ASTRI-Horn Legacy

Prepared by:	Name:	Vito Conforti	Signature:	[illegible]	Date:	Jan 14, 2021
Verified by:	Name:	J. Schwarz	Signature	[illegible]	Date:	Feb. 1, 2021
Approved by:	Name:	G. Tosti	Signature:	[illegible]	Date:	14/01/2021
Released by:	Name:	S. Scuderi	Signature	Salvatore Scuderi	Date:	05/02/2021

Main Authors: Vito Conforti, F. Gianotti, F. Lucarelli

Contributor Authors: G. Tosti, G. Schwarz, A. Bulgarelli, S. Lombardi

TABLE OF CONTENTS

INDEX OF FIGURES & TABLES

DOCUMENT HISTORY		
Issue	Date	Modification
1.0	14/01/2021	First release

1 Introduction

The ASTRI project (Astrofisica con Specchi a Tecnologia Replicante Italiana) was started by INAF (Istituto Nazionale di Astrofisica) in 2010 within the context of the "Progetti Bandiera" ("Flagship programs") of the Italian Ministry for Researcher and University (MUR, formerly MIUR). In the first phase of this project the ASTRI-Horn Cherenkov telescope was developed as an end-to-end prototype for the next generation of Imaging Atmospheric Cherenkov Telescopes (IACTs) of the Cherenkov Telescope Array (CTA) international observatory. In particular the ASTRI-Horn 4-m telescope was designed following the requirements for the CTA Small Sized Telescopes that will be dedicated to observe in the high energy part, above some TeV, of the energy range covered by CTA.

The ASTRI-Horn prototype was inaugurated in September 2014 and is located on mount Etna (Sicily) at 1730 m of altitude, at the Astronomical Station of Serra La Nave (SLN) of the Catania Astrophysical Observatory.

It is mainly a technological demonstrator. Indeed, it uses a dual mirror Schwarszchild-Couder telescope optical configuration, based on highly aspherical optics and it represents a novelty in the world of Very High Energy astrophysics. This particular design allowed the use of smaller camera pixels based on Silicon PhotoMultipliers (SiPMs) technology to substitute the larger Photomultiplier Tubes (PMTs), in use on current IACTs.

The second phase of the ASTRI project consists in the construction, deployment and operation of an array of 9 wide-field Cherenkov telescopes (The ASTRI Mini-Array), based on an upgraded version of the ASTRI-Horn prototype, at the Observatorio del Teide in Tenerife (Spain).

1.1 Purpose

This document describes the ASTRI – Horn software integration that will be taken as baseline for the ASTRI Mini-Array. This is part of the legacy of the ASTRI-Horn project used to develop the Final Architecture of the ASTRI Mini-Array.

1.2 Scope

The scope of this document is to provide the needed reference information to the designer and developer of the ASTRI Mini-Array software. Readers should not assume that procedures and tools described here will necessarily be used or mandated for the Mini-Array. While some, e.g., ACS and OPCUA, will certainly be carried over to the Mini-Array, others will be reviewed and may be revised or replaced taking account of experience gained with ASTRI-Horn and new developments in software engineering technology.

1.3 Content

1.4 Definitions and Conventions

1.4.1 Abbreviations and acronyms

- AMC - Active Mirror Control
- ASTRI - Astrofisica a Tecnologia Replicante Italiana
- CTA - Cherenkov Telescope Array
- DAQ -Data Acquisition system
- DHS - Data Handling System
- GUI - Graphical User Interface
- ICD - Interface Control Documents
- ICS - Instrument Control System
- INAF - Istituto Nazionale di Astrofisica
- MASS - Mini-Array Software System
- OCS - Observatory Control System
- PDE - Photon Detection Efficiency
- PDM - Photon Detection Module
- PMC - Pointing Monitoring Camera
- SiPM - Silicon Photomultiplier
- SLN - Serra La Nave
- SQM - Sky Quality Meter
- SST - Small Size Telescope
- TCS - Telescope Control System
- TCU - Technical Control Unit
- TDR - Technical Design Report
- THCU - Technical Health Control Unit

2 Applicable and reference documents

2.1 Applicable Documents

[AD1] ASTRI-INAF-DES-2100-001 ASTRI Mini-Array Top level software architecture

[AD2] ASTRI-INAF-PLA-2100-002 ASTRI Mini-Array Software Development Plan

[AD3] ASTRI-INAF-PRO-2100-001 ASTRI Mini-Array Software Integration Test Model

2.2 Reference Documents

[RD1] PROCEDURES OF SOFTWARE INTEGRATION TEST AND RELEASE FOR ASTRI SST-2M PROTOTYPE PROPOSED FOR THE CHERENKOV TELESCOPE ARRAY - 16th Int. Conf. on Accelerator and Large Experimental Control Systems ICALEPCS2017, Barcelona, Spain JACoW Publishing ISBN: 978-3-95450-193-9 doi:10.18429/

[RD2] Software Integration for the ASTRI SST-2M Prototype proposed for the Cherenkov Telescope Array - Proceedings of the Astronomical Data Analysis Software and Systems XXVI vol. 521

[RD3] Software use cases to elicit the software requirements analysis within the ASTRI project, Conforti et al. Proceedings of the SPIE Astronomical Telescopes + Instrumentation, 2016, Edinburgh,United Kingdom.

[RD4] Software design and code generation for the engineering graphical user interface of the ASTRI SST-2M prototype for the Cherenkov Telescope Array - C. Tanci et al - Proceedings Volume 9913, Software and Cyberinfrastructure for Astronomy IV; 99133X (2016) https://doi.org/10.1117/12.2232005

[RD5] The High-Level Interface Definitions in the ASTRI/CTA Mini Array Software System (MASS) - V. Confort et al - Astronomical Data Analysis Software and Systems: XXIV, ASP Conference Series, Vol. 495.

[RD6] Information and Communication Technology (ICT) Infrastructure for the ASTRI SST-2M telescope prototype for the Cherenkov Telescope Array - F. Gianotti et al - Proc. SPIE 9913, Software and Cyberinfrastructure for Astronomy IV, 99132C (8 August 2016); doi: 10.1117/12.2230150.

[RD7] ASTRI Virtual Test Bed: from Prototype to Mini Array - F. Gianotti et al - Proceedings of the Astronomical Data Analysis Software and Systems XXX

[RD8] "*ASTRI data reduction software in the framework of the Cherenkov Telescope Array*" - S. Lombardi, L. A. Antonelli, C. Bigongiari, et al. - Proceedings of SPIE 2018, Vol. 10707-29.

[RD9] *"Cherenkov Telescope Array Data Management"* - G. Lamanna, et al. (for the CTA Consortium) - Proc. 34th ICRC (2015) - arXiv:1509.01012.

[RD10] *"Data model issues in the Cherenkov Telescope Array project"* - Contreras, J. L., et al. (for the CTA Consortium) - Proc. 34th ICRC (2015) - arXiv:1508.07584.

[RD11] *"ASTRI SST-2M prototype and mini-array data reconstruction and scientific analysis software in the framework of the Cherenkov Telescope Array"* - Lombardi, S., L. A. Antonelli, D. Bastieri, et al. - Proc. SPIE 9913 (2016), 991315.
[RD12] *"ASTRI SST-2M data reduction and reconstruction software on low-power and parallel architectures"* - Mastropietro, M., et al. - Proc. SPIE 9913 (2016), 99133V.
[RD13] *"Definition of the Flexible Image Transport System (FITS), version 3.0"* - Pence, W. D., Chiappetti, L., Page, C. G., Shaw, R. A. and Stobie, E., Astronomy and Astrophysics 524, A42 (2010).
[RD14] http://heasarc.gsfc.nasa.gov/fitsio/fitsio.html
[RD15] http://heasarc.gsfc.nasa.gov/fitsio/ccfits/
[RD16] https://cmake.org/
[RD17] https://github.com/google/googletest/
[RD18] www.doxygen.org/
[RD19] http://www.sphinx-doc.org/
[RD20] https://docs.conda.io/en/latest/
[RD21] http://heasarc.gsfc.nasa.gov/docs/heasarc/caldb/caldb_intro.html
[RD22] http://heasarc.nasa.gov/lheasoft/
[RD23] Igel, C., Heidrich-Meisner, V., Glasmachers, T., "Shark," Journal of Machine Learning Research 9, 993-996 (2008).
[RD24] Beazley, D. M., "SWIG: An easy to use tool for integrating scripting languages with C and C++," Proc. 4th USENIX Tcl/Tk Workshop 4 (1996).
[RD25] https://www.mongodb.org
[RD26] https://opcfoundation.org/about/opc-technologies/opc-ua/
[RD27] https://confluence.alma.cl/display/ICTACS/ICT+ALMA+Common+Software
[RD28] https://www.jenkins.io/
[RD29] https://ieeexplore.ieee.org/abstract/document/5764064

3 ASTRI-Horn Context

The ASTRI-Horn Telescope (formerly known as the ASTRI SST-2M Prototype) is being verified and tested with the engineering software released in beta version [RD1, RD2]. Since we are adopting an iterative incremental approach for the development of the software, it is growing and changing rapidly. The software integration team goal is to integrate more software components, each of which functions properly in standalone tests, into the larger ASTRI software system that fulfills the system requirements. An additional goal is to support the maintenance activities such as debugging and implementation of new functionality. The drivers for the software integration activities are the ASTRI software requirements, both in text and use case form [RD3], the software architecture [RD4] and the component interfaces [RD5].

3.1 The Mini Array Software System

The main MASS components, which implement the required scenarios, are depicted in Fig. 1. Telescope hardware is locally controlled, with each assembly responsible for its own safety. The assemblies are grouped in a control hierarchy, in which each parent relays the commands to its children. At the highest level the Operator Control System (OCS) provides all the common services necessary for observations and the Data Handling System (DHS) manages the data flow from the control system to the data repositories.

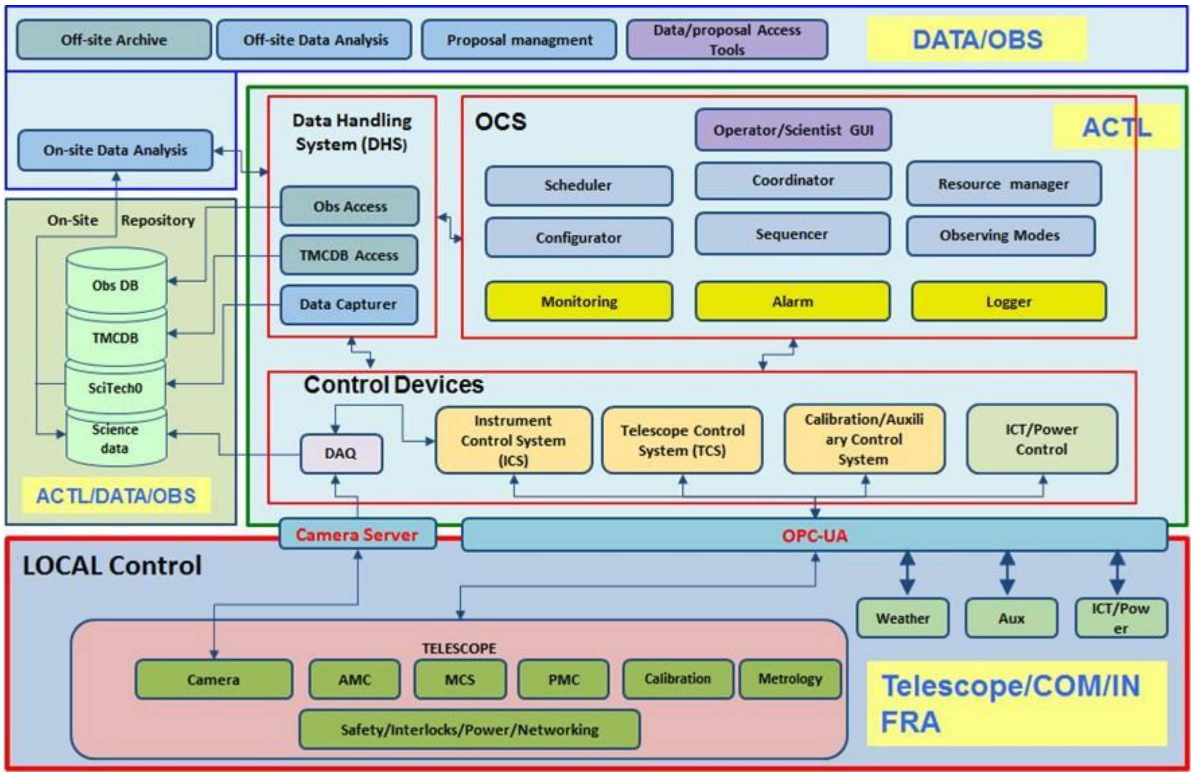


Figure 1 - ASTRI MASS functional decomposition

The architecture for the ASTRI mini-array is detailed in [AD1].

