Journal of Physics: Conference Series **396** (2012) 012052 doi:10.1088/1742-6596/396/1/012052

Simultaneous operation and control of about 100 telescopes for the Cherenkov Telescope Array

P Wegner[1], J Colomé[3], D Hoffmann[4], J Houles[4], H Köppel[1], G Lamanna[5], T Le Flour[5], A Lopatin[6], E Lyard[7], D Melkumyan[1], I Oya[1], L-I Panazol[5], M Punch[8], S Schlenstedt[1], T Schmidt[1], C Stegmann[1], U Schwanke[1] and R Walter[7] for the CTA Consortium[9]

E-mail : Peter.Wegner@desy.de

[1]Deutsches Elektronen-Synchotron, DESY, Platanenallee 6, D-15738 Zeuthen, Germany,
[2]Institut für Physik, Humboldt-Universität zu Berlin, Newtonstrasse 15, D-12489 Berlin, Germany;
[3]Institut de Ciències de l'Espai, IEEC-CSIC, Campus UAB, Facultat de Ciències, Torre C5 par-2, Bellaterra, 08193, Spain;
[4]Centre de Physique des Particules de Marseille 163, avenue de Luminy Case 902, 13288 Marseille cedex 09, Aix-Marseille Université, CNRS/IN2P3, Marseille, France;
[5]Laboratoire d'Annecy-le-Vieux de Physique des Particules, Universite´ de Savoie, CNRS/IN2P3, F-74941 Annecy-le-Vieux, France;
[6]Erlangen Centre for Astroparticle Physics (ECAP), University of Erlangen-Nuremberg, Department of Physics, Erwin-Rommel-Str. 1, D-91058 Erlangen, Germany;
[7]ISDC, Geneva Observatory, University of Geneva, Chemin d'Ecogia 16, CH-1290 Versoix, Geneva, Switzerland
[8]Laboratoire d'Astroparticule et Cosmologie-APC CNRS, Université P7, Observatoire de Paris, CEA
[9]See http://www.cta-observatory.org for full author & affiliation list

Abstract. The Cherenkov Telescope Array (CTA) project is an initiative to build the next generation ground-based very high energy (VHE) gamma-ray instrument. Compared to current imaging atmospheric Cherenkov telescope experiments CTA will extend the energy range and improve the angular resolution while increasing the sensitivity up to a factor of 10. With about 100 separate telescopes it will be operated as an observatory open to a wide astrophysics and particle physics community, providing a deep insight into the non-thermal high-energy universe. The CTA Array Control system (ACTL) is responsible for several essential control tasks supporting the evaluation and selection of proposals, as well as the preparation, scheduling, and finally the execution of observations with the array. A possible basic distributed software framework for ACTL being considered is the ALMA Common Software (ACS). The ACS framework follows a container component model and contains a high level abstraction layer to integrate different types of device. To achieve a low-level consolidation of connecting control hardware, OPC UA (OPen Connectivity-Unified Architecture) client

functionality is integrated directly into ACS, thus allowing interaction with other OPC UA capable hardware. The CTA Data Acquisition System comprises the data readout of all cameras and the transfer of the data to a camera server farm, thereby using standard hardware and software technologies. CTA array control is also covering conceptions for a possible array trigger system and the corresponding clock distribution. The design of the CTA observations scheduler is introducing new algorithmic technologies to achieve the required flexibility.

1. The Cherenkov Telescope Array research infrastructure

CTA is a new and unique facility to probe electromagnetic radiation from space in the very high energy domain – at energies between tens of Giga-electronvolts to hundreds of Tera-electronvolts – with unprecedented sensitivity, energy coverage, and angular resolution.

CTA will serve a number of science communities, including astronomy and astrophysics, cosmology, astroparticle physics, and particle physics. Its measurements will complement those in the radio, infrared, optical, X-ray and gamma-ray wavebands. Furthermore, CTA will form a crucial partner to facilities using other messengers that probe extreme processes in the universe, including ultra-high energy cosmic rays, high energy neutrinos, and gravitational waves.

CTA will consist of arrays of Imaging Atmospheric Cherenkov telescopes, detecting the faint light flashes that are generated by gamma-ray induced particle cascades in the upper atmosphere.

Three different types of telescopes are proposed, optimized for the detection of gamma rays over almost four orders of magnitude in energy:

(i) in the energy range of a few 10 GeV to 100 GeV over a detection area corresponding to a fraction of a square kilometer

(ii) for the detection in the 100 GeV to 10 TeV range over about a square kilometer, and

(iii) for the detection in the 10 TeV to 100 TeV range over up to 10 square kilometers.

