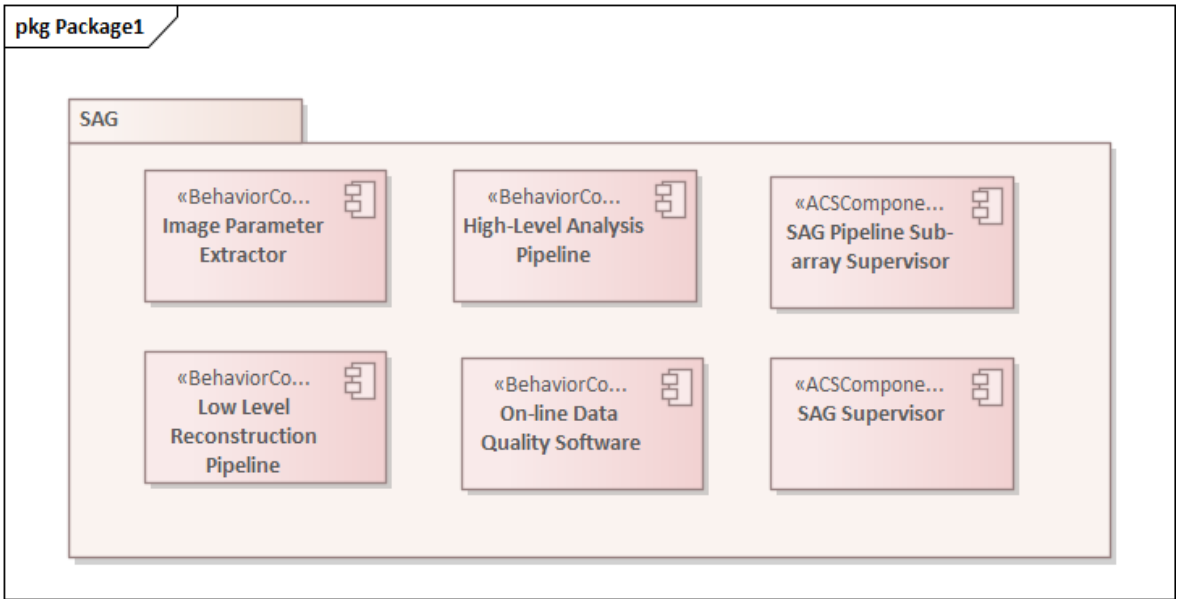


**Figure 3: Package diagram illustrating the internal structure of the SAG sub-system.**

### 5.3.1 Components

In the context of the SAG, the following components are present.

**Table 1: Components of the SAG sub-system**

| Component | Description |
|---|---|
| SAG Pipeline Sub-Array Supervisor (SAG-SUP) | A supervised component that supervises the operation of the SAG pipeline component associated to a sub-array. The SAG Pipeline Sub-Array Supervisor is composed of: SAGSubArrayManager, SAGSubArrayMonitor, SAGreco, SAGdq, SAGsci |
| SAG Supervisor (SAG-SUP) | A supervised component with the same lifetime of the nightly operations that manages the generation of scientific monitoring results to HMI, and provides input to the ACADA Reporting Subsystems for the generation of reports to HMI, DPPS, and SUSS. |
| Image Parameter Extractor | Part of the SAG-RECO, parameterizes Raw Cherenkov Camera images |
| Low-Level Reconstruction Pipeline | Part of the SAG-RECO, A component that receives Cherenkov Camera data from the ADH, selects signal-like events (i.e., it performs a quick-look background rejection/signal discrimination) and produces DL3 data products. Depending on the configuration, it informs ADH if the event should be stored, providing means for further data volume reduction. |

| | |
|---|---|
| On-Line Data Quality Software (SAG-DQ) | A component that performs a real-time data quality analysis to produce data quality indicators.. |
| High-Level Analysis Pipeline (SAG-SCI) | A component that receives the DL3 data and performs the high-level processing to produce DL4-DL5 data products. It includes the high-level scientific analysis pipelines, that will be used to check the observation status from a scientific point of view (sky maps, light curves, and so on) and to generate candidates for internal Science Alerts. |

The main components with a more refined view of the SAG-SUP (the SAG Supervisor and of the SAG Sub-Array Supervisor components developed in ACS) are provided in the following figure and are described in section 7.2.
Two DBMS that store the results of the SAG-DQ and SAG-SCI pipelines are shown. Slurm is the workload manager of the SAG system. The SAG subsystems external to ACS listed in the previous table are shown as grey components.

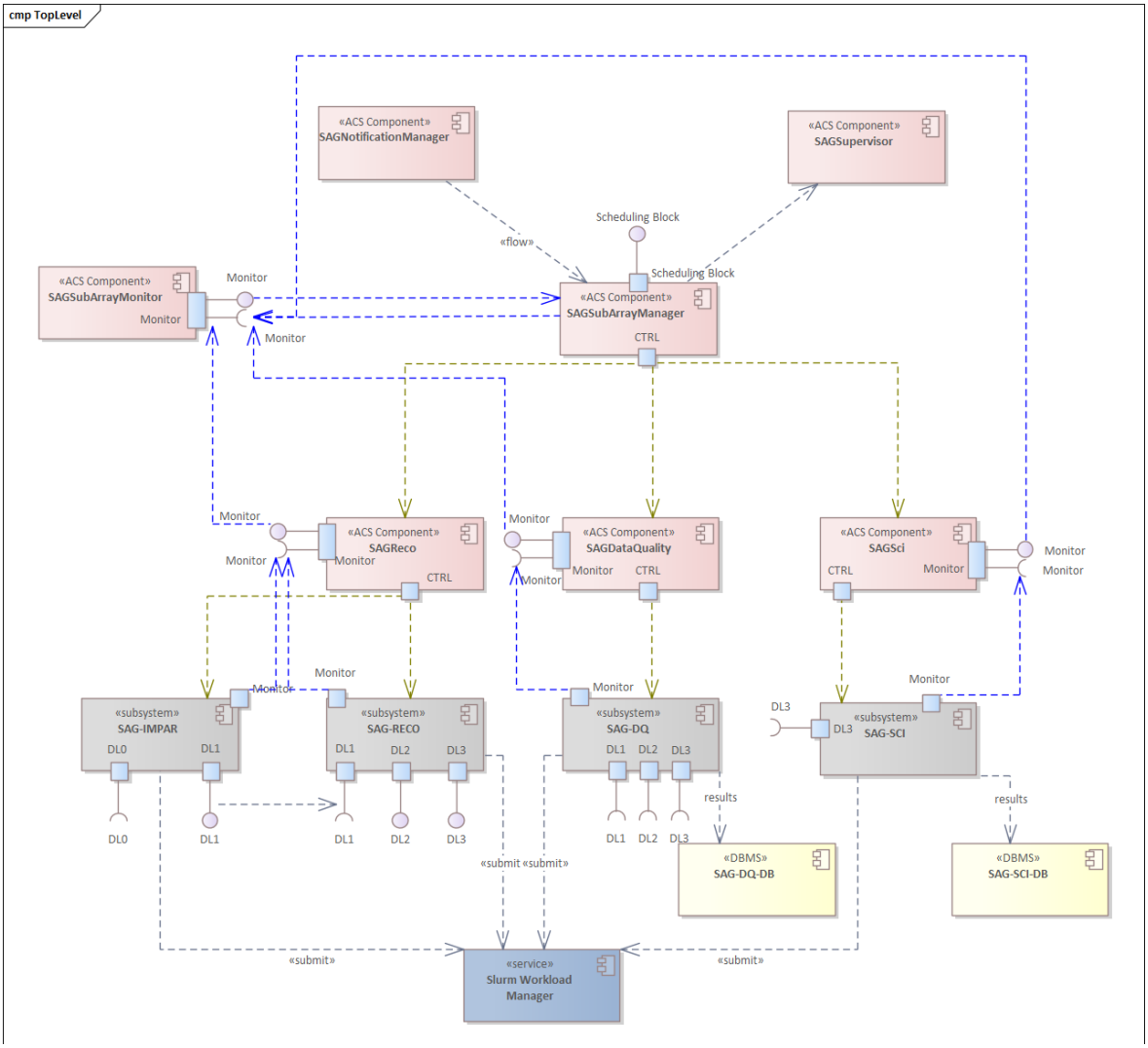


**Figure 4: Component diagram illustrating the internal structure of the SAG developed with ACS framework (red) and the relationship with the SAG subsystems external to ACS (grey). Two DBMS that store the results of the SAG-DQ and SAG-SCI pipelines are shown. Slurm is the workload manager of the SAG system. Green lines are the control path, blue lines are the monitoring path.**

Slurm is the workload manager of the SAG system. Slurm is an open-source, fault-tolerant, and highly scalable cluster management and job scheduling system for large and small Linux clusters. The different pipeline subsystems submit all the tasks to Slurm, Slurm manage the resources and the execution of the analysis. Slurm is capable of managing thousands of jobs and can be easily scaled from one machine to thousands of machines. It can submit jobs in parallel and manage different priorities between jobs. It can suspend low priority jobs to run high priority jobs (useful if the SAG receives a science alert with high priority). Another key feature of Slurm is the capability to log inside a database all the information about every job. This logging system is useful to monitor the pipeline in real-time, viewing statistics about the execution of jobs. Scaling the computing power used by Slurm does not require a change in the pipeline subsystems software and it is not necessary to coordinate the workload of the different pipelines.

## 5.4 Physical Architecture

The deployment diagram shows the structure of the subsystem to be designed in its physical eco-system of its use.
In the following figure, the scenario where the SAG is analysing the data of two sub-arrays is shown. Each sub-array has a sub-array manager that allocates one pipeline for each sub-array and related subsystems. The subsystems that are required to start Slurm jobs need to be configured. In this example, only RECO and DQ subsystem are running.

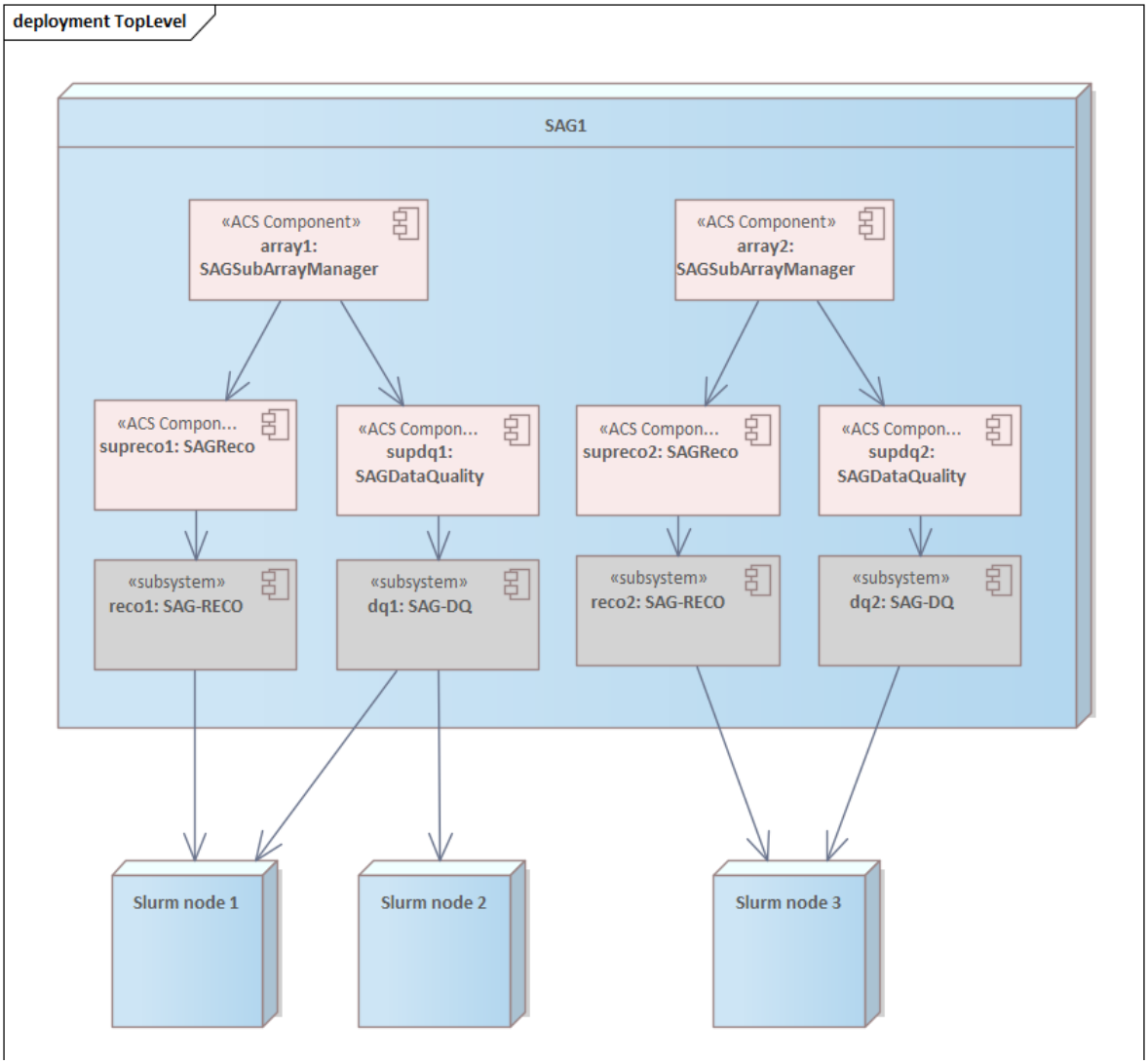


**Figure 5: SAG deployment diagram. In this scenario two sub-array are managed by the SAG. The workload is managed by Slurm in a transparent way for different SAG subsystems or it is possible to establish specific queues.**

# 6 Design Decisions

## 6.1 General Design Principles

The SAG system is implemented using a combination of Python and C/C++ programming languages. Python is used for the SAG-SUP, for the SAG-SCI and for some tasks of the SAG-DQ, while C/C++ is used for the SAG-RECO as they require the best possible performances. The logical interconnection between types of components, along with their type of interface is outlined in the next Figure.

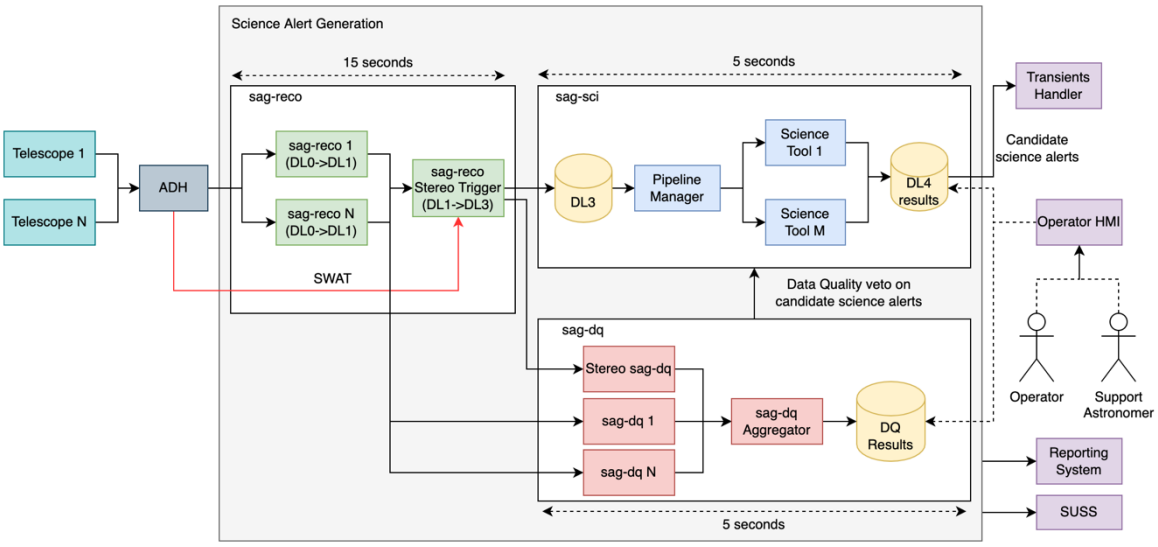


**Figure 6: SAG process view. In this scenario N lines of input data are processed separately by SAG-RECO and a final list of DL3 data is analysed by SAG-SCI. SAG-DQ processes each line of data in parallel.**

All HPC processing components are stateless, except for some tasks of the RTA-DQ that are stateful:

- Stateless components are simple functional components without a local state. They perform specific functions and are submitted to the workload manager.
- Stateful components can contain the history of the processed data between Observation Blocks. They are managed by the workload manager.

For instance, a data quality task can produce a summary of the results of the analysis of multiple Observation Blocks that must be kept by a stateful component.

The SAG Sub-Array Manager and the SAG Supervisor are the supervisors of the other sub-components of the SAG system.
The SAG-SUP is the interface with the rest of the ACADA subsystems: the SAG subsystem can be launched and controlled by other ACADA subsystems. All commands and monitoring are achieved via CORBA remote procedure calls and ACS notification channels. The exchange of Cherenkov, auxiliary and events timestamps data is achieved either by using the zeroMQ middleware or direct TCP/IP sockets with the ADH. This low-level approach was desired to ensure the best possible performances to handle high-throughput and high-frequency streams.

Some results of SAG-RECO and SAG-SCI (DL3, DL4 and DL5) are in FITS file format.

## 6.2 Assumptions and Dependencies

The SAG shall run on a Linux RedHat derivate distribution and the ACS framework. The version of the Linux distribution and ACS framework is defined by the Configuration Control Board for each release.

The SAG interfaces with other sub-systems inside ACADA shall be implemented with the ACS framework.

The SAG interfaces between internal components are implemented with the ACS framework.

## 6.3 General Constraints

The constraints that have a significant impact on the SAG are the following:

- The use of ACS that drives the design and the interfaces with the rest of the ACADA system
- The limited computing power, that requires HPC solutions for workflow and computing management and the use of C++ for the most compute-intensive tasks.

## 6.4 Design Decisions

This section describes sub-system level design decisions.

### 6.4.1 Design decisions applicable to all sub-components

| **ID** | **DD-SAG-1** |
|---|---|
| **Explanation** | |
| The ACS framework is the basic middleware of the ACADA sub-systems. The ACADA sub-systems are implemented as ACS components. These components run in the on-site data center and can interact with each other using this framework. | |
| **Pros & Opportunities** | |
| • ACS is used by all ACADA sub-systems providing standard interfaces<br>• ACS components can be implemented in different languages (Python, C++, Java)<br>• The share of interfaces through the IDL standard is easy<br>• ACS components can run in a distributed computing cluster | |
| **Cons & Risks** | |
| The learning curve of ACS is slow and the framework is complex<br>The already existing prototypes must be redesigned to support this framework | |
| **Assumptions and Quantification** | |
| The ACS framework is well maintained by the community<br>ACS workshops are offered by the community<br>There is a well-defined documentation | |
| **Trade-offs** | |
| • The slow learning curve is justified by the possibility to have the same interfaces for all ACADA sub-systems and the possibility to interface software developed with different programming languages | |

| **ID** | **DD-SAG-2** |
|---|---|
| **Explanation** | |
| Python for script | |
| **Pros & Opportunities** | |
| • Fast learning curve<br>• Reduced development time | |
| **Cons & Risks** | |
| Lower performances than compiled languages (Java, C++)<br>Many errors are found at run time (strongly and dynamically typed). **Strong** typing means that the type of a value doesn't change in unexpected ways. A string containing only digits doesn't magically become a number, as may happen in Perl. Every change of type requires an explicit conversion. **Dynamic** typing means that runtime objects (values) have a type, as opposed to static typing where variables have a type. | |

| **Assumptions and Quantification** |
|---|
| Python is largely used by the Scientific Community<br>Python is well documented and supported<br>Python allows the usage of Anaconda to manage packages |
| **Trade-offs** |
| • The fast development time and learning curve justifies the lower performances |

### 6.4.2 Reconstruction, Data Quality and High-Level analysis pipeline

| **ID** | **DD-SAG-3** |
|---|---|
| **Explanation** | |
| Slurm workload manager. Slurm is a framework for HPC. | |
| **Pros & Opportunities** | |
| • Slurm can manage processes in a cluster of machines<br>• It enables parallel processing<br>• There are failure management strategies<br>• It is possible to define priority, queues and sequences between processes<br>• It is possible to target specific machines and reserve resources<br>• All processes information are stored in a database or in the file system. | |
| **Cons & Risks** | |
| It requires the configuration of the service<br>It requires a fast network connection between machines | |
| **Assumptions and Quantification** | |
| Slurm is largely used<br>Slurm is well documented, free and open-source<br>Slurm will be maintained in the future years, and implements state-of-the-art technologies<br>High-throughput computing (HTC) is the deployment of resources to tackle a large computational burden where the individual computations do not need to interact while running. HTC differs from high-performance computing (HPC), where rapid interaction of intermediate results is required to perform the computations, that is the case of the SAG. | |
| **Trade-offs** | |
| • Slurm has many useful functionalities that justify the initial setup of the service | |

The scripting part of the RECO and DQ components is implemented with Python, but the core of the analysis is implemented with C++.

| **ID** | **DD-SAG-4** |
|---|---|
| **Explanation** | |
| C/C++ are general-purpose and well know languages. C++ is an object-oriented programming language. We decided to implement RECO and some DQ core functionalities using C++. | |
| **Pros & Opportunities** | |
| • C/C++ are programming language used to develop software with high performances.<br>• C++ is Object Oriented.<br>• C/C++ is largely used and known. | |
| **Cons & Risks** | |
| C/C++ requires more effort and time to develop software with respect to other languages<br>C/C++ learning curve is slower than other programming languages like Python. | |
| **Assumptions and Quantification** | |
| C/C++ will be maintained in the next years.<br>C/C++ has a large community. | |
| **Trade-offs** | |

| |
|---|
| • The high performances reached with these programming languages justify the slower learning curve and development process. |

### 6.4.3 High-Level Analysis Pipeline

| ID | DD-SAG-5 |
|---|---|
| **Explanation** | |
| MySQL is used as DBMS to implement the core functionalities of this component. The Science Logic implemented in this component to configure different Science Tools is implemented using MySQL Triggers. MySQL is one of the allowed database technologies by ACADA as indicated in AS-9 of AD1. | |
| **Pros & Opportunities** | |
| • MySQL is an open-source DBMS largely used in several domains.<br>• MySQL triggers are stored program invoked automatically in response to events.<br>• trigger management is done by MySQL.<br>• The data are not transferred to an external application during the invocation of triggers. | |
| **Cons & Risks** | |
| The architecture of this sub-system is hardly coupled with the MySQL framework.<br>It is hard to test and create an automated test for MySQL trigger. | |
| **Assumptions and Quantification** | |
| MySQL is a well-supported framework.<br>MySQL is known by all the developer's community. | |
| **Trade-offs** | |
| • The advantages of having automated triggers directly in the DBMS is greater than the risks of coupling the architecture with a framework. | |

### 6.4.4 SAG Pipeline Sub-array Supervisor and SAG Supervisor (SAG-SUP)

These functional components are highly based on the ACS framework. SAG Pipeline Sub-Array Supervisor supervises all processes of the SAG and it manages the interfaces with external sub-systems. Also, the SAG Supervisor is interfaced with ACADA subsystem. For this reason, we decided to implement these functional components with several ACS components in order to distribute the workload and to have higher fault tolerance.

# 7 Structure and Behavior

The internal structure is shown in the following section.

## 7.1 Product tree breakdown

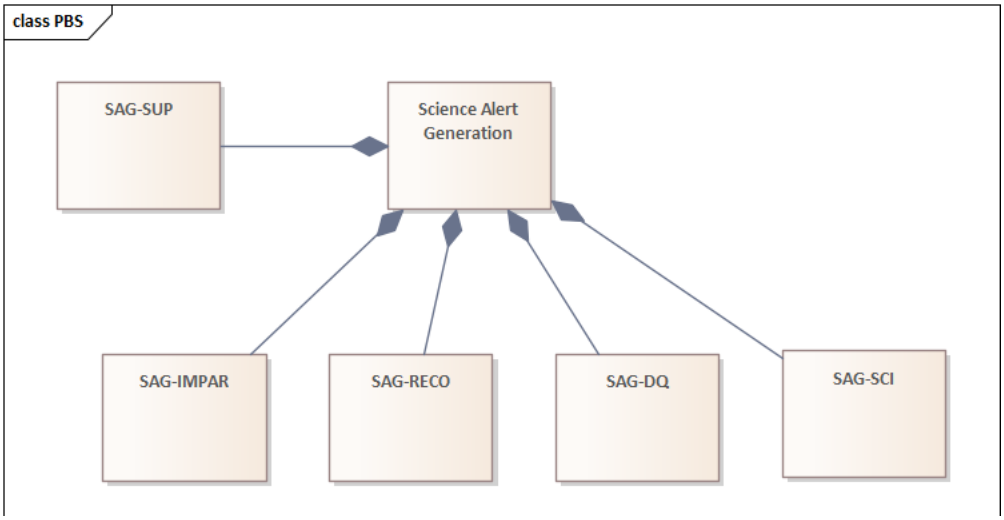


**Figure 7: Class diagram illustrating the product tree of the SAG.**

## 7.2 Component: SAG-SUP

### 7.2.1 General description

SAG-SUP consists of two main logical components: SAG Pipeline Sub-Array Supervisor and SAG Supervisor.
Both logical components are developed using ACS and they provide interfaces to communicate with the other ACADA subsystems. SAG Pipeline Sub-array Supervisor
The SAG Pipeline Sub-Array Supervisor is implemented with a dynamic ACS component whose features are described by an ACS interface named SAGSubArrayManager. It supervises the operations of SAG pipeline components (see Sections 7.3, 7.4 and 7.5) associated to a sub-array and it interfaces with the rest of the ACADA system. The list of managed components is: SAGreco, SAGdq, SAGsci and SAGSubArrayMonitor.

The Central Control allocates one SAGSubArrayManager component for each Scheduling Block, supervises its operation and shutdowns the component when the Scheduling Block is completed.

The SAGSubArrayManager shall allocate SAG pipeline components for a Scheduling Block and it shall supervise their operations: when a sub-array enters in observing state, the SAGSubArrayManager shall communicate this information to the SAG pipeline components and they will start the actual processes that performs the data processing. When the observation of a sub-array is stopped, the SAG pipelines components will be shut down in a controlled manner.
SAGSubArrayManager shall inform the SAG pipeline components about changes in the status or configuration of arrays, Observation Block changes, environmental condition changes, and NSB status changes. In addition, the SAGSubArrayManager shall receive the modified Scheduling Block when a sub-array configuration is modified, i.e. when telescopes are added or removed from current Observation Block. The SAGSubArrayManager is also responsible for meeting the following scenarios:

- when the execution of a Scheduling Block ends, all acquired data must be processed;
- when the execution of a Scheduling Block is cancelled, all acquired data must be processed.

The SAGSubArrayManager will be notified when the following scenario happens:

- ACADA performs an update of the observation position coordinates of the Array Elements by an amount less than the size of the field of view without stopping ongoing observations of the relevant sub-array. This scenario can happen, for example, . due to ToO coordinates update.
- the observation time originally assigned to an Observation Block under execution, is extended without the need to stop the observation.

The SAGSubArrayManager is also responsible for supervising the SAGSubArrayMonitor component, that monitors the SAG pipeline components to collect operational data, such as processing rates. The component shall send the data to the MON subsystem through an ACS notification channel.

The Supervisor tree described in [AD1] is realised deriving all ACS components from supervision::Supervisor, supervision::Supervised.

#### 7.2.1.1 SAG Supervisor

The SAG Supervisor is implemented with an ACS component whose features are described by an ACS interface named SAGSupervisor. This component is supervised by the RM subsystem, and it has the same lifetime of the nightly operations. It is responsible for the generation of scientific monitoring results of DL3 to be sent to the HMI, and also for providing input to the ACADA Reporting Subsystems for the generation of reports for HMI, DPPS, and SUSS.
To receive the Scheduling/Observation Blocks updates and data, it relies on the ACS notification channels defined by the CC subsystem [AD6]. It also supervises a specialized version of a SAGsci component (which interface is called SAGlazysci) used to process the DL3 data. It cannot supervise the same SAGsci component supervised by the SAGSubArrayManager due to conflicts in the role names of the supervision tree (the role name is generated using the interface name + Scheduling Block id).

### 7.2.2 Structure

The internal structure is shown in the following section.

### 7.2.3 Class Diagrams

The following figures show the class diagrams of the SAG-SUP component. They are defined using the UML notation.