4 ASTRI-Horn Software Development

- 4.1 Software Version Control

In order to perform the software versioning control, we selected Git as a repository to control both the component versions and the build versions.

We consider a software component version, a specific release of a software which includes all its dependencies, setup instructions and "getting started" instructions. The software component shall be executed either standalone to support the unit tests, or as a part of a system to support the business operations.

We consider a build version, a specific release of a set of components which satisfy the requirements. The build includes the dependencies for each component and the setup instruction to install the software components as a whole.

Git repo allows the developer to control the software prototypes through his or her Sandbox.

In order to apply the logic above we organized our repo (gitbox) tree as follow:

- ASTRI/Sandbox/USERNAME/*
- ASTRI/MODULE/*
- ASTRI/ICD/*
- ASTRI/CommonTools/*

where

- USERNAME is the Git account of a single developer who could control the version for each Git project. Usually USERNAME is the developer surname.
- MODULE is one of the ASTRI modules which composes the MASS software. The foreseen modules are:
    - AUX (auxiliary)
    - CAM (camera)
    - DAQ (data acquisition)
    - DHS (data handling system)
    - ICT (hardware monitoring and control)
    - OCS (observations control)
    - PMC (pointing camera)
    - TCS (telescope control).
- ICD contains the Interface Control Documents for each component, which was in charge of the management of the hardware devices, either in .xls or .xml format;
- CommonTools contains software that can be used by all the ASTRI developers (*e.g.* code generator, device simulator, ...).

- 4.2 How to obtain a Git account

For the ASTRI-Horn prototype we used gitbox detailed here: http://redmine.iasfbo.inaf.it/projects/sw_management_tools/wiki/IASFBO_Git-Tutorial

For the ASTRI mini-array activities we configured the gitlab installation of INAF here: https://www.ict.inaf.it/gitlab/groups/astri

The authentication in gitlab is managed through the LDAP credentials provided by INAF. Once you run the first log in then a new account is created in gitlab and then you can ask the authorization to the ASTRI project just sending an e-mail to vito.conforti@inaf.it

Detailed gitlab documentation is available here: https://docs.gitlab.com/

A dedicated wiki page provide more information concerning the gitlab to support ASTRI project: https://redmine.iasfbo.inaf.it/projects/software/wiki/Git_Repository

-

4.3 Development rules

The integration activities require specific development rules to support the integration processes.

1. The software components shall be tested (unit) and documented (with user manual and setup instructions). The first step shall be always to produce a simulator in order to not lock the other developer jobs;

2. We request our developers to use the virtual machine for the development which provides a standardized environment, very similar to what will be used in operations. OPC-UA [RD26] is used throughout to integrate the software with the hardware devices, and ACS [RD27] to integrate higher-level software components with each other.

3. A Git repository is used for the releasing of software. We require our developers to use the repository for the intermediate versions during the software development.

4. It is mandatory to perform the lowest-level integration (for components which interface hardware devices) and middle-level integration before any release. In case of success the developer shall tag the version.

5. It is mandatory to perform the top-level integration (exploiting the test bed) before any major release of the software, and before the deployment in the production environment. The software on-site can be updated and a new version of the Software Release Document (SrelD) shall be published.

6. Continuous integration, consisting of a full build and execution of all unit tests, is done via Jenkins.

4.3.1 Interface Control Document

In order to support the test activities we provide a hardware simulator which produces dummy data and states according to its ICD (Interface Control Document).

4.3.2 Code generation

Since some components may be automatically generated, we suggest paying particular attention to extend and commit these components. Otherwise the risk is that some updates could be overwritten by the code re-generation.

4.4 How to implement an ACS component

To ease the work of the developers, a predefined set of standard directories can be created using a simple command:

```
$ getTemplate
```

This is an interactive utility which provides templates for many different purposes. In this case we are interested in choosing the option "directoryStructure". Then a choice among different types of directory structures is proposed. Basically you will be interested in using two types:

- MODROOT (that can be WS type only, LCU type only or both WS and LCU according to the type of code you have to develop). Here you will put the software source code as you develop it;

- INTROOT (integration area where the developed code gets installed after successful compilation and by running "make install"). You can access this area through the environment variable $INTROOT. This environment variable can be found in the .bash_profile.acs. It is up to the user to define it in a proper way.

Another important area is called ACSROOT, which is fixed when ACS is installed and is not modifiable by the developer. The directory structure is basically the same as for the INTROOT but the meaning of the area is different: the ACSROOT is the repository for the ACS software. There, all the ACS libraries, tools and the acsMakefile (a makefile based on GNU make) are installed and available for every user, who can access that area through the environment variable $ACSROOT. Only the responsible persons for the installation of the ACS software release can create and populate an ACSROOT. The ownership and permission of the files into the ACSROOT should not allow other users different from the installation user to modify that area.

MODROOT and INTROOT are instead totally handled by each developer, who can create and populate them or delete them.

There are also man pages which explain how those areas are organized (just be careful that your ACS installation did build them; this is optional and not all machines installed with ACS have the man pages available). You can access them running:

man acsDirectoryStructure

If you do not want to go through the interface provided by getTemplate, you can also create in one command the directory structure using the commands (example in the case of INTROOT):

getTemplateForDirectory INTROOT $INTROOT

where $INTROOT is the path to your INTROOT directory.

- 4.4.1 MODROOT

The usual location for a MODROOT is somewhere in the user home directory. Each MODROOT corresponds to a software module containing a set of standard subdirectories created with getTemplate. A software module is a piece of software able to perform functions and having an interface available to an external user to access the functions provided. Technically a module is a way to organize functions in homogeneous groups. The interface hides the implementation and system dependencies from the user. Managerially the module is the basic unit for planning, project control and configuration control. There is no rule to define how big a module shall be. Common sense and programming experience should be enough to identify what can be gathered and treated as a unique item. Examples of modules are: a driver for a specific board (the driver itself, install utility, configuration data files, etc.), the logging system of ACS (libraries, utilities, etc.), the configuration database (the server in Java and C++ libraries to communicate with the server). Once a MODROOT has been created with getTemplate you will work under the different directories of the MODROOT, keeping or removing what is not needed. Be aware that some directories are mandatory:

- src is the directory where you will be putting the sources you are working on
- include is the directory that contains any needed header (".h") files
- lib is where the libraries produced by the build are stored, as well as any necessary third-party libraries that are not provided by ACS
- bin is where any executables produced by the build are stored
- idl is for the Interface Definition language files
- man is for the man pages
- object is where the dependencies files for the build are stored (when running "make all")
- doc is for the documentation generated with doxygen (see chapter 15)
- test is for the test source code

Note that the Makefile system requires the user to put *generated* source code under src as well (e.g., code generated for the ICD and TMCDB). This pollutes the src directory and leads to generated code getting committed to git, which is a bad idea. The Makefile procedure ought to be enhanced to look in an additional srcgen directory (or something similar) for source code. At the time of writing, this overdue enhancement has not been addressed by the ALMA ACS development group.

- 4.4.2 INTROOT

The usual location for the INTROOT is /introot/<username>, but any other area accessible to the users on a machine will be OK. This path should be passed to the program "getTemplate", after choosing the options "directoryStructure" -> "createINTROOTarea".

Then the variable $INTROOT must be made available to the user environment. As we saw in chapter 5, you should edit the file $HOME/.acs/.bash_profile.acs and set the INTROOT accordingly, e.g.:

export INTROOT=$HOME/introot/

and log out and in again.

- 4.4.3 The Makefile

To ensure homogeneity in the development of the software for the ALMA project, GNU make must be used at every site or consortium. This is by default the case on any system installed with the ACS release according to the ALMA standards. A wide set of make definitions have been coded and made available through a file called “acsMakefile”. Analogously, every developer should provide the src and test directories of his/her software module with a Makefile where he/she will indicate the actions to be taken when running “make all man install”. This Makefile has to be created starting from the utility “getTemplate” and choosing “code” and then “Makefile_for_WS” (if the code to be produced has to run on a UNIX workstation. Note that, if the module has been created with the directory structure from getTemplate, it will already contain the right Makefile. The module Makefile must include the acsMakefile. (The template contains the adequate instruction). To see the structure and use of the Makefile, please refer to the man pages:

```
$ man Makefile
```

The makefile for package option produces a high-level Makefile, intended to be used to build a set of modules with one command. Using this Makefile causes “make build” to iterate over all the modules specified, running either “make clean all install” or “make build” for each one. In the case of subsystems containing more than one module, it may be preferable to use one such high-level Makefile for each subsystem. This will enable a developer who wants to build only his own subsystem to do so as a single command.

Comments inside the generated template explain how to configure the Makefile to build the specific modules desired.

- 4.5 preliminary running and tests

During the coding activities the developer can run and test the software.

The developer configures an ACS CDB for test purposes creating the files in myACSComponent/test/CDB and sets the environment variable $ACS_CDB that shall refer to the CDB folder just prepared.

The developer then installs the software in the introot folder through the command:

```
make install
```

Finally the developer can experiment interactively with the software thanks to the GUI provided by ACS. (This is *not* the basis for automatic tests, which typically use JUnit, cppunit or pyunit.) The command is:

```
acscommandcenter
```

Once started the GUI, the developer can run ACS and the containers. ACS provides a default container for each programming language admitted:

- frodoContainer for java components;
- bilboContainer for C++ components;
- argoContainer for python components

5 ASTRI-Horn Integration Model

In information technology, the systems integration is the process of linking together different computing systems and software applications physically or functionally, to act as a coordinated whole. The ASTRI Collaboration in compliance with the CTA consortium selected the OPC-UA (OPC - Unified Architecture) to connect the software to the hardware devices, and ACS (ALMA Common Software) to interface the software components to each other within the application layer as depicted below:

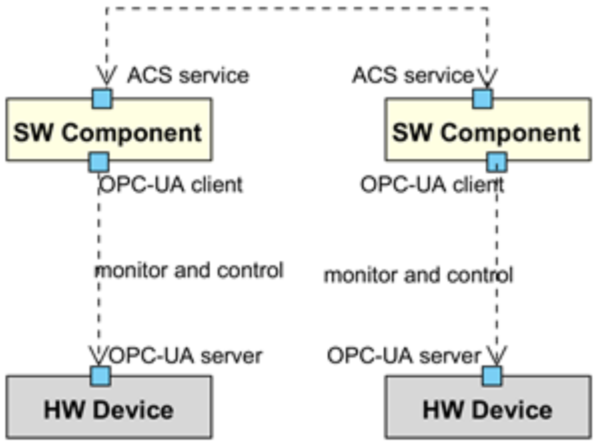


Figure 2 ASTRI integration levels

We plan three levels of integration:

- Lowest level: the goal is to test the interfaces between each hardware device and its related software. The software shall be able to monitor and control all the hardware device properties.
- Middle level: the goal is to test the interface between a software component and its neighbours, both the incoming and outgoing calls.
- Top level: the goal is to test the system as a whole by executing the scenarios detailed in the use cases.

We support the integration activities at every level, providing:

- Git repo for the version control;
- Jenkins to perform the continuous integration tests;
- Test report and release templates to document the software tests and the releases;

We are using Git for the software versioning at all levels (firmware, prototyping software, production software, third-party software). We use the Git tag system to set a component version or in general a software version for a specific physical machine. Jenkins exploits the setup and unit tests to execute the continuous integration.

