ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA092

PROTOTYPING THE RESOURCE MANAGER AND CENTRAL CONTROL SYSTEM FOR THE CHERENKOV TELESCOPE ARRAY

D. Melkumyan[*], I. Sadeh, T. Schmidt, P. Wegner, DESY, Zeuthen, Germany
S. Sah, M. Sekoranja, Cosylab d.d., Ljubljana, Slovenia
I. Oya, M. Füßling, CTAO gGmbH, Heidelberg, Germany
DI. Schwanke, Humboldt-Universitaet zu Berlin, Berlin, Germany
DII. Schwarz, INAF – Osservatorio Astronomico di Brera, Milan, Italy
for the CTA Observatory[†]

Abstract

The Cherenkov Telescope Array (CTA) will be the next generation ground-based observatory for gamma-ray astronomy at very-high energies. CTA will consist of two large arrays with 118 Cherenkov telescopes in total, deployed in the Paranal (Chile) and Roque de Los Muchachos (Canary Islands, Spain) Observatories. The Array Control and Data Acquisition (ACADA) system provides the means to execute observations and to handle the acquisition of scientific data in CTA. The Resource Manager & Central Control (RM&CC) sub-system is a core element of the ACADA system. It implements the execution of observation requests received from the scheduler sub-system and provides infrastructure services concerning the administration of various resources to all ACADA sub-systems. The RM&CC is also responsible of the dynamic allocation and management of concurrent operations of up to nine telescope sub-arrays, which are logical groupings of individual CTA telescopes performing coordinated scientific operations. This contribution presents a summary of the main RM&CC design features, and of the future plans for prototyping.

INTRODUCTION

The Cherenkov Telescope Array (CTA) is the planned next generation observatory for ground-based very-high-energy gamma-ray astronomy [1]. In order to provide full sky coverage, CTA will consist of two large arrays with 118 Cherenkov telescopes in total, to be deployed in the Southern (Paranal, Chile) and Northern (La Palma, Spain) Hemispheres. The southern array will be sensitive to entire energy range of CTA, covering gamma-ray energies from ~20 GeV to above 300 TeV. The northern array will focus on low- and mid-energy ranges from ~20 GeV to ~20 TeV. In addition to a large number of scientific instruments, CTA will comprise three types of telescopes covering different energy ranges: Small Sized Telescopes (~4 m diameter) to cover the highest energy gamma-rays, Medium Sized Telescopes (~12 m) to cover the core energy range of CTA and Large Sized Telescopes (~23 m) sensitive to low-energy gamma-rays (SSTs, MSTs and LSTs, respectively).

The Array Control and Data Acquisition (ACADA) system provides the means to execute observations and to handle the acquisition of scientific data in CTA [2, 3].

In this proceeding, we focus on the design and implementation of one of the top-level sub-systems of ACADA, the Resource Manager and Central Control (RM&CC) system. The purpose of RM&CC is to execute observations requests received from the Short-term Scheduler sub-system and to provide to all ACADA sub-systems infrastructure services concerning the administration of various resources.

ARRAY CONTROL AND DATA ACQUISITION SYSTEM

ACADA will provide the functionality required to monitor and control all telescopes and auxiliary instruments in CTA; to perform observations and calibration procedure; to handle, filter and store data from all the telescope and auxiliary instruments; and to produce status and quality reports.

The ACADA architecture was designed using the Software Platform Embedded System (SPES) methodology [4]. A combination of OMG systems Modeling Language (SysML) [5] and Unified Modeling Language (UML) [6] was employed to model the ACADA system architecture [7] (system requirements, system behavior and subsystems). The ACADA system is composed of several closely interrelated sub-systems (*Short-term Scheduler, Transient Handler, Resource Manager and Central Control, Array Data Handler, Human Machine Interface, Science Alert Generation Pipeline, Array Alarm, Configuration, Reporting, Monitoring and Logging*). These subsystems are split up further into individual components. Each sub-system and each component serves a well-defined easily comprehensible purpose. This architectural approach allows us to implement a flexible use-case driven software development approach thanks to the traceability from use cases to the logical software elements.