In each case the increasing detection area compensates the decrease in the flux of very high energy gamma rays with increasing energy. The basic detection methodology is well understood and builds upon the extensive experience in the CTA Consortium with instruments like H.E.S.S. – winner of the EC Descartes Prize in 2006 – and MAGIC and VERITAS. The design features of CTA allow it to deliver far superior performance compared to previous instruments, achieved through the larger amount of light collected from each gamma-induced cascade, the increased number of views provided for each cascade, through its significantly larger field of view, and through its larger overall detection area. Optimized design of the telescopes will make the arrays affordable and reliable.

In terms of energy density, the high energy Universe is comparable to the more familiar Universe of thermal radiation emitted by hot celestial objects – yet it remains far less explored. The data delivered by CTA will allow the detailed study of cosmic particle accelerators in and beyond our galaxy, directly probing the high-energy Universe. With its low energy threshold and high sensitivity, CTA will be able to detect sources at cosmological distances, allowing the probing of cosmological parameters, and the investigation of the evolution of matter and of radiation fields in the Cosmos. The search for dark matter through its self-annihilation into gamma rays will be a key theme of CTA, possibly resulting in the detection of the elusive particles that are thought to make up most of the mass of the Universe. CTA will address other fundamental physics questions, for example, its sensitivity to the effects of quantum gravity occurring at the Planck scale will allow searches for deviations from the theory of relativity. CTA will provide science early in its construction. Even with only 10% of its final telescopes deployed, CTA will surpass existing instruments and will start to deliver science, allowing data reduction tools to be tested and perfected, and to ramp up gradually the operation of the facility.

It is proposed to install arrays at two sites, in both the Northern and Southern hemispheres, providing full sky coverage. The detailed array layout at each site will reflect the opportunities

provided, with the Southern array emphasizing observations of Galactic objects in addition to the extragalactic objects that more or less uniformly populate the sky [1,2].

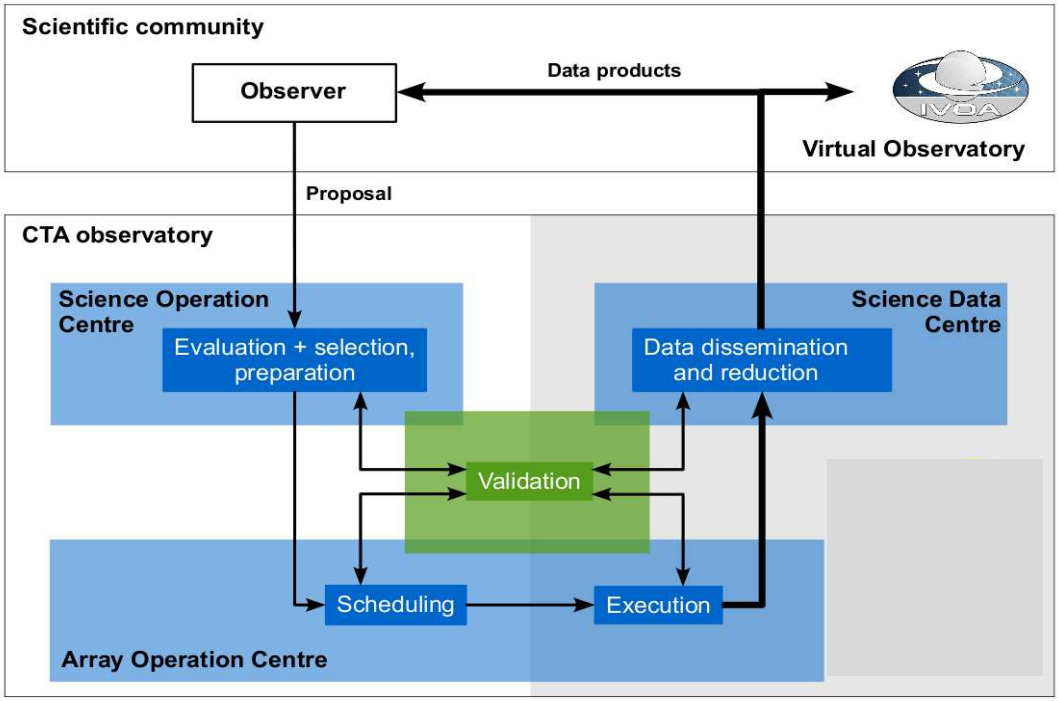

Figure 1. CTA Observatory

2. CTA Data Acquisition and Array Control – ACTL

The ACTL project comprises the Data Acquisition (DAQ), the entire Array Control of all Cherenkov telescopes and the Scheduling of observations.

The array control part includes the slow control and monitoring of all telescope devices, especially the cameras. The DAQ activities comprise buffering the readout data, processing of global trigger decisions and the subsequent building of camera-dependent events. DAQ computing hardware will consist of standard networking and computing hardware components, i.e. Ethernet local area networks and server farms. Similarly, common software layers and standard software frameworks will be used inside the DAQ and control layer wherever possible. Figure 2 shows a schematic view on the DAQ and control layout.