We provide the virtual machine to support the middle level integration. The virtual machines provide an environment similar to the real system in order to facilitate local tests, the test bed provides a virtual environment in which all the production machines are simulated.

We provide to the developers, who produce code through high level programming, a virtual machine with an environment similar to the real machine on site. We require the developers to test effectively their code before committing it to the repository in order to minimize potential errors during integration with other software components. We implemented the integration-manager workflow on Git repo (see Fig. below)

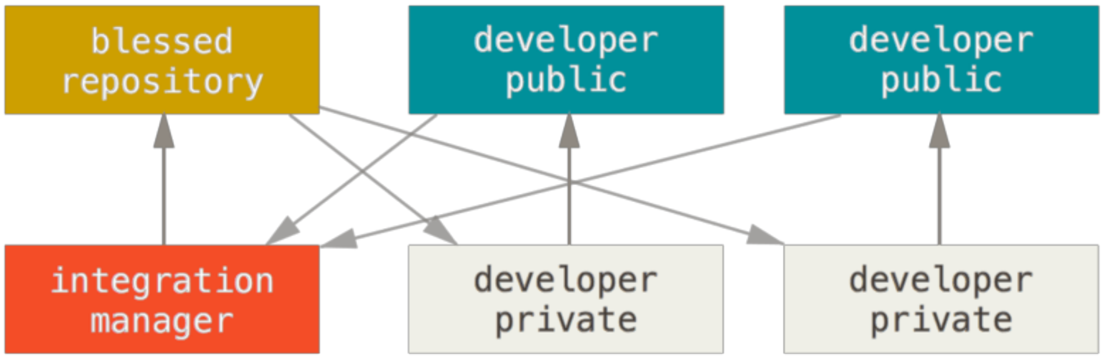


Figure 3 Integration Manager Workflow

in order to control the software versions and to support the developer who executes the local test of his component which interfaces other components. During the coding period the developer commits to the local (private) development branch. Once the software is ready for the release, the developer pushes the code to the remote (public) development branch. The integration manager clones the public development branches and performs the integration tests. In case of failure the developers have to fix any bug, in order to eventually close with success the integrations tests. Finally the integration manager merges the development branch on the master branch and pushes the code into the remote master branch (blessed repository). The developers pull the latest software version from the blessed repository into their development branch to continue the coding for the next release. The interface between the OPC-UA server installed on the hardware device and the OPC-UA client that is part of the ACS component is defined through an ICD (Interface Control Document) created with a predefined format. The ASTRI software has to provide the management of any hardware devices. In order to reduce the coding time, we implemented code generator software (python script) that takes as input the ICD (either in .xls or .xml format) and provides the ACS component which includes the OPC-UA client. We also implemented the OPC-UA server simulator that gets information about the device to simulate from a configuration file provided by the code generator. In this way the developer can easily execute the local software test without any real hardware. We also require the developer to implement the test for the business logic layer. The test consists of one or more programs that can be repeated by any user.

The [AD3] provides detail concerning the integration approved test model for the ASTRI Mini-Array project.

- 5.1 Test Bed environment

The test bed is a set of virtual machines installed at INAF-OAS Bologna which reproduce the same real machines on site. The virtual machines in the test bed have the same configuration (networking, users, operating system, dns, ...) of real machines in order to allow the tester to perform effective tests. The goal of these tests is to verify the software setup, and the interactions among software components. All the machines in the test bed as well as those ones on-site have the ASTRI Git read-only user in order to allow the integrator to download or update the software.

- 5.2 Continuous Integration with Jenkins

Continuous integration systems are a vital part of any Agile team because they help enforce the ideals of Agile development [RD29]. Jenkins [RD28], a continuous build tool, enables teams to focus on their work by automating the build, artefact management, and deployment processes. The Jenkins core functionality and flexibility allow it to fit in a variety of environments and can help streamline the development process for all stakeholders involved.

Martin Fowler provides the continuous integration definition: “Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible”.

At a regular frequency (ideally at every commit), the system is:

- Integrated
- Built (the code is compiled into an executable or package)
- Tested (Automated test suites are run)
- Archived (versioned and stored so it can be distributed as it is, if desired)
- Deployed (Loaded onto a system where developers can interact with it).

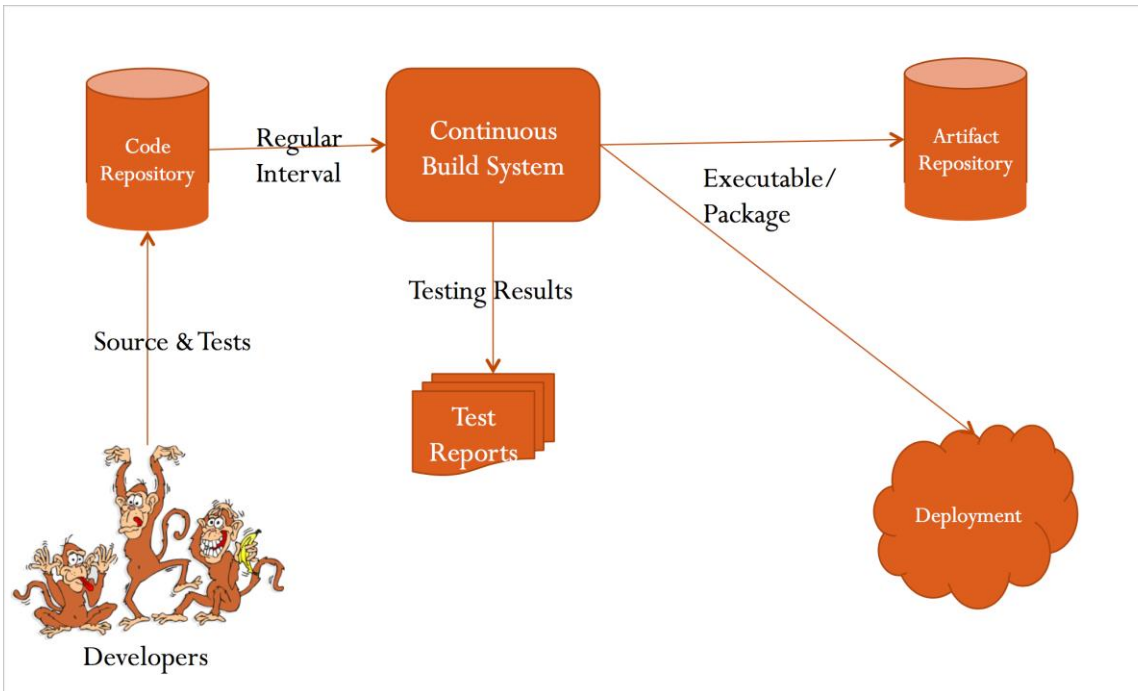


Figure 4 Continuous Integration with Jenkins

The best practices for the continuous integration are:

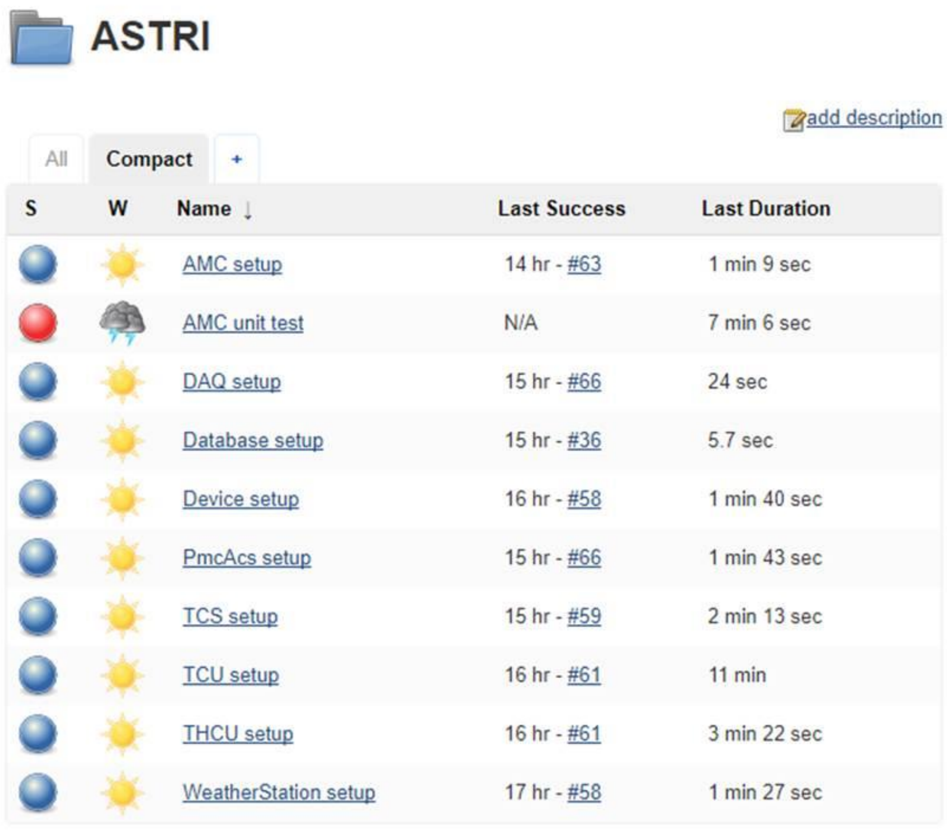
- Single Source Repository;
- Automate Build and Test;
- Everyone Commits Every Day;
- Keep the Build Fast;
- Everyone can see what's happening.

And the benefits are:

- Immediate bug detection;
- No integration step in the life cycle;
- A deployable system at any given point;
- Record of evolution of the project.

One of the basic principles of Continuous Integration is that a build should be verifiable. You have to be able to objectively determine whether a particular build is ready to proceed to the next stage of the build process, and the most convenient way to do this is to use automated tests. Without proper automated testing, you find yourself having to retain many build artifacts and test them by hand, which is hardly in the spirit of Continuous Integration.

The test suite has to be run by the developer before committing the software with the new code to the repository. Furthermore ACS includes TAT (Tool for Automated Tests) that is a framework which can run a test suite with only one command and reports the result in a simple and clear way: the word PASSED or FAILED is printed to the standard output. Although the ASTRI developers do not push new software with very high frequency, we keep continuously monitored, through a Jenkins server, the software status in order to detect very quickly any breaking build in our development work. We implemented two Jenkins jobs for each software component: the first compiles the last version on the development branch, the latter runs the unit test suite through TAT.


ASTRI

add description

All | Compact | +

S	W	Name ↓	Last Success	Last Duration
●	☀	AMC setup	14 hr - #63	1 min 9 sec
🔴	⛈	AMC unit test	N/A	7 min 6 sec
●	☀	DAQ setup	15 hr - #66	24 sec
●	☀	Database setup	15 hr - #36	5.7 sec
●	☀	Device setup	16 hr - #58	1 min 40 sec
●	☀	PmcAcs setup	15 hr - #66	1 min 43 sec
●	☀	TCS setup	15 hr - #59	2 min 13 sec
●	☀	TCU setup	16 hr - #61	11 min
●	☀	THCU setup	16 hr - #61	3 min 22 sec
●	☀	WeatherStation setup	17 hr - #58	1 min 27 sec

Figure 5 ASTRI-Horn Jenkins display

The figure above depicts the Jenkins page for the ASTRI project. The first column is the status of the build (a broken build is marked with a red bullet); in the second column there is the “weather report” showing aggregated status of recent builds; then follows the name of the job. The integration manager and the developers can display in any moment the details about the continuous integration through a jenkins server installed at the Astronomical Observatory of Catania.

We plan an integration test before any software release. Before we require that the component version candidates for the release locally passed the tests. We execute the preliminary tests on the ASTRI test bed. The test bed is a set of virtual machines which reproduce the same real machines on site.

We created an installer project for the management of the software components which exploit the ACS services. In particular it provides the functionalities to download the software either from development or master branch. In this project we configure also the parameters to connect the hardware devices, the parameters for the monitoring and the details for the deployment. We install this project and the ACS software components on a dedicated machine where the ACS manager is responsible for the software deployment. All the other machines are configured to link properly the node where the manager runs. Once the test succeeds on the test bed, the next step is to perform the integration test onsite. During this test we verify the setup procedures on the real servers, the interfaces among the software components, and here we also verify the connections between the software and the firmware installed in the hardware devices. This integration test is part of the ASTRI project AIV (Assembly Integration and Verification) plan.