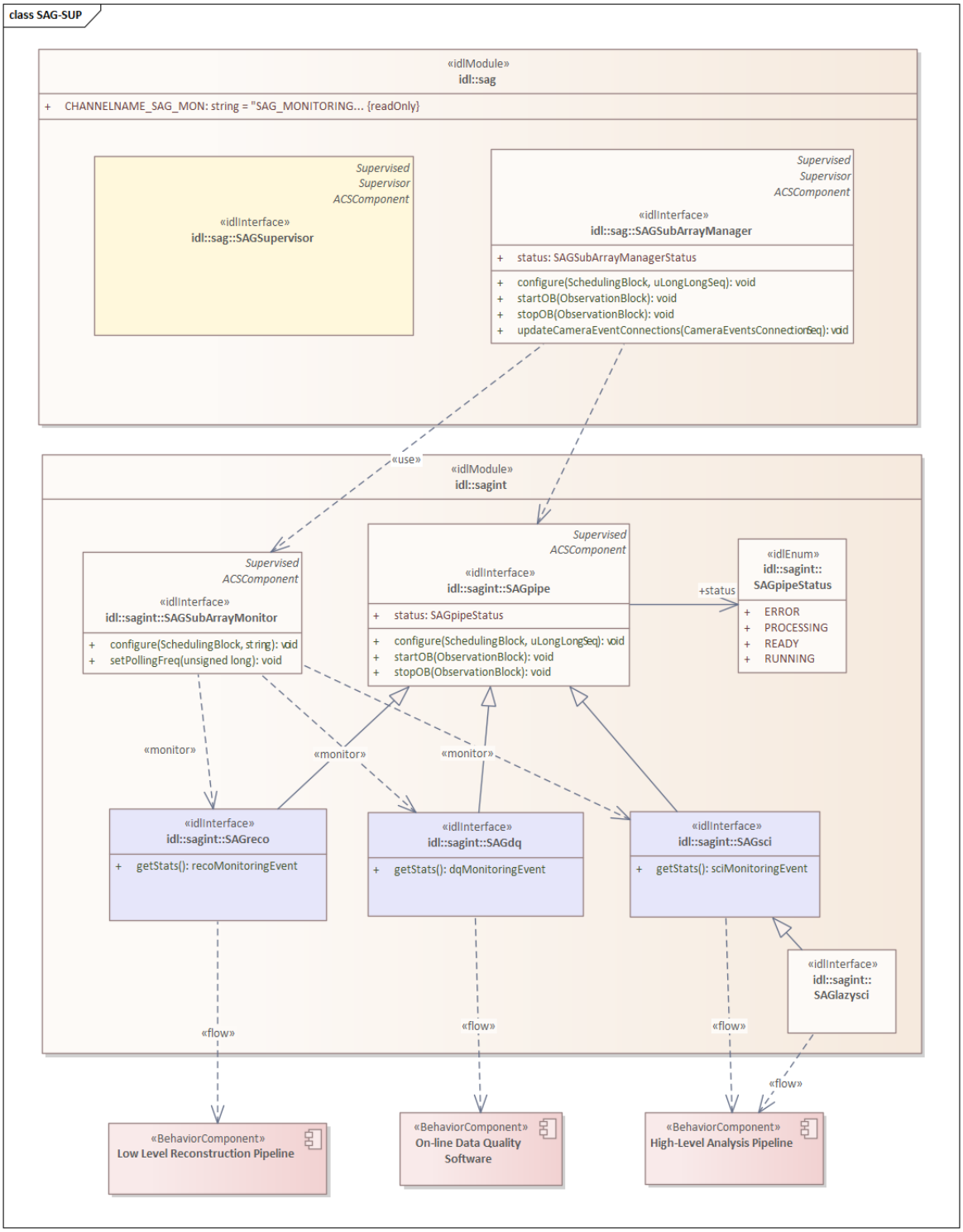


**Figure 8: SAG-SUP relationship with logical components and IDL modules**

### 7.2.3.1 Module::sag

The *sag* idl module contains the interfaces visible to the rest of the ACADA system. They are listed in the following table.

| |
|---|
| SAGSubArrayManager : «idlInterface»<br>This interface is used by the CC subsystem that supervises SAGSubArrayManager components, one for each Scheduling Block.<br><br>Each SAGSubArrayManager component isconfigured with a scheduling block and it is in charge to supervise SAGreco, SAGdq e SAGsci components that perform the data processing. |
| SAGSupervisor : «idlInterface»<br>This interface is used by the RM subsystem that supervises a SAGSupervisor component with the lifetime of an observing night.<br><br>This interface manages the generation of scientific monitoring results to be sent to the HMI and provides inputs to the ACADA Reporting Subsystems for the generation of reports to be sent to the HMI, DPPS, and SUSS. |

### 7.2.3.2 *IDLInterface*: SAGSubArrayManager

This interface is used by the CC subsystem that supervises SAGSubArrayManager components, one for each Scheduling Block.

| Method | Description |
|---|---|
| configure | configure (in sb::SchedulingBlock schedBlock, in ACS::uLongLongSeq telescope_ids)<br>: void Public<br><br>Description: this method will be called by the CC immediately after creating the SAGSubArrayManager for a particular SB, and when updating this SB or when updating the list of allocated telescopes. When this method is called for the first time, it will dispatch a worker thread that will instantiate in the background the the SAGreco, SAGdq and SAGsci components. |
| startOB | startOB (in sb::ObservationBlock obsBlock): void Public<br><br>Description: this method will be called by the CC when an OB state changes to RUNNING. The SAG pipelines will start to process the data. |
| stopOB | stopOB (in sb::ObservationBlock obsBlock): void Public<br><br>Description: this method will be called by the CC when an OB state changes to COMPLETED_SUCCEDED or COMPLETED_CANCELED or COMPLETED_TRUNCATED. |
| updateCameraEventConnections | updateCameraEventConnections (in CameraEventsConnectionSeq connections)<br>: void Public<br><br>Description: called by ADHSupervisor to inform us about ZMQ Camera Event connections to use in order to listen to incoming camera events. Called usually once when ADHSupervisor is initialized, but may be called anytime if SDH instances were replaced. The complete list of connections is always provided. Method should be sync and return quickly after taking note of the update. Reactions to the update (creation/configuration of pipelines) should be handled internally. |

### 7.2.3.3 *IDLInterface*: SAGSupervisor

This interface is used by the RM subsystem that supervises a SAGSubArrayManager component. This interface manages the generation of scientific monitoring results to be sent to the HMI and provides inputs to the ACADA Reporting Subsystems for the generation of reports to be sent to the HMI, DPPS, and SUSS.

The SAGSupervisor does not define any public interface methods, but it relies on the inherited ACS lifecycle management methods.

### 7.2.3.4 Module::sagint

The *sagint* module contains all the internal SAG interfaces listed in the following table.

| |
|---|
| SAGSubArrayMonitor : «idlInterface»<br><br>This interface allows the monitoring of the SAG pipeline components to collect operational data, such as processing rates. This data is sent to the MON subsystem through an ACS notification channel. |
| SAGpipe : «idlInterface»<br><br>This interface allows to start and stop SAG data analysis pipelines. The pipelines are managed by SAGpipe components, in particolar SAGreco, SAGdq and SAGsci that are supervised by the SAGSubArrayManager component. When the SAGSubArrayManager supervisor shutdowns SAGreco, SAGdq and SAGsci, these components will wait for the pipelines to finish the data processing and then they terminate. |
| SAGdq: «idlInterface»<br><br>This interface is used to start data quality analysis processes on Slurm. It is used internally within the SAG system. |
| SAGreco : «idlInterface»<br><br>This interface is used to start reconstruction analysis processes on Slurm. It is used internally within the SAG system. |
| SAGsci : «idlInterface»<br><br>This interface is used to start scientific analysis processes on Slurm. It is used internally within the SAG system and it is supervised by the SAGSubArrayManager. |
| SAGlazysci : «idlInterface»<br><br>This is used to start scientific analysis processes on Slurm. It is used internally within the SAG system and it is supervised by the SAGSupervisor. |

### 7.2.3.5 IDLInterface: SAGpipe

| Method | Description |
|---|---|
| configure | configure (in sb::SchedulingBlock schedBlock, in ACS::uLongLongSeq telescope_ids) : void Public<br><br>Description: this method will be called by the SAGSubArrayManager when a new SB is available and also when a SB is updated or when updating the list of allocated telescopes. This method will configure the configuration file of the corresponding pipeline and it may start the actual slurm jobs. |
| startOB | startOB (in sb::ObservationBlock obsBlock): void Public |

| Method | Description |
|---|---|
| | Description: This method will be called by the SAGSubArrayManager when an OB state changes to RUNNING. The SAG pipelines will start to process the data. |
| stopOB | stopOB (in sb::ObservationBlock obsBlock): void Public<br><br>Description: This method will be called by the SAGSubArrayManager when an OB state changes to COMPLETED_SUCCEDED or COMPLETED_CANCELED or COMPLETED_TRUNCATED. |

### 7.2.3.6 IDLInterface: SAGreco

| Method | Description |
|---|---|
| getStats | getStats() : sag::recoMonitoringEvent Public<br><br>Description: ask for monitoring point events. |

### 7.2.3.7 IDLInterface: SAGdq

| Method | Description |
|---|---|
| getStats | getStats() : sag::dqMonitoringEvent Public<br><br>Description: ask for monitoring point events. |

### 7.2.3.8 IDLInterface: SAGsci

| Method | Description |
|---|---|
| getStats | getStats() : sag::sciMonitoringEvent Public<br><br>Description: ask for monitoring point events. |

### 7.2.3.9 IDLInterface: SAGlazysci

| Method | Description |
|---|---|
| getStats | getStats() : sag::sciMonitoringEvent Public<br><br>Description: ask for monitoring point events. |

### 7.2.4 General behavior

The SAGSupervisor and the SAGSubArrayManager are the interfaces of the SAG w.r.t. ACADA.

### 7.2.5 State machine

Since the SAGSubArrayManager acts as the "controller" of the SAG subsystem, it must have an interface which provides information on the current subsystem state in any given moment. The SAGSubArrayManager is mapped to a state machine, which defines transitions between all the possible states. The state machine of the SAGSubArrayManager depends on the state machine of its supervised components. The figure below shows the state machine, where "sc" stands for "supervised component".
The state machine is reported in the following figure.

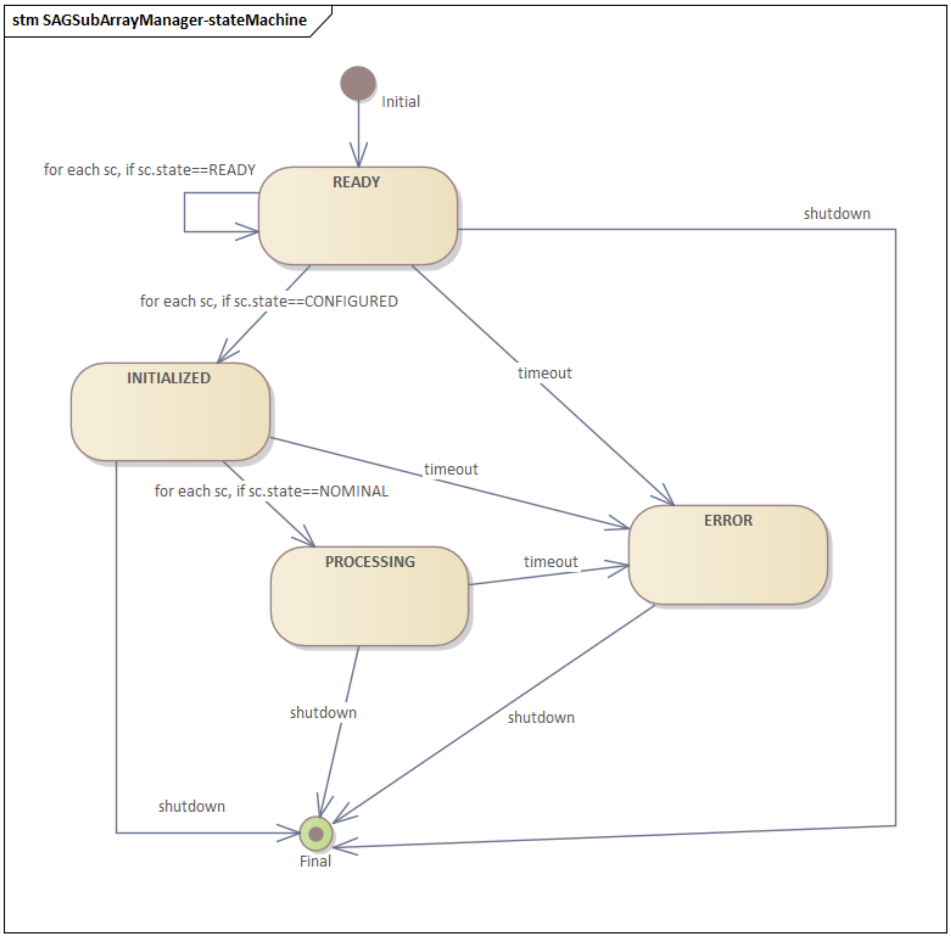


**Figure 9: State machine of the SAG-SUP ACS components.**

When the SAGSubArrayManager is created, its internal state will be equal to *READY*. The required action to start the SAG analysis processes (or SAG pipelines, i.e. slurm jobs) is to bring the SAGSubArrayManager's state to *INITIALIZED*. To be *INITIALIZED* the SAG-SubArrayManager needs a scheduling block and the zmq connection parameters.

The scheduling block is passed through the *configure(SchedBlock)* api method that will be called by the CC. When *configure(SchedBlock)* is called, the SAGSubArrayManager will

create the SAG{Reco,Dq,Sci} ACS supervised components. The configure(SchedBlock) call will return immediately, but it will trigger background configurations: the SAG{Reco,Dq,Sci} components are created in separate threads. In addition, those subcomponents will, in turn, do some internal configuration.

The zmq connection parameters are passed through the updateCameraEventConnections(CameraConnections []) API method that will be called by the ADH. The ADH can call this method as soon as the SAGSubArrayManager component is alive, so before or after the CC calls *configure(SchedBlock)*.

When the SAGSubArrayManager's status is *INITIALIZED*, the *startOB(ObsBlock)* API method can be called to start the SAG analysis processes, and its status will transact, after some delay, to *PROCESSING*. This status is reached if and only if all the expected Slurm jobs are up and running on the cluster. If this check fails, the status will transact to *ERROR*. If *startOB(ObsBlock)* is called and the status is not *INITIALIZED*, an exception will be raised.

The *shutdown()* call will kill immediately all the Slurm jobs associated to the scheduling block. If this method is called while the status of the SAGSubArrayManager's status is equal to *PROCESSING*, the data analysis will be stopped even if there's data left to analyze. The supervised components will be destroyed and the SAGSubArrayManager's status will transact back to *READY*.

The *stopOB(ObsBlock)* call will let the Slurm jobs to process all the data left and only then they are stopped. In this case, the supervised components are not destroyed but they will be ready to process the next observing block. The status of the SAGSubArrayManager will transact to *INITIALIZED*. The *stopOB(ObsBlock)* call lets SAG know when the repointing phase starts. This is important because we don't want to analyze the data taken during this phase, assuming that the data stream does not stop during the repointing.
The *stopOB(ObsBlock)* call also allows a smoother transition of SAG's available computing resources between an OB's end and the next one's start. Indeed during this time, SAG can release any pipeline that finished its processing.

### 7.2.6 Command Interface

See the following subsections of each subsystem.

### 7.2.7 Activities

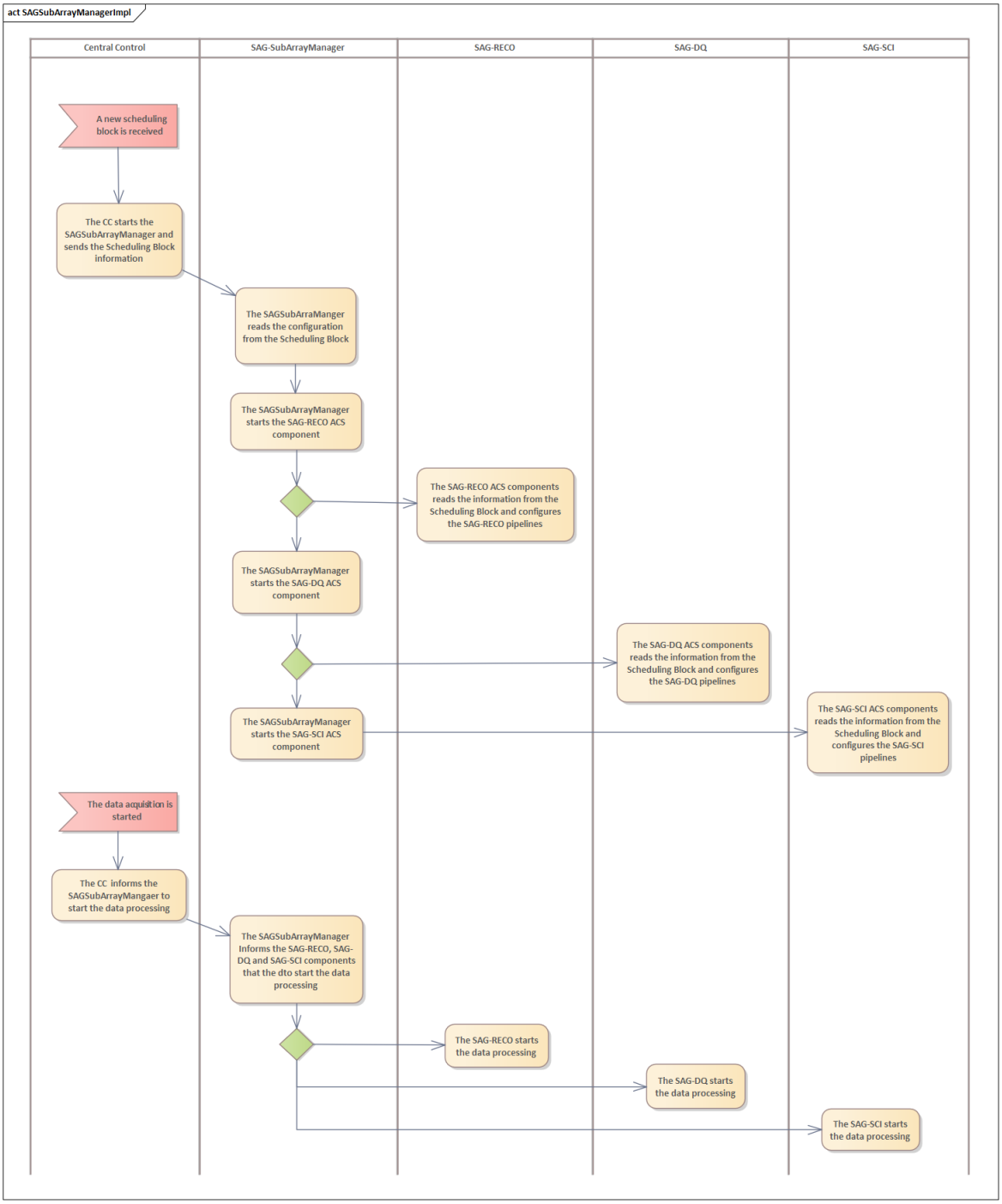


**Figure 10: SAGSubArrayManager activity diagram: Starts the pipelines for a new Scheduling Block.**

### 7.2.8 Sequences

In this section we show relevant sequence diagrams to document the handling of various use cases.

The following two diagrams put a relationship between the IDL interfaces and the implementation of the ACS components using Python language.

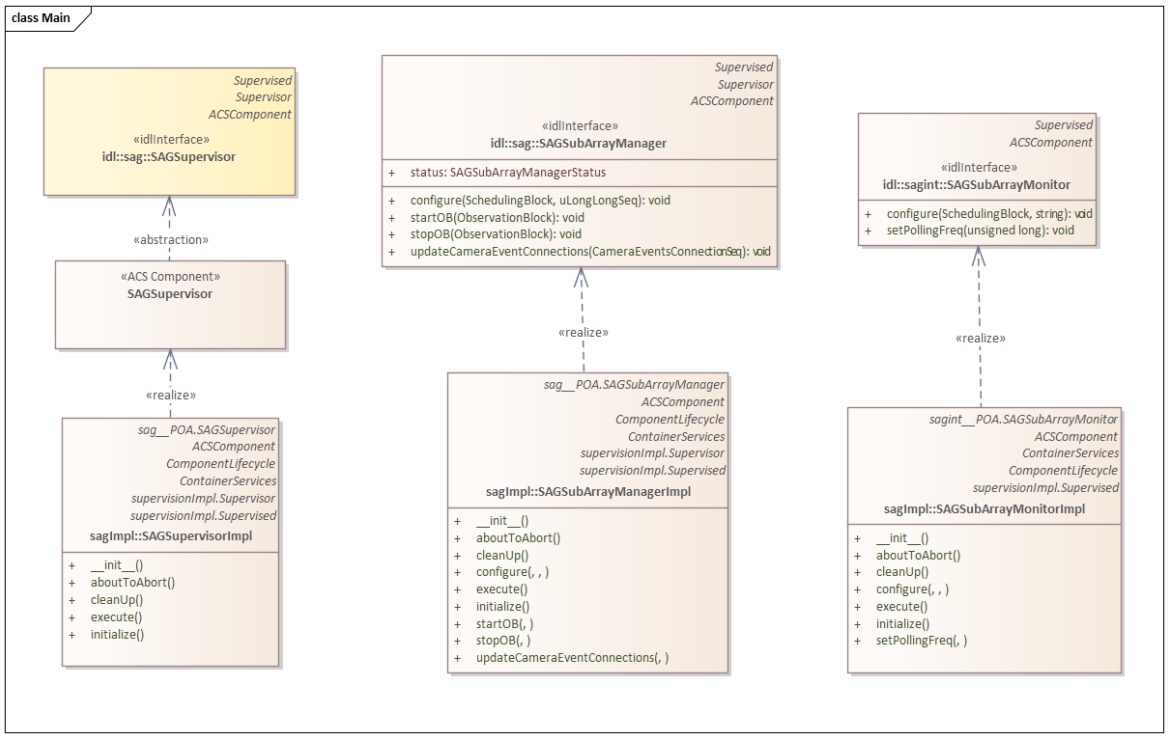


**Figure 11: SAG ACS components with implementation of the interfaces (part 1)**

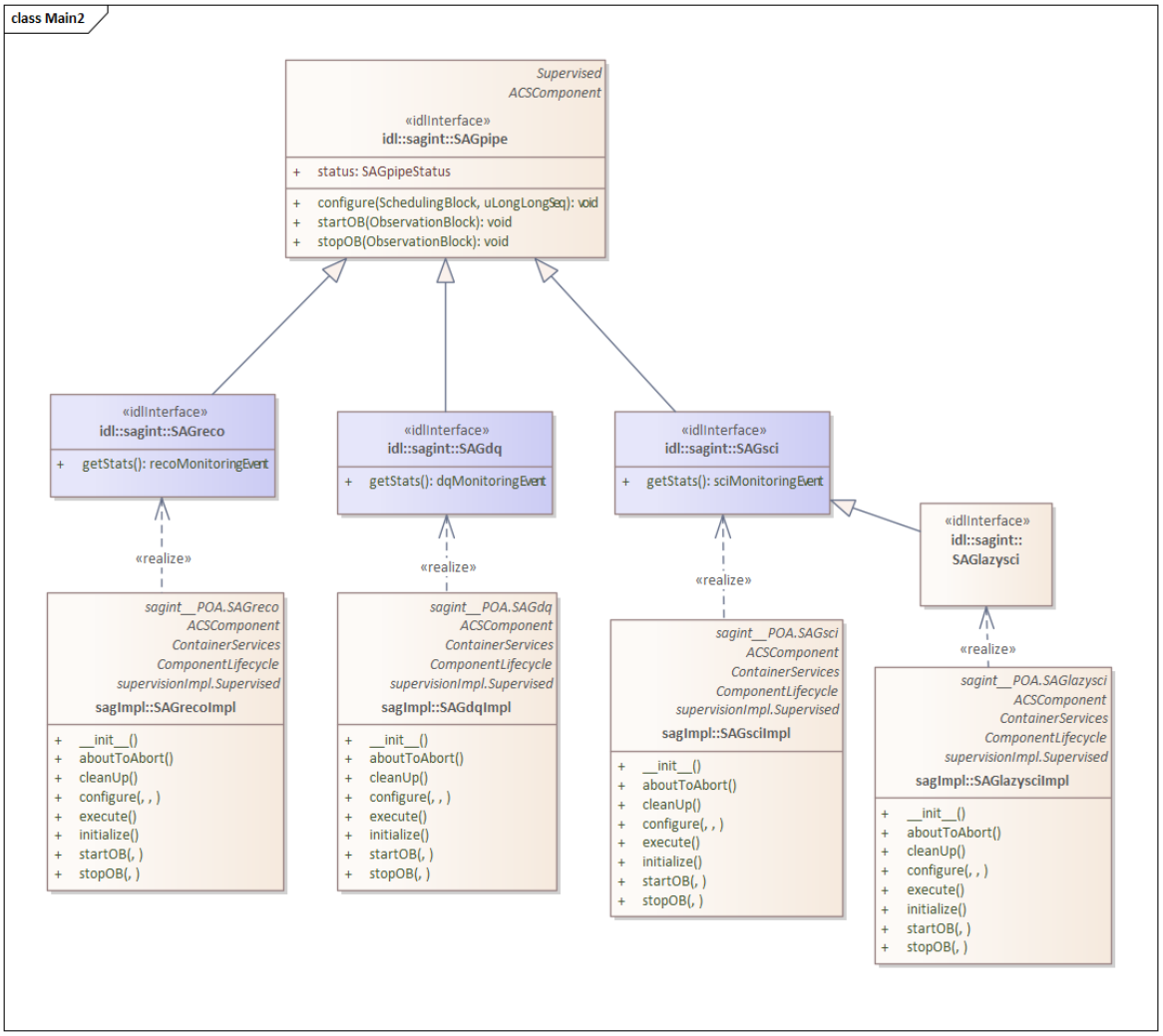


**Figure 12: SAG Supervisor with implementation of the interfaces (part 2)**

### 7.2.8.1 Sequence: initialize SAGSupervisor

At the beginning of the observing night the Resource Manager activates the SAGSupervisor ACS component.

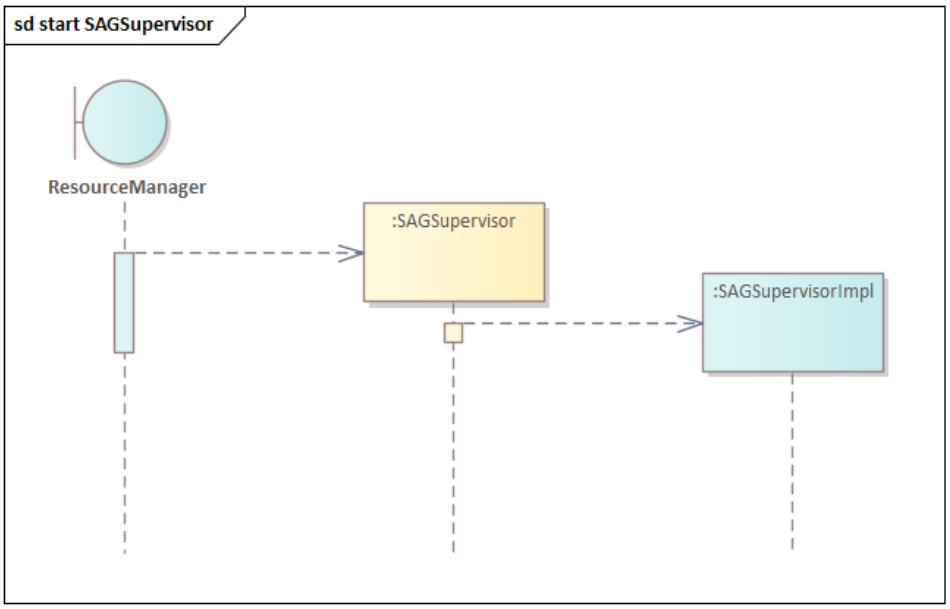


**Figure 13: Initialise**

### 7.2.8.2 Sequence: configure()

When a new SB is received, a dedicated SAG pipeline shall start in an automatic and controlled way.