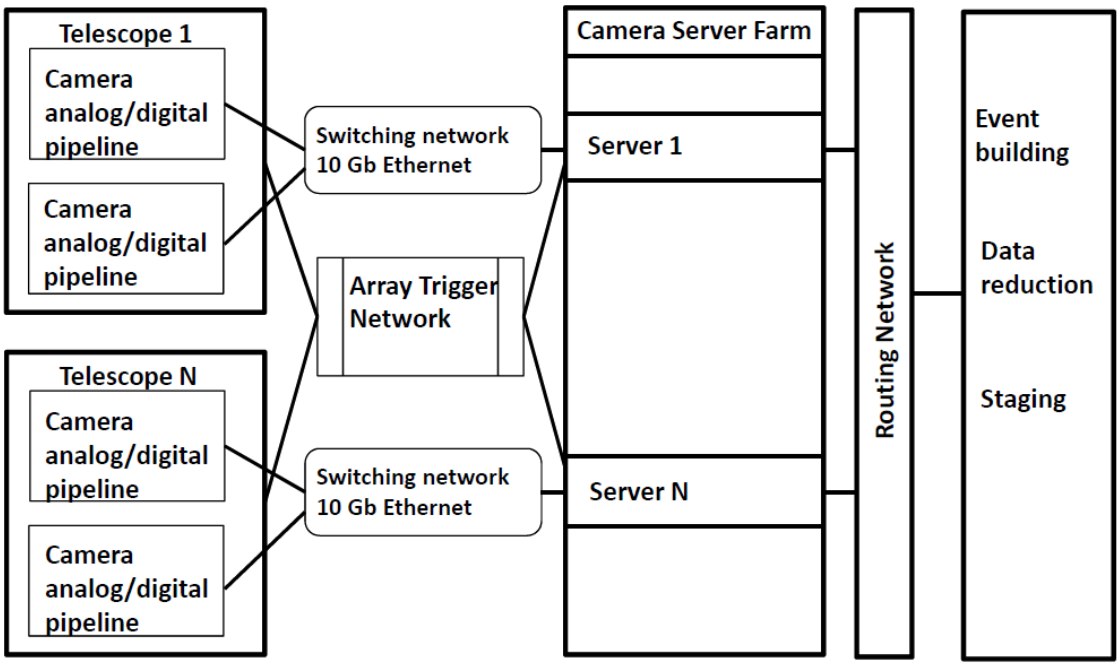

Figure 2. DAQ and Control network layout

3. Common software layer

3.1. Overview

The main part of the Common Software in ACTL will consist of a middleware layer, which extends the capabilities of the operating system and provides a basis for hosting and supporting the application layer. In doing so, it allows components of a distributed system to communicate and manage data in a homogeneous and simplified manner. Other typical and important functionalities of control middleware environments are higher-level communication, error handling, event notification, and process control, support of state machines. By using standard communication software, which in particular provides the exchange of objects, the distribution of the control software services and applications over different hosts and operating systems can be realized in an easy and general way.

Typical middleware frameworks for control covering different levels of functionality (often also referred as SCADA - Supervisory Control And Data Acquisition) are SIMATIC WinCC Open Architecture (former PVSS[12]), EPICS[13], ICE[14], ACS[3], DOOCS[15], Tango[16], and for low level control, LabView[17]. As CTA will comprise two arrays of about 100 telescopes of different sizes, each composed of large number of devices, for a stable and easy to maintain control system an object oriented control framework stands up as a natural choice. Moreover, due to the nature of CTA as a worldwide collaboration of universities and research centers, with different software development cultures, the creation of the array control software can become a sophisticated task. In order to mitigate the expected complication from the beginning, the selection of common software is necessary to funnel into a uniform architecture the disparate styles and approaches naturally thriving within the large number of participating institutes.

Due to the similar level of complexity and requirements between the *Atacama* Large Millimeter/submillimeter Array (ALMA) and CTA, the ALMA Control Software (ACS) middleware [3] is considered as a natural candidate to be investigated as a common Control and DAQ framework.

3.2. ALMA Common Software

ACS is a distributed middleware framework, which sits on top of the operating system, realizing a *container/ component* implementation based on CORBA, i.e. the core of ACS is based on a distributed component model, with components implemented as CORBA objects in any of the supported programming languages C++, Java, and Python. The ACS components form the basis for high-level control entities and for the implementation of devices such as a telescope drive system. The ACS provides common CORBA-based services such as logging, error and alarm management, and configuration database and lifecycle management, while hiding the complexities of CORBA programming. The framework itself, implements a Container/Component model which goes beyond the original CORBA philosophy but which was successfully used, for example in Sun's EJB or Microsoft's COM+ frameworks. The European Southern Observatory (ESO), based in Garching, Germany, is mainly developing ACS for ALMA.