- 5.3 ASTRI Software release

We officially release the software version when all tests are successfully completed on site. The integration manager updates the installer in order to download the specific tested version of the software component. Then the integration manager tags and pushes on the master branch all the updated software on the repository. We apply a specific pattern to the tag of the release that is V.X.Y.Z, where X denotes a major release not always compatible with the previous version, Y means a minor release which includes new capabilities and it is compatible with the previous version, and Z is used to indicate the bug fixes. These activities are reported in the release document which provides an overview of that version, the details and capabilities for every component which constitute the version. In this document we also refer to the test reports and user manuals. In addition we published a web based application built on redmine software, to collect the issues, to require new features or to report any bug. Redmine also provides issue-tracking functionality.

6 ASTRI-Horn Deployment Model

In this section we detail for each component the physical machine (generally the node) where it should be executed. You can see the kind of node and the kind of component through the stereotype (within the brackets <<>>) specified at the top of the element before its name.

This model is valid both for the SLN site and the test bed.

- 6.1 Observation control machines

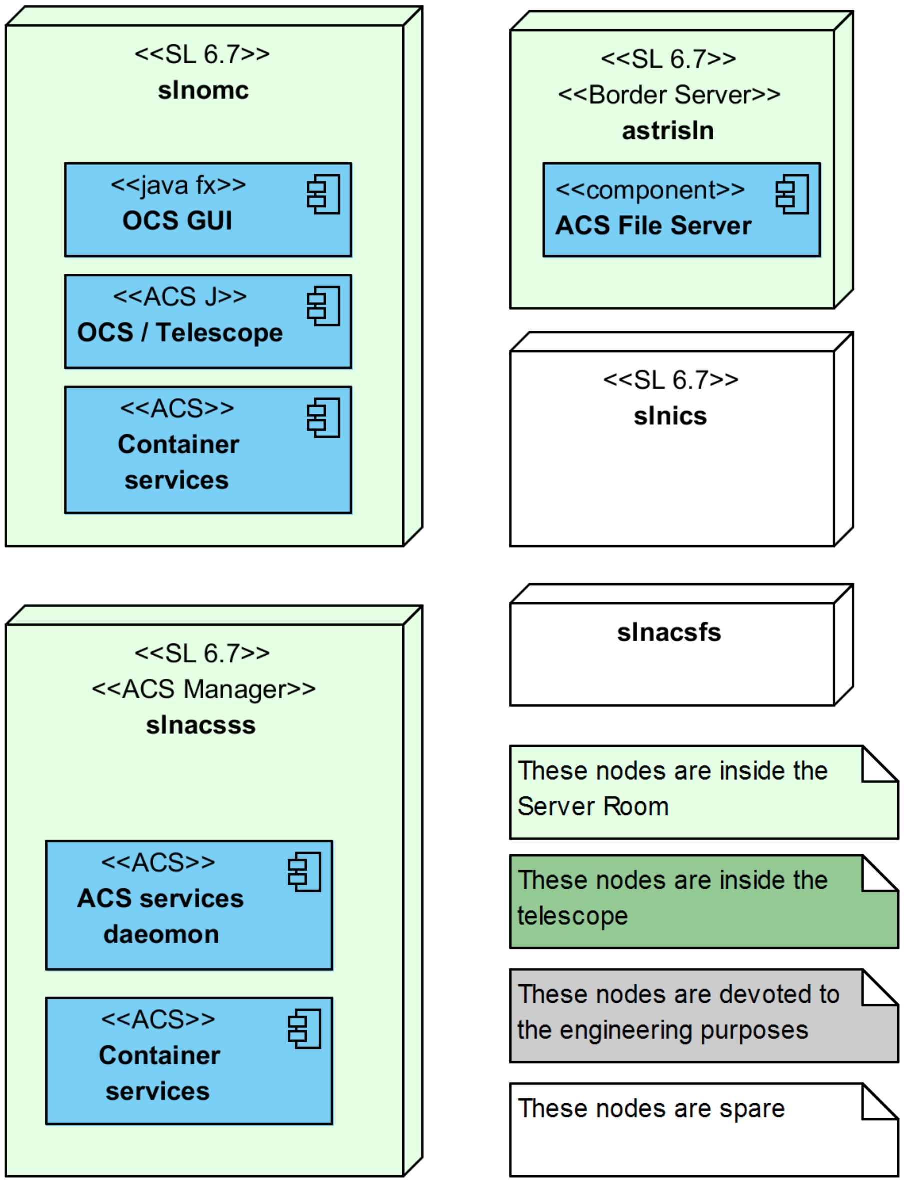


Figure 6 observation control UML deployment diagram

- 6.2 Telescope and auxiliary system management machines

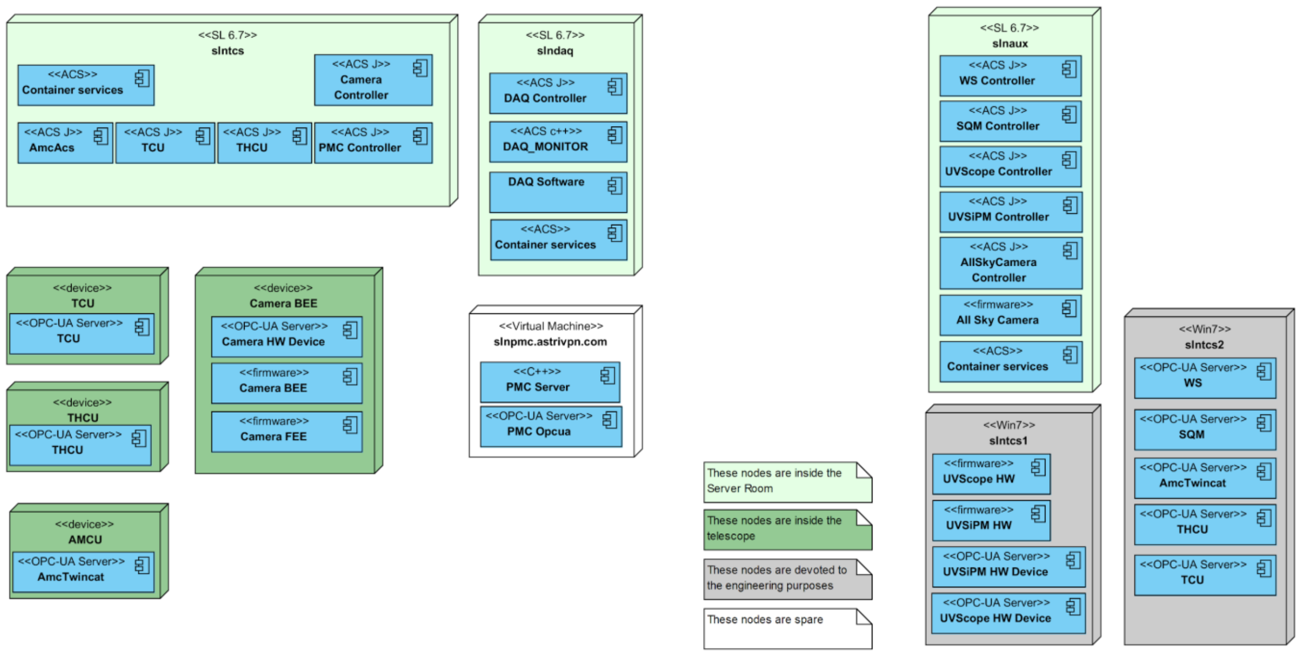


Figure 7 telescope and auxiliary UML deployment diagram

- 6.3 Data handling machines

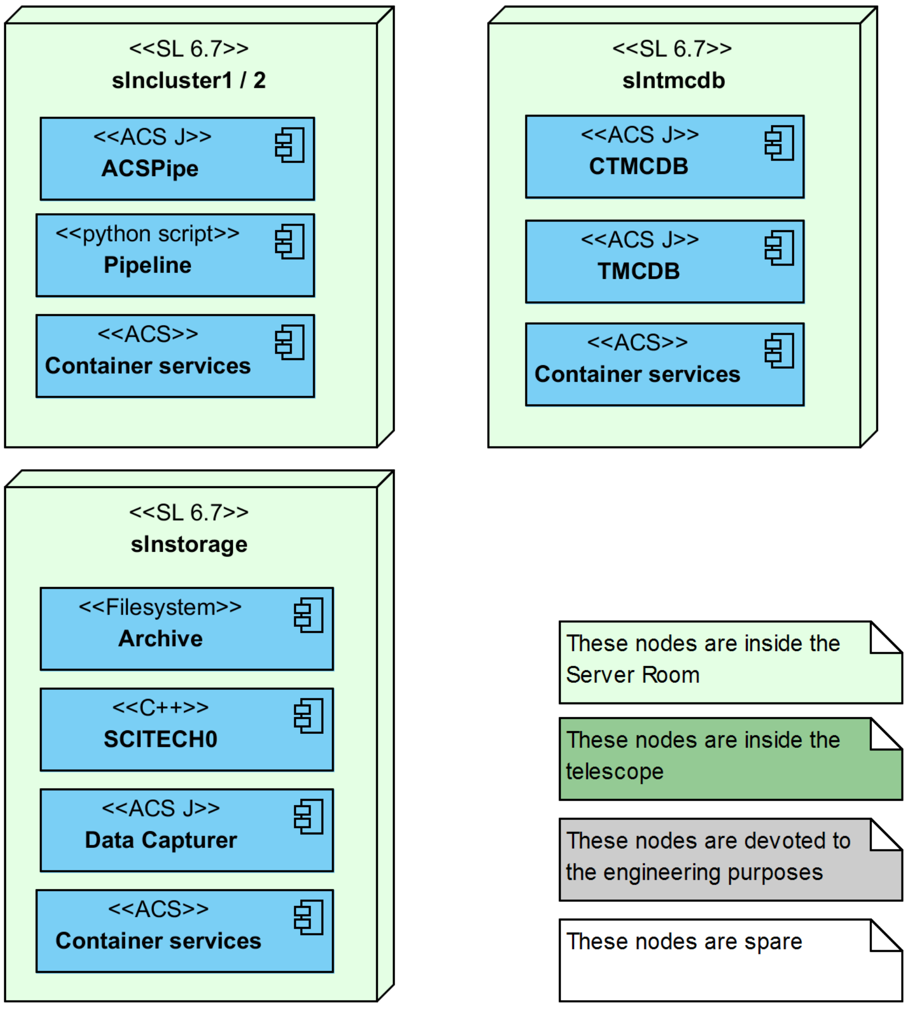


Figure 8 data handling UML deployment diagram

- 6.4 The ACS container configuration

In this section you can find detailed descriptions of the container (where it shall run and what components it will manage.

- 6.5 user and machine lists

The table below details the list of machines and users either in the test bed or on-site.

machine	O.S.	User	description
slntcs	SL 6.7	astrisw	telescope control software

snltcs1	Win7	tcs1user	pc used for the development and maintenance activities of the telescope control firmware.
slntcs2	Win7	tcs2user	pc used for the weather station software. In addition it is used for the development and maintenance activities of the telescope control firmware.
slnics	SL 6.7	astrisw	camera control software
slnaux	SL 6.7	astrisw	auxiliary control software
astrisln	SL 6.7	astrisw	border server and ACS file server
slndaq	SL 6.7	astrisw	data acquisition software
slntcmcdb	SL 6.7	astrisw	monitoring database
slncluster1	SL 6.7	pipeuser	science software

slncluster2	SL 6.7	pipeuser	science software
slnstorage	SL 6.7	storageuser	archive
slnomc	SL 6.7	astrisw	human machine interface
slnpmc.astrivpn.com	Fedora	TBD	virtual machine for the pointing model camera
slnacsfs	None	none	initially devoted as ACS file spare but eventually spare
slnacsss	SL 6.7	astrisw	ACS manager

-

At time of writing this document the operating system version in Scientific Linux 6.7. Nevertheless we foresee upgrading to the Scientific Linux or CentOS 7.6.