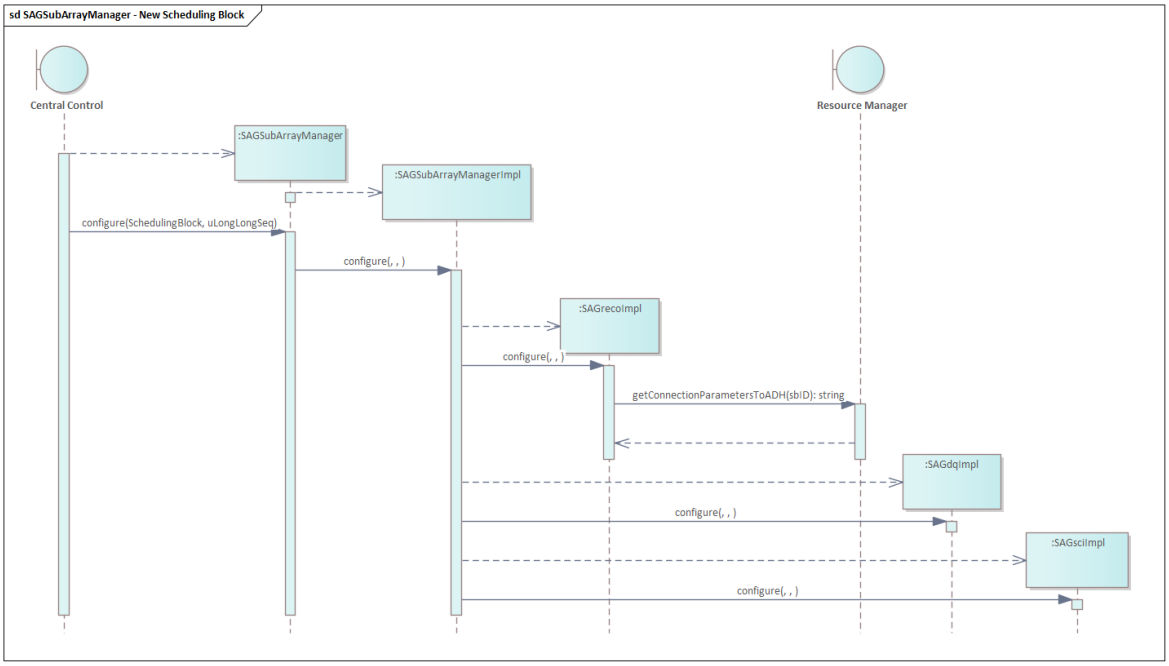


**Figure 14: newSchedulingBlock()**

### 7.2.8.3 Sequence: startOB()

When a sub-array enters in observing state, the CC informs the SAGSubArrayManager

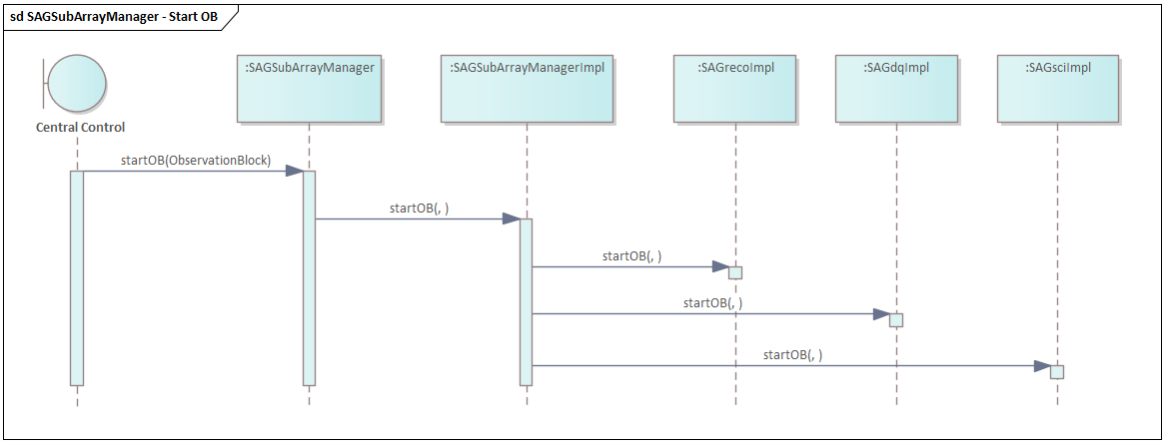


**Figure 15: startOB()**

#### *7.2.8.3.1 Sequence: stopOB()*

When a sub-array stops the observation, the Central Control shall inform the SAGSubArrayManager that shall shutdown data processing in an automatic and controlled way.

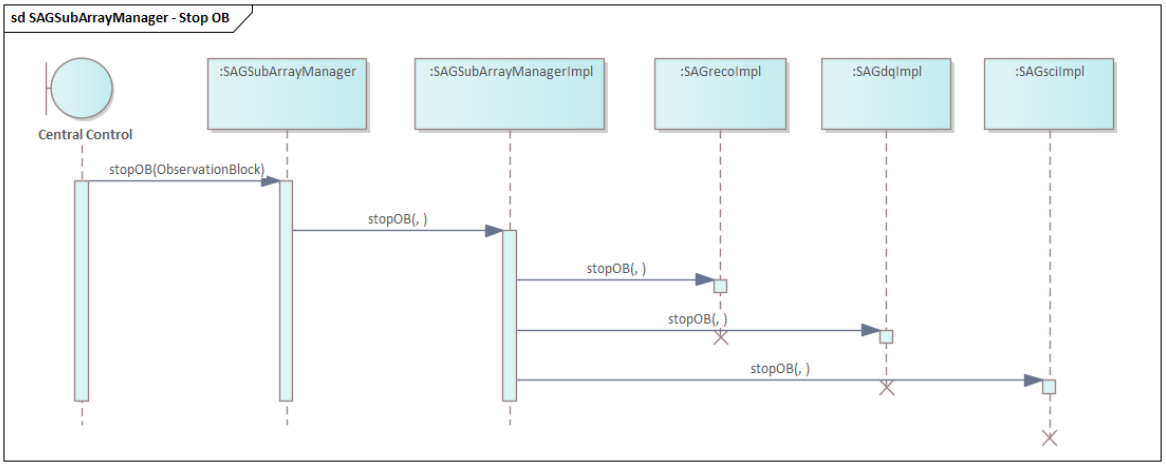


**Figure 16: stopOB()**

### 7.2.8.4 Sequence: shutdown()

SAG shall receive information when the execution of a Scheduling Blocks is cancelled. Analysis of the data taken until the cancellation command shall be executed.

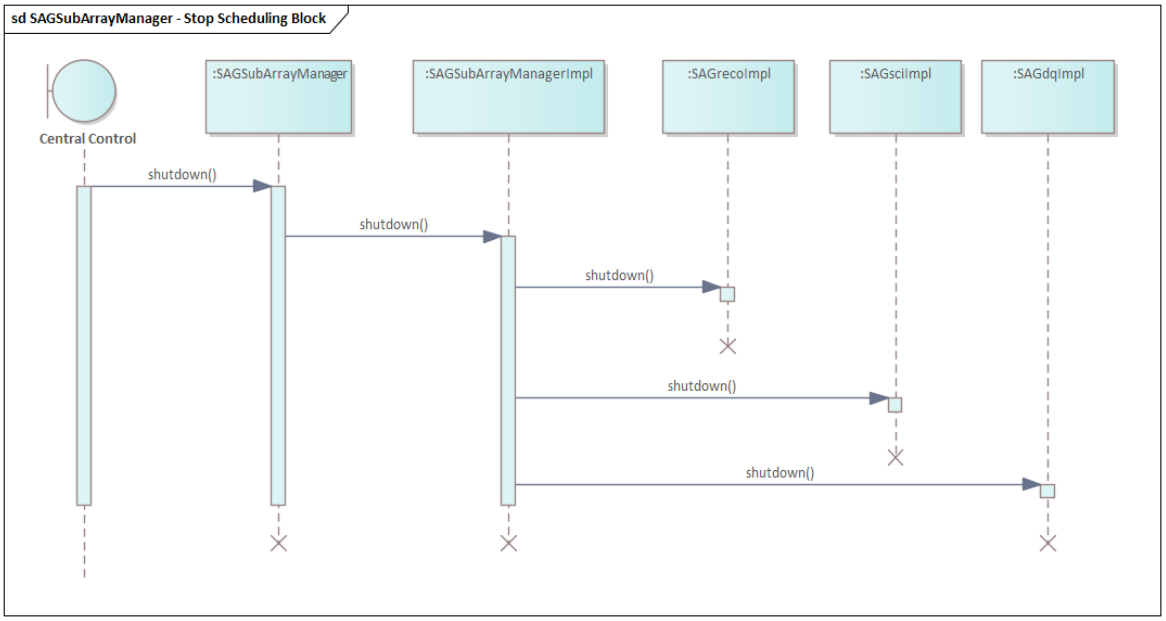


**Figure 17: shutdown()**

## 7.3 Component: Image Parameter Extractor and Low-Level Reconstruction Pipeline

This component, also called SAG-RECO in this document is designed to perform data reconstruction in real-time during the observations.
DL0 data is received from the camera data handling with information about telescopes triggered from sub-array triggering system. The image integration is performed to obtain a camera map of integrated charges and a camera map of mean arrival time. From these maps, a tailcut cleaning is performed and image parameters (first moments of the signal, global intensity, time gradient...) are extracted by the Image Parameter Extractor (data level 1). These parameters are fed into a model (such as random forests) trained on Monte-Carlo simulations to reconstruct the event physical parameters (direction, energy and particle type - data level 2). The last stage of SAG-RECO applies selection cuts to produce DL3 data that are then provided to the High-Level Analysis Pipeline (SAG-SCI). This component is also able, depending on the configuration, to inform ADH if an event should be stored, providing means for further data volume reduction.

### 7.3.1 General description

The document [AD2] defines a list of requirements for this component that are needed to fulfill the use cases of the SAG system. This pipeline shall execute reconstruction in parallels for different sub-array observing at the same time. It shall reconstruction an event with a latency of 15 seconds. These are some of the main requirements for the SAG-RECO pipeline.

The input data for this component is the DL0 (R0 for Mini-ACADA) produced by the camera data handler and the information about telescopes triggered from sub-array triggering system.

The SAG-RECO pipeline is composed of the following main components:

- Hillas parameters extraction (DL0 to DL1),
- Spatial reconstruction, energy estimation and background rejection (DL1 to DL2)
- Selections to keep good quality gamma events (DL2 to DL3)

This pipeline has interfaces with external systems and components:

- Camera data handling provides DL0 data to SAG-RECO
- Sub-array triggering provides information about telescopes triggered from sub-array triggering system to SAG-RECO (not for ACADA REL1)
- SAG-RECO provides SAG-SCI the DL3 data
- SAG-RECO provides Camera data handling DL3 information to reduce data volume (only for final ACADA release)
- SAG-RECO provides DL1 and DL2 data SAG-DQ for monitoring reconstruction processes

Many of these interfaces are managed through the SAGSubArrayManager component and not directly by the SAG-RECO pipeline.

### 7.3.2 Structure

For the DL1, the telescope streams are merged at the DL1 level, and the DL2 are merged at the sub-array level.

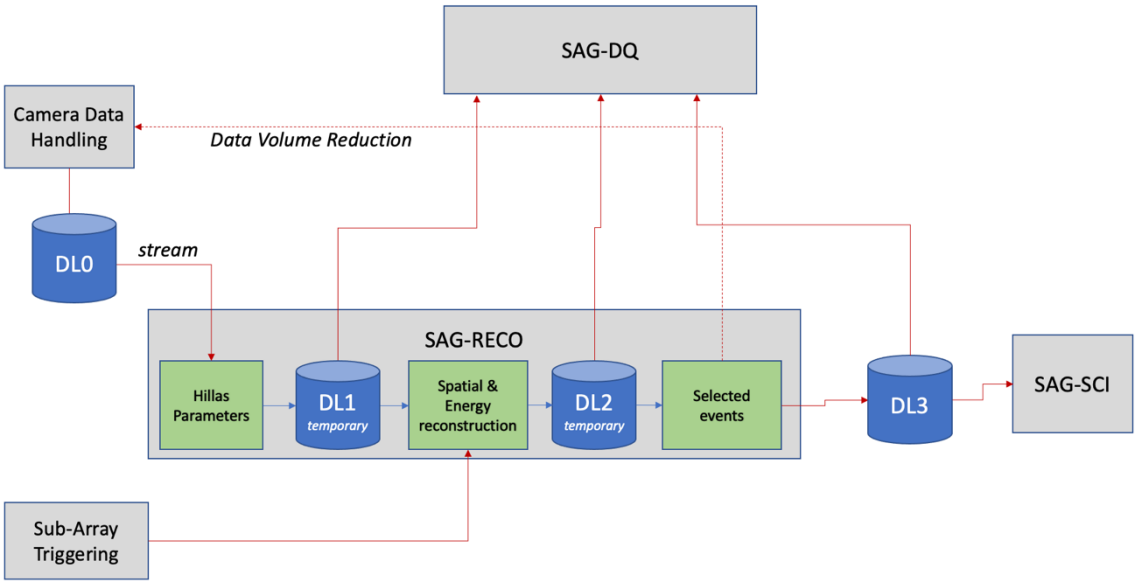


**Figure 18: Low level reconstruction pipeline.**

### 7.3.3 Class Diagrams

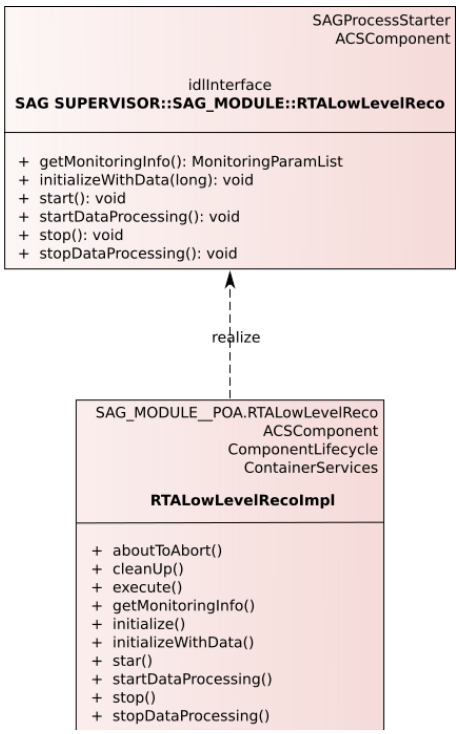


**Figure 19: Low level reconstruction interface.**

### 7.3.4 General behavior

Raw data are received from the Array Data Handler (ADH) via Protocol Buffer data format version 2.

First, charges are integrated. Integration consists in summing the calibrated signal for all samples in order to increase the signal over noise ratio. The expected output is one single image for the signal charge converted in photoelectrons and one image of the time of maxima for all pixels. Maximum time of waveform is obtained on each pixel independently and used to produce a time map.

Second, a cleaning step is performed. The cleaning step aim to remove the pixel containing only noise and to select only pixel containing signal in the calibrated and integrated images. A pixel is kept if it has a signal greater than a threshold and if at least one of its neigbours has a signal greater than another threshold.

Third, Hillas parameters are computed. Hillas parameters are barycenters and projections to extract position, orientation, asymmetry and other parameters of the electromagnetic shower from cleaned images. Timing parameters are also computed using a weighted mean of maximum time in pixel time map.

After these steps, DL1 are produced. DL2 consists in computing the gammaness, arrival direction and energy based on random forest trained on Monte Carlo simulations. These

random forest will be provided by DPPS. Then, conversion to DL3 consists in performing quality and gammaness selection on DL2, and in converting this to a gammapy compliant format.

In the release 1 of ACADA, we are using the lstchain software to produce the steps DL1->DL3. In the future, the DPPS data processing pipeline will be used.

### 7.3.5 State machine

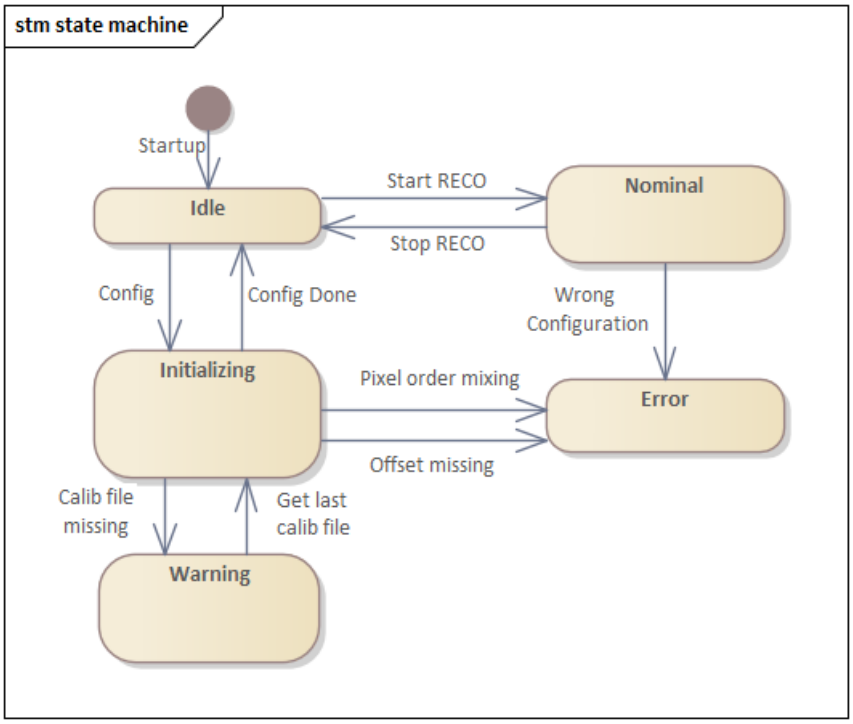


**Figure 20: SAG-RECO state machine.**

### 7.3.6 Command Interface

Starting the RECO analysis can be done with:

```
$ hiperta_stream_start --run_id <RUN_ID> [--config_file hiperta_stream_configuration.yml]
```

### 7.3.7 Activities

In the LST-1 framework, the camera is producing 4 streams of data. Each of this 4 stream can be shared in a tunable number of threads (this is independent events) in order to produce DL1. After the production of DL1, the production of DL2 and DL3 can be as well shared in multiple jobs in order to improve the speed of the DL3 production.

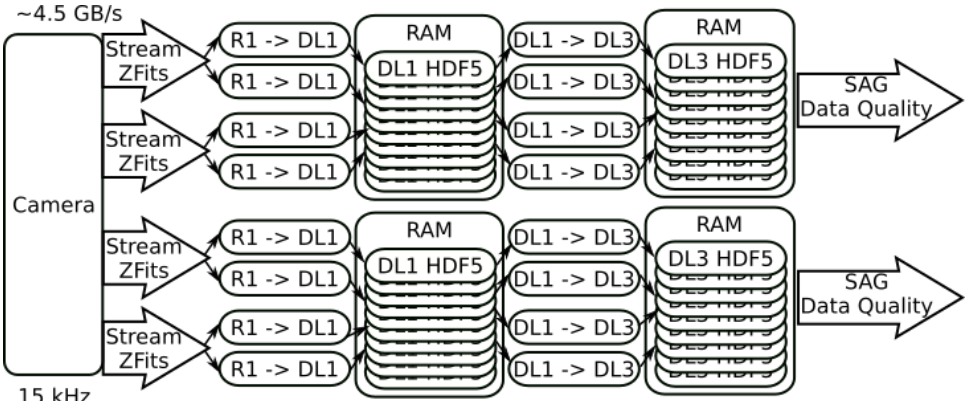


**Figure 21: SAG-RECO activities.**

## 7.3.8 Sequences

### 7.3.8.1 low-level reconstruction

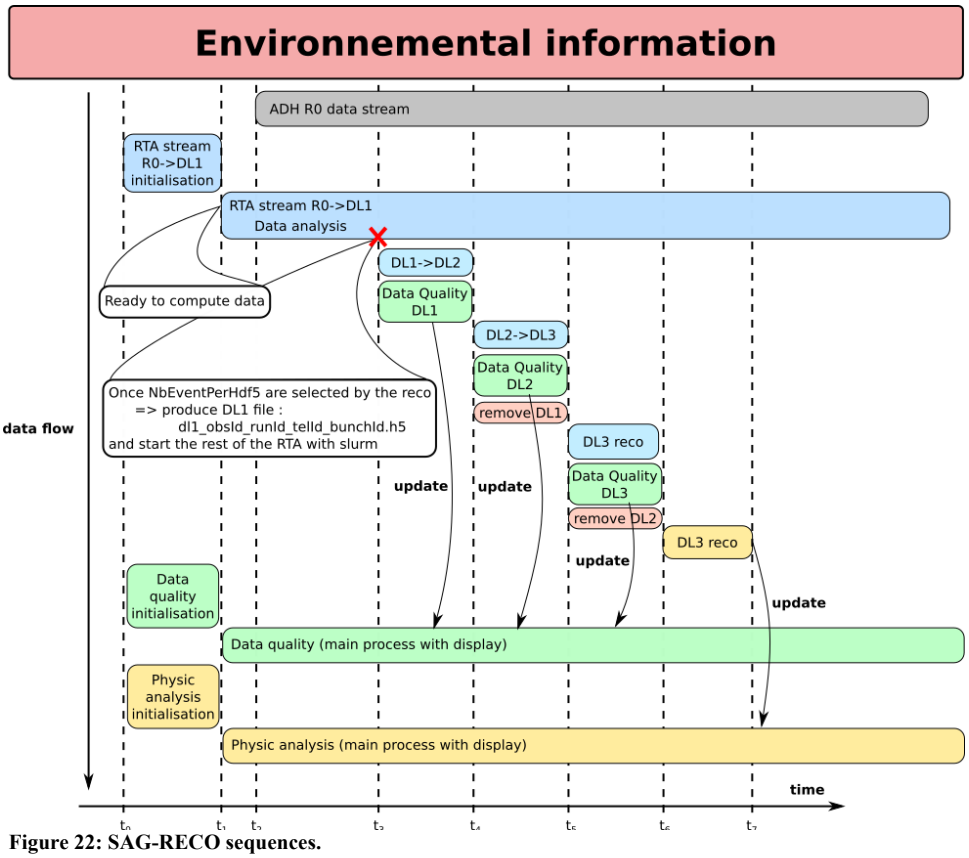


**Figure 22: SAG-RECO sequences.**

- DL0 (R0 in miniACADA) is reduced to DL1 data, containing:
  - Integrated charge per pixel
  - Mean arrival time per pixel
  - Extracted image parameters

- DL1 data is fed to previously trained random forests on Monte-Carlo simulations to produce DL2 data:
    - Event incoming direction in the sky as (altitude, azimuth)
    - Event reconstructed energy
    - Event gammaness (probability to be a gamma)
- DL3 data are produced from DL2 by applying selection cuts and converting sky co-ordinates in RA/DEC

At each stage, the produced files are written to disk to be read by later stage as well as the data quality.

## 7.4 Component: On-Line Data Quality Software

### 7.4.1 General description

The On-Line Data Quality Software (SAG-DQ) is a logical component that performs real-time data quality analysis. The latency to perform the data quality analysis must be less than 5 seconds with respect to the data provided as input. This software must be able to process every data level from DL0 to DL3.

The SAG-DQ is composed of two software libraries, called rta-dq-lib and rta-dq-pipe. Both software components have been developed using the Python programming language.
The rta-dq-lib library is dedicated to the rapid prototyping of data quality use cases. It allows the user to define, through XML configuration files, the structure of the input data and, for each input data field, which data quality results should be produced. There are two types of data quality results: statistical data (such as distributions or rms) and data quality checks. The aim of a data quality check is to verify if the values of the input data satisfy some constraints. If, for example, the pixel value of a telescope's camera exceeds an upper bound threshold , a warning or an alarm result will be generated.

The rta-dq-pipe is used to run in parallel an arbitrary number of data quality pipelines. The goal is to split the data processing to achieve greater throughput. The software must be configured with a configuration file in xml format, that specifies the number of pipelines and, for each pipeline, its type and its input and output directories, along with the input and output data formats.

The following paragraphs will describe both software packages, rta-dq-lib and rta-dq-pipe.

### 7.4.2 Structure

The next figure shows a process-level view of the SAG-DQ logical component. In this example, three data quality pipelines (in blue) are running. They can be executed as linux processes or slurm jobs and they execute the logic present within the rta-dq-pipe::DQAnalysis and rta-dq-pipe::DQAggregator classes. Those classes internally use the rta-dq-lib::DQLib in order to perform the computations.

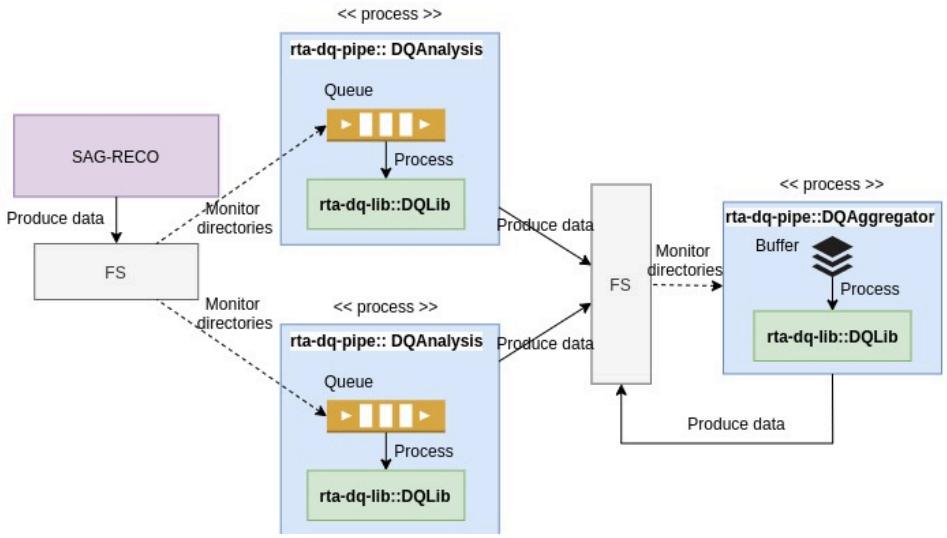


**Figure 23: SAG-DQ process view**

### 7.4.3 Class Diagram: rta-dq-lib

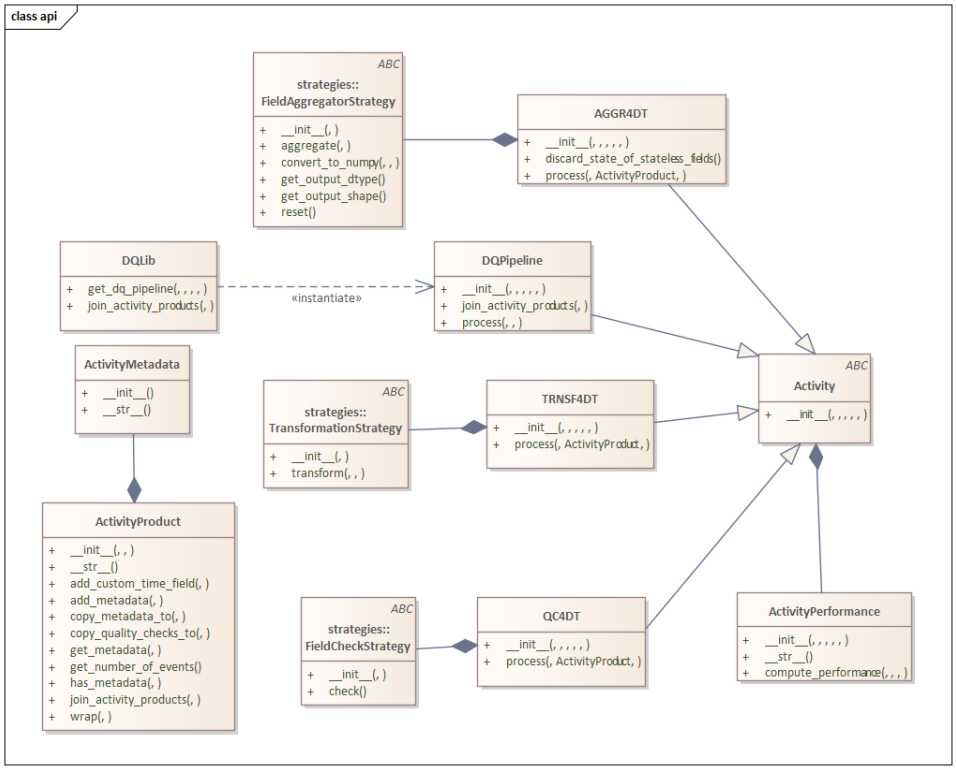


**Figure 24: SAG-DQ class diagram**

The class digram above shows the main classes present in the rta-dq-lib packages. The “entry point” is represented by the DQLib class, a static class that allows to construct DQPipeline objects. The latter is the core class that orchestrates the computations: it uses QC4DT, AGGR4DT and TRNSF4DT objects to produce the data quality results.
The QC4DT class is used to perform quality checks. The constructor reads the corresponding 'qualitychecks.xml' configuration file in order to set up the operational behavior of the QC4DT object. The AGGR4DT class is used to aggregate data. The constructor reads the

corresponding 'aggregations.xml' configuration file in order to set up the operational behavior of the AGGR4DT object. The TRNSF4DT class is used to transform data. The constructor reads the corresponding 'transaformations.xml' configuration file in order to set up the operational behavior of the TRNSF4DT objects.
The templates for the configuration files can be found under the “xml_templates” directory.

More details for the DQPipeline class are provided in the next section.