ACS provides the following high level abstraction layers:

- **Base tools (external products):** This layer contains base tools, like the CORBA middleware (TAO implementation) itself, that are distributed as part of ACS to provide a uniform development and run time environment on top of the operating system for all higher layers and applications. These are essentially off-the-shelf components and with ACS itself simply providing packaging, installation and distribution support. This ensures that all installations of ACS (development and run-time) will have the same basic set of tools.
- **Core packages:** This second layer ensures standard interface patterns and implements essential services, necessary for the development of any application. It contains the Basic Access Control Interface (BACI), Configuration Database, Data Channel, Error System, Logging System and Time System.
- **Services:** The third layer implements higher-level services. Among these the Management and Access Control Interface (MACI) Design patterns, protocols and meta-services for centralizing access to ACS services and Components, to manage the full life cycle of Components, including persistence, and to supervise the state of the system. It contains also the Archiving system – the endpoint of all control data.
- **Application Frameworks and High-level APIs:** The fourth and last layer provides higher-level APIs and tools. The main goal of these packages is to offer a clear path for the implementation of applications, in order to obtain implicit conformity to design standards.

The Basic Access Control Interface is a specification outlining the design patterns in the CORBA IDL language that may be used to specify the structure of the control system according to the device - property - monitor paradigm. It is a restriction of all possible specifiable IDL interfaces to a small "BACI-compliant" subset made with the purpose of being able to express a given control system functionality uniquely. In contrast, the Management and Access Control Interface is a fixed set of interfaces that parameterize control and management of BACI objects. Calls invoked through MACI return information about BACI objects, shut them down or turn them on, and perform similar functions. MACI obviously operates on a different level than BACI, and is therefore specified independently. Additionally ACS supports a mechanism to encapsulate the lower level access interfaces - called DeviceIO or shortly DevIO. A DevIO file is a class that contains all the required code (and libraries) for communication with a device, hiding these details from the ACS component itself, and allows ACS components to be created without being bound to a particular communication mechanism. For example and of particular importance for CTA, ACS and OPC UA (see corresponding section on this document) can interface via a DevIO class that implements an OPC UA client. The ESO team has developed a C++ DevIO for OPC UA [7] while within the ACTL working groups a Java equivalent was developed. The configuration of ACS is based on the use of XML. Our experience with the corresponding configuration data base (CDB) was that it became complicated and error prone, in particular when the project starts to scale. A better approach seems the one adopted by

ALMA (the hibernate based version, called TMCDB) or the use structured storage (for example with MongoDB).

The use of a life-cycle state machine for the components of a subsystem can constitute an advantage as it simplifies the integration of the instrumentation control system and allows a common and well known life-cycle management for all subsystems and devices. ACS implements a state machine that has been designed for the life-cycle management of the ALMA subsystems: the *Master Component* State Machine. These Master Components are coded in Java and can be configured via XML files in an analogous way as ACS components are configured in the CDB. It is possible to create Master Component state machines with the code generation framework mechanism under consideration by ACTL group (The *Comodo* software, see below).

3.3. Code Generation

A code generator was development inside the ACS project to optimize and speed up the creation of ACS components, which due to the complex structure and especially the definition of XML schemas of configuration data can be a repetitive and error prone task [5]. The code generator accelerates the development process by using predefined metamodels, in this case Unified Modelling Language (UML) diagrams in a XMI 2.0 format standard used as input in a Model Driven Code Generator (MDCG). A new Software component, Comodo, is an ACS open source code generator project supported by ESO. One of the main objectives of the Comodo project is to implement a Java component code generator for ACS, including IDL interfaces and Configuration Database (CDB) parameters. The UML models are created using the licensed software MagicDraw UML. These UML models are used to create the input required by Comodo, which then generates the interface definition language (IDL), configuration database files and Java implementation skeleton. By using the Java skeleton code, an extension to a fully functional ACS module was built, enabling a test of OPC UA client functionality in communication with an OPC UA server. This last approach was tested with a prototype Weather Station early stage component version [6].

3.4. ACS Evaluation

ACS is currently being evaluated in a number of CTA participating institutes. One major effort in the evaluation of ACS is being carried out in the context of the control software of the Medium Size Prototype Telescope (MST, [3]) prototype being developed by DESY Zeuthen and Humboldt University Berlin. It has been decided to use ACS for controlling the MST prototype because it will provide an excellent platform to test the behaviour of ACS with CTA hardware, while serving as a learning platform. Beginning with the drive system, a set of devices around the prototype telescope will be controlled by ACS – among them a weather station, CCD cameras for the calibration of camera movement, pointing and mirror adjustment procedures, and Active Mirror Control Units (AMCs) for aligning the telescope mirrors by use of actuators ([9], [6], [8]).