In order to access to this environment you need to jump to the border server:

- astrisln.iasfbo.inaf.it (for the test bed);
- astrisln.oact.inaf.it (for the SLN site).

6.6 Authorization and Authentication

Any user who needs to access the machines either in the test bed or on site shall authenticate through the LDAP system.

The ASTRI ICT manager releases access credentials to the people who request it. Currently encrypted software which contains all the needed passwords is available.

The user astrisw is a special read-only user with a common $HOME in all the machines which use ACS in order to share the same software version and configuration on the ACS framework.

6.7 ICT configuration for test bed and real environment

The RD6 and RD7 provide details concerning the test bed and real environment of the information and communication technology system.

The test bed provides a test environment aimed at performing the integration tests during the development and maintenance phase in order to perform the preliminary software verification before the integration tests with hardware devices on-site. The test bed is based on Oracle VM (OVM), which is a professional bare metal virtualization system. OVM is fast and reliable and free for small installations and it can ensure high availability. OVM allows a single control console to easily manage multiple hypervisor servers and dozens of Virtual Machines. The ASTRI test bed (see figure below) reproduces the SLN prototype environment like Network Services: Firewall, NAT, VPN, DNS, LDAP, Frontier Server, ISCSI/NFS Storage. Telescope Control and Monitor Servers like: Telemetry Data Base, Instrument Control Telescope control, Telescope Operation Gui, Auxiliary management. Data Acquisition, Archiving and Analysis Servers for: Digital Acquisition, Data reduction system and Storage systems In the test bed we have to reproduce: the same network as the ASTRI prototype and the same SLN servers, but virtualized, with the same IP addresses and server names as those of SLN. The benefit is to run the same software configuration in both test and production environments. In VTB all the physical devices are replaced by HW simulators so as to be able to integrate and test the Software in the VTB in a complete and meaningful way.

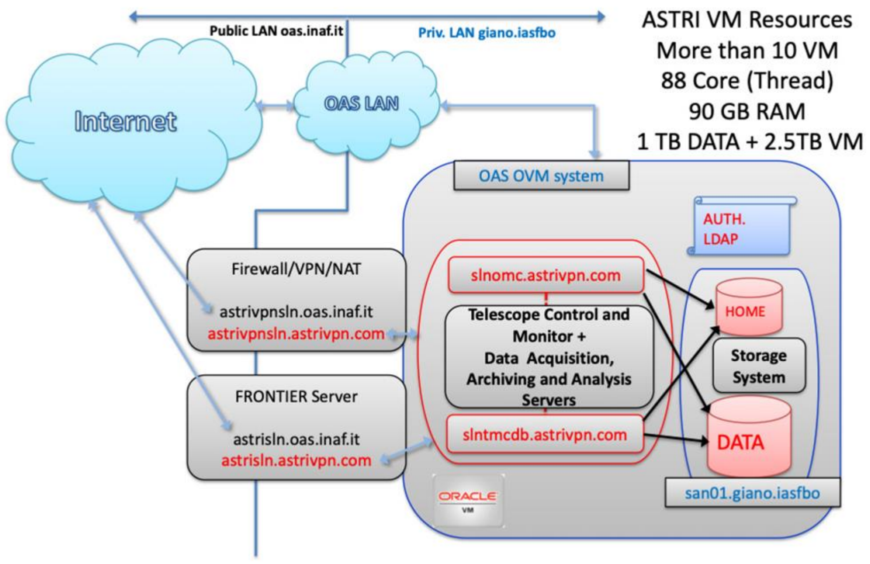


Figure 9 ASTRI test bed

7 ASTRI-Horn setup and running

- 7.1 Machine setup

The machines which provide the distributed software on ACS shall be properly configured.

The installation of ACS is the prerequisite and the starting point for developing any application for the ONLINE system. Access to the ACS libraries may also be necessary for building OFFLINE applications. When ACS is installed, a file called .bash_profile.acs, placed under the path $ACSROOT/config/.acs, is available for the user who performed the installation. As a user, you should copy the $ACSROOT/config/.acs directory in his/her home directory and source it from $HOME/.bash_profile .bash_profile.acs to properly set up the environment. You should set the INTROOT variable before sourcing .acs/.bash_profile.acs, leaving this latter file unmodified. To source the file, just run the command:

```
. $HOME/.acs/.bash_profile.acs
```

(Note that the environment variable $ACSROOT can expand in different ways, depending from how you handled the installation, but the default path is: /alma/ACS-X.Y/ACSSW, where X is the major number of the ACS release and Y is the minor number.)

This file .bash_profile.acs is written in bash style and supposes that the user uses bash as an interactive shell. Then, it is enough to source the file before starting any work and the developer will have the environment setup with the necessary environment variables. Note, however, that changing INTROOT after logging in will not perform properly. When the system $PATH and other environment variables are constructed at login time, the INTROOT that these variables reference will be the one encountered by the *first* value of INTROOT encountered in the login scripts. It is possible, but very tedious to change these values manually later. To change the INTROOT in use, you need to redefine it in your .bash_profile and then log out and login again.

A special machine master runs the ACS Manager which coordinates all the machines in the system. Each machine must have set the following environment variable, which provides reference to the manager:

```
export MANAGER_REFERENCE=corbaloc:slnacsss:3000/Manager
```

In addition all the machines, in order to execute the ACS containers and its components need the ACS daemons running (we usually run the daemon at the boot time of each machine).

The command to run the container services and ACS daemons are:

```
su -l csuser -c acscontainerdaemon &
su -l acsmgr -c acsservicesdaemon &
su -l acsmgr -c acscontainerdaemon &
```

The Annex D details the instructions for configuring the machines properly.

- 7.2 Download and compile

In order to setup the

- Makefile for downloading the modules from repository and build (that is compiled and installed in the introot defined).
- For the local test environment, each developer can define the introot in local path;
- For the integration test on the test bed it is mandatory to use the integration folder under /alma/ASTRI_SW on the file server mounted by any other participating machine.
- Test and Operation CDB (either XML or TMCDB);

The ACS software itself is installed on a single server: *astrisln*. It is mounted by all the machines which use ACS facilities and host ACS containers. This solution ensures that all the machines use only one instance of ACS software (the same version). Since ACS exists mainly to manage distributed objects (services, containers and components) on many network-connected hosts, a single instance of the ACS manager must keep track of all such objects in the system to do this task successfully. It is also a benefit for the ACS maintenance/upgrade activities.

The ACS manager shall be configured properly. Through the ACS command center you can create the configuration of each container (ip address and port). Then you can export this configuration in an XML file so that it may be passed as a parameter to the ACS start command line script.

At time of writing this document the latest stable version of the software on-site is:

gitbox:ASTRI/CONTROL/ASTRI-Control-Setup

The first step is to configure the environment and download the software modules and we can perform this task through the Makefile. The README file contains all the information to set the environment variables (see also the file loadEnvironment). The Makefile provides the target to download and install the software. You can find the Makefile in appendix A.

The command `make download_master` pulls the software module which produces the build in the control directory.

The command `make  build` installs the software in the introot which we defined in $HOME/introot.

Where the $HOME is that of the user astrisw.

The file ASTRI_DISTRIBUTED_CONFIG_WITHTMCDB contains the configuration to run the software for test purposes with the acscommandcenter. In particular we defined the machines in the system and the containers which run in each machine (see appendix B).

The configuration of the components/containers is held in the CDB directory tree which includes XML files. Here you can find the containers and their configurations, and the component configurations. The directory tree is in Appendix C.

We created the script to run/stop/kill the software (see ASTRI-Control-Setup/bin folder):

```
|-astriacsKill

|-astriacsKillContainers

|-astriacsStart

|-astriacsStartContainers

|-astriacsStop

|-astriacsStopContainers

|-kill
```

the script to start acs is quite simple because we need only to define the machine to host the acs manager, the path for the log files and the heap size:

```
#!/bin/bash



ssh -t astrisw@slnacsss.astrivpn.com "date && hostname &&  source
$ASTRI_ROOT/loadEnvironment && echo $ASTRI_ROOT && cdbChecker &&
HEAP_SIZE=\"-Xmx512m\" && export JAVA_OPTIONS=\"$HEAP_SIZE\" &&
nohup                        acsStart                          >
$ACSDATA/logs/slnacsss.astrivpn.com/acsStart_`date         "+%F-
%T"`_acstart_log && exit"
```

The script to run the containers is more interesting, because we run here the daemon to start each container on the specific machine (Appendix E).

All the scripts above are used directly in the operator GUI in order to be more user friendly. Finally the user shall only run this command in `slnomc`, the machine dedicated to the HMI:

```
./StartInstalledGUI.sh
```

The distributed systems run properly because all the machines know the location of the acs manager and the software version is the same for all the machines because each machine mounts the $HOME directory of the user astrisw.

- 8 Development and integration of the ASTRI reduction and scientific data analysis software (*A-SciSoft)*

The ASTRI data reconstruction and scientific analysis software (henceforth *A-SciSoft* [RD8]) is the official software package of the ASTRI Project for the reduction and analysis of the scientific data acquired by the ASTRI-Horn Telescope. Its design and development started in 2014 in full compliance with the general CTA data management requirements [RD9] and data model specifications [RD10] available at that time. The software has been designed to handle both real and Monte Carlo (MC) simulated data, and to provide all necessary algorithms and analysis tools for characterizing the scientific performance of the ASTRI-Horn Prototype. Because the software was also designed with an array configuration in mind, it will be also used to perform the reduction of data acquired with the ASTRI Mini-Array.

The main purpose of *A-SciSoft* is to reconstruct the physical characteristics of astrophysical gamma rays (and background cosmic rays) from the raw data generated by the ASTRI-Horn Telescope and the Mini-Array. The software is composed by a set of independent modules organized in an efficient pipeline that implements all the necessary algorithms to perform the complete scientific data reduction, from raw data to the final scientific products.

The overall framework adopted for the development of *A-SciSoft* follows the general high-level requirements listed in [RD11], and the main guidelines provided by the CTA Consortium.

In what follows, the main features of the *A-SciSoft* software development framework are presented [RD11].

- Programming languages: C++11 (International Standard ISO/IEC 14882:2011(E) – Programming Language C++) and Python. Due to our interest in exploring data processing on parallel and low-power architecture, we also ported computationally intensive algorithms to CUDA [RD12]. The Python language has been used for wrapping software executables and to provide a high-level user interface.
- Data format and I/O layer: The FITS format [RD13] has been adopted for all the data produced by the ASTRI-Horn prototype, from the raw data (converted to FITS files during data acquisition) up to the final scientific products. Throughout the whole analysis chain, *cfitsio* libraries [RD14] are used for FITS read/write and manipulation. As C++ is the common programming language used in all low-level modules, we used the C++ wrapped version of cfitsio, called *CCfits* [RD15].
- Build system and compilers: The code is built using the *gcc* compiler under Unix-like platforms. Since we target multiple architectures, we used *CMake* [RD16] to automatically configure the compilation. *CMake* easily allows the software to be compiled on heterogeneous systems and can automatically detect the presence of kernels targeted to accelerators (e.g., CUDA code for NVIDIA GPUs).
- Software development tools: Version control is made through *git*, which is integrated both with the *Redmine* issue tracker system and the *Jenkins* continuous integration system. Jenkins, at each new push on the remote code repository, automatically performs compilation of the last code version and execution of the unit tests written using the *GoogleTest* [RD17] framework. In the

current environment, *cppcheck* is used for static code quality inspection (check for duplicate code, potential bugs, coding standards and so on).