ACADA will be implemented as a distributed software system using the ALMA Common Software (ACS), which is a set of application frameworks built on top of CORBA middleware [8]. ACS is based on a container-component model and supports the programming languages C++, Java and Python. Each CTA site will contain one instance of the ACADA system.

[*] david.melkumyan@desy.de
[†] https://cta-observatory.org

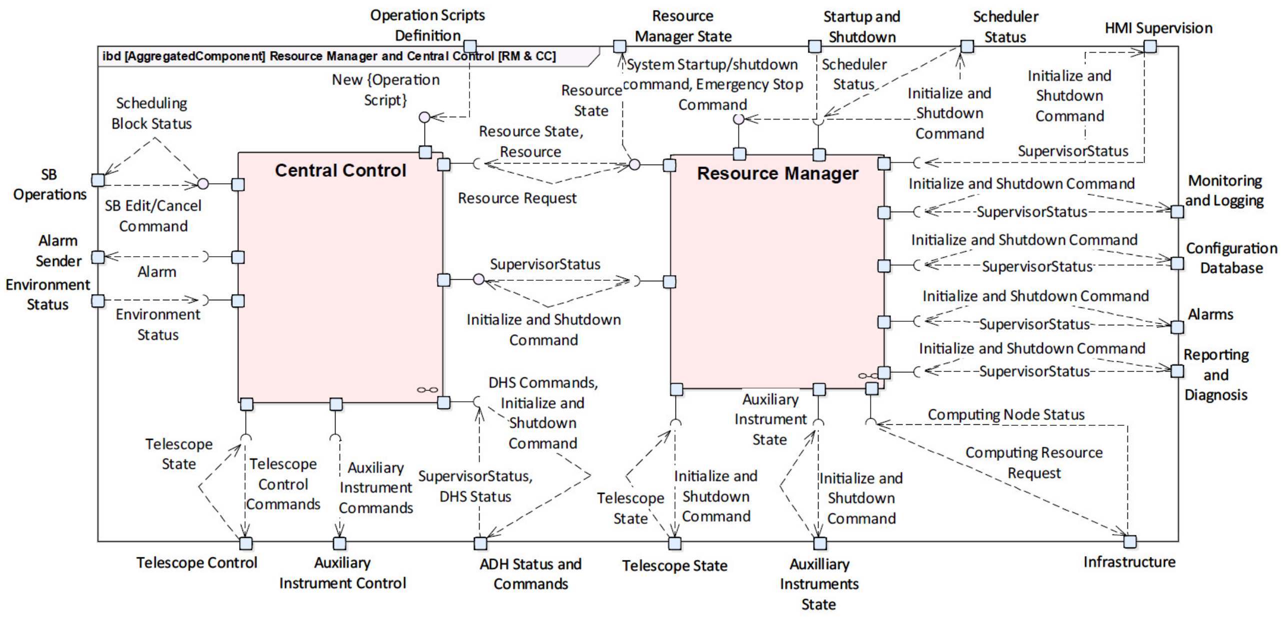


Figure 1: Logical view of RM&CC, representing the highest-level components and the most relevant data elements and interfaces of the system.

RESOURCE MANAGER AND CENTRAL CONTROL

RM&CC is one of the top-level sub-systems of ACADA. It was prototyped following the Model-Driven Architecture (MDA) approach of ACADA [9]. RM&CC comprises two components; namely, *Resource Manager* and *Central Control*. Figure 1 shows an UML diagram indicating the main components of the RM&CC sub-system. The Resource Manager component is the master element that provides the ACADA system with start-up, shut-down, supervision and registry functionality. The Central Control component is responsible for execution of scheduling blocks (SB) received from the Short-term Scheduler (Scheduler) [10]. It creates sub-arrays and sends corresponding commands to the telescopes, to the auxiliary instruments and to the Array Data Handler System (ADH) [11]. The CC then monitors observation progress and provides this information to the Human Machine Interface (HMI) [12]. One instance of RM&CC exists per ACADA system.