The ACS software default operating system is Scientific Linux (SL). Using the available source Tar-Balls from ESO, rpm packages are built for installing ACS under SL5, SL6 in 32-Bit and 64-Bit releases. Moreover ACS has also been tested under Fedora Core.

4. ACS hardware interface layer

4.1. Slow control software

Each telescope and array environment will have to be monitored in order to get, in real time, the current status and the availability of each component of the array. The various set of devices (PLC, front-end boards, specific electronic boards) composing telescopes, camera, security system etc., will publish a large amount of data used to indicate the current status of these devices. Also, a subset of devices should be controlled via a set of commands. To avoid dependencies on proprietary interfaces, such as field busses and serial lines, a common standard interface based on Ethernet and the IP protocol should be used.

4.2. OPC UA

For the automated systems driven by PLCs (e.g. the telescope drive system), an already used common software layer is based on the OPC UA standard [18]. From a software point of view, it is required to access all devices in a most common and standardized way. To access device information, embedded software layers must not depend on a specific implementation (dedicated or propriety hardware busses and communication layer, libraries, APIs). Moreover possible hardware changes should not affect the higher layer of communication and control software.

The OPC Foundation [16] has released the OPC specifications in 2006 as an improvement upon its predecessor, Classic OPC. OPC UA is designed to be independent of the operating system, by using SOAP/XML over HTTP and a TCP/IP protocol. OPC UA can be deployed on Linux, Windows XP Embedded, VxWorks, Mac, Windows 7, and Classical Windows platforms. This also enables OPC to communicate over networks through WEB Services. Finally, the main features of OPC have been preserved which means that (classic MS Windows based) OPC DA and OPC UA environments can be mixed.

It is planned that inside ACTL all control devices will communicate via an OPC UA server interface, either by using native OPC UA server firmware (as already existent in most of the drive PLCs) or by implementing OPC UA server software on the device access level. Therefore all information of different devices can be accessed in a standard manner by OPC UA clients. OPC UA client functionality is already implemented inside ACS as part of the low level DevIO layer.

For testing and prototyping the OPC UA environment, cross-platform software development frameworks from Unified Automation Company and Prosys has been chosen. The provided software development kits (SDKs) allow developing OPC UA server and client applications by using the languages C++ and JAVA, respectively.

5. The DAQ system

5.1. DAQ principles

The CTA DAQ system will acquire the data coming from the telescope cameras. It implies data transfer for each telescope from the front-end electronics/buffers to the event builders, real-time computing (central triggering, event building, quick-look analysis...) and final transfer of the raw products to the on-site archive. To achieve this goal, the designed system must be able to operate with a high throughput (up to several Gbps). It must be robust and automatically recover from errors. Its cost needs to be minimized, for example by using commercial off-the-shelf components and standard technologies with a reasonable expected time-on-market as much as possible.

Each telescope will be assigned a Camera Server as part of a server farm, which collects data from the front-end modules via Ethernet. Such a server would allow reconstructing and buffering the individual events coming from each individual telescope in the computer memory, thus allowing to buffer up to several seconds of data.

5.2. Simulation of Camera DAQ and data transfer services

According to the proposed global networking architecture of the DAQ/Control system, today's standard commodity hardware and software should be used to test the basic functionalities required by CTA ACTL. For the purpose of validation of the local Event Builder as well as in-situ tests on telescope prototypes, a simulator device for the Camera Server DAQ is being developed, which simulates the fully-fledged output of any type of CTA camera on 300 physical Ethernet sockets. In a first series of tests, we have deployed 300 simulator software agents on ten servers, each of them equipped with two 1000baseT Ethernet outputs respectively. Theses servers are connected to the switch which is connected to the camera server through two 10 Gb links [9].

The data delivery service will be a central piece of the DAQ system. Various options are available ranging from a fully home-built system, to commercial off-the-shelf packages. One promising specification to address communication needs of distributed applications is the "Data Distribution Service for Real-Time Systems" (DDS, [19]) governed by the Object Management Group (OMG). DDS is following a Data-Centric Publish-Subscribe (DCPS) approach for communication and provides data transfer constraints according to Quality of Service mechanisms, i.e. ensuring deterministic data delivery and controlling the bandwidth. Various DDS packages - OpenDDS, RTI DDS, OpenSplice DDS - were tested so far but no decision regarding which one to use has been taken yet.

6. The Array Trigger and Clock distribution

For CTA, a global triggering system is under discussion. We are considering, as baseline approach, that each camera is triggered locally, i.e. independent from all other telescopes. Based on the timing information of all local triggers, a global array trigger decision will be derived, by requiring, for example, two or more telescopes within a localized region of order 500m diameter to be triggered within a given coincidence time interval of order 100ns.