- Code documentation: Code documentation is written in *Doxygen* [RD18] for C++ and CUDA code, and *Sphinx* [RD19] for Python code and software general description.
- Deployment and packaging: *A-SciSoft* will be distributed both as source code and as binary packages (.tgz, .deb, .rpm, etc.) generated from the source code by the build system. A *Conda* package [RD20] has also been built, which includes final binary executables and libraries for the installation on several target operating system platforms.
- Calibration database: *A-SciSoft* makes use of a calibration database (CALDB) system based on standard HEASARC tools [RD21] for retrieving basic instrumental and analysis inputs needed throughout the data reduction, such as nominal hardware-related quantities, look-up-tables and matrices of instrument response functions.
- External Dependencies: *A-SciSoft* depends essentially on the HEASoft [RD22] astronomical software package which includes the cfitsio, CCfits, CALDB, and hoops libraries needed for I/O and parameter system handling. These are the only external dependencies which can be satisfied installing the HEASoft software libraries. To perform model training for event reconstruction, the *Shark* [RD23] open-source Machine Learning libraries have been used.
- Python bindings: Python is rapidly becoming the standard for end-user analysis in astronomy, and nearly all new science analysis tools provided by other observatories provide at least a python interface. Since it is easy to interface C and Python code, we provided Python bindings to the library we implemented. We adopted SWIG (Simple Wrapper Interface Generator) [RD24] to generate python modules. The C++ library code that we have written can be then easily called.
- Event processing and parallelization: Processing for the low-level data reduction is done by *chunks of events*. This allows for an easy parallelization on multi-core systems and GPUs.
- Logging class: We designed a dedicated *logging class* called *ctalog* able to stream logging and error messages to different targets. Error and log streams may be redirected differently (by changing a configuration parameter), for example to:
  - standard streams (stdout and stderr)
  - text file
  - socket (may be used as shared memory, *a la MPI* (Message Passing Interface), internet socket, etc.)
  - database (SQL, MongoDB [RD25])

Thanks to its C interface, *ctalog* can be easily wrapped and used also in Python modules and pipeline scripts.

- 9 Lessons learned

The methodology implemented for ASTRI-Horn provided a lot of benefits:

- software well tested with different levels (unit, integration, validation on-site with the hardware);
- hardware platform performance corresponding to the requirements;
- effective software version control;
- user friendly setup and operations;

The improvement points for the future are:

- to implement the continuous integration pipeline which really stress the software (more unit tests);
- to configure the static code analysis with SonarQube;
- to include the off-site software in this loop;
- to improve the release workflow to ensure that we produce all the deliverables before the release (test reports and user manuals).

- 10 Appendix

- Appendix A - Makefile

This makefile refers the repo:

gitobox:ASTRI/CONTROL/ASTRI-Control-Setup

commit id: bfbbd5b3

```
#*******************************************************************
****************

# ALMA - Atacama Large Millimeter Array

# Copyright (c) UNSPECIFIED - FILL IN, 2016

#

# This library is free software; you can redistribute it and/or

# modify it under the terms of the GNU Lesser General Public

# License as published by the Free Software Foundation; either

# version 2.1 of the License, or (at your option) any later
version.

#

# This library is distributed in the hope that it will be useful,

# but WITHOUT ANY WARRANTY; without even the implied warranty of

# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU

# Lesser General Public License for more details.

#

# You should have received a copy of the GNU Lesser General Public

# License along with this library; if not, write to the Free
Software

# Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-
1307  USA

#
```

```
# "@(#) $Id: Makefile_PACKAGE.template,v 1.12 2010/07/09 12:48:42
alopatin Exp $"

#

# Makefile of ........

#

# who       when      what

# --------  --------  ------------------------------------------
----

# Vito Conforti  2016-12-15  created

#



# ALMA - Atacama Large Millimeter Array

# Copyright (c) ESO - European Southern Observatory, 2014

# (in the framework of the ALMA collaboration).

# All rights reserved.

#

# This library is free software; you can redistribute it and/or

# modify it under the terms of the GNU Lesser General Public

# License as published by the Free Software Foundation; either

# version 2.1 of the License, or (at your option) any later
version.

#

# This library is distributed in the hope that it will be useful,

# but WITHOUT ANY WARRANTY; without even the implied warranty of

# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU

# Lesser General Public License for more details.

#
```

```
# You should have received a copy of the GNU Lesser General Public

# License along with this library; if not, write to the Free
Software

# Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-
1307  USA

#*****************************************************************
****************



#*****************************************************************
****************

# This Makefile follows ACS Standards (see Makefile(5) for more).

#*****************************************************************
****************

# REMARKS

#    None

#-----------------------------------------------------------------
---------



#

# Modules in the various sub-packages.

#

# The following example is for a subsystem called Voyager One

# which is composed of two directories, Alpha and Beta.


# Alpha contains the modules alpha1, alpha2 and alpha3

# Beta contains the modules beta1, beta2 and beta3

#

# Modify as appropriate to your subsystem
```

```
SUBSYSTEM = "ASTRI Control Setup"


MODULES_ASTRI_CONTROL_MODULES    = TCS  UaDataSupportExtensions
device  ASC  SQM  AMC  TCU  THCU  Database  PmcAcs  TMCDB  TMCDB_MySQL
WeatherStation astritel OMC/AcceptanceGUI

#MODULES_BETA = beta1 beta2 beta3


MODULES  =   $(foreach  dir,  $(MODULES_ASTRI_CONTROL_MODULES),
control/$(dir)) :

          # $(foreach dir, $(MODULES_BETA), Beta/$(dir))


#

# If option KEEP_GOING=on is present in the make command line
gnu_make is NOT interrupted

# when the first error is encountered

#


ifdef KEEP_GOING

   KEEP_GOING="on"

else

   KEEP_GOING="off"

endif


RETURN_CODE=return_code

TMP_RETURN_CODE=tmp_return_code


MAKE_FLAGS = "-k"

PLATFORM := $(shell uname)
```

```
SHELL=/bin/bash

ECHO=echo


ifdef MAKE_VERBOSE

    AT =

    OUTPUT =

else

    AT = @

    OUTPUT = > /dev/null

endif

#

os      = $(shell uname)

osrev   = $(shell uname -r)


ifeq ($(os),SunOS)

      realtime=YES

endif


#

# "Failed all" error management

#

define mng_failed_all

      if [[ -a  $(TMP_RETURN_CODE) ]]; then\

            $(ECHO) "### ==> FAILED all ! " | tee -a build.log |
tee -a $(RETURN_CODE);\\

            rm $(TMP_RETURN_CODE);\
            if [[ $(KEEP_GOING) = "off" ]]; then \
                  if [[ -a $(RETURN_CODE) ]]; then \
                        rm $(RETURN_CODE);\
                  fi;\
                  exit 2;\
            fi;\
      fi
endef


#
# "Failed install" error management
#
define mng_failed_install
      if [[ -a  $(TMP_RETURN_CODE) ]]; then\
            $(ECHO) "### ==> FAILED install ! " | tee -a build.log
| tee -a $(RETURN_CODE);\
            rm $(TMP_RETURN_CODE);\
            if [[ $(KEEP_GOING) = "off" ]]; then \
                  if [[ -a $(RETURN_CODE) ]]; then \
                        rm $(RETURN_CODE);\
                  fi;\
                  exit 2;\
            fi;\
      fi
endef
```

```
#

# This target just forward any make target to all modules

#

define canned

      @$(ECHO) "############ Executing '$@' on all $(SUBSYSTEM) 
modules ##################"

      @for member in  $(foreach name, $(MODULES), $(name) ) ; do 
\

               $(ECHO) "############ $${member}" ;\

               if [ ! -d $${member} ]; then \

                         echo  "###  ==>  $${member}  MODULE  NOT 
FOUND! FAILED! " | tee -a build.log;\

                    fi;\

               if [ -f $${member}/src/Makefile ]; then \

                $(MAKE)  $(MAKE_FLAGS)  -C  $${member}/src/  $@  || 
break ;\

               elif [ -f $${member}/ws/src/Makefile ]; then \

                $(MAKE) $(MAKE_FLAGS) -C $${member}/ws/src/ $@ || 
break ;\

               fi;\

               if [ "$(realtime)" == "YES" ]; then \

                if [ -f $${member}/lcu/src/Makefile ]; then \

                $(MAKE)  $(MAKE_FLAGS)  -C  $${member}/lcu/src/  $@ 
|| break ;\

                fi;\

               fi;\

            done

endef
```

```
clean_log:

      @$(ECHO)  "############  Clean  Build  Log  File:  build.log
##################"

      @rm -f build.log

      @touch build.log



#

# building all modules

#

build:

      @$(ECHO)   "############   build   $(SUBSYSTEM)   Software
##################"| tee -a build.log

      @# Deletion of temporary files used to store make return
code

      @if [[ -a $(TMP_RETURN_CODE) ]]; then \

            rm $(TMP_RETURN_CODE);\

      fi

      @if [[ -a $(RETURN_CODE) ]]; then \

            rm $(RETURN_CODE);\

      fi

      @for member in  $(foreach name, $(MODULES), $(name) ) ; do
\

                  if [ ! -d $${member} ]; then \

                   echo  "###  ==>  $${member}  MODULE  NOT  FOUND!
FAILED! " | tee -a build.log;\

                  fi;\

                  if [ -f $${member}/src/Makefile ]; then \

                       $(ECHO) "############ $${member} MAIN" | tee
-a build.log;\
                    $(MAKE) $(MAKE_FLAGS) -C $${member}/src/ clean
>> build.log 2>& 1;\

                    $(MAKE) $(MAKE_FLAGS) -C $${member}/src/ all >>
build.log 2>& 1 || echo $$? >> $(TMP_RETURN_CODE) ;\

                    $(mng_failed_all);\

                    $(MAKE) $(MAKE_FLAGS) -C $${member}/src/ install
>> build.log 2>& 1 || echo $$? >> $(TMP_RETURN_CODE) ;\

                    $(mng_failed_install);\

                    continue ;\

                 fi;\

                 if [ -f $${member}/ws/src/Makefile ]; then \

                    $(ECHO) "############ $${member} WS" | tee -a
build.log;\

                    $(MAKE)  $(MAKE_FLAGS)  -C  $${member}/ws/src/
clean >> build.log 2>& 1;\

                    $(MAKE) $(MAKE_FLAGS) -C $${member}/ws/src/ all
>> build.log 2>& 1 || echo $$? >> $(TMP_RETURN_CODE) ;\

                    $(mng_failed_all);\

                    $(MAKE)  $(MAKE_FLAGS)  -C  $${member}/ws/src/
install >> build.log 2>& 1 || echo $$? >> $(TMP_RETURN_CODE) ;\

                    $(mng_failed_install);\

                 fi;\

                 if [ "$(realtime)" == "YES" ]; then \

                   if [ -f $${member}/lcu/src/Makefile ]; then \

                       $(ECHO) "############ $${member} LCU" | tee
-a build.log;\

                       $(MAKE) $(MAKE_FLAGS) -C $${member}/lcu/src/
clean >> build.log 2>& 1;\

                       $(MAKE) $(MAKE_FLAGS) -C $${member}/lcu/src/
all >> build.log 2>& 1 || echo $$? >> $(TMP_RETURN_CODE) ;\

                       $(mng_failed_all);\
                    $(MAKE) $(MAKE_FLAGS) -C $${member}/lcu/src/
install >> build.log 2>& 1 || echo $$? >> $(TMP_RETURN_CODE) ;\

                    $(mng_failed_install);\

                fi;\

              fi;\

          done

          @$(ECHO) "... done"




download_dev:

     rm -rf control

     mkdir  control;

     @  $(ECHO) "Extracting from Git the ext lib to support the
execution of characteristic components "; \

     git clone -b dev  gitbox:ASTRI/TOOLS/ext_lib control/ext_lib

     cp control/ext_lib/* $(INTROOT)/lib/

     @  $(ECHO) "Extracting from Git the astro tool  "; \

     git clone -b dev  gitbox:ASTRI/TOOLS/astro control/astro

     cp control/astro/lib/* $(INTROOT)/lib/

     @  $(ECHO) "Extracting from Git PMC "; \

     git clone -b dev gitbox:ASTRI/PMC/PmcAcs control/PmcAcs

     @  $(ECHO) "Extracting from Git ASC "; \

     git clone -b dev gitbox:ASTRI/AUX/ASC control/ASC

     @  $(ECHO) "Extracting from Git SQM "; \

     git clone -b dev gitbox:ASTRI/AUX/SQM control/SQM

     @  $(ECHO) "Extracting from Git AMC "; \
	git clone -b dev  gitbox:ASTRI/AMC/AmcAcs control/AMC

	@  $(ECHO) "Extracting from Git Database "; \

	git   clone   -b   dev   gitbox:ASTRI/ICD/ARCHIVE/Database
control/Database

	@  $(ECHO) "Extracting from Git device "; \

	git  clone  -b  dev   gitbox:ASTRI/CONTROL/TCS/common/device
control/device

	@  $(ECHO) "Extracting from Git idl common files of TCS (will
be put in TCS folder) "; \

	git clone -b dev gitbox:ASTRI/ICD/TCS control/TCS

	@  $(ECHO) "Extracting from Git TCU "; \

	git clone -b dev  gitbox:ASTRI/CONTROL/TCU control/TCU

	@  $(ECHO) "Extracting from Git THCU "; \

	git clone -b dev  gitbox:ASTRI/CONTROL/THCU control/THCU

	@  $(ECHO) "Extracting from Git TMCDB "; \

	git clone -b dev  gitbox:ASTRI/Archive/TMCDB control/TMCDB

	@  $(ECHO) "Extracting from Git TMCDB_MySQL "; \

	git   clone   -b   dev      gitbox:ASTRI/DHS/TMCDB_MySQL
control/TMCDB_MySQL

	@  $(ECHO) "Extracting from Git WeatherStation "; \

	git    clone    -b    dev        ASTRI/AUX/WeatherStation
control/WeatherStation

	@  $(ECHO) "Extracting from Git ASTRI Telescope component ";
\

	git   clone   -b   dev      gitbox:ASTRI/CONTROL/astritel
control/astritel

	@  $(ECHO) "Extracting from Git Hardware Simulator "; \

	git  clone  -b  dev     gitbox:ASTRI/TOOLS/OpcuaSimulator
control/OpcuaSimulator

	@  $(ECHO) "Extracting from Git the OPCUA extensions "; \
	git clone -b dev  gitbox:ASTRI/TOOLS/UaDataSupportExtensions control/UaDataSupportExtensions

	@  $(ECHO) "Extracting from Git Hardware code generatore PyHWGen "; \

	git clone -b dev  gitbox:ASTRI/TOOLS/PyHwGen control/PyHwGen

	@  $(ECHO) "Extracting from Git the OMC GUI  "; \

	git clone -b FedeDev  gitbox:ASTRI/OCS/OMC control/OMC

	@  $(ECHO) "Extracting from Git Camera Control "; \

	git clone -b dev  gitbox:ASTRI/CAM/AstriCameraControl

	@  $(ECHO) "Extracting from Git ICT ICMP "; \

	git clone -b dev  gitbox:ASTRI/ICT/ICMP

	@  $(ECHO) "Extracting from Git IMO GUI (GUI for the ICT monitoting)  "; \

	git clone -b dev  gitbox:ASTRI/ICT/IMO_GUI

	@  $(ECHO) "Install log4j.properties in the introot "; \

	cp conf/log4j.properties $(INTROOT)/config/




download_master:

	rm -rf control

	mkdir  control;

	@  $(ECHO) "Extracting from Git the ext lib to support the execution of characteristic components "; \

	git  clone  --branch  V.0.1.0  gitbox:ASTRI/TOOLS/ext_lib control/ext_lib

	cp control/ext_lib/* $(INTROOT)/lib/

	@  $(ECHO) "Extracting from Git the astro tool  "; \
    git clone --branch V.0.1.0 gitbox:ASTRI/TOOLS/astro
control/astro

    cp control/astro/lib/* $(INTROOT)/lib/

    @ $(ECHO) "Extracting from Git PMC "; \

    git clone --branch V.0.1.0 gitbox:ASTRI/PMC/PmcAcs
control/PmcAcs

    @ $(ECHO) "Extracting from Git ASC "; \

    git clone --branch V.0.2.0 dev gitbox:ASTRI/AUX/ASC
control/ASC

    @ $(ECHO) "Extracting from Git SQM "; \

    git clone --branch V.0.2.0 dev gitbox:ASTRI/AUX/SQM
control/SQM

    @ $(ECHO) "Extracting from Git AMC "; \

    git clone -b master gitbox:ASTRI/AMC/AmcAcs control/AMC

    @ $(ECHO) "Extracting from Git Database "; \

    git clone -b master gitbox:ASTRI/ICD/ARCHIVE/Database
control/Database

    @ $(ECHO) "Extracting from Git device "; \

    git clone --branch V.0.1.0
gitbox:ASTRI/CONTROL/TCS/common/device control/device

    @ $(ECHO) "Extracting from Git idl common files of TCS (will
be put in TCS folder) "; \

    git clone --branch V.0.1.0 gitbox:ASTRI/ICD/TCS control/TCS

    @ $(ECHO) "Extracting from Git TCU "; \

    git clone --branch V.0.1.0 gitbox:ASTRI/CONTROL/TCU
control/TCU

    @ $(ECHO) "Extracting from Git THCU "; \

    git clone --branch V.0.1.0 gitbox:ASTRI/CONTROL/THCU
control/THCU

    @ $(ECHO) "Extracting from Git TMCDB "; \
    git clone -b master gitbox:ASTRI/Archive/TMCDB control/TMCDB

    @  $(ECHO) "Extracting from Git TMCDB_MySQL "; \

    git    clone    -b    master    gitbox:ASTRI/DHS/TMCDB_MySQL
control/TMCDB_MySQL

    @  $(ECHO) "Extracting from Git WeatherStation "; \

    git   clone   --branch   V.0.1.0   ASTRI/AUX/WeatherStation
control/WeatherStation

    @  $(ECHO) "Extracting from Git ASTRI Telescope component ";
\

    git   clone   -b   master   gitbox:ASTRI/CONTROL/astritel
control/astritel

    @  $(ECHO) "Extracting from Git Hardware Simulator "; \

    git           clone           --branch           V.0.1.0
gitbox:ASTRI/TOOLS/OpcuaSimulator control/OpcuaSimulator

    @  $(ECHO) "Extracting from Git the OPCUA extensions "; \

    git           clone           --branch           V.0.1.0
gitbox:ASTRI/TOOLS/UaDataSupportExtensions
control/UaDataSupportExtensions

    @  $(ECHO)  "Extracting  from  Git  Hardware  code  generatore
PyHWGen "; \

    git   clone   --branch   V.0.1.0   gitbox:ASTRI/TOOLS/PyHwGen
control/PyHwGen

    @  $(ECHO) "Extracting from Git the OMC GUI  "; \

    git clone --branch V.0.1.0 gitbox:ASTRI/OCS/OMC control/OMC

    @  $(ECHO) "Install log4j.properties in the introot "; \

    cp conf/log4j.properties $(INTROOT)/config/
```

```
#

# Test target

#


.PHONY: test


Test = test

$(Test):

      @$(ECHO)  "############  Clean  Test  Log  File:  test.log
#################"

      @rm -f test.log

      @touch test.log

      @$(ECHO)   "############   TEST   $(SUBSYSTEM)   Software
#################"| tee -a test.log

      @for member in $(foreach name,$(MODULES),$(name)); do\

            if [ -d $${member}/ws/test ]; then\

                  $(ECHO) "############ $${member}/ws/test WS TEST
############" | tee -a test.log ;\

                  $(MAKE)  -k  -C  $${member}/ws/test/  $@  |  tee  -a
test.log | egrep '(Nothing to|FAILED.|PASSED.|Error:)';\

                  if [ -d $${member}/lcu/test ]; then\

                        $(ECHO)  "############  $${member}/lcu/test
LCU TEST ############" | tee -a test.log;\

                        $(MAKE) -k -C $${member}/lcu/test/ $@ | tee
-a test.log | egrep '(Nothing to|FAILED.|PASSED.|Error:)';\

                  fi;\

            elif [ -d $${member}/test ]; then\
                $(ECHO) "############ $${member}/test MAIN TEST ############" | tee -a test.log ;\
                $(MAKE) -k -C $${member}/test/ $@ | tee -a test.log | egrep '(Nothing to|FAILED.|PASSED.|Error:)';\
           else\
                $(ECHO) "### ==> $${member} TEST DIRECTORY STRUCTURE NOT FOUND! FAILED!" | tee -a test.log ;\
           fi;\
     done
     @$(ECHO) "... done"
#
# Standard canned targets
#
clean:
     $(canned)
all:
     $(canned)
install:
     $(canned)

man:
     $(canned)

buildClean: build clean

buildMan: build man
```

```
#

# ___oOo___
```