RESOURCE MANAGER

The Resource Manager (RM) provides to all ACADA sub-systems administrative and infrastructure services concerning various resources. The later comprise telescopes, auxiliary instruments, computing nodes and ACADA components. The Scheduler and the RM share responsibility for the management of telescopes. A telescope that should not be scheduled for sub-array actions can be marked unavailable in the RM by the operator via HMI. The RM and the Scheduler work in tandem: on the one hand, on any change of telescope availability, the RM informs the schedulers; on the other hand, at the beginning of every short-term scheduler run, the Scheduler asks the RM for current telescope availability information. Any unavailable telescopes will not be used for the schedule that is being created. Other resources (e.g. computing nodes) are entirely managed by the RM.

The Resource Manager consists of the following components:

- **The Root Supervisor** is the root node of the ACADA supervision tree. All components started by the root supervisor are deployed to run in a persistent way, and are restarted by the supervisor in case of a problem. For fault tolerance, the Root Supervisor component is instantiated in two instances (“cold” and “hot” instances), deployed on different computing nodes. The list of all components to be supervised by the Root Supervisor is statically configured through the ACS configuration database.
- **The Component Starter** provides services to any supervisor component that needs to instantiate and start another component. The Component Starter is capable of finding a computing node appropriate for hosting a particular supervised component to be started. It helps a supervisor component to initialise and run, providing pertinent run-time initialization parameters, in a process which may be component-specific.
- **The Device Registry** tracks availability for service of telescopes and auxiliary devices. If some device has been taken out of service due to a problem, for maintenance, or for other reasons, this information is stored in the Device Registry.
- **The Computing Node Registry** tracks availability and utilization of computing nodes used by ACADA. It can be used by the operator via HMI to obtain a list of all nodes utilized by ACADA components; to check individual availability and idle status; and to take some computing nodes out of service, or reactivate them.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA092

- **The Persistence Service** is an ACADA-wide service to persist the state of those components that require it. A simple-to use development version of the Persistence Service is implemented. This development version reads its content from a flat file in JSON format.
- **The Role Look-up Service** allows look-up of components by role name, facilitating identification of a service component by functionality. It provides convenient access to the currently active member of the successor chain of an ACADA component (see section Supervision Tree). The Role Look-up Service also keeps references of all instances of all components, as well as the particular control ticket each one holds (see section Control Tickets). Almost every ACADA component uses the role look-up functionality of the Resource Manager.
- **The Array Elements Supervisor** provides the capability to start-up, shutdown, and supervise software components of Array Elements.

Figure 2 shows the logical view of the Resource Manager component.

Supervision Tree

Reliable operation is a primary quality requirement of ACADA. To handle potential failure, ACADA employs a supervision tree concept, used in Erlang's fault management system [13]. For that purpose, the system is structured as a tree, where each tree-node supervises its child nodes. The supervision includes the following functions: starting a supervised component; monitoring the status of a supervised component, and replacing a supervised component with its successor if something goes wrong (e.g. if a supervised component becoming network-wise unreachable).

Control Tickets

In the supervisor tree, the life time of a predecessor and the corresponding successor component might overlap. In order to avoid conflict, the API methods of each service component include a control ticket. The latter will be used to guard against conflicting concurrent access, where only a single control ticket can be valid at a time (i.e. control tickets constructed later invalidate those constructed earlier). The client should then operate with the particular component instance which owns the valid ticket. All commands that change state as a part of the ACADA command tree have to carry a control ticket.