For current generation Gamma Ray experiments (HESS, HEGRA, MAGIC), such stereoscopic or array triggers have been implemented as a hardware coincidence of the camera triggers. The camera trigger signals are sent to a central point, corrected for different delays due to the pointing direction, and an array-trigger signal is returned to start the full camera readout for each triggered telescope if there was a coincidence. Such a hardware-based strategy is strongly disfavoured as a global array trigger of CTA, since the distribution of the array over around $10km^2$ would result in the introduction of much larger latency times. Also, for CTA there is the requirement to be able to operate independent sub-arrays; i.e. to reconfigure the coincidence strategy easily to form sub-arrays. Thus, a more flexible concept for an array trigger for CTA is considered.

It is based on local camera triggering and readout, and on the transmission of the camera-trigger time-tags to a central decision unit, where a search for coincidences is carried out at software level. The clock distribution from a central point will be done either by a custom-made solution integrated with the reception of the camera-trigger time-tags, or alternatively, the use of the White Rabbit concept - a recent Ethernet extension from CERN with sub-nsec precision local clocks [21,22] is being explored.

When a trigger from a Cherenkov Camera is formed in the Camera Local Trigger module (by coincidence between pixel comparators, in time and/or in topology) this signal serves to start the readout of the pixel Analogue Memories and ADC information. Each local trigger signal is then time stamped based on the timing of the local trigger with respect to the Cherenkov pulse. A stream of all time stamps of local Camera trigger signals is sent to the Array Trigger system where all time stamps can be compared with those coming from the Camera trigger of other telescopes. When the Array Trigger processor determines coincidences, the telescopes concerned are sent the orders to conserve and transmit the information of the events corresponding to the time-stamps.

7. The scheduler for CTA

7.1. Introduction

The CTA observatory will have to deal with a large variety of observation modes, ranging from the whole-array on single target configuration, to set-ups in which the array is split in to small groups of telescopes each with independent targets. This requires a well-designed, failure-tolerant software system to control all the CTA observatory systems. The large flexibility provided by the CTA array raises new challenges concerning the scheduling of the observations and other telescope related tasks, taking into account the state of the array and weather conditions and other constraints that may affect each of those tasks, and may prevent them from being executed at a certain time or forcing others to be executed at a determined time. The main purpose of the Scheduler for CTA (SCTA) is the allocation of multiple tasks to one single array or to multiple sub-arrays of telescopes, while minimizing the total execution time and maximizing the scientific return of the CTA projects. This is a key element in the control layer for the observatory time optimization.

Artificial intelligence technology is required to solve this problem of high mathematical and computational complexity. In most cases there is no single best solution for the planning system and, therefore, various mathematical algorithms (Genetic Algorithms, Ant Colony Optimization algorithms, Multi-Objective Evolutionary algorithms, etc.) are usually considered to adapt the application to the system configuration and task execution constraints. The scheduling time-cycle is also an important ingredient to determine the best approach. The scheduler will consider long- and short-term varying conditions to optimize the prioritization of tasks among the pending science, calibration and maintenance operations. A short-term scheduler, for instance, will incorporate the dynamic response and will have to find a good solution with the minimum computation time, providing the system with the capability to adapt, in almost real-time, the selected task to the varying execution constraints (i.e., health or status of the system components, environment conditions). A long-term scheduler, on the other hand, has to provide a full night or season planning to optimize the subsequent short-term selection and adding the long-term perspective. The algorithm selection might have an important impact on the final system performance. A tool to simulate the observatory operation is, then, required to check the suitability of each possible solution.

The SCTA has to interact with the array control system. It has to access the CTA data archive, where all the pending tasks (science, calibration and maintenance operations) are stored, and has to receive real-time housekeeping data to have a continuous knowledge of the system status. The communication interface has to guarantee the efficient exchange of data and must be based on ACS, the software infrastructure evaluated for this purposes in the ACTL work package framework.

Finally, the SCTA must be highly configurable and modular in order to be easily modified in case the scheduler does not reach the expected performance or the CTA system requirements suffer a significant change in the future.

7.2. SCTA design and development

Figure 3 illustrates the high level design of the application. This design includes different subsystems defined to implement the system features:

- The Scheduler is the process that receives the tasks, computes the schedule and returns the tasks to be executed.
- The Simulator is in charge of the full system simulation. It creates an observatory with environment and system conditions similar to those expected for CTA. It manages the execution of each task and the communication with the Scheduler.

- The CTA Common Software (CCS) includes the basic libraries that are used by different subsystems.
- The Input/Output interface is used to configure the basic settings of the application (including parameters of the scheduler and the simulator, the software environment where the processes run, etc., represented as XML files) and to handle the interaction with users. The latter is planned to be a web-based interface devoted to the specification of tasks to schedule and their execution constraints and, also, the simulation framework (site location, telescope and detector setting, telescope allocation to a sub-array, simulation period, etc.). The Output refers to the data obtained with the simulation that will be mainly used to check the scheduler performance.
- The database (DB) is an external entity that provides input and output data storage support to the main subsystems.