- Appendix B - ACS Command Center Configuration File

The ACS Command center configuration file is available in the repo gitbox:ASTRI/CONTROL/ASTRI-Control-Setup.

The file is ASTRI_DISTRIBUTED_CONFIG_WITHTMCDB

In order to run acascommandcenter with the above configuration you have to use the following command:

```
$  acscommandcenter ASTRI_DISTRIBUTED_CONFIG_WITHTMCDB
```

In this xml file you need to fill the following tags:

- mode: remote_daemon or local
- servicesLocalJavaRoot: the root folder of source files
- remoteHost: ip address or naming of machine which hosts the acs manager
- containers. For each container the following data are required:
  - name: the container name
  - type: java or python or cpp
  - heapSizeMB: size of the container in MB for java containers
  - remoteHost: the machine which hosts the container.

The contents of ASTRI_DISTRIBUTED_CONFIG_WITHTMCDB is reported below.

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

<AcsCommandCenterProject
xmlns="Alma/Acs/AcsCommandCenterProject"

    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

    creator="acc-v10.2" xsi:type="AcsCommandCenterProject">

    <mode>remote_daemon</mode>

    <servicesLocalJavaRoot>/home/astrisw/ASTRI-Control-
Setup</servicesLocalJavaRoot>
```

```
    <scriptBase>0</scriptBase>

    <remoteHost>slnacsss</remoteHost>

    <remoteAccount></remoteAccount>

    <remotePassword></remotePassword>

<toolRunAgainstDedicatedSettings>false</toolRunAgainstDedicatedS
ettings>

    <toolAgainstManagerHost></toolAgainstManagerHost>

    <toolAgainstManagerPort></toolAgainstManagerPort>

<toolAgainstInterfaceRepository></toolAgainstInterfaceRepository
>

    <toolAgainstNameService></toolAgainstNameService>

    <containers>

        <select>6</select>

        <againstManagerHost></againstManagerHost>

        <againstManagerPort></againstManagerPort>

        <againstCDB></againstCDB>

<againstInterfaceRepository></againstInterfaceRepository>

        <container>

            <name>slnauxJContainer</name>

            <type>java</type>

            <heapSizeMB>512</heapSizeMB>

            <useDedicatedSettings>true</useDedicatedSettings>

            <scriptBase>0</scriptBase>

            <remoteHost>slnaux</remoteHost>

            <remoteAccount></remoteAccount>
```

```
</container>
<container>
    <name>slntcsJContainer</name>
    <type>java</type>
    <heapSizeMB>2048</heapSizeMB>
    <useDedicatedSettings>true</useDedicatedSettings>
    <scriptBase>0</scriptBase>
    <remoteHost>slntcs</remoteHost>
    <remoteAccount></remoteAccount>
</container>
<container>
    <name>frodoContainer</name>
    <type>java</type>
    <heapSizeMB>2048</heapSizeMB>
    <useDedicatedSettings>true</useDedicatedSettings>
    <scriptBase>0</scriptBase>
    <remoteHost>slntcs</remoteHost>
    <remoteAccount></remoteAccount>
</container>
<container>
    <name>bilboContainer</name>
    <type>cpp</type>
    <heapSizeMB></heapSizeMB>
    <useDedicatedSettings>true</useDedicatedSettings>
    <scriptBase>0</scriptBase>
    <remoteHost>slntcs</remoteHost>
```