#### 7.4.3.1 ***Class***: DQPipeline

This class provides the methods to interact with the rta-dq-lib.

| Method | Description |
|---|---|
| init() | __init__ (self : , activity_id : , configuration_file_path : , obs_id : , debug_lvl : , compute_pipe_performance : , compute_activities_performance : ) : Public<br><br>Description:<br>It returns a DQPipeline instance.<br><br>:param activity_id: the DQPipeline id. :type activity_id: :class:`str`.<br>:param configuration_file_path: the relative or absolute path to the xml configuration file. :type configuration_file_path: :class:`str`.<br>:param debug_lvl: it controls the verbosity of the output (0 => no output, 1 => info, warning and critical messages, 2 => debug, info, warning and critical messages) :type debug_lvl: :class:`bool`.<br><br>:returns: :class:`DQPipeline` -- A DQPipeline object. |
| join_activity_products() | join_activity_products (activity_products : ) : Public<br><br>Description: join the output coming from different lines of parallel executions of this class. |
| process() | process (self : , events : , run_id : ) : Public<br><br>Description: It runs the data quality analysis on the input events data.<br><br>:param events: a numpy structured array or a python dictionary.<br><br>:returns: :class:`dict` -- A dictionary representing a aggregated or transformed event. |

### 7.4.4 Class Diagram: rta-dq-pipe

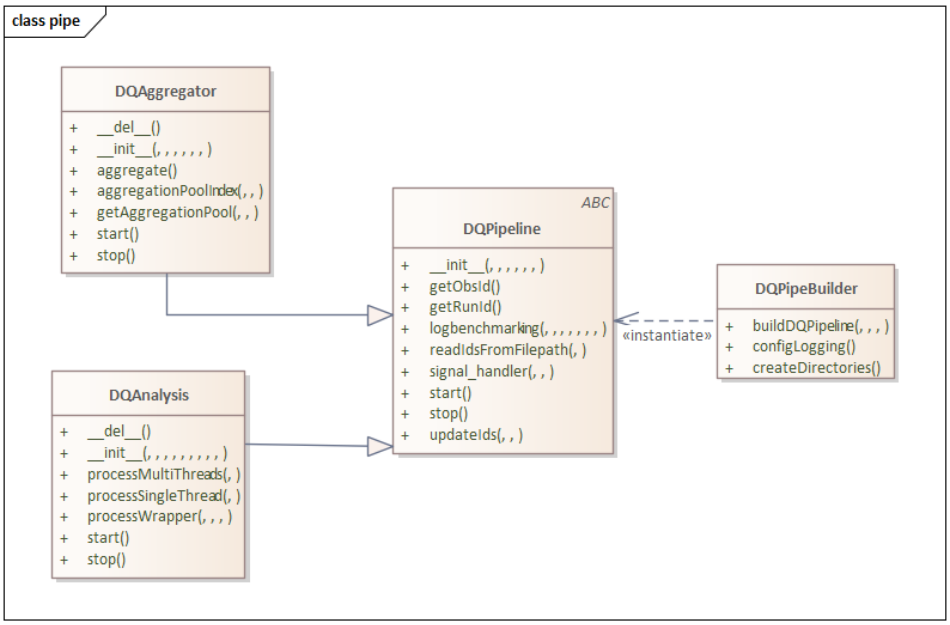


**Figure 25: rta-dq-pipe class diagram (part 1)**

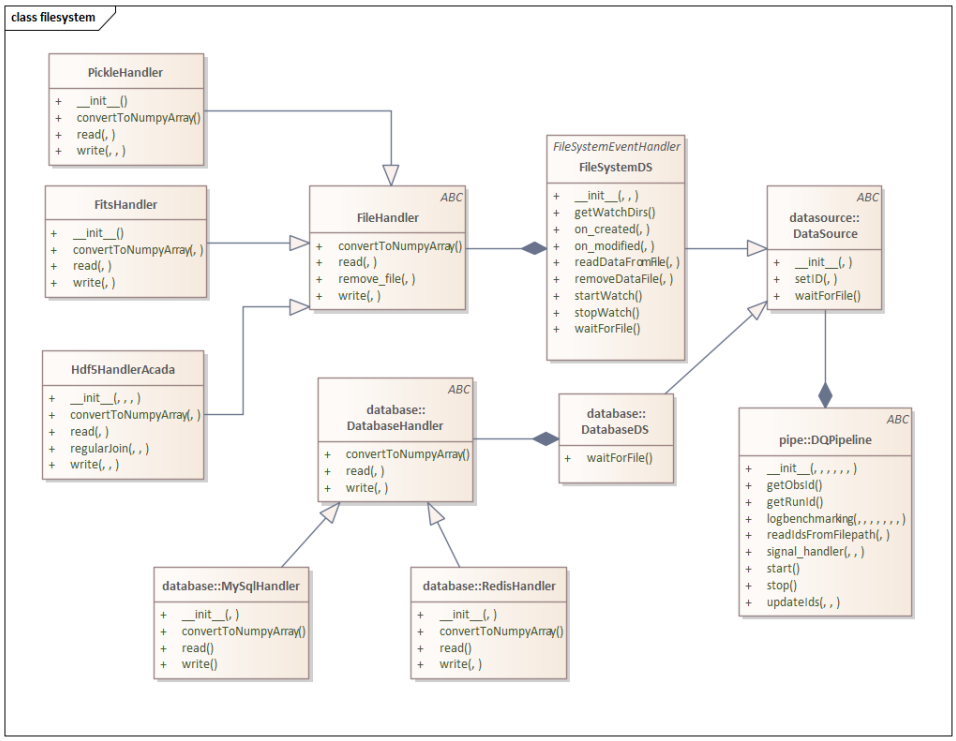


**Figure 26: rta-dq-pipe class diagram (part 2)**

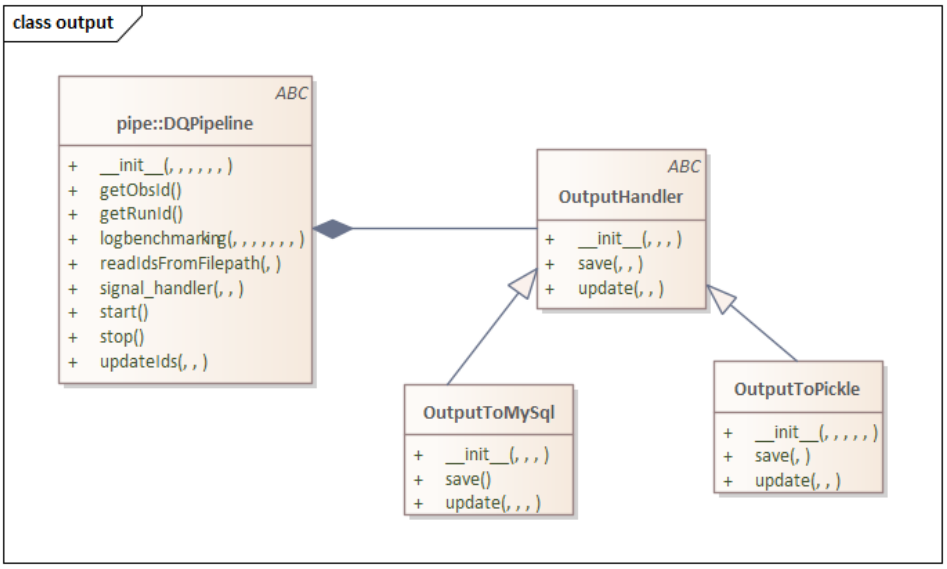


**Figure 27: rta-dq-pipe class diagram (part 3)**

Figure 25 shows the class diagrams that describe the rta-dq-pipe software structure. Figure 25 shows the core classes. DQPipelineBuilder is a static class that is needed to construct rta-dq-pipe::DQPipeline objects. It needs a configuration file in xml format that describes how many DQPipeline objects must be constructed and other details, such as the definition of the input directories. The configuration file details will be explained further in this document. The DQAnalysis class is a wrapper around the rta-dq-lib::DQPipeline core class: it adds three main functionalities:

it handles the input, managing a queue of HDF5 files to process;
it can split a single input HDF5 file in several batches to speed up the computation in a multithreading fashion;
it handles the output of the rta-dq-lib::DQPipeline: it can write it on file or it can interact with a database.

Figure 26 shows the hierarchy of classes that manage the input. Classes FileSystemDS and DatabaseDS implements the DataSource interface. They abstract the way new data is available: the first class hides the details of monitoring file system directories while the other hides the details of polling a database. In addition, for each data source abstraction, there are different classes that abstract the file type (Pickle, HDF5, Fits) and database connections (Mysql, Redis).
Figure 27 shows the classes that manage the output: one class can interact with a database, while the other writes binary files.

## 7.4.5 General behavior

### 7.4.5.1 rta-dq-lib

The rta-dq-lib performs a set of standard operations to transform the input data in data quality indicators. Two types of quality indicators are currently supported: quality checks (that defines warning and alarms), and statistical data (such as distributions or averages). This library is implemented in Python and it relies on Numpy to perform fast computations. In particular, the following Numpy operations are used:
Np.sum (to sum the element of an array) (axis=None)
Np.square (to apply the radix square functionto each array element)

Min (to get the minimum element of an 1D/2D array)
Max (to get the maximum element of an 1D/2D array)
Np.add (to sum two arrays together)
indexing operations (to cut an array or select elements row-wise and column-wise)
Vector operations (to apply the same math operator to each array element)

The library assumes that the input data format is HDF5, i.e. a table composed of several rows (the events) and several columns (events' features). The feature's data types can be scalar, 1d array or 2d array.

A data quality check can be applied to any feature, and it verifies if the feature's values satisfy a custom constraint. The quality check is applied "column wise", so the same constraint is checked against each row. The different types of supported constraints are:

- "upperboundthreshold": an alert of type "warning" is raised if a value exceeds the "ub_warning" threshold, an alert of type "alarm" is raised if a value exceeds the "up_alarm" threshold. It must be "ub_warning" < "up_alarm".
- "lowerboundthreshold": an alert of type "warning" is raised if a value exceeds the "lb_warning" threshold, an alert of type "alarm" is raised if a value exceeds the "lb_alarm" threshold. It must be "lb_alarm" < "lb_warning".
- "rangethreshold": in this case, the value must be within the range ["lb_warning", "ub_warning"]. If the value exceeds this range but it still contained in the range ["lb_alarm", "up_alarm"], an alert of type "warning" is raised. If the value exceeds the range ["lb_alarm", "up_alarm"], an alert of type "alarm" is raised.
- The output of a quality check is a python dictionary: the "batch_event_ids" key contains the event_ids of the processed events for a particular feature. Next figure shows an example of a data check output for the "image" feature.

```
{  'batch_event_ids': array([1749002, 1749003, 1749004, ..., 1801998, 1801999, 1802000]),
   'image': {  'alarms': array([[    0,    22],
       [    0,    56],
       [    0,   100],
       ...,
       [52998,  1818],
       [52998,  1823],
       [52998,  1827]]),
              'alarms_evt_ids': array([    0,     0,     0, ..., 52998, 52998, 52998]),
              'warnings': array([[    0,   458],
       [    3,  1373],
       [    3,  1504],
       ...,
       [52998,   721],
       [52998,   819],
       [52998,   917]]),
              'warnings_evt_ids': array([    0,     3,     3, ..., 52998, 52998, 52998])}}
```

The output dictionary will contain an "image" key which value is another dictionary containing the following keys:

- the "alarms_event_id" is a list that contains the index of the event that raised the alarm. This index can be used to retrieve the event id, e.g. evt_id = batch_event_ids[index];
- the "alarms" key contains a matrix: each element is an "alarm". Each "alarm" is an array of 2 elements (e.g. [0,22]) with the following meaning: [index of the event that raised the alarm, index of the pixel with the bad value];
- Note that the same event can raise multiple alarms: one for each camera pixel that exceeds the alarm threshold value.

The same logic is applied for the “warnings” key.

In order to define a quality check on a particular feature, the user of the library must write a configuration file called “qualitychecks.xml”. The following code shows an example to perform the quality check of a single feature, although it is possible to specify any number of <qualitycheck> tags for a single <qualitychecker>.

```
<?xml version="1.0" encoding="UTF-8"?>
<qualitychecks>
  <qualitychecker id="pixelcheck" data_type_in="lst_dl1_image">
    <qualitycheck input_field="image" type="rangethreshold" ub_warning="12" ub_alarm="14" lb_alarm="16" lb_warning="18" />
  </qualitychecker>
</qualitychecks>
```

At least two more configuration files are needed to start the software. As shown in the previous code snippet, the <qualitychecker> tag specifies a “data_type_in” attribute. The string value associated to it (e.g. “lst_dl1_image”) links to another configuration file, called “datatypes.xml”. The next code snippet shows an example:

```
<datatypes>
  <datatype id="lst_dl1_image" id_field="event_id" time_field="trigger_time" desc="DL1 real data (March 2020)">
    <field name="event_id" data_shape="scalar" data_type="int" />
    <field name="image" data_shape="array1d" data_type="float" x_dim="1855" />
    <field name="num_trig_px" data_shape="scalar" data_type="int" />
    <field name="obs_id" data_shape="scalar" data_type="int" />
    <field name="pulse_time" data_shape="array1d" data_type="float" x_dim="1855" />
    <field name="tel_id" data_shape="scalar" data_type="int" />
    <field name="trigger_time" data_shape="array1d" data_type="float" x_dim="1" />
    <field name="trigger_type" data_shape="scalar" data_type="int" />
  </datatype>
</datatypes>
```

The <datatype> tag defines the structure of a data level. Each <field> tag specify the name, the data shape and the data type of a column present in the tabular data within the HDF5 input file.

The last required configuration file is called “dqpipelines.xml”. It comprises a pipeline putting together several operations: quality checks, aggregations and transformations. The next code snippet shows an example:

```
<?xml version="1.0" encoding="UTF-8"?>
<dqpipelines>
  <dqpipeline id="lst_dl1_pipeline_1" data_type_in="lst_dl1_image" data_type_out="lst_dl1_image_transformed">
    <activity type="quality_check" id="pixelcheck" />
    <activity type="aggregation" id="lst_dl1_aggregation_step_1" />
    <activity type="transformation" id="lst_dl1_transformation_1" />
  </dqpipeline>
</dqpipelines>
```

The aggregations and transformations activity also need to be defined in their own xml configuration file. The software provides the xml templates for every configuration file that describe their syntax and the possible values.

#### 7.4.5.2 rta-dq-pipe

The rta-dq-pipe package is used to manage the parallel execution of multiple data quality analysis pipelines. There are two types of supported pipelines: analysis pipelines (class DQAnalysis) and aggregation pipelines (class DQAggregator). The idea is to split the input data stream into different directories. Each directory is processed by a data quality analysis pipeline. The results of those pipelines must be aggregated. Hence, a data qualitity aggregation pipeline can be used to perform this task. Each pipeline uses the objects of the rta-dq-lib library that contain the logic to process the data.

### 7.4.6 State machine

The following diagram shows the state machine of the SAG-DQ component.

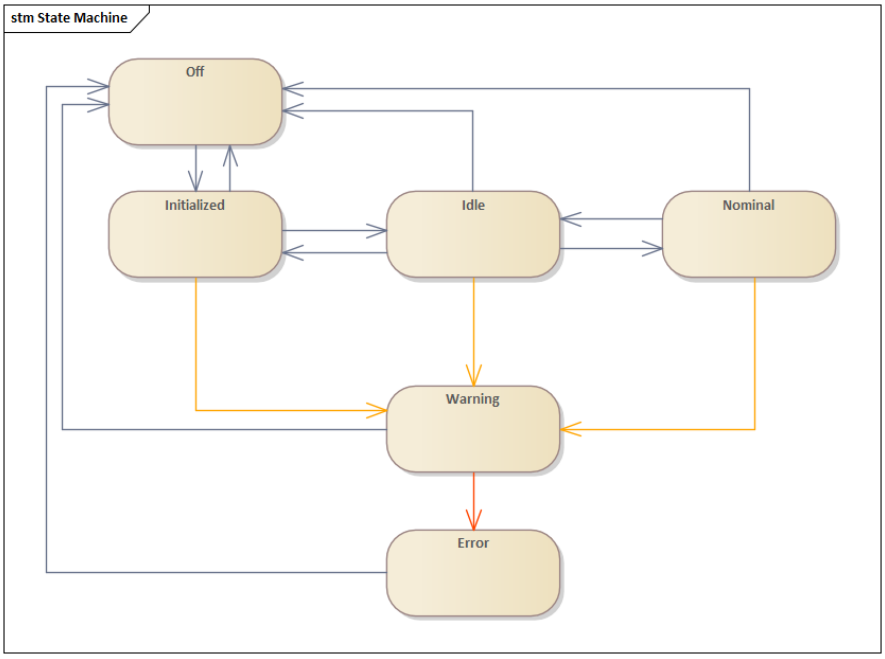


**Figure 28: SAG-DQ state diagram**

**Table 2: State transitions of the state machine of the component**

| Transition | Description |
|---|---|
| <Off → Initialized> | Initialize the pipeline and configure the system. |
| <Initialized →Idle> | Start the pipeline processes. |
| <Idle →Nominal> | Start the watchdog process that waits for new data. |
| <Initialized → Warning><br><Idle → Warning><br><Nominal → Warning> | From all operational states it is possible to go in Warning state. |
| <Warn →Error> | If the Warning state exceeds a critical level the pipeline goes in Error state. |
| <Initialized → Off><br><Idle → Off><br><Nominal → Off><br><Warning → Off><br><Error → Off> | From all states it is possible to power off the pipeline. |

### 7.4.7 Command Interface

The following code snippet shows how to start the data quality pipelines:

```
1  from rta_dq_pipe.pipe.DQPipeBuilder import DQPipeBuilder
2
3  DQPipeBuilder
4      .buildDQPipeline(confFilePath, pipeID, obsId, runId)
5      .start()
```

The first argument of the buildDQPipeline method specifies the configuration file that describes how many pipelines should be executed in parallel, their behaviour and how to submit them to Slurm. The following code snippet shows an example:

```
<?xml version="1.0" encoding="UTF-8"?>
<rta_dq_pipe_config>

    <logging dir="$RTADQPIPE_TEST_OUTPUT/output_logs/test_full_pipeline_singlethread" level="DEBUG" />

    <exec>
        <submit_command submit="True">sbatch --parsable</submit_command>
        <pipeline_commands>
            <command>source activate rta-dq-dev</command>
            <command>export DQ_PIPE_PATH=/opt/dev/rta-dq/rta-dq/rta-dq-pipe/rta_dq_pipe/</command>
            <command>export DQ_PIPE_CONF=/opt/dev/rta-dq/rta-dq-pipe-conf/</command>
            <command>export DQLIBCONF=/opt/dev/rta-dq/rta-dq-lib-conf/rta-dq-configurations-lst1/</command>
            <command>python -u $DQ_PIPE_PATH/dqpipeline_bootstrap.py \
                        --confFilePath=$DQ_PIPE_PATH/testing/test_conf/test_full_pipeline/test_fullpipeline_hdf5_in_pickle_out.xml
                        --pipelineId=#pipeid#
                        --obsId=#obsid#
                        --runId=#runid#
            </command>
        </pipeline_commands>
    </exec>

    <dqpipelines>

        <dqpipeline type="dq_analysis" id="line_1_camera" dqchain_id="camera_pipe_step1" nthreads="0">
            <input_data  type="h5" input_dirs="$RTADQPIPE_TEST_OUTPUT/line_1" file_pattern="*" reading_string="/dl1/event/telescope/images/tel_001"
                         join_with="" join_keys="" filter_column="" filter_value=""/>
            <output_data type="pickle" output_loc="$RTADQPIPE_TEST_OUTPUT/analysis_dir_out" overwrite="false"  suffix="line_1" />
        </dqpipeline>

        <dqpipeline type="dq_analysis" id="line_2_camera" dqchain_id="camera_pipe_step1" nthreads="0">
            <input_data  type="h5" input_dirs="$RTADQPIPE_TEST_OUTPUT/line_2" file_pattern="*" reading_string="/dl1/event/telescope/images/tel_001"
                         join_with="" join_keys="" filter_column="" filter_value=""/>
            <output_data type="pickle" output_loc="$RTADQPIPE_TEST_OUTPUT/analysis_dir_out" overwrite="false" suffix="line_2" />
        </dqpipeline>

        <dqpipeline type="dq_aggregator" id="camera_aggregator" dqchain_id="camera_pipe_step2" nthreads="0">
            <input_data  type="pickle" input_dirs="$RTADQPIPE_TEST_OUTPUT/analysis_dir_out" file_pattern="*"/>
            <output_data type="pickle" output_loc="$RTADQPIPE_TEST_OUTPUT/aggregator_dir_out" overwrite="true" suffix="test_out"/>
        </dqpipeline>

    </dqpipelines>

</rta_dq_pipe_config>
```

### 7.4.8 Activities

The activity diagram shown below describes the main SAG-DQ workflow during the data quality checks of different data levels generated by the SAG-RECO. The interface between

the two components are the HDF5 files generated by the SAG-RECO. The SAG-DQ periodically checks that new files are available for the analyses and then processes these files.

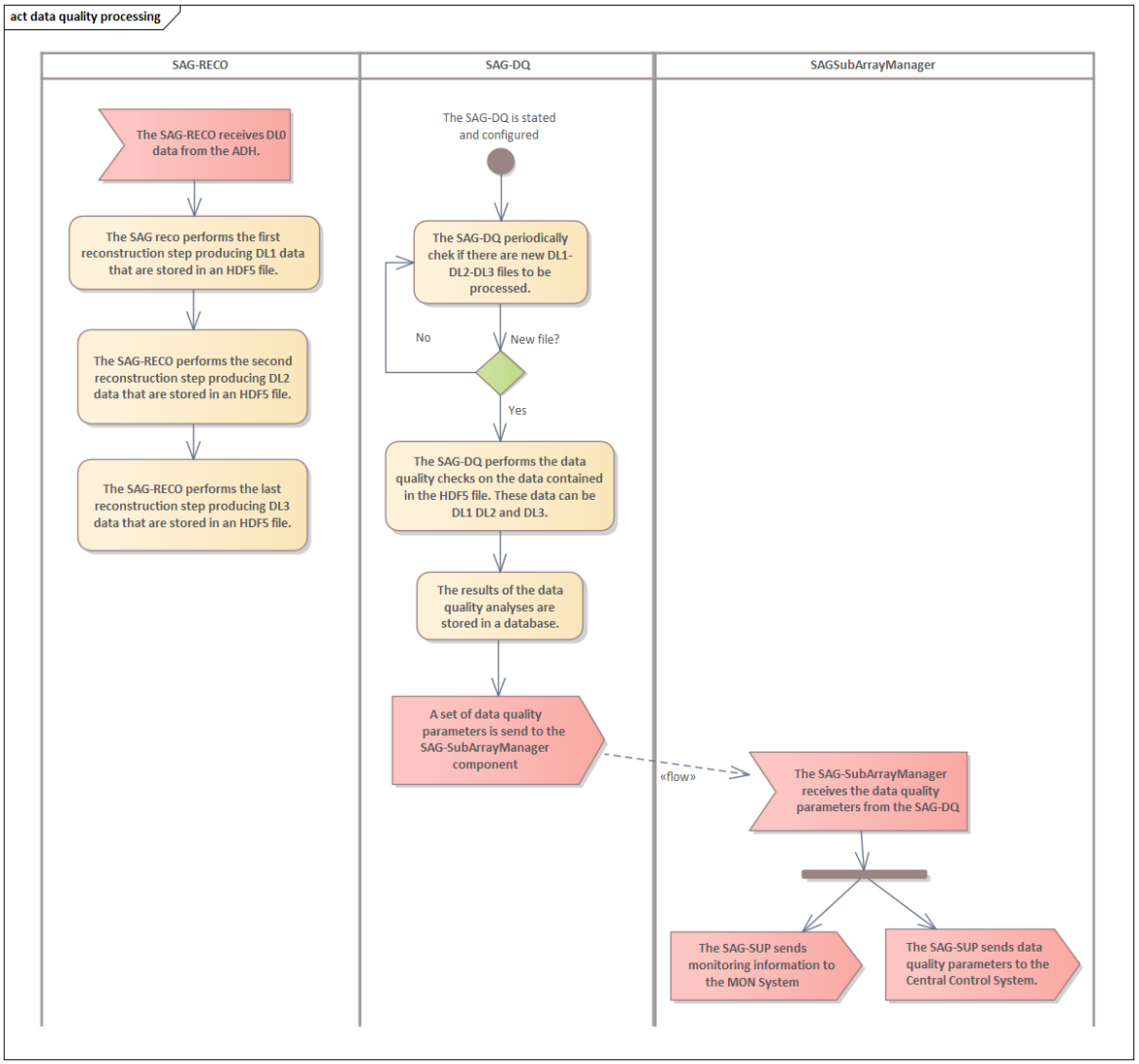


**Figure 29: Activity diagram of the SAG-DQ.**

## 7.4.9 Sequences

### 7.4.9.1 Sequence: high level data quality analysis

The following figures show the sequence diagrams of the SAG-DQ component during the startup and shutdown of the pipelines. They are defined using the UML notation.

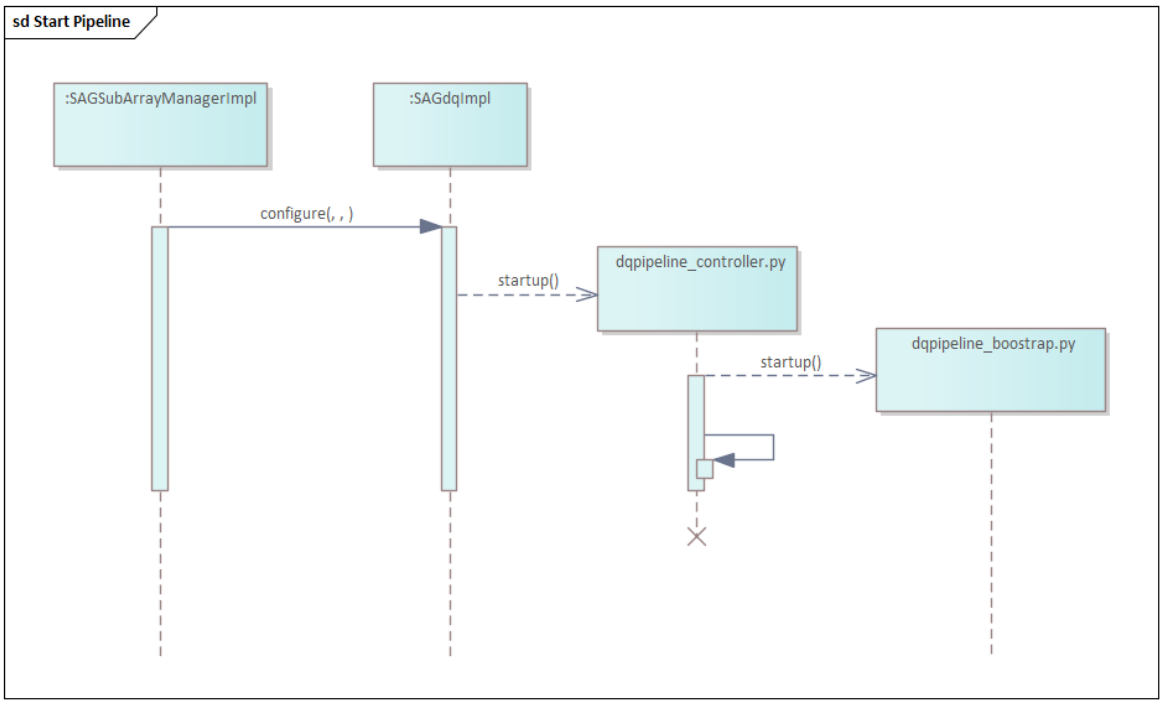


**Figure 30: Sequence diagram for the startup of the SAG-DQ pipelines.**

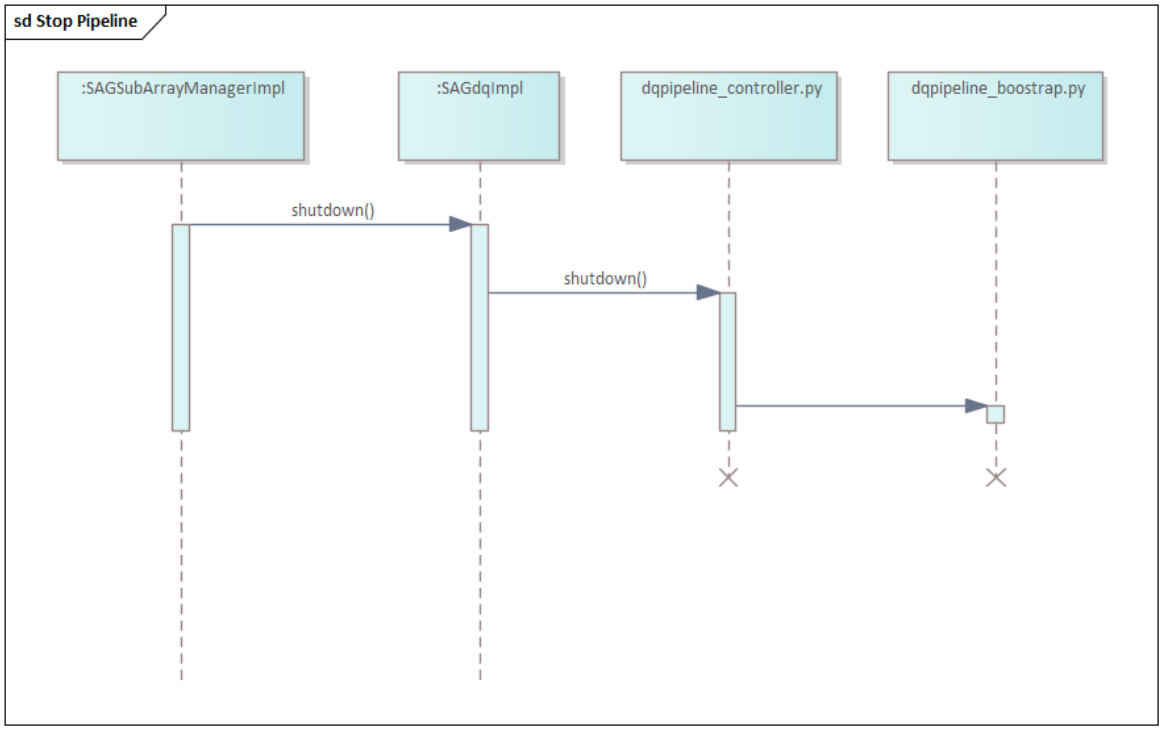


**Figure 31: Sequence diagram for the shutdown of the SAG-DQ pipelines.**

### 7.4.9.2 Sequence: data quality of camera data (DL1)

Figure 32 shows how the rta-dq-pipe and rta-dq-lib can be used to satisfy the data quality of camera data (DL1) use case.
In the diagram below, each red box represents a Python process and each process uses the rta-dq-lib to perform the data quality operations: in particular, the goal for the implementation described below, is to generate four data quality indicators:
the distribution (1D histogram) of the camera pixel values for each pixel ;
the sum of the camera pixels;
the averages of the camera pixels;
the rms of the camera pixels.

The red box is a python process that executes the class rta-dq-pipe::DQAnalysis, reads the HDF5 input file and processes it with an instance of the rta-dq-lib::DQPipeline.. The diagram shows the operational configuration of the rta-dq-lib used by the analysis process. Finally, the Python process writes the rta-dq-lib's output as pickled binary objects. The diagram shows also several "analysis processes" executing in parallel to speed up the data analysis.