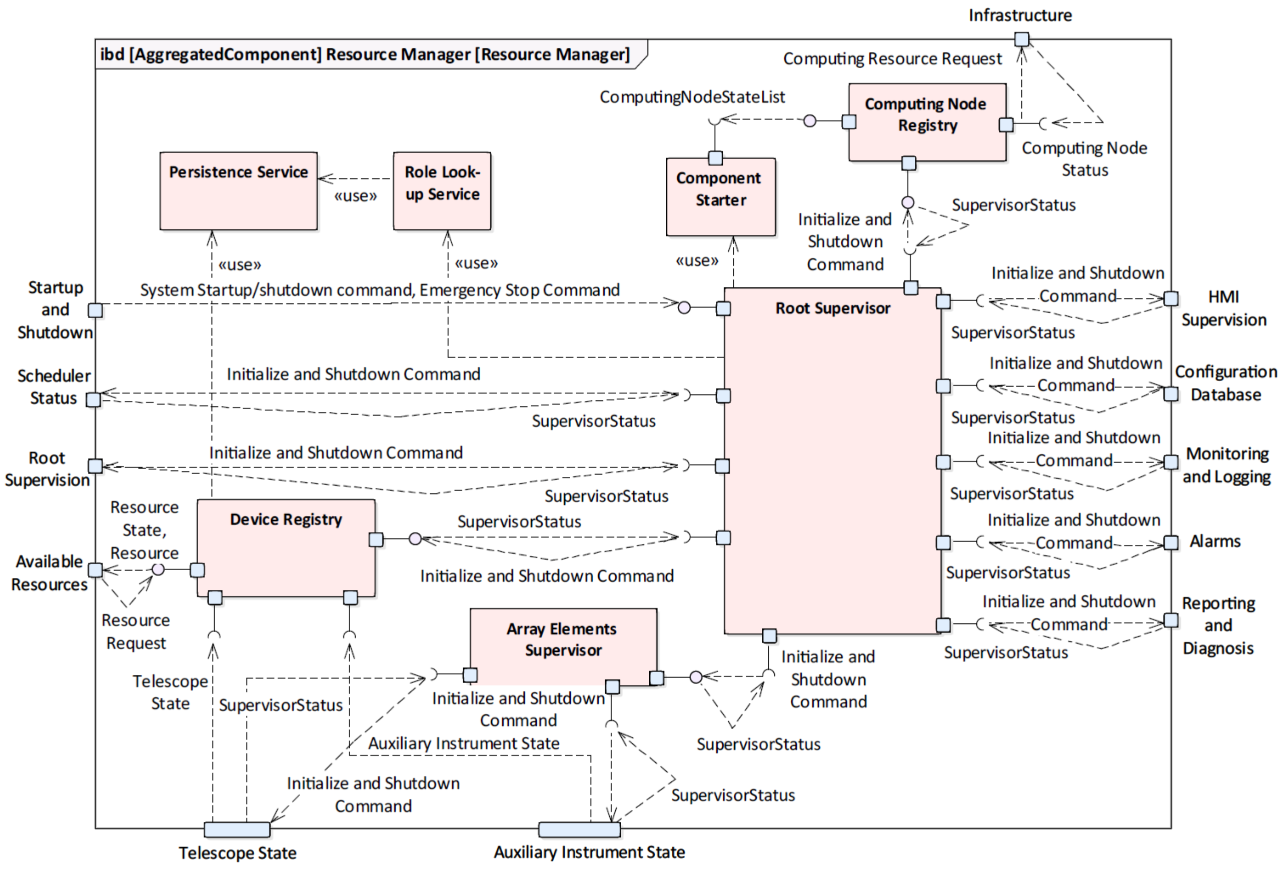


Figure 2: Diagram showing the logical view of the Resource Manager component. Only the highest-level components, and the most relevant data elements and interfaces are shown.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA092