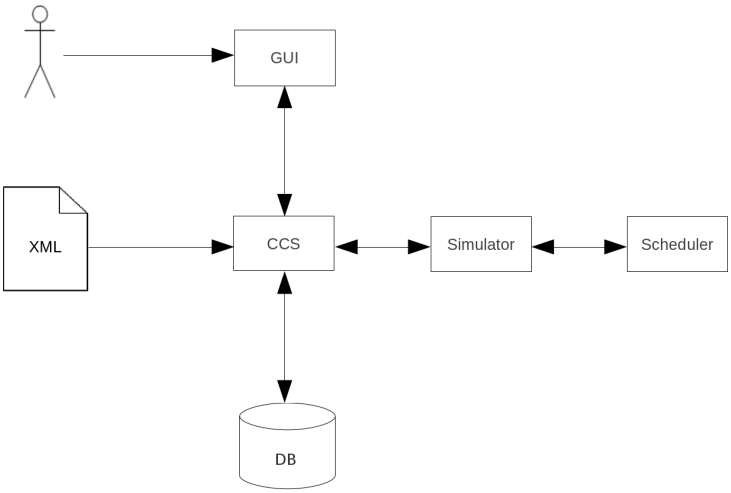

Figure 3. SCTA high level design

Figure 4 shows the designed architecture, including the ACS software infrastructure layer for subsystems communication and a detailed view of the Scheduler subsystem [22]. The most relevant components of the Scheduler subsystem are mentioned here:

- The Scheduler subsystem is an ACS component that allows the communication between the Simulator and the Scheduler subsystem.
- The GDSN component implements a scheduling algorithm as a Guarded Discrete Stochastic Neural Network following the explanations found [23]. A different algorithm might be used just adding a similar component and reconfiguring the application accordingly.
- The Events subsystem provides thread-safe event handling capabilities. It is used by the Scheduler and CTA Scheduling subsystem to allow asynchronous communications between the Simulator and the Scheduler while the last one is computing new schedules.