The "merger process" uses the rta-dq-lib to read the pickled results written in parallel by the "analysis processes", to unpickle and merge them and write the results in a DB.

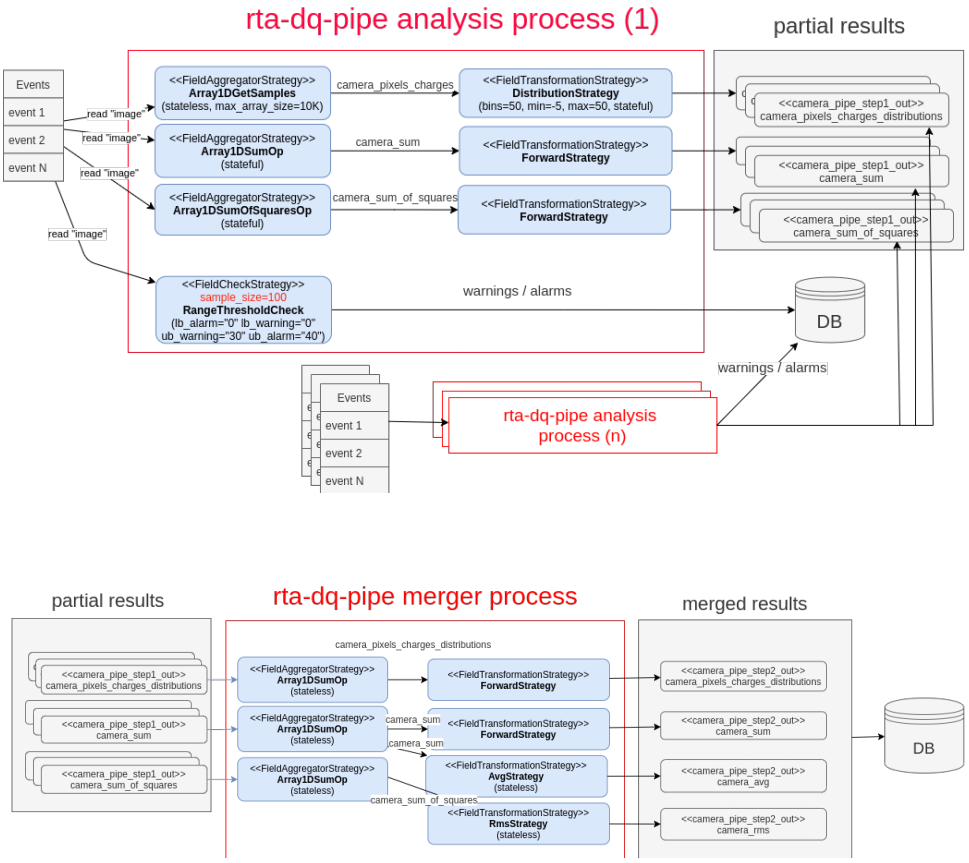


**Figure 32: 33communication diagram of the data quality of camera data (DL1) use case.**

## 7.5 Component: High-Level Analysis pipeline

This component is designed to perform scientific analyses in real-time during the observations and to detect candidate science alerts that must be sent to the Transient Handler system. The High-Level Analysis pipeline is also called SAG-SCI within this document.

### 7.5.1 General description

The document [AD2] defines a list of requirements for this component that are needed to fulfil the use cases of the SAG system. This pipeline shall execute analysis in parallels for different sub-array observing at the same time. It shall detect candidate science alerts with a latency of five seconds and with an integral sensitivity within a factor two of that required for the off-line analysis. These are some of the main requirements for the SAG-SCI pipeline.

The input data for this component is the DL3 produced by the low-level reconstruction pipeline. This data is saved into a MySQL database and triggers the SAG-SCI pipeline analysis automatically.

The results of the scientific analyses are stored into a MySQL database, and the Human Machine Interface can read this information and show it to the Science User.

The SAG-SCI pipeline is composed of the following main components: DL3 Merger, Data Model, Science Logic, Pipeline Manager, Wrappers to the Science Tools and Task Manager. This pipeline has interfaces with external systems and components:

- Low-level reconstruction pipeline as Data Source for input DL3 data
- science tools to perform analysis and to grant high flexibility
- Transient Handler to send candidate science alerts and to receive internal and external science alerts
- Monitoring system to send all the monitoring information from software processes
- Alarm system to send all the alarms during the scientific analyses
- Source catalogues to be used during the scientific analysis [AD5].

Many of these interfaces are managed through the SAGSubArrayManager component and not directly by the SAG-SCI pipeline. They are defined in more detail in the next section.

### 7.5.2 Structure

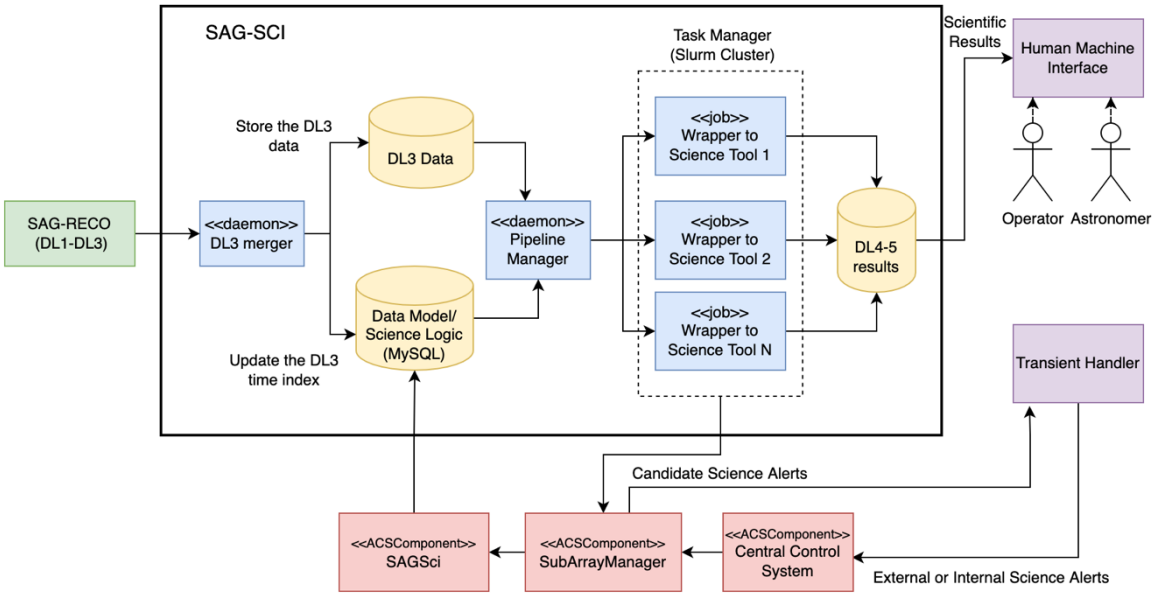


**Figure 34: High-Level pipeline architecture**

The input for the SAG-SCI is the DL3 data produced by the SAG-RECO pipeline and collected by the DL3 merger. The SAG-RECO stores the DL3 in HDF5 files, the DL3 merger can detect new files and import the information contained in these files into the DL3 database. In addition, the DL3 merger updates the Data Model with the time window contained in the imported DL3 files.

The Data Model (DM) consists of all the entities present in this framework: Data Sources, Instruments, Observations, Analysis, Science Tools, and more. The DM is very flexible and allows different analysis and Science Tools configurations. The DM is designed to satisfy the SAG-SCI requirements. A part of this DM is static and must be configured manually by the Configuration Manager before starting data acquisition. The other part is dynamically updated during the operations by the Pipeline Manager.

The Science Logic (SL) defines the set of rules used by the pipeline to know *When* and *How* to execute the configured analysis during operations. The Science Logic is implemented with MySQL triggers to improve the performances and reduce the latency. This component retrieves the analysis configuration from the static part of the DM and then updates the dynamic part. This set of rules, defined in advance, allows the pipeline to operate in real-time autonomously.

The Pipeline Manager (PM) and the Task Manager (TM) execute the analysis and manage priority between different tasks and queues. The PM component executes the analyses generated by the SL rules on the available computational resources. Since the number of these analyses can potentially be very high, the pipeline manager submits these processes to the TM component, which executes them in parallel, optimising the computational resources. The TM performs analyses on multiple computing nodes and coordinates them.

The PM is a software written in Python that periodically queries the DM to check if the triggers have created new analyses to execute. When the pipeline manager finds new analyses, it creates a new directory in the file system, queries the DM to obtain all the information necessary to execute these analyses (time window, science tools, configurations, and more) and saves this information in a file written using the Extensible Markup Language (XML). The following figure shows an example of XML files that the PM create during an analysis. Each science tool requires the development of a specific wrapper. A wrapper is a Python script that interfaces the science tools with the pipeline reading the configurations required to execute the analysis from the XML file written by the PM. The wrapper can be used to execute analysis using the same science tool with different configurations without changing the system architecture. It is possible to add new science tools without modifying the pipeline architecture; the developer needs to add a few rows into the DM database and to create the Python wrapper. This interface between the pipeline and the science tools provides great flexibility to upgrade the pipeline with new software when required.

Each wrapper is structured in order to provide an executable for the analysis and one for the post analysis. The former takes three configuration files in input: a) a file containing the observation configuration; b) a file containing the analyses configuration; c) a target file containing the spatial and spectral models of any sources to be considered for the analysis, included the background. The latter uses the updated results file produced by the analysis to store all information in the database.

Currently, the SAG-SCI consists of two science tools wrappers: one implementing analyses with a dedicated, highly optimised Real-Time Aperture PHotometry *(rtaph*) tool and another implementing the use of the *gammapy* software package.

The wrapper to the *rtaph* tool enables the computation of all photometric quantities (on counts, off counts, alpha parameter, counts excess), the Li & Ma significance, an analytic estimation for the integral flux and the counts map

The wrapper to the *gammapy* software package implements the computation of all photometric quantities (on counts, off counts, alpha parameter, count excess), the Li & Ma significance, a spectral model fitting procedure and the resulting estimation of integral flux.

Both wrappers enable analyses to be executed canonically or in stack mode, provided that results of previous connected analyses are found in the database. Stacking can be performed throughout a set of observations when operating in wobble mode. Analyses can be performed for targeted sources as well as candidate sources localised through a hotspot search within the field of view of the observation.

The parameters contained inside this XML file are configured for each science tools inside the DM as a template of the XML file that the pipeline manager shall produce at run-time. A section of the XML file contains dynamical parameters with \#@@\# tag that are dynamically defined by the pipeline for each run while the static part is customized by the Configuration Manager and can contain fixed parameters for a specific science tool. After preparing all the configuration files necessary to interface the pipeline with the science tools, the PM submits the analysis to the TM.

```
<run id="#@runid@#">
    <parameter name="AlertInfo" triggerid="#@triggerid@#" seqnum="#@seqnum@#" notice_type="#@notice_type@#"/>
    <parameter name="TimeIntervals" tmin="#@tstart@#" tmax="#@tstop@#" t0="#@tzero@#" timeunit="#@timerefunit@#"/>
    <parameter name="Energy" emin="#@emin@#" emax="#@emax@#" energyBinID="" />
    <parameter name="DirectoryList" run="#@diroutputrun@#" results="#@diroutputresults@#" fitspath="/"  />
    <parameter name="DeleteRun" value="0" />
</run>
```

**Figure 35: Example of XML configuration file.**

We implement the TM using Slurm, an open-source, fault-tolerant, and highly scalable cluster management and job scheduling system for large and small Linux clusters. The PM submits all the analysis created by the SL to Slurm. Slurm manages the resources and the execution of the analysis.
We decided to use Slurm because it can manage thousands of jobs and is easily scaled from one machine to thousands of machines. It can manage jobs in parallel and different priorities between jobs. It can suspend low priority jobs to run high priority jobs. Another key feature of Slurm is the capability to log inside a database all the information about every job. This logging system is useful to monitor the pipeline in real-time, viewing statistics about the execution of jobs. Scaling the computing power used by Slurm does not require a change in the pipeline manager software. Within the DM, it is possible to configure the priorities between jobs, which will then be respected by Slurm, and to reserve resources for a specific type of analysis.

Sky maps of DL4-DL5 are stored in FITS format in the filesystem.

When the SAG-SCI detect a Candidate Science Alerts, the related SubArrayManager sends the information to the Transient Handler using the provided interface defined in [AD6]. On the other hand, when the Transient Handler receives an External or Internal Science Alerts it sends information to the SAG-SCI. The interface is through a new Scheduling Block that is received from the Central Control using the control flow described in the sequence diagram of Sect. 7.2.8.2.

### 7.5.3 Class Diagrams

The SAG-SCI module is structured in packages, each tasked with a specific purpose. The module contains all scripts for running multiple scientific analyses in parallel, collecting the results and store them in the database. It can interrogate the database for query of previous analyses results as well. The core of the module also comprises a set of tools that reduce the DL3 taken as input in DL4 and DL5 scientific results.

#### 7.5.3.1 Package: gammapipecommon

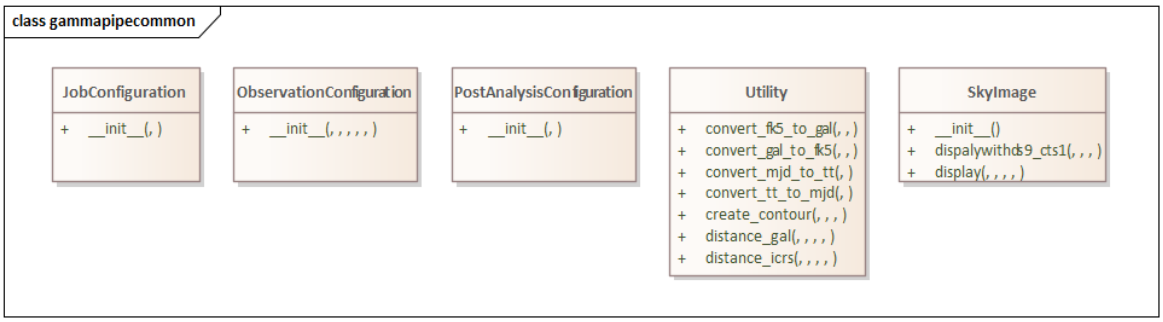


**Figure 36: Class diagram of the package gammapipecommon**

The *gammapipecommon* package contains the parent configuration classes. These classes initialise the common configuration of each job, observation and post-analysis. They store the values of common parameters, and are thus inherited by the individual configuration of each science tools wrapper. In this class there is also a collection of common utility, such as methods for the conversion of time and coordinates.

### 7.5.3.2 Package: pipelinemanager

This package contains the software that manages the pipeline to execute the scientific analyses automatically. It contains a class Utility with several methods that are used by the pipelines, a class Watcher, and a class Handler that are in charge of monitoring a directory and executing a function whenever a new file is written in the directory (e.g., when the SAG-RECO completes a new DL3 file).

Several components of this software are not classes but scripts. In future releases, the goal is to refactor this part of the software with an object-oriented pattern.

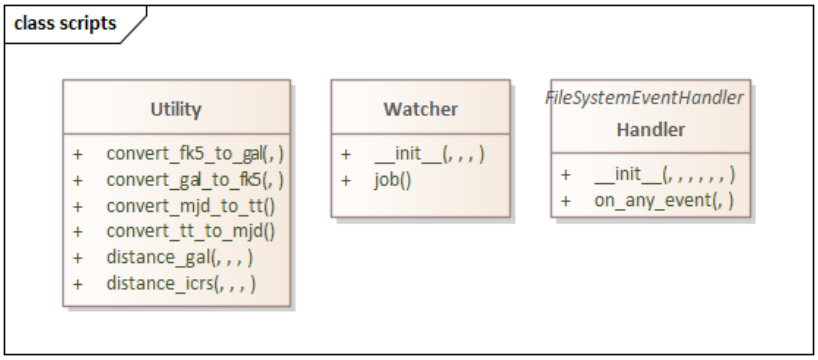


**Figure 37: Class diagram of the package pipelinemanager**

### 7.5.3.3 Package: tools


**Figure 38: Class diagram of the package tools**

The *tools* package comprises the core classes of the High-Level Analysis, alongside a collection of functions to allow for querying the database and other support utilities for the scientific analysis. The main classes are:

- *Fits*, this class contains all methods and functionalities to read, update, and create data files in FITS format. Among its utilities are the methods to update header information, compute counts map, smoothed counts map and produce empty templates;
- *IRF*, this class allows to extract data from the Instrument Response Functions in FITS format;
- *EffectiveArea*, this class specifically operates on the Effective Area extension of an Instrument Response Function in FITS format. It has methods that allow to extract the effective area for a given region in the field of view;
- *PSF*, this class specifically operates on the Point Spread Function extension of an Instrument Response Function in FITS format. Its methods allow to correct the effective area extracted within a region for the containment radius of the instrument resolution;
- *MyXml*, this class is a wrapper of the *lxml* software package. It provides methods for reading, manipulating and creating XML files that are specifically used for the SAG-SCI configuration files, target files and results outputs.
- *Photometrics,* this class comprises the algorithms for dedicated tools of aperture photometry. The algorithms supported are the “cross” and “reflection” background estimation methods, both of which can function in Wobble mode;
- *SkyImage*, this class allows the creation of counts map PNG images. It contains methods that can display DL4 files in FITS format, but also methods that can take a DL3 file as input, reduce it on memory to a DL4 and produce the resulting plot. It can produce plots of counts map as well as smoothed counts map;
- *PipeLoggerConfig* and *Singleton*, these classes allow to log information throughout the entire SAG-SCI module.

### 7.5.3.4 Package: rtaph

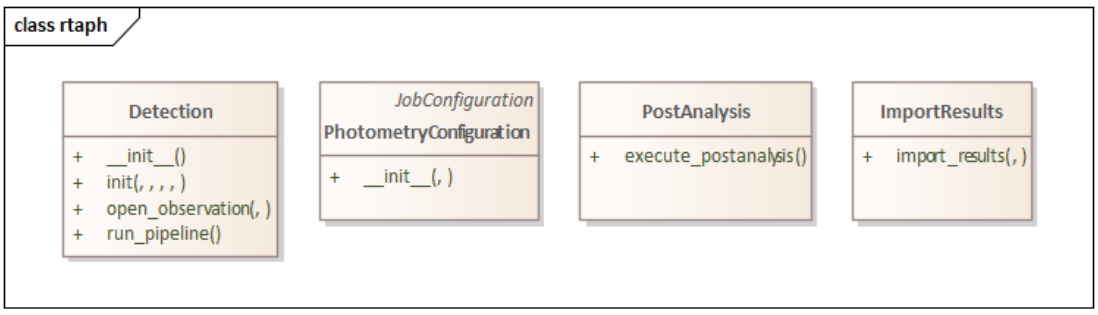


**Figure 39: Class diagram of the package rtatools**

This package is the wrapper of a dedicated Real-Time Aperture PHotrometry (*rtaph*) tool, for the SAG-SCI module. It contains all the scripts necessary to perform aperture photometry analyses, using independent tools based on the analytic Li & Ma formula. The main classes are:

- *PhotometryConfiguration*, it initialises the tool-specific analysis configuration;

- *Detection*, it contains the full length of the scientific analysis;
- *ImportResults*, it allows to store the analysis results inside the database;
- *PostAnalysis*, handles the post-analysis operations, including the database update.

#### 7.5.3.5 Package: gammapy

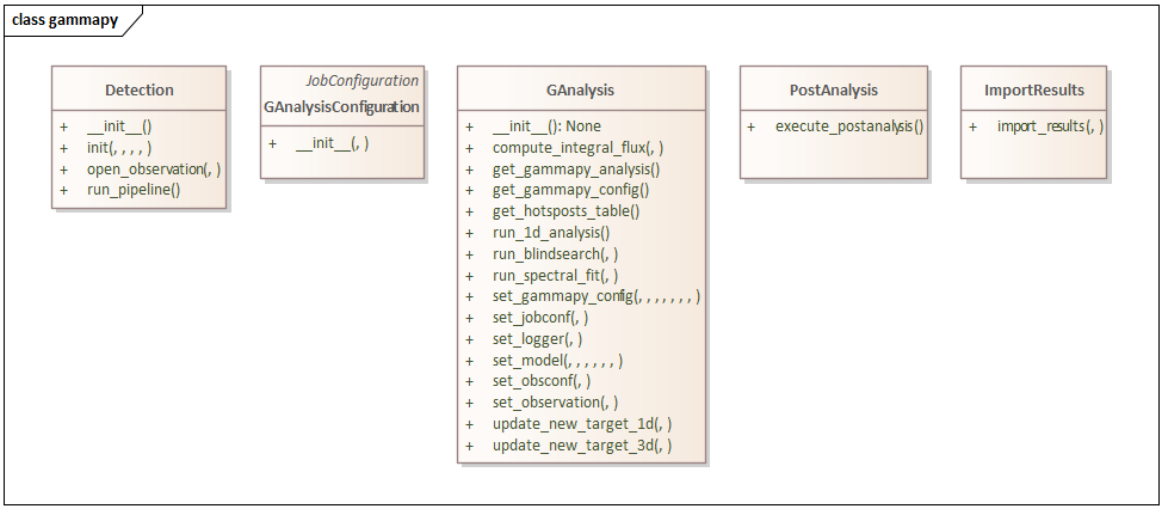


**Figure 40: Class diagram of the package gammapy**

This package is the wrapper of the *gammapy* software package, for the SAG-SCI module. It allows to perform scientific analyses both on/off and full field of view, using the science tool API. The main classes are:

- *GAnalysisConfiguration*, it initialises the tool-specific analysis configuration, including the option of running a hotspots search within the field of view before the aperture photometry;
- *GAnalysis*, it contains the *gammapy* analysis wrapping functions to perform the configuration, hotspots search, spectral fit and analysis;
- *Detection*, it contains the full length of the scientific analysis;
  *ImportResults*, it allows to store the analysis results inside the database;
  *PostAnalysis*, handles the post-analysis operations, including the database update.

### 7.5.4 General behavior

We designed the SAG-SCI to perform new analyses when one of the following two conditions occurs. The first condition occurs when the DL3 merger checks the presence of new data and updates the data index into the database to communicate to the pipeline that new data has arrived. The second condition occurs when the pipeline receives external or internal science alerts from the Transient Handler. This last condition is used to search for a counterpart of the science alert inside the data stream of the ongoing observation. Using the data index , the pipeline verifies  if the data archive contains the required time window for the analysis. If yes, the analysis is performed; otherwise, the pipeline waits until the required data arrives to start suspended analyses.

### 7.5.5 State machine

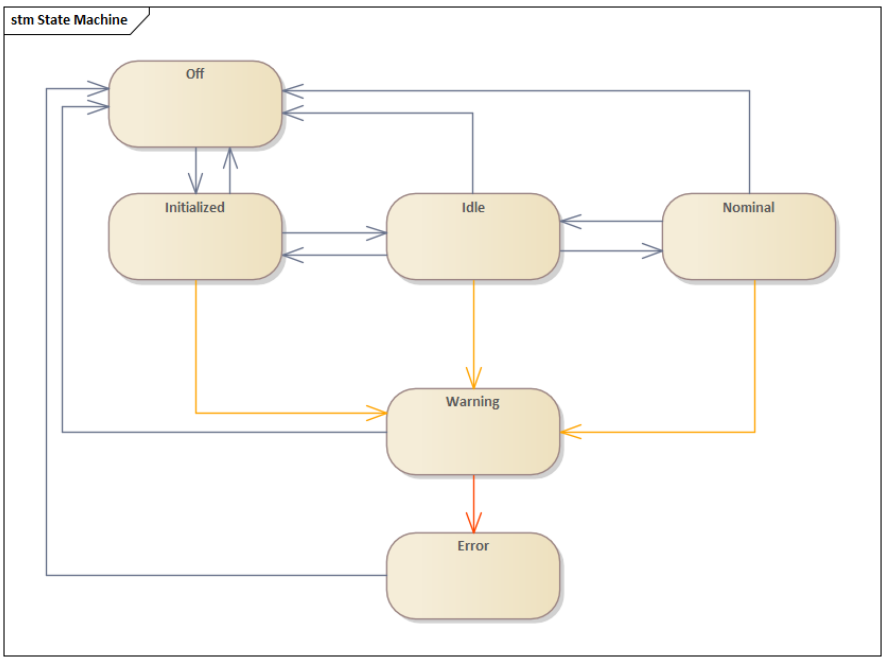


**Figure 41: SAG-SCI state diagram**

**Table 3: State transitions of the state machine of the component**

| Transition | Description |
|---|---|
| <Off → Initialized> | Initialize the pipeline and configure the system. |
| <Initialized →Idle> | Start the pipeline processes. |
| <Idle →Nominal> | Start the watchdog process that waits for new data. |
| <Initialized → Warning><br><Idle → Warning><br><Nominal → Warning> | From all operational states it is possible to go in Warning state. |
| <Warn →Error> | If the Warning state exceeds a critical level the pipeline goes in Error state. |
| <Initialized → Off><br><Idle → Off><br><Nominal → Off><br><Warning → Off><br><Error → Off> | From all states it is possible to power off the pipeline. |

### 7.5.6 Command Interface

The SAG-SCI pipeline can be started from the command line by executing python scripts that run in an infinite loop until the stop signal.

The submit_job.py and cancel_job.py manage the submission and the removal of jobs to Slurm. The update_job_status.py read the status of each job from Slurm and updates the database.