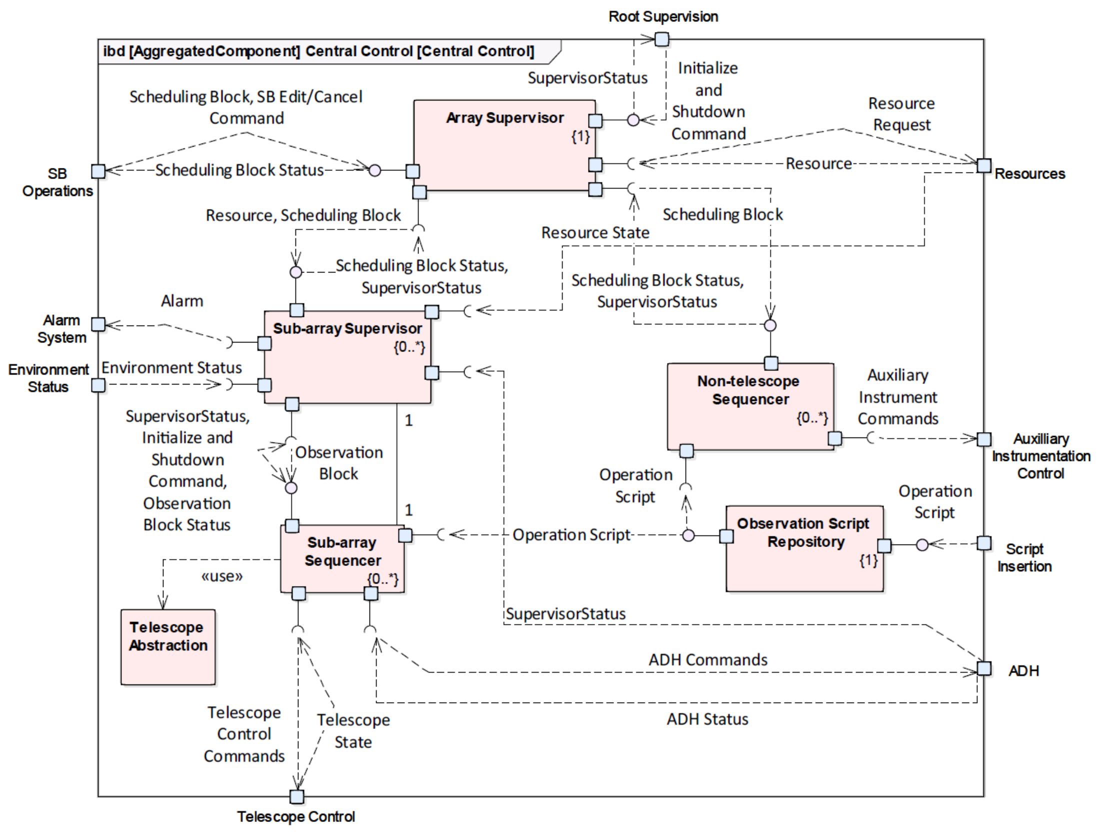

Root Supervision
ibd [AggregatedComponent] Central Control [Central Control]
SupervisorStatus
Initialize and Shutdown Command
Scheduling Block, SB Edit/Cancel Command
Array Supervisor {1}
Resource Request
SB Operations
Scheduling Block Status
Resource
Resources
Resource, Scheduling Block
Scheduling Block
Resource State
Scheduling Block Status, SupervisorStatus
Scheduling Block Status, SupervisorStatus
Alarm
Alarm System
Sub-array Supervisor {0..*}
Environment Status
Environment Status
Non-telescope Sequencer {0..*}
Auxiliary Instrument Commands
Auxilliary Instrumentation Control
SupervisorStatus, Initialize and Shutdown Command, Observation Block Status
Observation Block
1
1
Operation Script
Operation Script
Observation Script Repository {1}
Sub-array Sequencer {0..*}
Operation Script
Script Insertion
«use»
Telescope Abstraction
SupervisorStatus
ADH Commands
ADH
Telescope Control Commands
Telescope State
ADH Status
Telescope Control

Figure 3: Logical view of the Central Control component, representing the main components of the system.

CENTRAL CONTROL

The Central Control component implements the execution of scheduling blocks received from the Scheduler sub-system. CC uses the monitoring system to oversee the conditions to continue the execution of a SB, and cancels that execution if the conditions become adverse. Figure 3 shows the logical view of the Central Control component.