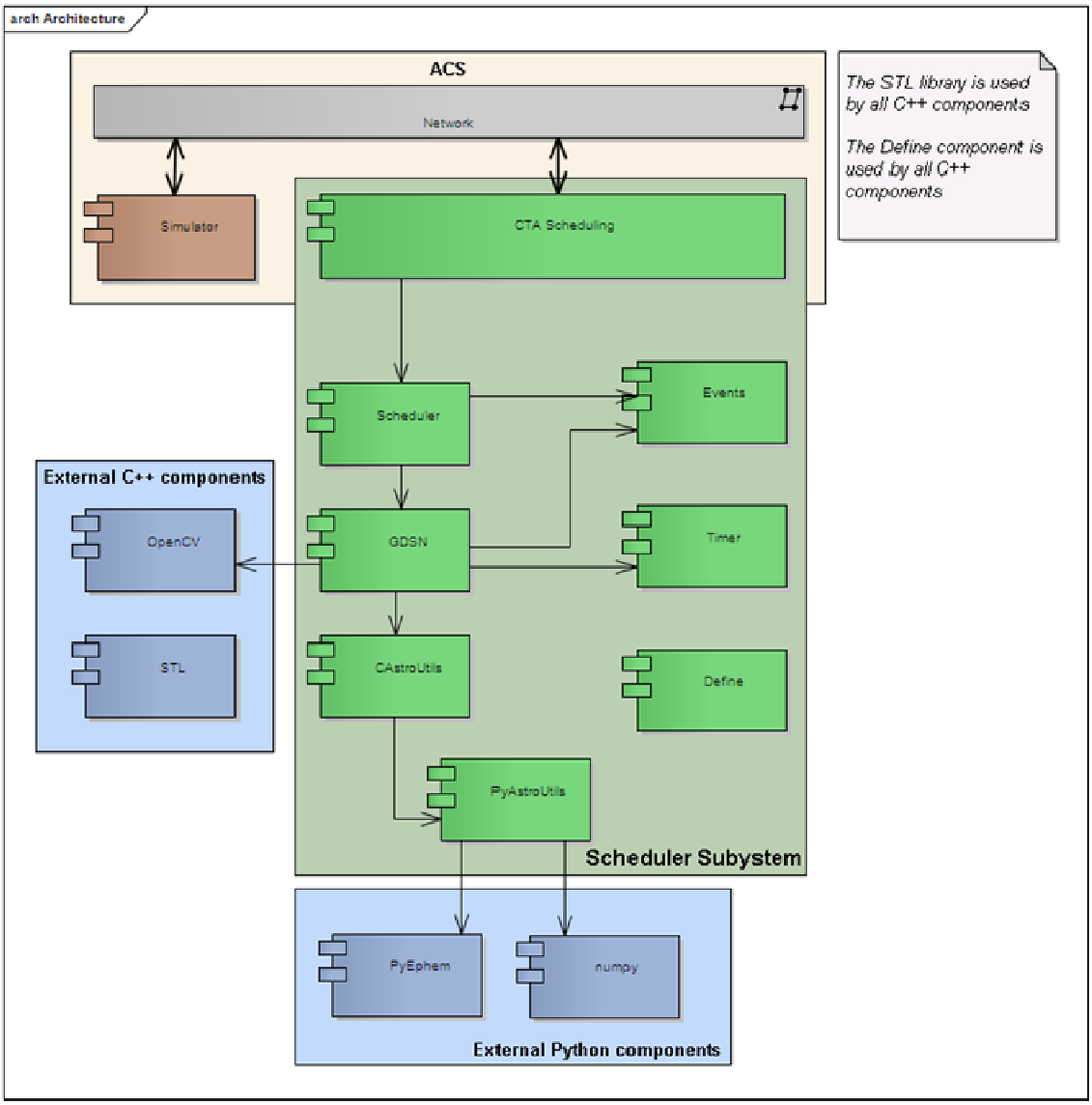

Figure 4. Scheduler Architecture (courtesy of P. Colomer, GTD S.I.)

8. Summary

Due to the widely distributed and complex structure of the planned Cherenkov Telescope Array the Design of Data Acquisition and Control system –ACTL- is characterized by a comprehensive multi-layer structure comprising the SCADA like framework ACS and the standardized device access layer OPC UA. Moreover, new technologies such as the Data Distribution Services for Real-Time systems and new algorithmic approaches are tested for being incorporated into the ACTL software. Comprehensive tests of all layers have already started and show promising results in the direction of establishing an efficient and reliable DAQ and control environment based on standard software and hardware components.

References

[1] CTA Consortium, *Design Concepts for the Cherenkov Telescope Array*, Experimental Astronomy Volume 32, Number 3, 193-316 (2011).
[2] CTA-PP, FP7-INFRASTRUCTURES-2010-1 INFRA-2010-2.2.10: CTA (Cherenkov Telescope Array for Gamma-Ray Astronomy)
[3] E. Davis, et al. *Mechanical Design of a Medium-Size Telescope for CTA*. CTA internal note
[4] Chiozzi G, et al., *The ALMA Common software: a developer friendly CORBA based framework*, SPIE Procs. 5496-23 (2004)
[5] Troncoso N, et al. *A Code Generation Framework for the ALMA Common Software*, SPIE Procs. 7740-121 (2010)
[6] Oya I, et al. *A Readout and Control System for a CTA Prototype Telescope*, Proceedings of ICALEPCS, Grenoble, France, 2011
[7] Di Marcantonio, P., et al. *Evaluation of Software and Electronics Technologies for the Control of the E-ELTInstruments: a Case Study*, proceedings of ICALEPCS, Grenoble, France, 2011
[8] Oya I, et al. "*Evaluating the control software for CTA in a medium size telescope prototype*." in these proceedings.
[9] Hoffmann D, et al. *Prototyping a 10Gigabit-Ethernet Event-Builder for a Cherenkov Telescope Array*, in these proceedings.
[10] Föster A, et al. *Mirror Development for CTA*, Proc of the 32nd International Cosmic Ray Conference, Beijing, China, 2011
[12] http://www.pvss.com
[13] http://www.aps.anl.gov/epics
[14] http://www.zeroc.com/ice.html
[15] http://tesla.desy.de/doocs
[16] http://www.tango-controls.org
[17] http://www.ni.com/labview
[18] http://www.opcfoundation.org
[19] http://www.opendds.org
[20] Serrano J, Alvarez P, Cattin M, Cota E G et al.,
The White Rabbit Project, in ICALEPCS, Kobe, Japan, 2009
[21] http://www.ohwr.org/projects/white-rabbit
[22] Colomer P, *Software Design Document for the CTA scheduler*, GTD-IEEC Ref: E1241-10.00_SDD-01, v0.1
[23] Johnston M D, Adorf H M, *Scheduling with Neural Networks - The Case of Hubble Space Telescope*, Computers & Operations Research, v.19, DOI: 10.1016/0305-0548(92)90045-7 (1992)

Acknowledgments

The Authors gratefully acknowledge support from the agencies and organizations forming the CTA consortium (http://www.ctaobservatory.org)
They also like to thank the ACS experts from ESO Garching, in particular G. Chiozzi and H. Sommer, for their excellent support.