### 7.5.7 Activities

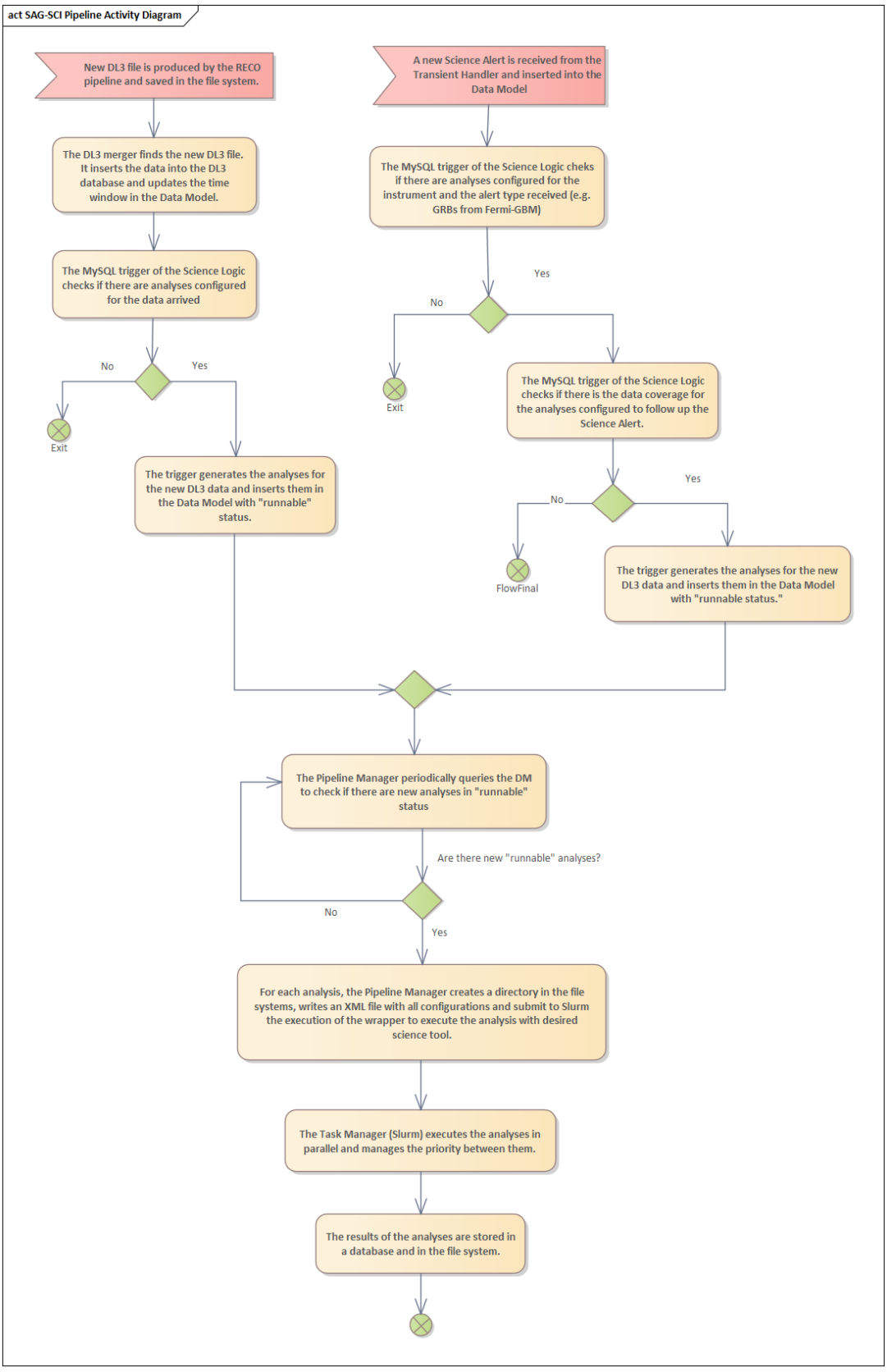


**Figure 42: SAG-SCI activity diagram**

### 7.5.8 Sequences

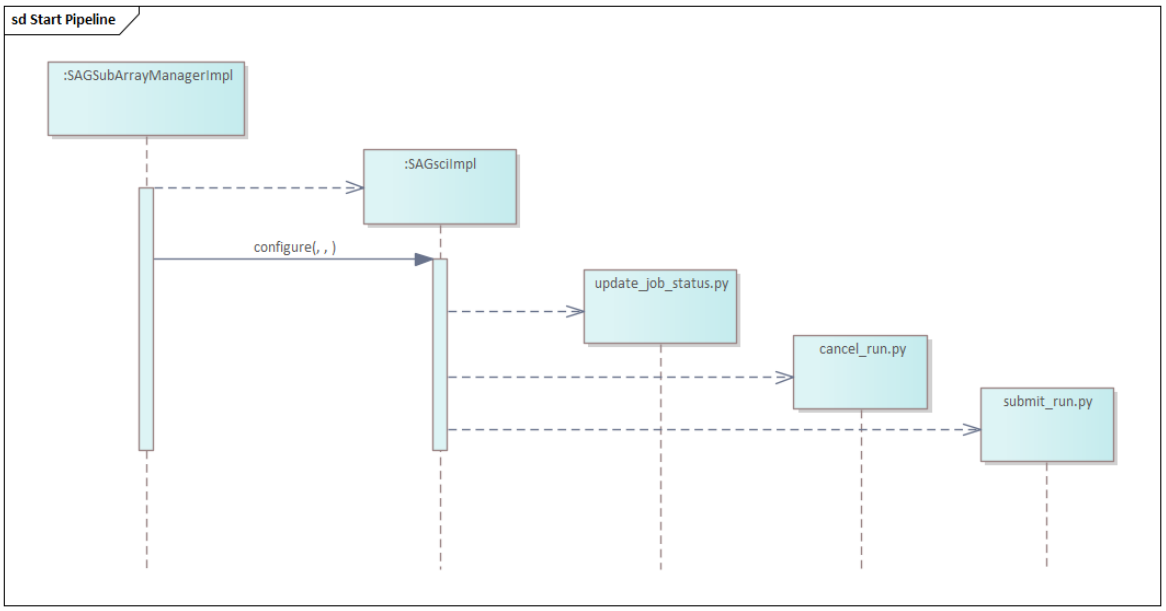


**Figure 43: SAG-SCI sequence diagram: the SubArrayManager starts the pipelines**

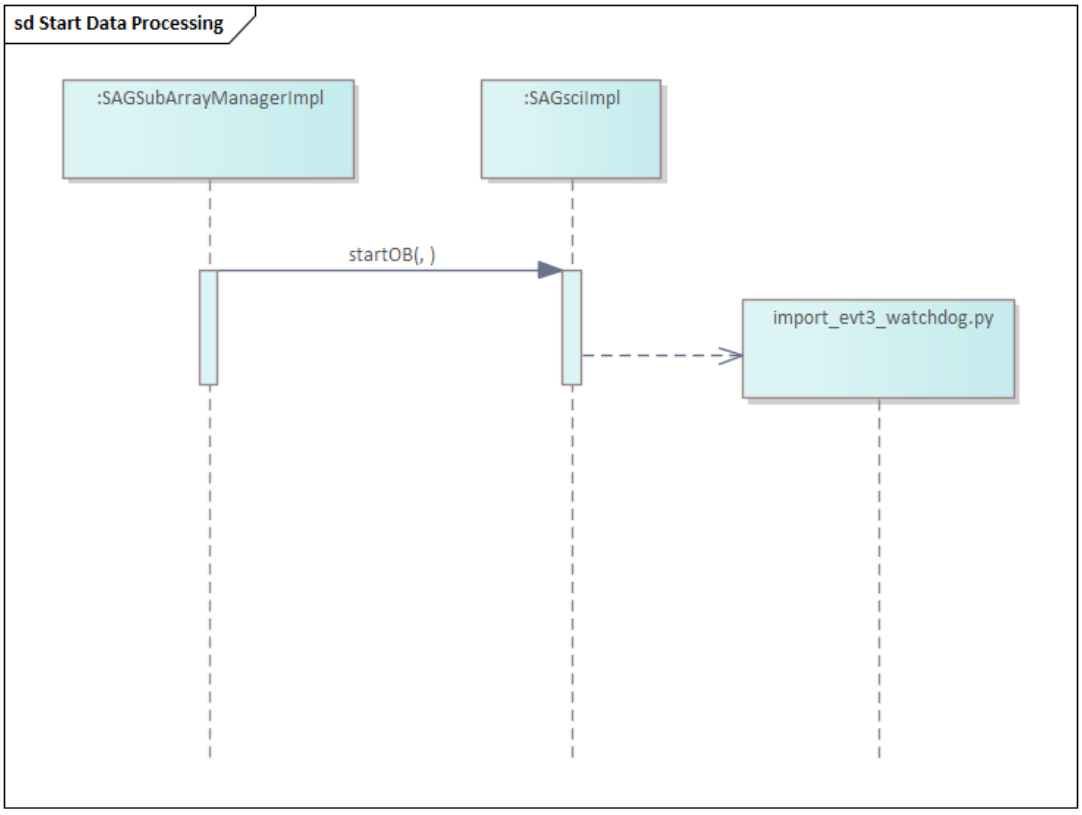


**Figure 44: SAG-SCI sequence diagram: the SubArrayManager starts the data processing**

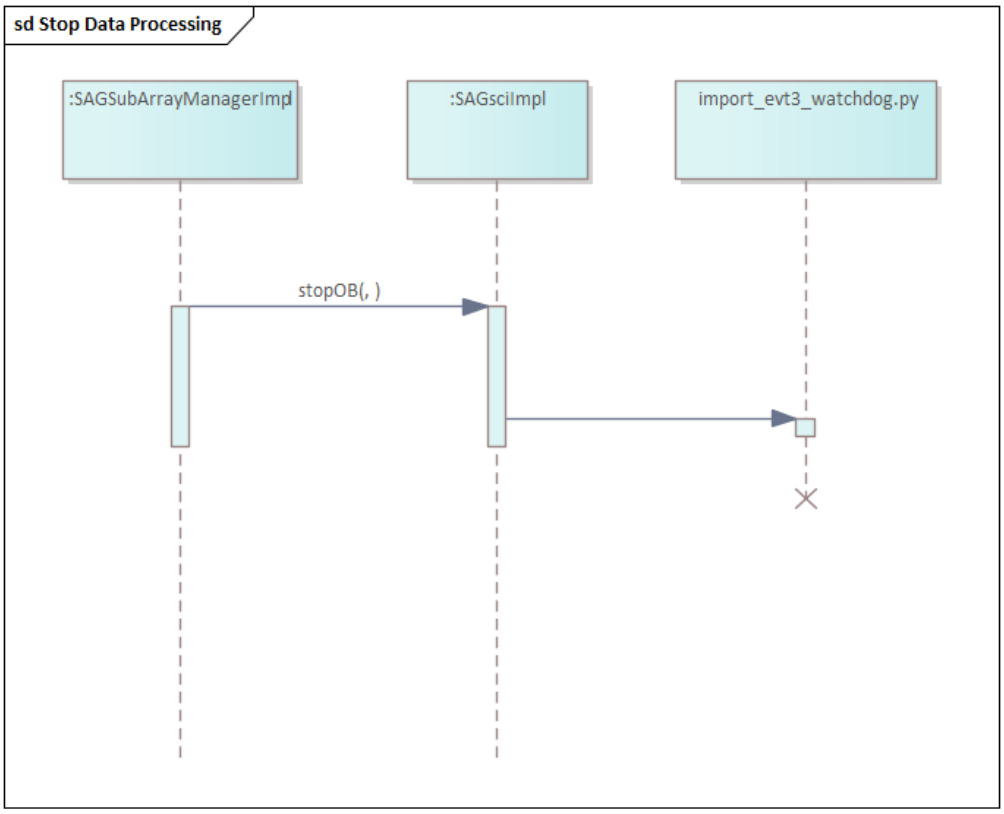


**Figure 45: SAG-SCI sequence diagram: the SubArrayManager stop the data processing**

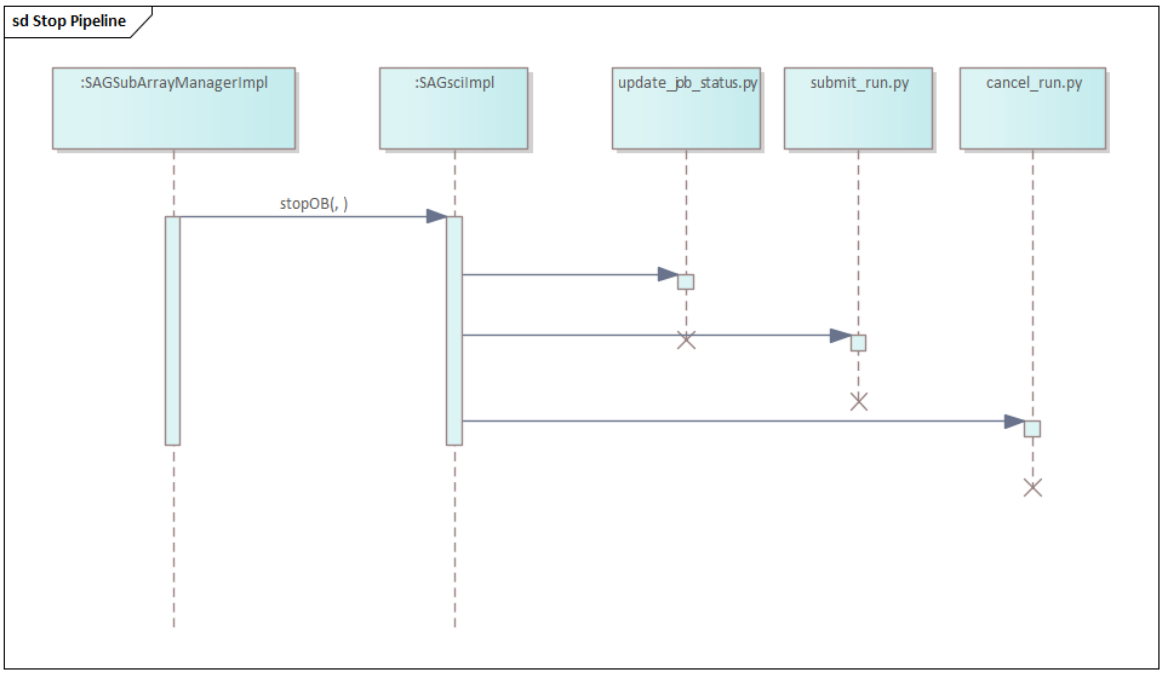


**Figure 46: SAG-SCI sequence diagram: the SubArrayManager stop the pipelines**

# 8 Data Model

## 8.1 Overall Information Flow Diagram

In this section we report the overall data flow. Some data types are reported in pseudo-code.

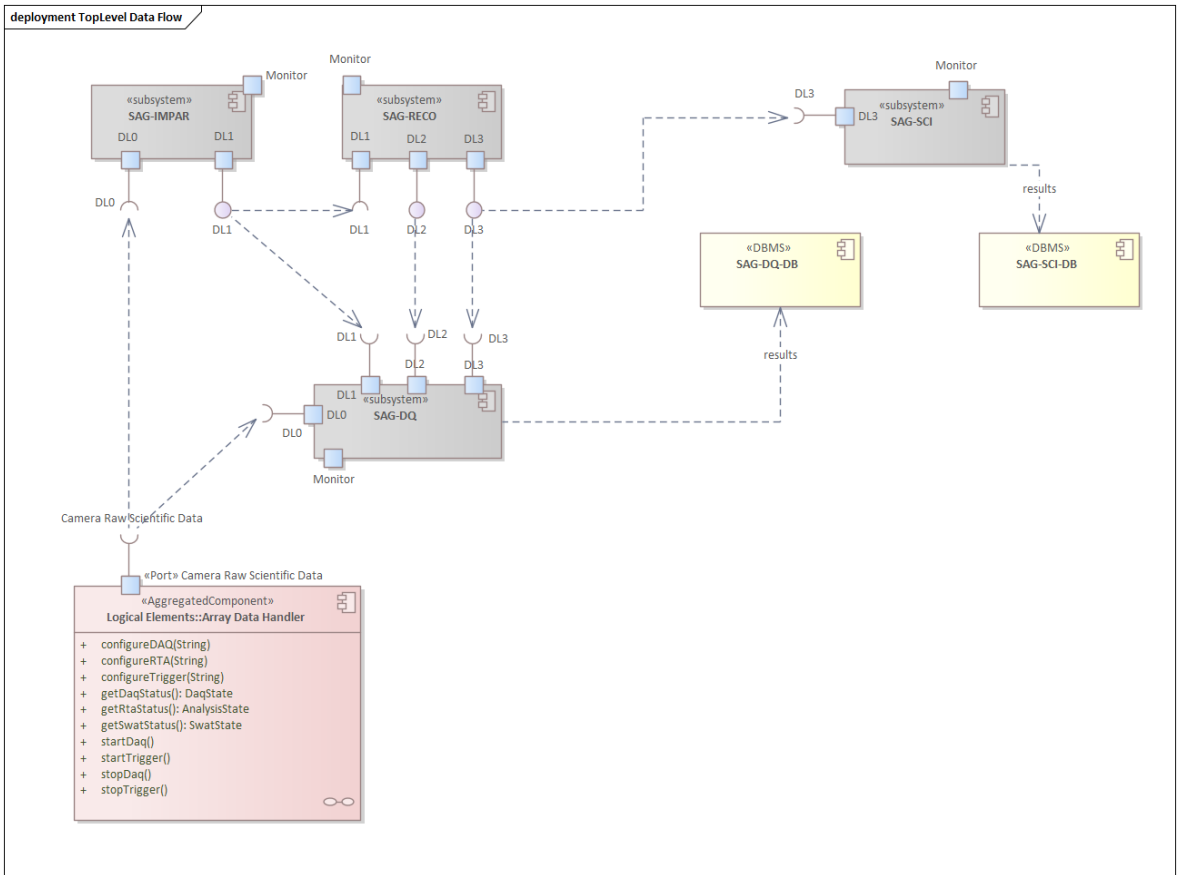


**Figure 47: Overall data flow diagram**

### 8.1.1 DL0

Since the CTAO has not yet released the DL0 data model, the specifications shown here can only be considered as preliminary.

```
///Table of waveform of the high gain signal(nbrow = 5896)
class TelescopeEvent{
        uint64 event_id
        uint16 waveformHi[Nsamp, Npix]
        uint16 waveformLo[Nsamp, Npix]
}
///All simulated Corsika events(nbrow = 10965)
class Shower{
        uint64 event_id
        uint64 obs_id
        float32 true_alt
        float32 true_az
        float32 true_core_x
        float32 true_core_y
```

```
        float32 true_energy
        float32 true_h_first_int
        uint64 true_shower_primary_id
        float32 true_x_max
}
///Configuration of the simulated events(nbrow = 1)
class ShowerConfiguration{
        uint64 atmosphere
        uint64 core_pos_mode
        float32 corsika_bunchsize
        int32 corsika_high_E_detail
        int32 corsika_high_E_model
        int32 corsika_iact_options
        int32 corsika_low_E_detail
        int32 corsika_low_E_model
        int32 corsika_version
        float32 corsika_wlen_max
        float32 corsika_wlen_min
        uint64 detector_prog_id
        int32 detector_prog_start
        int32 diffuse
        float32 energy_range_max
        float32 energy_range_min
        float32 injection_height
        float32 max_alt
        float32 max_az
        float32 max_scatter_range
        float32 max_viewcone_radius
        float32 min_alt
        float32 min_az
        float32 min_scatter_range
        float32 min_viewcone_radius
        uint64 num_showers
        uint64 obs_id
        float32 prod_site_B_declination
        float32 prod_site_B_inclination
        float32 prod_site_B_total
        float32 prod_site_alt
        float32 run_array_direction[Ngain]
        uint64 shower_prog_id
        int32 shower_prog_start
        uint64 shower_reuse
        int32 simtel_version
        float32 spectral_index
}
///Layout of the subarray(nbrow = 4)
class InstrumentLayout{
        string camera_type
        string name
        uint64 num_mirrors
        float32 pos_x
        float32 pos_y
        float32 pos_z
        string tel_description
        uint64 tel_id
        string type
        uint64 type_id
}
///Geometry of LSTCam(nbrow = Npix)
class CameraGeomatry{
        float32 pix_area
        uint64 pix_id
        float32 pix_x
        float32 pix_y
}
///Reference shape of LSTCam(nbrow = Nrefsamp)
class CameraReadout{
        float32 reference_pulse_sample_time
        float32 reference_pulse_shape_channel0
        float32 reference_pulse_shape_channel1
}
```

### 8.1.2 DL1

Since the CTAO has not yet released the DL1 data model, the specifications shown here can only be considered as preliminary.

```
///(nbrow = 5896)
class ImageParameter{
        float32 hillas_x
        float32 hillas_y
        float32 hillas_phi
        float32 hillas_width
        float32 hillas_length
        float32 hillas_intensity
        float32 morphology_num_pixels
        float32 hillas_skewness
        float32 hillas_r
        float32 hillas_kurtosis
        float32 hillas_psi
        float32 is_good_event
        float32 timing_slope
        float32 timing_intercept
        float32 leakage_pixels_width_1
        float32 leakage_pixels_width_2
        float32 leakage_intensity_1
        float32 leakage_intensity_2
        float32 morphology_num_islands
        float32 total_intensity
float32 telAltitude
float32 telAzimuth
        uint64 event_id
        uint64 obs_id
        uint64 tel_id
float64 triggerTime
uint64 eventType
int32 quality
}
///(nbrow = 5896)
class CalibratedImage{
        uint64 event_id
        uint64 obs_id
        uint64 tel_id
float64 triggerTime
uint64 eventType
        float32 image[Npix]
        float32 peak_time[Npix]
}
///Trigger information(nbrow = 10965)
class Trigger{
        uint64 event_id
        uint64 event_type
        uint64 obs_id
        float64 time
uint32 timestampSecond
uint32 timestampQns
}
///All simulated Corsika events(nbrow = 10965)
class Shower{
        uint64 event_id
        uint64 obs_id
        float32 true_alt
        float32 true_az
        float32 true_core_x
        float32 true_core_y
        float32 true_energy
        float32 true_h_first_int
        uint64 true_shower_primary_id
        float32 true_x_max
}
///Configuration of the simulated events(nbrow = 1)
class ShowerConfiguration{
        uint64 atmosphere
        uint64 core_pos_mode
        float32 corsika_bunchsize
```

```
        int32 corsika_high_E_detail
        int32 corsika_high_E_model
        int32 corsika_iact_options
        int32 corsika_low_E_detail
        int32 corsika_low_E_model
        int32 corsika_version
        float32 corsika_wlen_max
        float32 corsika_wlen_min
        uint64 detector_prog_id
        int32 detector_prog_start
        int32 diffuse
        float32 energy_range_max
        float32 energy_range_min
        float32 injection_height
        float32 max_alt
        float32 max_az
        float32 max_scatter_range
        float32 max_viewcone_radius
        float32 min_alt
        float32 min_az
        float32 min_scatter_range
        float32 min_viewcone_radius
        uint64 num_showers
        uint64 obs_id
        float32 prod_site_B_declination
        float32 prod_site_B_inclination
        float32 prod_site_B_total
        float32 prod_site_alt
        float32 run_array_direction[Ngain]
        uint64 shower_prog_id
        int32 shower_prog_start
        uint64 shower_reuse
        int32 simtel_version
        float32 spectral_index
}
///Layout of the subarray(nbrow = 4)
class InstrumentLayout{
        string camera_type
        string name
        uint64 num_mirrors
        float32 pos_x
        float32 pos_y
        float32 pos_z
        string tel_description
        uint64 tel_id
        string type
        uint64 type_id
}
///Geometry of LSTCam(nbrow = Npix)
class CameraGeomatry{
        float32 pix_area
        uint64 pix_id
        float32 pix_x
        float32 pix_y
}
///Reference shape of LSTCam(nbrow = Nrefsamp)
class CameraReadout{
        float32 reference_pulse_sample_time
        float32 reference_pulse_shape_channel0
        float32 reference_pulse_shape_channel1
}
```

### 8.1.3 DL2

Since the CTAO has not yet released the DL2 data model, the specifications shown here can only be considered as preliminary.

```
class ImageParameter{
        float32 hillas_x
        float32 hillas_y
        float32 hillas_phi
        float32 hillas_width
```

```
        float32 hillas_length
        float32 hillas_intensity
        float32 morphology_num_pixels
        float32 hillas_skewness
        float32 hillas_r
        float32 hillas_kurtosis
        float32 hillas_psi
        float32 is_good_event
        float32 timing_slope
        float32 timing_intercept
        float32 leakage_pixels_width_1
        float32 leakage_pixels_width_2
        float32 leakage_intensity_1
        float32 leakage_intensity_2
        float32 morphology_num_islands
        float32 total_intensity
       float32 telAltitude
       float32 telAzimuth
       Float32 log_reco_energy
       Float32 reco_energy
       Float32 reco_disp_dx
       Float32 reco_disp_dy
       Float32 reco_src_x
       Float32 reco_src_y
       Float32 reco_alt
       Float32 reco_az
       Float32 gammaness
        uint64 event_id
        uint64 obs_id
        uint64 tel_id
       float64 triggerTime
       uint64 eventType
       int32 quality
}///Trigger information(nbrow = 10965)
class Trigger{
        uint64 event_id
        uint64 event_type
        uint64 obs_id
        float64 time
uint32 timestampSecond
uint32 timestampQns
}
///All simulated Corsika events(nbrow = 10965)
class Shower{
        uint64 event_id
        uint64 obs_id
        float32 true_alt
        float32 true_az
        float32 true_core_x
        float32 true_core_y
        float32 true_energy
        float32 true_h_first_int
        uint64 true_shower_primary_id
        float32 true_x_max
}
///Configuration of the simulated events(nbrow = 1)
class ShowerConfiguration{
        uint64 atmosphere
        uint64 core_pos_mode
        float32 corsika_bunchsize
        int32 corsika_high_E_detail
        int32 corsika_high_E_model
        int32 corsika_iact_options
        int32 corsika_low_E_detail
        int32 corsika_low_E_model
        int32 corsika_version
        float32 corsika_wlen_max
        float32 corsika_wlen_min
        uint64 detector_prog_id
        int32 detector_prog_start
        int32 diffuse
        float32 energy_range_max
        float32 energy_range_min
        float32 injection_height
        float32 max_alt
```

```
        float32 max_az
        float32 max_scatter_range
        float32 max_viewcone_radius
        float32 min_alt
        float32 min_az
        float32 min_scatter_range
        float32 min_viewcone_radius
        uint64 num_showers
        uint64 obs_id
        float32 prod_site_B_declination
        float32 prod_site_B_inclination
        float32 prod_site_B_total
        float32 prod_site_alt
        float32 run_array_direction[Ngain]
        uint64 shower_prog_id
        int32 shower_prog_start
        uint64 shower_reuse
        int32 simtel_version
        float32 spectral_index
}
///Layout of the subarray(nbrow = 4)
class InstrumentLayout{
        string camera_type
        string name
        uint64 num_mirrors
        float32 pos_x
        float32 pos_y
        float32 pos_z
        string tel_description
        uint64 tel_id
        string type
        uint64 type_id
}
///Geometry of LSTCam(nbrow = Npix)
class CameraGeomatry{
        float32 pix_area
        uint64 pix_id
        float32 pix_x
        float32 pix_y
}
///Reference shape of LSTCam(nbrow = Nrefsamp)
class CameraReadout{
        float32 reference_pulse_sample_time
        float32 reference_pulse_shape_channel0
        float32 reference_pulse_shape_channel1
}
```

### 8.1.4 DL3

Since the CTAO has not yet released the DL3 data model, the specifications shown here can only be considered as preliminary.

General Structure :

| Index | Extension | Type | Dimension |
|---|---|---|---|
| 0 | Primary | Image | 0 |
| 1 | EVENTS | Binary | 5 cols X 211 rows |
| 2 | GTI | Binary | 2 cols X 1 rows |
| 3 | POINTING | Binary | 0 cols X 0 rows |
| 4 | EFFECTIVE AREA | Binary | 5 cols X 1 rows |
| 5 | ENERGY DISPERSION | Binary | 7 cols X 1 rows |

Events Header:

```
XTENSION= 'BINTABLE'           / binary table extension
BITPIX  =                    8 / array data type
NAXIS   =                    2 / number of array dimensions
NAXIS1  =                   40 / length of dimension 1
NAXIS2  =                  211 / length of dimension 2
PCOUNT  =                    0 / number of group parameters
GCOUNT  =                    1 / number of groups
TFIELDS =                    5 / number of table fields
HDUDOC  = 'https://github.com/open-gamma-ray-astro/gamma-astro-data-formats'
HDUVERS = '0.2     '
HDUCLASS= 'GADF    '
HDUCLAS1= 'EVENTS  '
OBS_ID  =                 2990
DATE_OBS= '2020-11-22'
TSTART  =    1606014661.090049
TSTOP   =    1606014669.470045
MJDREFI = '40587   '
MJDREFF = '0       '
TIMEUNIT= 's       '
TIMESYS = 'UTC     '
OBJECT  = 'None    '
OBS_MODE= 'WOBBLE  '
N_TELS  =                    1
TELLIST = 'LST-1   '
RA_PNT  =    83.97853411061054
DEC_PNT =    22.24453703366607
ALT_PNT =            80.837301
AZ_PNT  =           226.315166
RA_OBJ  =    83.97853411061054
DEC_OBJ =    22.24453703366607
FOVALIGN= 'ALTAZ   '
ONTIME  =    8.379996538162231
DEADC   =   0.9321401938851603
LIVETIME=    7.811331597839515
EXTNAME = 'EVENTS  '           / extension name
TTYPE1  = 'EVENT_ID'
TFORM1  = 'D       '
TUNIT1  = ''
TTYPE2  = 'TIME    '
TFORM2  = 'D       '
TUNIT2  = 's       '
TTYPE3  = 'RA      '
TFORM3  = 'D       '
TUNIT3  = 'deg     '
TTYPE4  = 'DEC     '
TFORM4  = 'D       '
TUNIT4  = 'deg     '
TTYPE5  = 'ENERGY  '
TFORM5  = 'D       '
TUNIT5  = 'TeV     '
END
```

Events Table:

| | EVENT_ID | TIME | RA | DEC | ENERGY |
|---|---|---|---|---|---|
| Select | D | D | D | D | D |
| All | | s | deg | deg | TeV |
| Invert | Modify | Modify | Modify | Modify | Modify |

GTI Header:

```
XTENSION= 'BINTABLE'           / binary table extension
BITPIX  =                    8 / array data type
NAXIS   =                    2 / number of array dimensions
NAXIS1  =                   16 / length of dimension 1
NAXIS2  =                    1 / length of dimension 2
PCOUNT  =                    0 / number of group parameters
GCOUNT  =                    1 / number of groups
TFIELDS =                    2 / number of table fields
HDUDOC  = 'https://github.com/open-gamma-ray-astro/gamma-astro-data-formats'
HDUVERS = '0.2     '
HDUCLASS= 'GADF    '
HDUCLAS1= 'GTI     '
OBS_ID  =                 2990
MJDREFI = '40587   '
MJDREFF = '0       '
TIMESYS = 'UTC     '
TIMEUNIT= 's       '
EXTNAME = 'GTI     '           / extension name
TTYPE1  = 'START   '
TFORM1  = 'D       '
TUNIT1  = ''
TTYPE2  = 'STOP    '
TFORM2  = 'D       '
TUNIT2  = ''
END
```

GTI Table:

| | START | STOP |
|---|---|---|
| Select | D | D |
| All | | |
| Invert | Modify | Modify |

Pointing Header:

```
XTENSION= 'BINTABLE'           / binary table extension
BITPIX  =                    8 / array data type
NAXIS   =                    2 / number of array dimensions
NAXIS1  =                    0 / length of dimension 1
NAXIS2  =                    0 / length of dimension 2
PCOUNT  =                    0 / number of group parameters
GCOUNT  =                    1 / number of groups
TFIELDS =                    0 / number of table fields
HDUDOC  = 'https://github.com/open-gamma-ray-astro/gamma-astro-data-formats'
HDUVERS = '0.2     '
HDUCLASS= 'GADF    '
HDUCLAS1= 'POINTING'
OBS_ID  =                 2990
RA_PNT  =    83.97853411061054
DEC_PNT =    22.24453703366607
ALT_PNT =            80.837301
AZ_PNT  =           226.315166
TIME    =    1606014661.090049
EXTNAME = 'POINTING'           / extension name
END
```

Effective Area Header:

```
XTENSION= 'BINTABLE'           / binary table extension
BITPIX  =                    8 / array data type
NAXIS   =                    2 / number of array dimensions
NAXIS1  =                 1808 / length of dimension 1
NAXIS2  =                    1 / length of dimension 2
PCOUNT  =                    0 / number of group parameters
GCOUNT  =                    1 / number of groups
TFIELDS =                    5 / number of table fields
CREATOR = 'pyirf v0.4.1'
HDUDOC  = 'https://github.com/open-gamma-ray-astro/gamma-astro-data-formats'
HDUVERS = '0.2     '
HDUCLASS= 'GADF    '
HDUCLAS1= 'RESPONSE'
HDUCLAS2= 'EFF_AREA'
HDUCLAS3= 'FULL-ENCLOSURE'
HDUCLAS4= 'AEFF_2D '
DATE    = '2021-04-30 12:24:53.521'
TELESCOP= 'CTA-N   '
INSTRUME= 'LST-1   '
FOVALIGN= 'RADEC   '
GH_CUT  =                  0.6
EXTNAME = 'EFFECTIVE AREA'     / extension name
TTYPE1  = 'ENERG_LO'
TFORM1  = '21D     '
TUNIT1  = 'TeV     '
TDIM1   = '(21)    '
TTYPE2  = 'ENERG_HI'
TFORM2  = '21D     '
TUNIT2  = 'TeV     '
TDIM2   = '(21)    '
TTYPE3  = 'THETA_LO'
TFORM3  = '8D      '
TUNIT3  = 'deg     '
TDIM3   = '(8)     '
TTYPE4  = 'THETA_HI'
TFORM4  = '8D      '
TUNIT4  = 'deg     '
TDIM4   = '(8)     '
TTYPE5  = 'EFFAREA '
TFORM5  = '168D    '
TUNIT5  = 'm2      '
TDIM5   = '(21,8)  '
END
```

Effective Area Table:

| | ENERG_LO | ENERG_HI | THETA_LO | THETA_HI | EFFAREA |
|---|---|---|---|---|---|
| Select | 21D | 21D | 8D | 8D | 168D |
| All | TeV | TeV | deg | deg | m2 |
| Invert | Modify | Modify | Modify | Modify | Modify |

Energy Dispersion Header:

```
XTENSION= 'BINTABLE'           / binary table extension
BITPIX  =                    8 / array data type
NAXIS   =                    2 / number of array dimensions
NAXIS1  =                41264 / length of dimension 1
NAXIS2  =                    1 / length of dimension 2
PCOUNT  =                    0 / number of group parameters
GCOUNT  =                    1 / number of groups
TFIELDS =                    7 / number of table fields
CREATOR = 'pyirf v0.4.1'
HDUDOC  = 'https://github.com/open-gamma-ray-astro/gamma-astro-data-formats'
HDUVERS = '0.2     '
HDUCLASS= 'GADF    '
HDUCLAS1= 'RESPONSE'
HDUCLAS2= 'EDISP   '
HDUCLAS3= 'FULL-ENCLOSURE'
HDUCLAS4= 'EDISP_2D'
DATE    = '2021-04-30 12:24:53.559'
TELESCOP= 'CTA-N   '
INSTRUME= 'LST-1   '
FOVALIGN= 'RADEC   '
GH_CUT  =                  0.6
EXTNAME = 'ENERGY DISPERSION'  / extension name
TTYPE1  = 'ENERG_LO'
TFORM1  = '21D     '
TUNIT1  = 'TeV     '
TDIM1   = '(21)    '
TTYPE2  = 'ENERG_HI'
TFORM2  = '21D     '
TUNIT2  = 'TeV     '
TDIM2   = '(21)    '
TTYPE3  = 'MIGRA_LO'
TFORM3  = '30D     '
TUNIT3  = ''
TDIM3   = '(30)    '
TTYPE4  = 'MIGRA_HI'
TFORM4  = '30D     '
TUNIT4  = ''
TDIM4   = '(30)    '
TTYPE5  = 'THETA_LO'
TFORM5  = '8D      '
TUNIT5  = 'deg     '
TDIM5   = '(8)     '
TTYPE6  = 'THETA_HI'
TFORM6  = '8D      '
TUNIT6  = 'deg     '
TDIM6   = '(8)     '
TTYPE7  = 'MATRIX  '
TFORM7  = '5040D   '
TUNIT7  = ''
TDIM7   = '(21,30,8)'
END
```

Energy Dispersion Table:

| | ENERG_LO | ENERG_HI | MIGRA_LO | MIGRA_HI | THETA_LO | THETA_HI |
|---|---|---|---|---|---|---|
| Select | 21D | 21D | 30D | 30D | 8D | 8D |
| All | TeV | TeV | | | deg | deg |
| Invert | Modify | Modify | Modify | Modify | Modify | Modify |

### 8.1.5 Monitoring point

The SAG sends monitoring points to the MON system through an ACS notification channel. The data model of these monitoring points is defined in the diagram below.