CC consists of the following components:

- **Array Supervisor:** A single Array Supervisor exists per site. Receiving new sub-array actions from the Scheduler causes the array supervisor to respond, instantiating as needed any sub-array supervisors. When creating the Sub-array Supervisor, the Array Supervisor hands over run-time information such as the list of telescopes allocated to the sub-array to act on. The Array Supervisor is supervised by the Root Supervisor of the Resource Manager. The Array Supervisor provides SB execution statuses to other subsystems; in particular, it informs the Scheduler when an observation has been completed.
- **The Sub-array Supervisor** receives from the Array Supervisor a list of telescopes and a sub-array action to execute on those telescopes. Given the sub-array action type, the Sub-array Supervisor has the facility to determine which sub-array sequencer to instantiate. It derives this from an identifier in the scheduling block. After initialization, the Sub-array Supervisor asks the sub-array sequencer whether any additional components (e.g. ADH) need to be initialized and supervised. The sub-array supervisor also checks that the conditions associated with that sub-array, such as good weather conditions or allocation of telescopes are being fulfilled. These conditions are specified in the scheduling block. Whenever one of them fails to be met, the observations are stopped. The Sub-array Supervisor informs the Array Supervisor when processing of the corresponding sub-array action has come to an end.
- **The Sub-array Sequencer** is responsible for the execution of one sub-array action, as derived from the scheduling block. This corresponds to the execution of a script, which governs the actions at the sub-array level. The scripting environment provides the means to execute the main observatory operation modes. The tasks or script to be executed via the Scripting Environment are extracted from the Operation Scripts Repository. Being responsibility of the sub-array sequencer, it steers any non-telescope-related components required to support the sub-array action. The Sub-array Sequencer also executes a sequence of steps and synchronizes the actions of the individual telescopes. For this, it uses the abstraction services of telescopes. Figure 4 illustrates the functional diagram of Sub-array Sequencer script execution.
- **The Non-telescope Sequencer** controls non-telescope related hardware assemblies, specifically those that are not directly involved in sub-array actions (such as an all-sky monitoring device). Non-telescope sequencers are supervised by the Root Supervisor of the Resource Manager.

- **The Telescope Abstraction** provides a convenient interface to be used by the sub-array sequencer. It makes it easy to deal with one individual telescope. The prime focus here is uniformity of access across telescope types.
- **The Observation Script Repository** stores the different observation scripts that define the observation tasks to be performed. At its simplest level this is a directory location in the system used by the sequencer to retrieve the observation scripts. Eventually, the Observation Script Repository will be implemented as a database or a code repository.

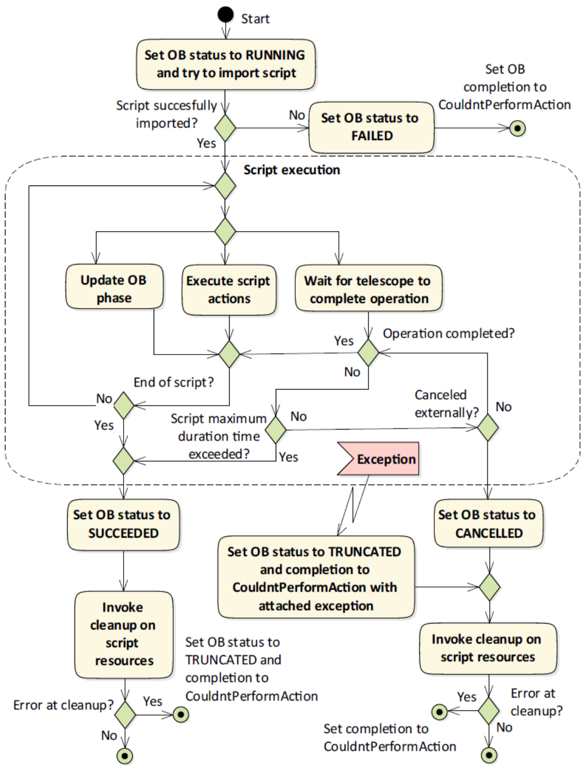
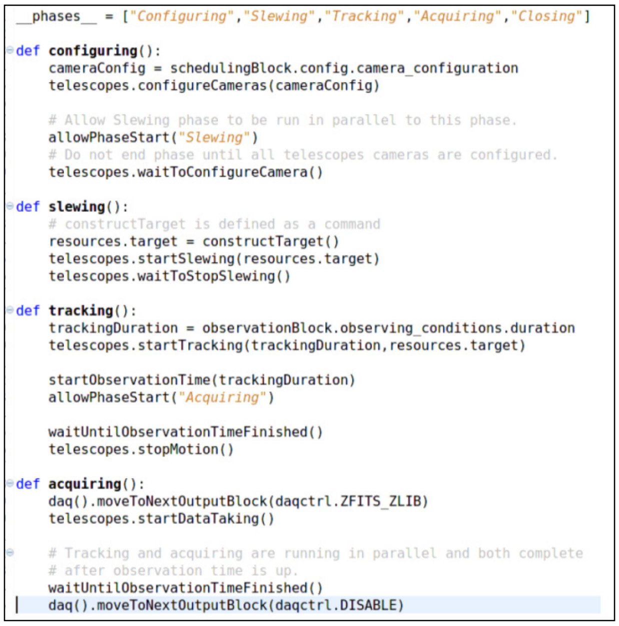