```
            <remoteAccount></remoteAccount>
        </container>
        <container>
            <name>slntcsCppContainer</name>
            <type>cpp</type>
            <heapSizeMB></heapSizeMB>
            <useDedicatedSettings>true</useDedicatedSettings>
            <scriptBase>0</scriptBase>
            <remoteHost>slntcs</remoteHost>
            <remoteAccount></remoteAccount>
        </container>
        <container>
            <name>slnauxCppContainer</name>
            <type>cpp</type>
            <heapSizeMB></heapSizeMB>
            <useDedicatedSettings>true</useDedicatedSettings>
            <scriptBase>0</scriptBase>
            <remoteHost>slnaux</remoteHost>
            <remoteAccount></remoteAccount>
        </container>
    </containers>
</AcsCommandCenterProject>
```

- Appendix C - ASTRI-Horn CDB tree

```
|-MACI
  |   |-Channels
|   |   |-Channels.xml
|   |-Components
|   |   |-ARCHIVE
|   |   |   |-TMCDB
|   |   |   |   |-MONITOR_BLOBBER
|   |   |   |   |   |-MONITOR_BLOBBER.xml
|   |   |   |   |-MONITOR_CONTROL
|   |   |   |   |   |-MONITOR_CONTROL.xml
|   |   |   |-ARCHIVE.xml
|   |   |-Components.xml
|   |-Containers
|   |   |-aragornContainer
|   |   |   |-aragornContainer.xml
|   |   |-bilboContainer
|   |   |   |-bilboContainer.xml
|   |   |-frodoContainer
|   |   |   |-frodoContainer.xml
|   |   |-slnauxCppContainer
|   |   |   |-slnauxCppContainer.xml
|   |   |-slnauxJContainer
|   |   |   |-slnauxJContainer.xml
|   |   |-slnclusterJContainer
|   |   |   |-slnclusterJContainer.xml
|   |   |-slndaqCppContainer
|   |   |   |-slndaqCppContainer.xml
|   |   |-slndaqJContainer
|   |   |   |-slndaqJContainer.xml
|   |   |-slnomcJContainer
|   |   |   |-slnomcJContainer.xml
|   |   |-slnstorageJContainer
|   |   |   |-slnstorageJContainer.xml
|   |   |-slntcsCppContainer
|   |   |   |-slntcsCppContainer.xml
|   |   |-slntcsJContainer
|   |   |   |-slntcsJContainer.xml
|   |-Managers
|   |   |-Manager
|   |   |   |-Manager.xml
|-alma
|   |-AMC
|   |   |-AMC.xml
|   |-ASC
|   |   |-ASC.xml
|   |-CameraControl
|   |   |-CameraControl.xml
|   |-ICT_ICMP
|   |   |-ICT_ICMP.xml
|   |-PMC
|   |   |-PMC.xml
|   |-SQM
|   |   |-SQM.xml
|   |-TCU
|   |   |-TCU.xml

|   |-THCU

|   |   |-THCU.xml

|   |-WeatherStation

|   |   |-WeatherStation.xml

|-build.log
```

- Appendix D - System configuration

- D-1 Scientific Linux 6.x at SLN

- D-1-1 System installation, startup and automatic configuration

In the case of the Scientific Linux 6.x operating system, the scripts must be copied to the directory:

- /etc/init.d

They start and stop with the commands:

- service acscontainerdaemon start
- service acsservicesdaemon start

- service acscontainerdaemon stop
- service acsservicesdaemon stop

Verify that they started with:

- service acscontainerdaemon status
- service acsservicesdaemon status

Once you see that everything is fine, they are enabled to be started at the system boot with the commands:

- chkconfig acscontainerdaemon on
- chkconfig acsservicesdaemon on

command to see if a script is enabled:

- chkconfig --list acscontainerdaemon
  result:

-
  ```
   acscontainerdaemon       0:off    1:off    2:on    3:on    4:on    5:on    6:off
  ```

If you need to disable the scripts use the commands:

- chkconfig acscontainerdaemon on
- chkconfig acsservicesdaemon on

- D-1-4 Scripts Code

The scripts must be started with the system user that you have decided to use to run the Software. In the scripts he must specify his username he belongs to.
The scripts are in the directory:

- /etc/init.d
-

```
[root@slnacsss init.d]# more acscontainerdaemon
#!/bin/sh
#
# Startup script for program
#
# chkconfig: 345 85 15
# description: Description of program
# processname: acscontainerdaemon
# pidfile: /var/run/containerdaemon.pid

. /etc/init.d/functions


RETVAL=0
prog="acscontainerdaemon"
HOST=`hostname`
STAMP=`date "+%Y-%m-%d_%H.%M.%S"`
USER=astrisw
LOGDIR=~$USER/.acscontainerdaemon
LOG=$LOGDIR/acscontainerdaemon_$HOST-$STAMP
# we should use the daemon facility provided by functions script, but
# acscontainerdaemon does no fork into the background so it lockup the script.
# For that reason i had to implement my onw flaky daemon start. I creates lock
# and pid file, but they are just to mantain standard since they are of little
# use.
#If there is an instance of the daemon running the script will pass.
case "$1" in
  start)
        echo -n "Starting $prog:"
      PID=`pidof $prog`
        if [ $? -eq 1 ];then
        su -l $USER -c "mkdir -p $LOGDIR" && su -l $USER -c "$prog &> $LOG&"
```

```
            PID=`pidof $prog`
            RETVAL=$?
            if [ $? -eq 1 ]; then
                failure
            else
                success
            fi
        else
            passed
            echo "acscontainerdaemon is already running"
            RETVAL=0
        fi
        echo
        if [ $RETVAL -eq 0 ];then
             touch /var/lock/subsys/$prog
             PID=$(pidofproc "$prog")
             echo $PID > /var/run/$prog.pid
        fi
        ;;
  stop)
        echo -n "Shutting down process-name: "
        killproc $prog
        RETVAL=$?
        echo
        if [ $RETVAL -eq 0 ];then
            rm -f /var/lock/subsys/$prog
            rm -f /var/run/$prog.pid
        fi
        ;;
  status)
        status $prog
        ;;
  restart)
        $0 stop
        $0 start
        ;;
  *)
        echo "Usage: $0 {start|stop|restart|status}"
        exit 1
esac

exit 0
```

```
[root@slnacsss init.d]# more acsservicesdaemon
#!/bin/sh
#
# Startup script for program
#
# chkconfig: 345 84 16
# description: Description of program
# processname: acsservicesdaemon
# pidfile: /var/run/acsservicesdaemon.pid

. /etc/init.d/functions


RETVAL=0
prog="acsservicesdaemon"
HOST=`hostname`
STAMP=`date "+%Y-%m-%d_%H.%M.%S"`
USER=astrisw
LOGDIR=~$USER/.acsservicesdaemon
LOG=$LOGDIR/acsservicesdaemon_$HOST-$STAMP
# we should use the daemon facility provided by functions script, but
# acsservicesdaemon does no fork into the background so it lockup the script.
# For that reason i had to implement my onw flaky daemon start. I creates lock
# and pid file, but they are just to mantain standard since they are of little
# use.
#If there is an instance of the daemon running the script will pass.
case "$1" in
  start)
         echo -n "Starting $prog:"
       PID=`pidof $prog`
         if [ $? -eq 1 ];then
         su -l $USER -c "mkdir -p $LOGDIR" && su -l $USER -c "$prog &> $LOG&"
         PID=`pidof $prog`
         RETVAL=$?
         if [ $? -eq 1 ]; then
             failure
         else
             success
         fi
       else
         passed
         echo "acsservicesdaemon is already running"
         RETVAL=0
```

```
        fi
        echo
        if [ $RETVAL -eq 0 ];then
            touch /var/lock/subsys/$prog
            PID=$(pidofproc "$prog")
            echo $PID > /var/run/$prog.pid
        fi
        ;;
  stop)
        echo -n "Shutting down process-name: "
        killproc $prog
        RETVAL=$?
        echo
        if [ $RETVAL -eq 0 ];then
           rm -f /var/lock/subsys/$prog
           rm -f /var/run/$prog.pid
        fi
        ;;
  status)
        status $prog
        ;;
  restart)
        $0 stop
        $0 start
        ;;
  *)
        echo "Usage: $0 {start|stop|restart|status}"
        exit 1
esac

exit 0
```

- D-2 CentOS 7.x for the Test Bed

- D-2-1 System installation, startup and automatic configuration
  In the case of the CentOS7.x operating system, the scripts must be copied to the directory:
  - /etc/systemd/system/

  They start and stop with the commands:
  - systemctl start acscontainerdaemon.service
  - systemctl start acsservicesdaemon.service

- systemctl stop acscontainerdaemon.service
- systemctl stop acsservicesdaemon.service

Verify that they started with:

- systemctl status acscontainerdaemon.service
- systemctl status acsservicesdaemon.service

Once you see that everything is fine, they are enabled to be started at the system boot with the commands:

- systemctl enable acscontainerdaemon.service
- systemctl enable acsservicesdaemon.service

command to see if a script is enabled:

- systemctl list-unit-files | grep acs

result:

- acscontainerdaemon.service   enabled

If you need to modify the scripts, they must be reloaded into the system with the commands:

- systemctl daemon-reload

If you need to disable the scripts use the commands:

- systemctl disable acscontainerdaemon.service
- systemctl disable acsservicesdaemon.service

- D-2-2 Script Code

The scripts must be started with the system user that you have decided to use to run the Software. In the scripts he must specify his username and the group he belongs to.
In the script, the hostname of the server on which it will be run must be specified.
The scripts are in the directory:

- /etc/systemd/system/

```
[root@slnacsss system]# more acscontainerdaemon.service
[Unit]
Description=Daemon to start ACS containers -- remote or local
After=auditd.service systemd-user-sessions.service time-sync.target

[Service]
User=astrisw
Group=cta
Environment=HOST=slnacsss.astrivpn.com
ExecStart=/bin/bash -c 'source /home/astrisw/.bash_profile; exec acscontainerdaemon'
ExecReload=/bin/kill -HUP $MAINPID
```

```
KillMode=process

[Install]
WantedBy=multi-user.target

[root@slnacsss system]# more acsservicesdaemon.service
[Unit]
Description=Daemon to start ACS services -- remote or local
After=auditd.service systemd-user-sessions.service time-sync.target

[Service]
User=astrisw
Group=cta
Environment=HOST=slnacsss.astrivpn.com
ExecStart=/bin/bash -c 'source /home/astrisw/.bash_profile; exec acsservicesdaemon'
ExecReload=/bin/kill -HUP $MAINPID
KillMode=process

[Install]
WantedBy=multi-user.target
```

- E script to run containers

```
#!/bin/bash



echo "[INFO]Start slntcsJContainer on slntcs"

acsdaemonStartContainer  -i 0 -t java -c slntcsJContainer -H
slntcs -m corbaloc::slnacsss.astrivpn.com:3000/Manager

echo "[INFO]Start frodoContainer on slntcs"

acsdaemonStartContainer  -i 0 -t java -c frodoContainer -H slntcs
-m corbaloc::slnacsss.astrivpn.com:3000/Manager

echo "[INFO]Start bilboContainer on slntcs"

acsdaemonStartContainer  -i 0 -t cpp -c bilboContainer -H slntcs
-m corbaloc::slnacsss.astrivpn.com:3000/Manager

echo "[INFO]Start slntcsCppContainer on slntcs"

acsdaemonStartContainer  -i 0 -t cpp -c slntcsCppContainer -H
slntcs -m corbaloc::slnacsss.astrivpn.com:3000/Manager
```

```
echo "[INFO]Start slnauxCppContainer on slnaux "

acsdaemonStartContainer  -i 0 -t cpp -c slnauxCppContainer -H
slnaux -m corbaloc::slnacsss.astrivpn.com:3000/Manager

echo "[INFO]Start slnauxJContainer on slnaux"

acsdaemonStartContainer  -i 0 -t java -c slnauxJContainer -H
slnaux -m corbaloc::slnacsss.astrivpn.com:3000/Manager

#echo "[INFO]Start frodoContainer on slnaux"

#acsdaemonStartContainer  -i 0 -t java -c frodoContainer -H slnaux
-m corbaloc::slnacsss.astrivpn.com:3000/Manager

#echo "[INFO]Start bilboContainer on slnaux"

#acsdaemonStartContainer  -i 0 -t cpp -c bilboContainer -H slnaux
-m corbaloc::slnacsss.astrivpn.com:3000/Manager
```