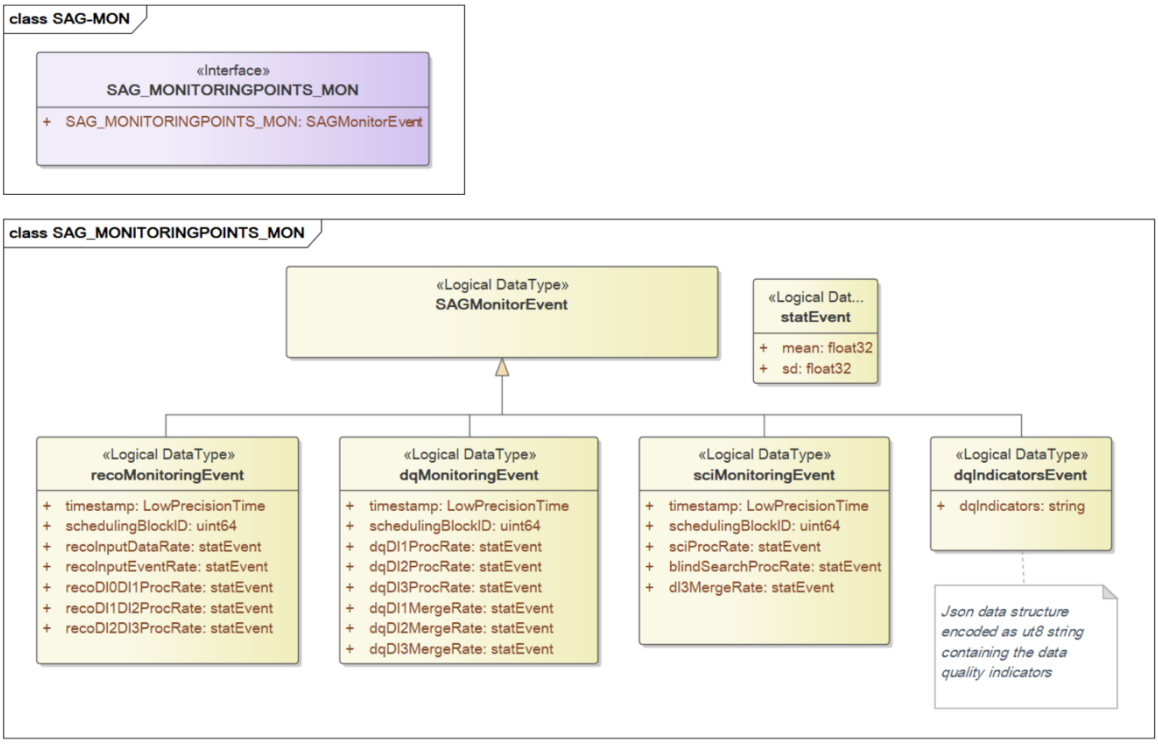


**Figure 48: Data model of the monitoring points.**

## 8.2 Component: SAG-SUP

### 8.2.1 Configuration

The SAG-SUP receives the configurations from the Central Control System and Resource Manager when the ACS components are started.

### 8.2.2 Run-Time Setup

During the operations the configurations are received through the Scheduling Block sent by the CC. The data model of the Scheduling Block is described in [AD7].

### 8.2.3 Operations Logging

The logging infrastructure is provided by the ACADA MON sub-system. The SAG-SUP ACS components generate ACS logs that are collected by the MON sub-system.

**Table 4: Logging information component SAG-SUP**

| Log name | Log level | Condition to trigger | Description |
|---|---|---|---|
| ACS components logs | TBD | periodically | |

### 8.2.4 Alarms Triggered

This section lists the alarm generated by the SAG-SUP during the operations.

**Table 5: Alarm triggered component SAG-SUP**

| Alarm name | Alarm Family | Alarm Criticallity | Description |
|---|---|---|---|
| Unable to start the system | | | The sub-system cannot be started and this is reported as an Alarm. |
| Unable to get the status of ACS components | | | The sub-system is not able to get the status of ACS components |
| One of the ACS component is in the Alarm state | | | |

### 8.2.5 Monitoring Points

The SAG-SUP collects and sends to the MON system a list of monitoring points shown in the table below.

**Table 6: Monitoring Points for Component SAG-SUP**

| Point name | Description | Type | Sample rate | Alarm Condition |
|---|---|---|---|---|
| Status of the ACS components | The staus of all ACS components | state | 1 Hz | |
| Status of the pipelines | The status of SAG-RECO, SAG-DQ and SAG-SCI pipelines for each SB. | state | 1Hz | |

### 8.2.6 Input Data

The SAG-SUP component does not receive input data but only configurations.

### 8.2.7 Output Data

The SAG-SUP component does not produce output data.

### 8.2.8 Internal Data

The internal data are sent and received using the ACS components. These interfaces are defined in the IDL.

## 8.3 Component: Image Parameter Extractor and Low-Level Reconstruction Pipeline

### 8.3.1 Configuration

*DL0 -> DL1 configuration :*

```
# Standard configuration for offline and online analysis
threshold_bad_adc: 4094
threshold_bad_ped: 100000

# Integration (GLOBAL_PEAK_INTEGRATOR, NEIGHBOUR_PEAK_INTEGRATOR, LOCAL_PEAK_INTEGRATOR)
integration_type: GLOBAL_PEAK_INTEGRATOR
nb_slice_before_peak: 3
nb_slice_window: 7

# Useful on real data. Set to 3 and 2, respectively, for LST-1
nb_first_slice_to_reject: 3
nb_last_slice_to_reject: 2

# Cut on calibrated/integrated signal
hillas_threshold_signal_tel: 30

# Maximum integrated signal in the camera (60000 is fine to remove the flatfield inside real data)
hillas_threshold_signal_tel_max: 1000000

# Cleaning configuration (TAILCUT, WAVELET)
cleaning_type: TAILCUT
hillas_threshold_center: 8
hillas_threshold_neighbours: 4
min_number_neighbour_pixels: 2
keep_isolated_pixel: false

wavelet_threshold: 4

min_selected_pixel: 2

################################################
# Stream configuration for hiperta_stream_r0_dl1 #
################################################

# Used by hiperta_stream_r0_dl1
nb_event_average_ped: 10000
nb_event_per_hdf5: 10000
keep_rejected_event: true
stream_read_zfit: false  # False only when we do stream with .hdf5 files
stream_real_data: true   # False only when we stream simulated data

```

DL1 -> DL2 configuration :

```
"events_filters": {
  "intensity": [0, Infinity],
  "width": [0, Infinity],
  "length": [0, Infinity],
  "wl": [0, 1],
  "r": [0, 1],
  "leakage_intensity_width_2": [0, 1]
},


"tailcut": {
  "picture_thresh":8,
  "boundary_thresh":4,
  "keep_isolated_pixels":false,
  "min_number_picture_neighbors":2
},
```

```
  "random_forest_regressor_args": {
  "max_depth": 50,
  "min_samples_leaf": 2,
  "n_jobs": 4,
  "n_estimators": 150,
  "bootstrap": true,
  "criterion": "mse",
  "max_features": "auto",
  "max_leaf_nodes": null,
  "min_impurity_decrease": 0.0,
  "min_impurity_split": null,
  "min_samples_split": 2,
  "min_weight_fraction_leaf": 0.0,
  "oob_score": false,
  "random_state": 42,
  "verbose": 0,
  "warm_start": false
},
 "random_forest_classifier_args": {
   "max_depth": 100,
   "min_samples_leaf": 2,
   "n_jobs": 4,
   "n_estimators": 100,
   "criterion": "gini",
   "min_samples_split": 2,
   "min_weight_fraction_leaf": 0.0,
   "max_features": "auto",
   "max_leaf_nodes": null,
   "min_impurity_decrease": 0.0,
   "min_impurity_split": null,
   "bootstrap": true,
   "oob_score": false,
   "random_state": 42,
   "verbose": 0.0,
   "warm_start": false,
   "class_weight": null
 },
"regression_features": [
  "log_intensity",
  "width",
  "length",
  "x",
  "y",
  "psi",
  "phi",
  "wl",
  "skewness",
  "kurtosis",
  "r",
  "time_gradient",
  "leakage_intensity_width_2"
],
```

```
"classification_features": [
  "log_intensity",
  "width",
  "length",
  "x",
  "y",
  "psi",
  "phi",
  "wl",
  "skewness",
  "kurtosis",
  "r",
  "time_gradient",
  "leakage_intensity_width_2",
  "log_reco_energy",
  "reco_disp_dx",
  "reco_disp_dy"
],
"allowed_tels": [1],
"max_events": null,
"custom_calibration": false,
"write_pe_image": false,
"mc_image_scaling_factor": 1,
"image_extractor": "LocalPeakWindowSum",
"image_extractor_for_muons": "GlobalPeakWindowSum",
"gain_selector": "ThresholdGainSelector",
"gain_selector_config": {
  "threshold":  3500
},
"charge_scale": [1.18,1.09],
"LocalPeakWindowSum":{
   "window_shift": 4,
   "window_width": 8
  },
"GlobalPeakWindowSum":{
   "window_shift": 4,
   "window_width": 8
  },
"FixedWindowSum": {
   "window_shift": 4,
   "window_width": 8
},
"timestamps_pointing":"ucts",


"source_dependent": false,
"mc_nominal_source_x_deg": 0.4,
"mc_nominal_source_y_deg": 0.0,

"volume_reducer":{
  "algorithm": null,
  "parameters": {
  }
},
```

```
"calibration_product": "LSTCalibrationCalculator",

"LSTCalibrationCalculator":{
   "minimum_hg_charge_median": 5000,
   "maximum_lg_charge_std": 300,
  "squared_excess_noise_factor": 1.222,
   "flatfield_product": "FlasherFlatFieldCalculator",
   "pedestal_product": "PedestalIntegrator",
   "PedestalIntegrator":{
     "sample_size": 10000,
     "sample_duration":100000,
     "tel_id":1,
     "charge_median_cut_outliers": [-10,10],
     "charge_std_cut_outliers": [-10,10],
     "charge_product":"FixedWindowSum"
   },
   "FlasherFlatFieldCalculator":{
     "sample_size": 10000,
     "sample_duration":100000,
     "tel_id":1,
     "charge_product":"LocalPeakWindowSum",
     "charge_median_cut_outliers": [-0.5,0.5],
     "charge_std_cut_outliers": [-10,10],
     "time_cut_outliers": [2,38]
   },
  "LocalPeakWindowSum":{
    "window_shift": 5,
    "window_width":12
   },
  "FixedWindowSum":{
     "window_start": 12,
     "window_width":12
  }
}
```

DL2 -> DL3 configuration :

```
{
  "events_filters":{
    "intensity": [200, Infinity],
    "r": [0, 1],
    "wl": [0.1, 1],
    "is_good_event": [0.5,1]
  },
  "fixed_cuts":{
    "gh_score": [0.6,1],
    "theta_cut": {"value": 0.2, "unit": "deg"},
    "source_fov_offset": {"value": 2.83, "unit": "deg"}
  },
```

### 8.3.2 Run-Time Setup

The SAG-RECO is configured using the parameters contained in the Scheduling Block [AD7].

**Table 7: Run-time Setup Data component SAG-RECO**

| Parameter | Data Type | Description |
|---|---|---|
| Scheduling Block | struct | The Scheduling Block is needed to retrieve all the information about the sub-array configuration and scientific targets. |

### 8.3.3 Operations Logging

The SAG-RECO ACS components use the ACS log system while the software components external to ACS provide the logs to the ACS infrastructure using an ACS client.

**Table 8: Logging information component SAG-RECO**

| Log name | Log level | Condition to trigger | Description |
|---|---|---|---|
| `SAG-RECO-logs` | TBD | periodically | |

### 8.3.4 Alarms Triggered

The Alarms are managed through the ACS Alarm System.

**Table 9: Alarm triggered component SAG-RECO**

| Alarm name | Alarm Family | Alarm Criticallity | Description |
|---|---|---|---|
| Unable to start the system | | | The sub-system cannot be started and this is reported as an Alarm. |
| Data flow interrupted | | | The sub-system is not able to receive the input data. |

### 8.3.5 Monitoring Points

The table below lists the monitoring points of the SAG-RECO component.

**Table 10: Monitoring Points for Component SAG-RECO**

| Point name | Description | Type | Sample rate | Alarm Condition |
|---|---|---|---|---|
| recoInputDataRate | The DL0 data rate received by SAG | double | 1 Hz | monitoring point outside the nominal range |
| recoInputEventRate | The DL0 events rate received by SAG | double | 1 Hz | monitoring point outside the nominal range |
| recoDL0DL1ProcRate | The SAG-RECO processing rate in Hz of the step DL0->DL1 | double | 1 Hz | monitoring point outside the nominal range |
| recoDL1DL2ProcRate | The SAG-RECO processing rate in Hz of the step DL1->DL2 | double | 1 Hz | monitoring point outside the nominal range |

| | | | | |
|---|---|---|---|---|
| re-coDL2DL3ProcRate | The SAG-RECO processing rate in Hz of the step DL2->DL3 | double | 1 Hz | monitoring point outside the nominal range |

### 8.3.6 Input Data

The SAG-RECO receives input data from the ADH with the DL0 data format.

**Table 11: Input data of the component SAG-RECO**

| Data | Description |
|---|---|
| DL0 | See Sect. 8.1.1 |

### 8.3.7 Output Data

The SAG-RECO produces DL1, DL2 and DL3 data level as output data. These data are used by the SAG-DQ and SAG-SCI components.

**Table 12: Output data of the component SAG-RECO**

| Data | Description |
|---|---|
| DL1 | See Sect. 8.1.1 |
| DL2 | See Sect. 8.1.1 |
| DL3 | See Sect. 8.1.1 |

### 8.3.8 Internal Data

The SAG-RECO component receives DL0 as input from the ADH and reconstructs the data from DL0 to DL3 through other data levels. For this reason, the output of one layer of the SAG-RECO pipeline is the input of the following layer.

## 8.4 Component: On Line Data Quality

### 8.4.1 Configuration

The Data Quality component load configurations from XML files during start-up of the system. These configurations define the input data format, aggregation analysis and data quality checks.

The configurations are contained in different XML files:

1) Quality check

```xml
<?xml version="1.0" encoding="UTF-8"?>
<qualitycheckers>

  <qualitychecker id="pixelcheck" data_type_in="lst_dl1_camera" sample_size="1000">
    <qualitycheck id="image_pixel_range" input_field="image" type="rangethreshold" lb_alarm="-100" lb_warning="-16" ub_warning="30" ub_alarm="40" />
  </qualitychecker>

  <qualitychecker id="hillascheck" data_type_in="lst_dl1_parameters" sample_size="1000">
    <qualitycheck id="length_check" input_field="length" type="lowerboundthreshold" lb_alarm="0" lb_warning="0"  />
    <qualitycheck id="intensity_check" input_field="intensity" type="lowerboundthreshold" lb_alarm="0" lb_warning="0" />
    <qualitycheck id="kurtosis_check" input_field="kurtosis"  type="lowerboundthreshold" lb_alarm="0" lb_warning="0" />
    <qualitycheck id="phi_check" input_field="phi" type="rangethreshold" lb_alarm="-10" lb_warning="-5" ub_warning="5" ub_alarm="10" />
    <qualitycheck id="psi_check" input_field="psi" type="rangethreshold" lb_alarm="-10" lb_warning="-5" ub_warning="5" ub_alarm="10" />
    <qualitycheck id="r_check" input_field="r" type="lowerboundthreshold" lb_alarm="0" lb_warning="0" />
    <qualitycheck id="skewness_check" input_field="skewness" type="rangethreshold" lb_alarm="-10" lb_warning="-5" ub_warning="5" ub_alarm="10" />
    <qualitycheck id="width_check" input_field="width" type="lowerboundthreshold" lb_alarm="0" lb_warning="0" />
    <qualitycheck id="x_check" input_field="x" type="rangethreshold" lb_alarm="-10" lb_warning="-5" ub_warning="5" ub_alarm="10" />
    <qualitycheck id="y_check" input_field="y" type="rangethreshold" lb_alarm="-10" lb_warning="-5" ub_warning="5" ub_alarm="10" />
  </qualitychecker>


</qualitycheckers>
```

**Figure 49: Example of quality check configuration**

### 2) Aggregations

```xml
   <!-- step 1 -->
   <aggregator id="dl1_hillas_aggregator_step1" data_type_in="lst_dl1_parameters" >

      <!-- Collecting samples of hillas parameters -->
      <aggregation input_field="intensity" type="increasedim" output_field="intensity_samples" mode="stateless" max_array_size="100000" />
      <aggregation input_field="skewness" type="increasedim" output_field="skewness_samples" mode="stateless" max_array_size="100000" />
      <aggregation input_field="width" type="increasedim" output_field="width_samples" mode="stateless" max_array_size="100000" />
      <aggregation input_field="length" type="increasedim" output_field="length_samples" mode="stateless" max_array_size="100000" />
      <aggregation input_field="kurtosis" type="increasedim" output_field="kurtosis_samples" mode="stateless" max_array_size="100000" />
      <aggregation input_field="length" type="increasedim" output_field="length_samples" mode="stateless" max_array_size="100000" />
      <aggregation input_field="phi" type="increasedim" output_field="phi_samples" mode="stateless" max_array_size="100000" />
      <aggregation input_field="psi" type="increasedim" output_field="psi_samples" mode="stateless" max_array_size="100000" />
      <aggregation input_field="r" type="increasedim" output_field="r_samples" mode="stateless" max_array_size="100000" />
      <aggregation input_field="x" type="increasedim" output_field="x_samples" mode="stateless" max_array_size="100000" />
      <aggregation input_field="y" type="increasedim" output_field="y_samples" mode="stateless" max_array_size="100000" />
    </aggregator>
```

**Figure 50: Example of aggregations configuration**

### 3) Input Data Format

```
<datatype id="lst_dl1_parameters" id_field="event_id" time_field="trigger_time" desc="DL1 real data
  <field name="alt_tel" data_shape="scalar" data_type="float" um="?"/>
  <field name="az_tel" data_shape="scalar" data_type="float" um="?"/>
  <field name="dragon_time" data_shape="scalar" data_type="float" um="?"/>
  <field name="event_id" data_shape="scalar" data_type="int" um="?"/>
  <field name="intensity" data_shape="scalar" data_type="float" um="?"/>
  <field name="intercept" data_shape="scalar" data_type="float" um="?"/>
  <field name="kurtosis" data_shape="scalar" data_type="float" um="?"/>
  <field name="leakage" data_shape="scalar" data_type="float" um="?"/>
  <field name="length" data_shape="scalar" data_type="float" um="m"/>
  <field name="log_intensity" data_shape="scalar" data_type="float" um="?"/>
  <field name="mc_core_distance" data_shape="scalar" data_type="float" um="?"/>
  <field name="n_island" data_shape="scalar" data_type="int" um="?"/>
  <field name="num_trig_pix" data_shape="scalar" data_type="int" um="?"/>
  <field name="obs_id" data_shape="scalar" data_type="int" um="?"/>
  <field name="phi" data_shape="scalar" data_type="float" um="rad"/>
  <field name="psi" data_shape="scalar" data_type="float" um="rad"/>
  <field name="r" data_shape="scalar" data_type="float" um="m"/>
  <field name="skewness" data_shape="scalar" data_type="float" um="?"/>
  <field name="tel_id" data_shape="scalar" data_type="int" um="?"/>
  <field name="tel_pos_x" data_shape="scalar" data_type="float" um="?"/>
  <field name="tel_pos_y" data_shape="scalar" data_type="float" um="?"/>
  <field name="tel_pos_z" data_shape="scalar" data_type="float" um="?"/>
  <field name="tib_time" data_shape="scalar" data_type="float" um="?"/>
  <field name="time_gradient" data_shape="scalar" data_type="float" um="?"/>
  <field name="trigger_time" data_shape="array1d" data_type="float" x_dim="1" um="?"/>
  <field name="trigger_type" data_shape="scalar" data_type="int" um="?"/>
  <field name="ucts_time" data_shape="scalar" data_type="float" um="?"/>
  <field name="width" data_shape="scalar" data_type="float" um="m"/>
  <field name="wl" data_shape="scalar" data_type="float" um="?"/>
  <field name="x" data_shape="scalar" data_type="float" um="m"/>
  <field name="y" data_shape="scalar" data_type="float" um="m"/>
</datatype>
```

**Figure 51: Example of input data format**

### 8.4.2 Run-Time Setup

The SAG-DQ is configured using the parameters contained in the Scheduling Block [AD7].

**Table 13: Run-time Setup Data component SAG-DQ**

| Parameter | Data Type | Description |
|---|---|---|
| Scheduling Block | struct | The Scheduling Block is needed to retrieve all the information about the sub-array configuration and scientific targets. |

### 8.4.3 Operations Logging

The SAG-DQ ACS components use the ACS log system while the software components external to ACS provide the logs to the ACS infrastructure using an ACS client.

**Table 14: Logging information component SAG-DQ**

| Log name | Log level | Condition to trigger | Description |
|---|---|---|---|
| `SAG-DQ-logs` | TBD | periodically | |

### 8.4.4 Alarms Triggered

The Alarms are managed through the ACS Alarm System.

**Table 15: Alarm triggered component SAG-DQ**

| Alarm name | Alarm Family | Alarm Criticallity | Description |
|---|---|---|---|
| Unable to start the system | | | The sub-system cannot be started and this is reported as an Alarm. |
| Data flow interrupted | | | The sub-system is not able to receive the input data. |

### 8.4.5 Monitoring Points

The table below lists the monitoring points of the SAG-DQ component.

**Table 16: Monitoring Points for Component SAG-DQ**

| Point name | Description | Type | Sample rate | Alarm Condition |
|---|---|---|---|---|
| dqDL1ProcRate | The SAG-DQ processing rate in Hz of the DL1 data. | double | 1 Hz | Monitoring point outside the nominal range |
| dqDL2ProcRate | The SAG-DQ processing rate in Hz of the DL2 data. | double | 1 Hz | Monitoring point outside the nominal range |
| dqDL3ProcRate | The SAG-DQ processing rate in Hz of the DL3 data. | double | 1 Hz | Monitoring point outside the nominal range |
| dqDL1MergeRate | The SAG-DQ merge rate in Hz of the DL1 data. | double | 1 Hz | Monitoring point outside the nominal range |
| dqDL2MergeRate | The SAG-DQ merge rate in Hz of the DL2 data. | double | 1 Hz | Monitoring point outside the nominal range |
| dqDL3MergeRate | The SAG-DQ merge rate in Hz of the DL3 data. | double | 1 Hz | Monitoring point outside the nominal range |

### 8.4.6 Input Data

The SAG-DQ uses as input data the DL1, DL2 and DL3 data generated by the SAG-RECO pipelines.

**Table 17: Input data of the component SAG-DQ**

| Data | Description |
|---|---|
| DL1 | See Sect. 8.1.1 |
| DL2 | See Sect. 8.1.1 |
| DL3 | See Sect. 8.1.1 |

### 8.4.7 Output Data

The SAG-DQ generates different data types in output to show the results of the data quality checks.

**Table 18: Output data of the component SAG-DQ**

| Data | Description |
|---|---|
| Histograms | 1D and 2D histograms describing the distribution of analysed values (e.g. Hillas parameters). |
| Time Series | Time series of a value processed by the data quality pipeline (e.g. trigger rate). |
| Camera Plots | The image of a single event trigger seen by a camera. |

### 8.4.8 Internal Data

Internal data are data saved in the MySQL database.

## 8.5 Component: High-Level Analysis Pipeline

### 8.5.1 Configuration

The SAG-SCI pipeline is configured using the information stored in the Data Model described in Section 7.5.2. The main information are related to the sub-array configuration, observation parameters and targets.

### 8.5.2 Run-Time Setup

During the SAG-SCI run-time, all the analyses executed by the pipeline uses three XML files to retrieve configurations. These configurations contain static information (e.g. Observation parameters) and dynamic information updated for each analysis (e.g. time window, sky position etc.).