Start
Set OB status to RUNNING and try to import script
Script succesfully imported?
No
Yes
Set OB status to FAILED
Set OB completion to CouldntPerformAction
Script execution
Update OB phase
Execute script actions
Wait for telescope to complete operation
Operation completed?
Yes
No
End of script?
No
Yes
Script maximum duration time exceeded?
No
Yes
Canceled externally?
No
Exception
Set OB status to SUCCEEDED
Set OB status to TRUNCATED and completion to CouldntPerformAction with attached exception
Set OB status to CANCELLED
Invoke cleanup on script resources
Set OB status to TRUNCATED and completion to CouldntPerformAction
Invoke cleanup on script resources
Error at cleanup?
Yes
No
Set completion to CouldntPerformAction
Yes
Error at cleanup?
No

Figure 4: Sub-array Sequencer script execution.

Scripting Environment

The scripting environment provides the means to execute the main observatory operation modes. These are implemented as high-level Python scripts, and are executed by the Sub-array Sequencer component. This environment simplifies the script code by specifying a standard way of structuring the scripts and provides ways to test it. Most observation operations follow a standard set of phases (e.g. telescope camera configuration, slewing of telescopes, data acquisition). The scripting environment therefore allows a script to be defined in terms of phases. These detail conditions for completing a phase and moving on to the next one, or for running phases in parallel. The latter enables reduction of the configuration time and an increase of data acquisition time. The scripting environment supports the definition of commands for common operations shared between various operation modes. It provides a convenient access to the telescope and other resources, where definitions are imported automatically based on context. An example of a high-level Python script is shown in Fig. 5.


```python
__phases__ = ["Configuring","Slewing","Tracking","Acquiring","Closing"]

def configuring():
    cameraConfig = schedulingBlock.config.camera_configuration
    telescopes.configureCameras(cameraConfig)

    # Allow Slewing phase to be run in parallel to this phase.
    allowPhaseStart("Slewing")
    # Do not end phase until all telescopes cameras are configured.
    telescopes.waitToConfigureCamera()

def slewing():
    # constructTarget is defined as a command
    resources.target = constructTarget()
    telescopes.startSlewing(resources.target)
    telescopes.waitToStopSlewing()

def tracking():
    trackingDuration = observationBlock.observing_conditions.duration
    telescopes.startTracking(trackingDuration,resources.target)

    startObservationTime(trackingDuration)
    allowPhaseStart("Acquiring")

    waitUntilObservationTimeFinished()
    telescopes.stopMotion()

def acquiring():
    daq().moveToNextOutputBlock(daqctrl.ZFITS_ZLIB)
    telescopes.startDataTaking()

    # Tracking and acquiring are running in parallel and both complete
    # after observation time is up.
    waitUntilObservationTimeFinished()
    daq().moveToNextOutputBlock(daqctrl.DISABLE)
```

Figure 5: Example of a high-level sub-array sequencer script (written in Python).