The following figures show examples of XML configuration files.

```
<job id="1487">
    <parameter name="RegionOfInterest" ra="31.582" dec="-53.211" skypositiontype="3" radius="5.0" AlertContour="None" frame="fk5" unit="deg" />
    <parameter name="TimeIntervals" tmin="0" tmax="1200" timeunit="s" timesys="tt" />
    <parameter name="ScienceToolReference" timestart="0" timeunit="s" timesys="tt" timeref="LOCAL" skyframe="fk5" skyframeunit="deg" />
    <parameter name="Energy" emin="0.03" emax="100.0" energyBinID="" />
    <parameter name="DirectoryList" job="$SAGSCI/sagsci/examples/data/grb_reflection/" results="$SAGSCI/sagsci/examples/results/grb_reflection/" jobprefix="test"/>
    <parameter name="OnOff" method="reflection" radius="0.2" save_off_regions="off_regions.reg" pixel_size="0.05" event_type="events_filename"/>
    <parameter name="Calibration" path="$SAGSCI/sagsci/caldb/data/cta" />
    <parameter name="Stack" value="0" />
    <parameter name="ComputeFlux" value="1" />
    <parameter name="CountsMap" value="1" binsize="0.02" zoomroi="3" smooth="1" reg="1" />
    <parameter name="BlindSearch" value="1" exclusion="" sigmathresh="5" maxsrc="10" regfile="hotspots.reg" />
    <parameter name="WorkInMemory" value="1" />
    <parameter name="Verbose" value="0" />
    <parameter name="DeleteJob" value="0" />
    <parameter name="Logging" level="debug" />
</job>
```

**Figure 52: Example of job configuration for scientific analyses**

```
<?xml version="1.0" standalone="no"?>
<source_library title="source library">
  <source name="GRB_RUN0406_ID000126" type="PointSource" tscalc="1">
    <spectrum type="PowerLaw">
      <parameter name="Prefactor"   scale="1e-16" value="5.7"  min="1e-07" max="1000.0" free="1"/>
      <parameter name="Index"       scale="-1"    value="2.4" min="0.0"   max="+5.0"   free="1"/>
      <parameter name="PivotEnergy" scale="1e6"   value="0.3"  min="0.01"  max="1000.0" free="0"/>
    </spectrum>
    <spatialModel type="PointSource">
      <parameter name="RA"  scale="1.0" value="33.057" min="-360" max="360" free="0"/>
      <parameter name="DEC" scale="1.0" value="-51.841" min="-90"  max="90"  free="0"/>
    </spatialModel>
  </source>
  <source name="CTABackgroundModel" type="CTAIrfBackground" instrument="CTA-SOUTH">
    <spectrum type="PowerLaw">
      <parameter name="Prefactor"   scale="1.0"  value="1.0"  min="1e-3" max="1e+3"   free="1"/>
      <parameter name="Index"       scale="1.0"  value="0.0"  min="-5.0" max="+5.0"   free="1"/>
      <parameter name="PivotEnergy" scale="1e6"  value="1.0"  min="0.01" max="1000.0" free="0"/>
    </spectrum>
  </source>
</source_library>
```

**Figure 53: Example of target configuration for scientific analyses**

```
<observation name="GRB_RUN0406_ID000126" id="1" instrument="CTA-SOUTH">
  <parameter name="Pointing" ra="31.582" dec="-53.211" frame="fk5" unit="deg"/>
  <parameter name="GoodTimeIntervals" tstartreal="0" tendreal="1200" tstartplanned="0" tendplanned="1200" timeunit="s" timesys="tt"/>
  <parameter name="RegionOfInterest" ra="31.582" dec="-53.211" rad="5.0" frame="fk5" unit="deg"/>
  <parameter name="Deadtime" deadc="1"/>
  <parameter name="Calibration" database="prod3b-v2" response="South_z20_0.5h"/>
  <parameter name="Energy" emin="0.1" emax="100.0"/>
</observation>
```

**Figure 54: Example of observation configuration for scientific analyses**

### 8.5.3 Operations Logging

The SAG-SCI ACS components use the ACS log system while the software components external to ACS provide the logs to the ACS infrastructure using an ACS client.

**Table 19: Logging information component SAG-SCI**

| Log name | Log level | Condition to trigger | Description |
|---|---|---|---|
| SAG-SCI-logs | TBD | periodically | |

### 8.5.4 Alarms Triggered

The Alarms are managed through the ACS Alarm System

**Table 20: Alarm triggered component SAG-SCI**

| Alarm name | Alarm Family | Alarm Criticallity | Description |
|---|---|---|---|
| Unable to start the system | | | The sub-system cannot be started and this is reported as an Alarm. |
| Data flow interrupted | | | The sub-system is not able to receive the input data. |

### 8.5.5 Monitoring Points

The table below lists the monitoring points of the SAG-SCI component.

**Table 21: Monitoring Points for Component SAG-SCI**

| Point name | Description | Type | Sample rate | Alarm Condition |
|---|---|---|---|---|
| sciProcRate | The time required by the SAG-SCI to perform the scientific analysis. | statEvent | 1 Hz | |
| blindSearchProcRate | The time required by the SAG-SCI to perform a blind search scientific analysis. | statEvent | 1 Hz | |
| dl3MergeRate | The time required by the SAG-SCI to perform the merge of DL3 files produced by the SAG-RECO analysis. | statEvent | 1 Hz | |

### 8.5.6 Input Data

The SAG-SCI uses the DL3 data generated by the SAG-RECO pipelines as input to generate scientific results.

**Table 22: Input data of the component SAG-SCI**

| Data | Description |
|---|---|
| DL3 | See Sect. 8.1.1 |

### 8.5.7 Output Data

The SAG-SCI pipelines generate several types of results that depend on the configuration and on the science tool that performed the analysis.

**Table 23: Output data of the component SAG-SCI**

| Data | Description |
|---|---|
| Lightcurves | Time series plot that shows the flux or the significance value as a function of time of a sky position. |
| Counts Maps | Images showing for each pixel the binned number of photons. |
| Analysis results | The results obtained with the scientific analysis (e.g. flux, source position, etc.) |

| | |
|---|---|
| Candidate Science Alerts | The SAG-SCI sends to the TH the detection with a significance over a configured threshold. |

### 8.5.8 Internal Data

Internal data are data saved in the MySQL database.

# 9 Human Machine Interfaces

The HMI interface of the SAG System is provided by HMI System.

# 10 Performance

## 10.1 System Performance

The SAG must handle 0.3 Gbps assuming 5% of pixels survived zero-suppression, with a maximum data rate of 9 Gbps in case no zero-suppression is applied.

## 10.2 Data Rates

The data streams handled by the component and the performance properties of each are listed in the following table (from [RD1]):

**Table 24: Data streams handled by the component**

| Data Stream | Description |
|---|---|
| Telescope triggers | Max: 15kHz<br>Typical SST: 600Hz<br>Typical MST: 7kHz<br>Typical LST: 15kHz |
| Array trigger | Max: 40 kHz<br>Average multiplicity: 5 |
| DL0 input for SAG | Average: 0.3 Gbps assuming 5% of pixels survived zero-suppression<br>Max: 9 Gbps in case no zero-suppression is applied |

## 10.3 Synchonization

The synchronization needed between this component and other components within the subsystem or other sub-systems, or stakeholders are listed in the following table:

**Table 25: Synchronization items for the component**

| Data Stream | Description |
|---|---|
| Scheduling Block | An input Scheduling Block used to start sub-array pipelines. This is received from Central Control [AD6]. |
| Alarm | Alarms are received and used to decide the workflow of the processes. Alarms are received from Alarm System [AD6]. |
| Monitoring points | Some monitoring points are used to decide the workflow of the processes. These monitoring points are received from the Monitoring System [AD6]. Monitoring points could be: |

| | |
|---|---|
| | 1. environmental conditions (cloudiness, atmospheric transmission, atmospheric extinction profile): to select the right IRF.<br>2. NSB: to select the right IRF.<br>3. pointing info: to select the center of the FoV.<br>4. status for each telescope: to select the right IRF. |
| Candidate Science Alerts | Candidate Science Alert are sent to the Transient Handler [AD6]. |

## 10.4 Control Loops

The control loops implemented by these components are listed in the following table:

**Table 26: Control loops deployed within the component**

| Data Stream | Description |
|---|---|
| Incoming commands from CC | Waiting for new commands or new Scheduling Blocks from CC change the control loop. |
| DL0 | Incoming DL0 data starts the data processing of the reconstruction and data quality pipelines |

## 10.5 Start-up and Shut-down

The start-up of the SAG Supervisor and of the SAG Sub-Array Pipeline Supervisor requires a few seconds. The remaining part of the SAG pipelines is started when new a Scheduling Block is received and requires a few seconds. The same time is required to shutdown the pipelines and the SAG Supervisor.

## 10.6 State Transitions

The performance required/expected for each state transition are listed in the following table:

**Table 27: Performance of the state transitions of the SAG-DQ and SAG-SCI pipelines**

| Transition | Execution Time (s) | Comments |
|---|---|---|
| <Off → Initialized> | Some milliseconds | Initialize the pipeline and configure the system. |
| <Initialized →Idle> | Some milliseconds | Start the pipeline processes. |
| <Idle → Nominal> | Some milliseconds | Start the watchdog process that waits for new data. |
| <Nominal -> Idle > | Up to 20 seconds | The pipelines are not receiving new data but need to analyse the data buffer. |
| <Initialized → Warning><br><Idle → Warning><br><Nominal → Warning> | Some milliseconds | From all operational states it is possible to go in Warning state for example if data stream or data flow connection are lost even if the data is expected. |
| <Warn →Error> | Some milliseconds | If the Warning state exceeds a critical level the pipeline goes in Error state. |
| < Error -> Off > | Some milliseconds | Nothing is happening in Error state. Pipelines need to be restarted. |
| <Initialized → Off><br><Idle → Off><br><Nominal → Off><br><Warning → Off><br><Error → Off> | Some milliseconds | From all states it is possible to power off the pipeline. |

## 10.7 Command Handling

The performance required/expected for handling each command, are listed in the following table. Note, it may be necessary to take the parameters or data submitted with the command into account, as it may influence the execution time:

**Table 28: Performance of the command execution of the component**

| Transition | Execution Time (s) | Comments |
|---|---|---|
| SAG-RECO | 2 s | |
| SAG-DQ | 2 s | |
| SAG-SCI | 5 s | |

# 11 Frameworks & Libraries

The following external/3rd party frameworks (SW packages) are needed for the implementation of the component:

**Table 29: SW Frameworks/Packages used by the component**

| Framework/SW Package | Description |
|---|---|
| ACS | |
| ZeroMQ | Handles network sockets |

## 11.1 External dependencies:

This section lists the used external libraries for each SAG component. It is indicated the version used when this document is written but the software will be compatible also with newer versions of libraries.

| Library | Description |
|---|---|
| pytest | This is a python testing tool used to implement unit tests. |
| pyest-cov | Handles network sockets |
| scipy | Scipy is a library with modules useful for the scientific computations. |
| lxml | Library for processing XML in python. |
| regions | A library to manage regions. |
| matplotlib | A library to generate diagrams and plots. |
| swig | A wrapper to integrate C and C++ programs in Python. |
| opencv | Computer vision library. |
| mysql-connector-python | A library to work with MySQL database in Python. |
| blosc | High performance compressor optimized for binary data. |
| hdf5 | A library to work with the HDF5 data format. |
| zmq (ZeroMQ) | Asynchronous messaging library that can be used in distributed application. |
| coverage | A tool to measure the code coverage of Python programs. |

| | |
|---|---|
| unittest-xml-reporting | A runner of unittest to save the test results in XML. |
| cmake | A tool used to control the software compilation process. |
| make | An automation tool to build executable programs from source code. |
| ctapipe | Low-level data processing pipeline software for CTA. |
| gammapy | A Python package for gamma-ray astronomy. |
| h5py | A Python library to work with HDF5 data format. |
| jupyter | A web-based interactive platform to develop and run Python programs. |
| notebook | A web-based interactive platform to develop and run Python programs. |
| joblib | A set of tools to provide lightweight pipelining in Python. |
| pytest-runner | A plugin for the pytest library. |
| pytest-ordering | A pytest plugin to run the test in any order. |
| ctapipe_io_lst | A plugin for ctapipe to add new input data sources for LST zfits. |
| lstchain | Library for high-level analysis of LST telescope. |
| ctaplot | Python plotting library for CTA. |
| pyirf | Python library to generate the Instrument Response Functions for CTA |
| numpy | A Python library of mathematical functions, random number generation and other useful features. |
| tables | A Python library to format data into plaint-text tables. |
| astropy | A library of common features used in Astronomy. |
| watchdog | A library to monitor the file system events (e.g. new file created). |
| pandas | A Python tool for data manipulation and analysis. |
| fast-histogram | High performance library to generate histogram in python. |
| hurry.filesize | A Python library to print the byte size in human-readable format. |
| defusedxml | A python library to work with XML data format. |

### 11.1.1 SAG Pipeline Sub-Array Supervisor and SAG Supervisor (SAG-SUP)

Python packages:

- Python >= 3.6.9
- pytest >= 6.2.4
- pytest-cov >= 2.10.1
- unittest-xml-reporting >= 3.0.4
- coverage >= 5.3

### **11.1.2** Image Parameters Extractor and Low-Level Reconstruction Pipeline

- hdf5 103.0.0 (hdf5_serial 103.0.0 + hdf5_cpp 103.0.0)
- zmq >= 5.2.2blosc >= 1.17.1
- Slurm >= 17.11.12g++/gcc : >= 7.3
- cmake : >= 3

- make : >= 4

Python packages:

- Python>=3.7
- pip>=21.0
- ctapipe>=0.8.0
- gammapy>=0.18
- h5py>=3.1.0
- jupyter>=4.7.1
- Notebook>=6.2.0
- Joblib>=1.0.0
- pip
- pytest_runner>=5.2
- pytest-ordering>=0.6
- https://github.com/cta-observatory/ctapipe_io_lst/archive/v0.6.0.zip and above
- lstchain>=0.6.3
- ctaplot>=0.5.6
- pyirf>=0.4.0

### 11.1.3 On-Line Data Quality Software

Python packages:

- Python>=3.8
- numpy >= 1.20.3
- tables >= 3.6.1
- astropy >= 4.2.1
- watchdog >= 2.1.2
- pandas >= 1.1.4
- fast-histogram >= 0.9
- hurry.filesize >= 0.9
- defusedxml >= 0.7.1
- mysql-connector-python >= 8.0.22
- matplotlib >= 3.4.2 [Testing and benchmarking]
- pytest >= 6.2.4 [Testing and benchmarking]
- pytest-cov >= 2.10.1 [Testing and benchmarking]
- unittest-xml-reporting >= 3.0.4 [Testing and benchmarking]
- coverage >= 5.3 [Testing and benchmarking]
- psutil >= 5.8.0 [Testing and benchmarking]

### 11.1.4 High-Level Analysis Pipeline

Python packages:

Pipeline:

- Python>=3.6

- astropy>=3.0
- lxml>=4.1.1
- mysql-connector-python>=8.0.18
- watchdog>=2.1.6

Science Tools wrappers:

- astropy>=4.2.1
- scipy>=1.6.2
- lxml>=4.6.3
- regions>=0.4
- matplotlib>=3.4.2
- swig>=4.0.2
- opencv>=3.4.2
- mysql-connector-python>=8.0.18
- Gammapy>=0.18.2

## 11.2 Framework/Library: Munge

### 11.2.1 Justification

Munge is an authentication service for creating and validating credentials used to manage the connections of Slurm nodes.

### 11.2.2 Licensing

The software is distributed under the GNU General Public License.

### 11.2.3 Version Control

The SAG works with versions of Munge >= 0.5.11.

### 11.2.4 Distribution/Installation

It can be installed as a Linux service.

### 11.2.5 Obsolescence Handling

This service is cited in the official Slurm documentation so we can assume that it will be maintained.

## 11.3 Framework/Library: ACS

### 11.3.1 Justification

ACS framework is the main framework of the ACADA system.

### 11.3.2 Licensing

The software licensing of the SAG is based on CTA software policy and licensing.

### 11.3.3 Version Control

The updating of the framework follows the ACADA plans.

### 11.3.4 Distribution/Installation

ACS is distributed with ACADA SDK.

### 11.3.5 Obsolescence Handling

The obsolescence of ACS is managed by ACADA management team.

## 11.4 Framework/Library: ZeroMQ

ZeroMQ is a networking library and provides sockets that carry atomic messages across various transports like in-process, inter-process, TCP, and multicast. Connection patterns are: N-to-N with patterns like fan-out, pub-sub, task distribution, and request-reply. Its asynchronous I/O model gives you scalable multicore applications, built as asynchronous message-processing tasks. It has a score of language APIs and runs on most operating systems.

### 11.4.1 Justification

ZeroMQ is needed to receive DL0 from the ADH (see [RD1]).

### 11.4.2 Features

The library is message-based, supports multi-languages and is multi-protocol (allows to send/receive messages via a variety of protocols).

### 11.4.3 Licensing

GNU Lesser General Public License V3

### 11.4.4 Version Control

The current code works with any version of the ZeroMQ >= 4.1

### 11.4.5 Distribution/Installation

This package comes via yum: `yum install zmq-devel`

### 11.4.6 Obsolescence Handling

This library is open source.

## 11.5 Framework/Library: Slurm

Slurm is a job scheduler for Linux and Unix-like kernel used in HPC context. It is free and open-source.

### 11.5.1 Justification

The Slurm features satisfy the SAG requirements related to more than one component:

*GEN-0030 SAG pipelines shall mitigate the impact of all foreseeable partial failures, including loss of computing nodes and software component failures, implementing automatic recovery procedures of the pipeline workflow.*

*GEN-0060 SAG pipelines shall use parallel processing methods to speed up the processing.*

*SUP-035 SAG shall dynamically allocate the needed computing resources and free them when they are no more needed.*

*SCI-210 SAG/High-Level pipeline shall manage the priority between different processes, executing first the high-priority tasks and suspending low-priority task if all resources are busy.*

### 11.5.2 Features

Slurm has several fueatures, the most important for the SAG are:

- *No single point of failur*e is managed with backup daemons and fault-tolerant job options.
- *High performance (*up to 10000 job submissions per second)
- *High scalability (*up to thousands of cores)
- *The possibility to create queues and reservations* for specific jobs
- *The possibility* to define the priority between jobs, suspend low priority jobs to execute high priority jobs and then resume low priority jobs.
- *Para*llel execution of the jobs
- *Storage of* information about jobs in a database.
- *The status and other parameters of jobs can be retrieved in real-time.*

### 11.5.3 Licensing

GNU General Public License

### 11.5.4 Version Control

The current code works with any version of the Slurm >= 17.11.12

### 11.5.5 Distribution/Installation

The service can be installed using the source code or the yum package manager.

### 11.5.6 Obsolescence Handling

This library is open source and the large usage of this software leads to the assumption that it will be maintained in the future.

# 12 Miscellaneous Aspects

## 12.1 Deployment

The components are deployed in the ACADA standard way.

## 12.2 Exception and Error Handling

The components of the SAG Supervisor and of the SAG Sub-Array Pipeline Supervisor use the standard error handling defined for ACADA components (based on ACS ).

## 12.3 Logging and Tracing

The components of the SAG Supervisor and of the SAG Sub-Array Pipeline Supervisor use the standard logging and tracing defined for ACADA components (based on ACS).

## 12.4 Concurrency and Threading

The concurrency of the jobs submitted by SAG-RECO, SAG-DQ and SAG-SCI are managed by Slurm. Slurm is also able to manage the multithreading capabilities of SAG-DQ.

## 12.5 Debugging and Troubleshooting

Debugging and troubleshooting is performed using standard development tools.

## 12.6 Open Points and Issues

This section lists the open points and issues of the SAG sub-systems. Each issue has a severity and a comment.

**Table 30: Issues**

| Issue | Severity (L/M/H) | Comments |
|---|---|---|
| Lack of official DL0, DL1, DL2 and DL3 data model | High | SAG-RECO, SAG-DQ and SAG-SCI use these data formats, that currently are preliminary and not complete. Changes in these data formats will have an impact on the software developed. |
| Lack of official DPPS scheme to produce IRFs | High | SAG-SCI needs IRFs, there will be provided by DPPS but the current status is that no clear scheme of production of these IRFs is provided. |
| Lack of official DL4 and DL5 data models | High | The SAG-SCI produces results in DL4 and DL5 data format. We need to know the right format that the system shall generate. |
| Lack of reference test dataset for scientific analyses | Medium | To test different SAG-SCI pipeline configurations we need an official test dataset with all possible data that will be analysed by the SAG (e.g. GRBs, AGNs and more) |
| Specify Log Level for Logging information component from SAG. | Low | Logging information component from SAG must be specified in the next ACADA releases. |
| Details on data quality and scientific analysis | Low | Details on data quality and scientific analysis for each telescope will be provided in Annexes of this document for both LST, MST, SST. |

| | | |
|---|---|---|
| Comparison with the DPPS DL0 to DL3 offline pipeline is missing | Low | Since the DPPS DL0 to DL3 pipeline does not exist yet, a comparison with the offline pipeline is not possible yet, it will be done when the DL0 to DL3 pipeline will exist. |
| Pending SAG-HMI interface definition | Low | Discussion with HMI team is needed to define this interface |
| ACADA-SUSS ICD on source catalogues | Low | Offical ACADA-SUSS ICD must be released. |

# 13 Traceability Matrix

## 13.1 From Requirements

The following table defines the mapping between level C (sub-system) requirements, and design items.

**Table 31: Traceability Matrix of requirements to design items**

| Requirement | Design Item | Comments |
|---|---|---|
| ARC-010 | SAG-RECO | The SAG-RECO results are stored in HDF5 files. |
| DQ-010 | SAG-DQ | The design of this component is described in Section 7.4. In particular, the performances of the analyses are obtained using high-performance libraries to execute the analyses (Section 7.4.5) |
| DQ-020 | SAG-DQ | The SAG-DQ pipeline can be configured using XML files to process different data levels (DL1, DL2, DL3). This behaviour is described in Section 7.4.8. |
| DQ-030 | SAG-DQ | The analyses performed by the SAG-DQ are based on external libraries that allow the execution of required analyses, Section 7.4.5. |
| DQ-040 | SAG-DQ | In Section 7.4 all the analysis types performed by the SAG-DQ are described. |
| EXP-010 | All SAG components | The SAGSubArrayManager component has a method “updateSchedulingBlock” that can be used to update the configuration of the system when the array configuration is changed. The pipelines are flexible enough to react to these updates. |
| FL-0010 | All SAG components | The design decisions taken and described in Section 6 take into account the CTA life-time. |
| GEN-0010 | All SAG ACS components | The SAG implements ACS components for the supervision of the processes and to manage the interfaces between component inside and outside the SAG (Section 6). |
| GEN-0020 | All SAG components | All the SAG software components are developed and tested in a Linux environment. |
| GEN-0040 | All SAG components | |
| GEN-0050 | All SAG components | |
| GEN-0060 | All SAG components | The design decisions taken (Slurm workload manager and programming languages) allow parallel processing. |
| GEN-0150 | All SAG components | |
| GEN-0200 | All SAG components | The design of the SAG components are described in Section 7. |

| | | |
|---|---|---|
| GEN-0210 | All SAG components | |
| GEN-0220 | SAG-SCI | This component receives the DL3 from the low-level reconstruction pipeline and performs a quick-look analysis. It is described in Section 7.5. |
| GEN-0230 | All SAG components | The SAG software component, as described in Section 7 and 8, can be configured using XML, JSON and Databases to allow high flexibility. |
| GEN-0240 | All SAG components | Section 7 |
| HMI-010 | SAG-SCI and SAG-DQ | The SAG pipelines describe in Section 7.4 and 7.5 stores the results of analyses in a database and the HMI is able to retrieve this information. |
| HMI-020 | All SAG components | |
| HMI-060 | SAG-Supervisor, SAG-DQ | The SAGSupervisor described in Section 7.2.1.2 generates summaries of data quality results that are sent to the SUSS. |
| HMI-070 | SAG-SUP | The results of the pipelines are stored to be retrieved by the SAG-HMI. |
| INT-010 | All SAG components | The input data models are described in Section 8. |
| INT-020 | SAG-RECO | Section 7.3 |
| INT-030 | SAG-RECO | Section 7.3 |
| INT-040 | SAG-RECO | Section 7.3 |
| INT-050 | SAG-SUP | Section 7.2 |
| INT-070 | SAGSubArrayManager | A dedicated interface receives all information needed for SAG-RECO and the IRF needed by SAG-SCI. Not implemented for ACADA REL1 [AD3] |
| INT-080 | SAG-RECO, SAG-DQ, SAG-SCI | The outputs of the pipelines are described in Section 7. |
| INT-090 | SAG-RECO and SAG-SCI | The SAG-RECO reconstructs data from DL0 to DL3 and the SAG-SCI performs scientific analysis on DL3 producing DL4 and DL5. |
| INT-100 | SAG-Supervisor, SAG-SCI | Section 7.5 |
| INT-130 | SAG-Supervisor | https://forge.in2p3.fr/projects/acada-coordination/wiki/SAG-MON |
| INT-140 | SAG-SCI, SAG-Supervisor | Section 7.2.1.2 |
| INT-150 | SAGSubArrayManager | The SAGSubArrayManager collects information about the analysis status from other software components and sends this status to the CC (Section 7.2.1.1). |
| REC-005 | SAG-RECO | Section 7.3 |
| REC-010 | SAG-RECO | Section 7.3 |
| REC-020 | SAG-RECO | Section 7.3 |
| REC-030 | SAG-RECO | Section 7.3 |
| REC-040 | SAG-RECO | Section 7.3 |
| REC-050 | SAG-RECO | Section 7.3 |
| REL-0010 | All SAG components | The design decisions taken (Slurm workload manager, ACS etc) allow the management of failure to improve the availability of the system. |
| SCI-010 | SAG-SCI | The SAG-SCI is designed to perform the required scientific analyses (Section 7.5) |
| SCI-020 | SAG-SCI | Section 7.5 |
| SCI-030 | SAG-SCI | Section 7.5 |

| | | |
|---|---|---|
| SCI-040 | SAG-SCI | The SAG-SCI implements a local storage system to collect DL3 results (Section 7.5) |
| SCI-045 | SAG-SCI | Section 7.5 |
| SCI-050 | SAG-SCI | Section 7.5 |
| SCI-060 | SAG-SCI | The SAG-SCI can be configured to perform analyses on different time scales (Section 7.5) |
| SCI-070 | SAG-SCI | The SAG-SCI can read configuration in a database to define the threshold to generate Candidate Science Alerts (Section 7.5) |
| SCI-090 | SAG-SCI | Section 7.5 |
| SCI-120 | SAG-SCI | The SAG-SCI can read configuration in a database to define the threshold to avoid the duplication of candidate science alerts. |
| SCI-140 | SAG-SCI | The SAG-SCI can read configuration in a database to define the threshold to avoid the duplication of candidate science alerts. |
| SCI-160 | SAG-SCI | The SAG-SCI can visualize if there are data quality alarms and avoid the generation of candidate science alerts. |
| SCI-180 | SAG-SCI | |
| SCI-200 | SAG-SCI | The SAG-SCI can be configured to perform analyses using different science tools (Section 7.5). |
| SCI-210 | SAG-SCI | The SAG-SCI can manage priority between processes using the Slurm workload manager (Section 7.5). |
| SUP-010 | SAG-SUP | Section 7.2.1.1 |
| SUP-020 | SAG-SUP | Section 7.2.1.1 |
| SUP-030 | SAG-SUP | The usage of Slurm allows to execute pipelines in parallel. |
| SUP-035 | SAG-SUP | The usage of Slurm allows to dynamically allocate resources when needed. |
| SUP-040 | SAG-SUP | Section 7.2.1.1 |
| SUP-050 | SAG-SUP | Section 7.2.1.1 |
| SUP-060 | SAG-SUP | Section 7.2.1.1 |
| SUP-070 | All SAG components. | The SAG pipelines have a buffer to store the input data and process these data when a stop signal is received. |
| SUP-080 | SAG-SUP | Section 7.2.1.1 |
| SUP-110 | SAG-SUP | Section 7.2.1.1 |
| SUP-140 | SAG-SUP | The SAGSubArrayManager collects information from other components and sends it to the MON system. |

## 13.2 From Use Cases

The following table defines the mapping between ACADA use cases, and design items for REL1.

**Table 32: Traceability Matrix of interfaces to design items**

| Use Case | Design Item | Comments |
|---|---|---|

| | | |
|---|---|---|
| ACADA-UC-110-1.2 | All SAG components | This UC is described in diagrams of Sections 7.2.7 and 7.2.8. |
| ACADA-UC-110-3.1 | All SAG components | Sections 7.2.8.4 and 7.2.8.5 |
| ACADA-UC-110-3.2 | All SAG components | Sections 7.2.8.4 and 7.2.8.5 |
| ACADA-UC-110-3.3 | All SAG components | Sections 7.2.8.4 and 7.2.8.5 |
| ACADA-UC-110-3.4 | All SAG components | Sections 7.2.8.4 and 7.2.8.5 |
| ACADA-UC-120-2.1 | SAG-SUP | Sections 7.2.8.2 and 7.2.8.3. |
| ACADA-UC-120-2.2 | All SAG components | Secions 7.2.7, 7.2.8.2, 7.2.8.3, 7.3.7, 7.4.8, and 7.5.7. |
| ACADA-UC-120-2.3 | SAG-DQ | Secction 7.4.8. |
| ACADA-UC-120-2.4 | SAG-SCI | Section 7.5.2. |
| ACADA-UC-120-3.1 | All SAG components | Section 7.2.8.3.1 |
| ACADA-UC-130-3.2 | SAG-SCI | |
| ACADA-UC-130-3.3 | SAG-SCI | |
| ACADA-UC-170-1.1 | SAG-SUP | Section 7.2.1.2 |
| ACADA-UC-170-1.3 | SAG-SUP | Section 7.2.1.2 |

## 13.3 From Interfaces of the Sub-system

The following table defines the mapping between sub-system interfaces, and design items. Details are provided in [AD6].

**Table 33: Traceability Matrix of interfaces to design items**

| Interface/ICD | Design Item | Comments |
|---|---|---|
| SAG-ALARM | SAGSubArrayManager and SAGSupervisor | This interface is implemented using the ACS Alarm system. |
| SAG-MON | SAGSubArrayManager and SAGSupervisor | This interface is implemented with the ACS Notification Channel https://redmine.cta-observatory.org/projects/acada-coordination/wiki/SAG-MON |
| SAG-TH | SAG-SCI and SAGSubArrayManager | This interface is implemented using the ACS components. https://redmine.cta-observatory.org/projects/acada-coordination/wiki/TH-SAG . Section 7.5.2. |
| SAG-ADH | SAGReco | This interface is implemented using ZeroMQ and Protocol Buffer. Section 7.3. |
| SAG-RM | SAGSupervisor | Section 7.2 |
| SAG-CC | SAGSubArrayManager | Section 7.2 |
| SAG-SUSS | SAGSubArrayManager and SAGSupervisor | [AD5] |