CONCLUSIONS

In this paper, we present the main design features and the current implementation status of the RM&CC system, a part of the ACADA system of CTA. This prototype is intended as a proof of concept, applying a model driven approach to component based modelling of the ACADA system. The current version of the RM&CC system is capable of executing basic observation modes and running of multiple operations on various sub-arrays simultaneously. It also provides the main functionality of a supervision tree that includes dynamic instantiation, start, supervision, shutdown and replacement a supervised component with a successor, in the case it vanished or reached an error state.

RM&CC has been successfully integrated with the Scheduler and HMI sub-systems.

Our plans for development in the immediate future include the following tasks:

- Extending the scripting environment, to support a full set of CTA observatory operation modes.
- Expanding the functionality, performance and stability tests of the system.
- Designing and implementing a Scripts Repository environment to handle high-level Python scripts submitted by user.
- Continuing integration efforts with other ACADA sub-systems.

REFERENCES

[1] B. S. Acharya *et al.,* “Introducing the CTA concept”, Astroparticle Physics, vol. 43, pp. 3-18, 2013. doi:10.1016/j.astropartphys.2013.01.007

[2] I. Oya *et al.,* “The Array Control and Data Acquisition System of the Cherenkov Telescope Array”, presented at ICALEPCS’19, New York, NY, USA, October 2019, paper WEMPR05, this conference.

[3] E. Antolini *et al.,* “Quality Assurance of SCADA System of the Cherenkov Telescope Array Observatory”, presented at ICALEPCS’19, New York, NY, USA, October 2019, paper MOMPL001, this conference.

[4] K. Pohl, H. Hönninger, R. Achatz, and M. Broy (Eds.) *Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology*, Berlin: Springer, 2012. ISBN:978-3642346132

[5] SysML – "OMG Systems Modeling Language 1.4", OMG 2015, https://www.omg.org/spec/SysML/1.4/

[6] OMG UML – "OMG Formal Versions of UML", OMG 2017, https://www.omg.org/spec/UML/2.5.1/

[7] I. Oya *et al.,* “The software architecture to control the Cherenkov Telescope Array”, in *Proc. SPIE 9913, Software and Cyberinfrastructure for Astronomy IV*, paper 991303, 2016. doi:10.1117/12.2232520

[8] G. Chiozzi *et al.*, “CORBA-based Common Software for the ALMA project”, in *Proc. SPIE 4848*, 43, 2002. doi:10.1117/12.461036

[9] M. Füssling *et al.,* “Status of the array control and data acquisition system for the Cherenkov Telescope Array” in *Proc. SPIE 9913, Software and Cyberinfrastructure for Astronomy IV*, paper 99133C, 2016. doi:10.1117/12.2233174

[10] J. Colome *et al.,* “Artificial intelligence for the CTA Observatory scheduler”, in *Proc. SPIE 9149, Observatory Operations: Strategies, Processes, and Systems V*, paper 91490H, 2014. doi:10.1117/12.2057090

[11] E. Lyard and R. Walter, “End-to-end data acquisition pipeline for CTA”, in *Proc. ICRC’17*, Busan, Korea, 2017. doi:10.22323/1.301.0843

[12] I. Sadeh, I. Oya, J. Schwarz, E. Pietriga, and D. Dezman, “The Graphical User Interface of the Operator of the Cherenkov Telescope Array”, in *Proc.* ICALEPCS’17, Barcelona, Spain, Oct. 2017, paper TUBPL06, pp. 186-191. doi:10.18429/JACoW-ICALEPCS2017-TUBPL06

[13] Erlang - Supervision Principles, http://erlang.org/doc/design_principles/sup_princ.html