Software design and code generation for the engineering graphical user interface of the ASTRI SST-2M prototype for the Cherenkov Telescope Array

Tanci, Claudio, Tosti, Gino, Antolini, Elisa, Gambini, Giorgio, Bruno, Pietro, et al.

Claudio Tanci, Gino Tosti, Elisa Antolini, Giorgio Francesco Gambini, Pietro Bruno, Rodolfo Canestrari, Vito Conforti, Saverio Lombardi, Federico Russo, Pierluca Sangiorgi, Salvatore Scuderi, "Software design and code generation for the engineering graphical user interface of the ASTRI SST-2M prototype for the Cherenkov Telescope Array," Proc. SPIE 9913, Software and Cyberinfrastructure for Astronomy IV, 99133X (9 August 2016); doi: 10.1117/12.2232005

SPIE.

Event: SPIE Astronomical Telescopes + Instrumentation, 2016, Edinburgh, United Kingdom

Software design and code generation for the engineering graphical user interface of the ASTRI SST-2M prototype for the Cherenkov Telescope Array

Claudio Tanci[a,b], Gino Tosti[a,b], Elisa Antolini[a,b], Giorgio Francesco Gambini[a], Pietro Bruno[c], Rodolfo Canestrari[b], Vito Conforti[d], Saverio Lombardi[e], Federico Russo[f], Pierluca Sangiorgi[g], Salvo Scuderi[c], on behalf of the ASTRI Collaboration[h], and the CTA Consortium[i]

[a]Università di Perugia - Dipartimento di Fisica e Geologia, v. Pascoli, 06123 Perugia, Italy
[b]INAF - Osservatorio Astronomico di Brera, v. Bianchi, 46 23807 Merate (Lc), Italy
[c]INAF - Osservatorio Astrofisico di Catania, v. Sofia 78, 95123 Catania, Italy
[d]INAF - Istituto di Astrofisica Spaziale e Fisica Cosmica di Bologna, v. Gobetti 101, 40129 Bologna, Italy
[e]INAF - Osservatorio Astronomico di Roma, v. di Frascati 33, 00040 Monteporzio Catone (Roma), Italy
[f]INAF - Osservatorio Astrofisico di Torino, via Osservatorio 20, 10025 Pino Torinese (TO), Italy
[g]INAF - Istituto di Astrofisica Spaziale e Fisica Cosmica di Palermo, v. La Malfa 153, 90146 Palermo, Italy
[h]http://www.brera.inaf.it/astri/
[i]http://www.cta-observatory.org

ABSTRACT

ASTRI is an on-going project developed in the framework of the Cherenkov Telescope Array (CTA). An end-to-end prototype of a dual-mirror small-size telescope (SST-2M) has been installed at the INAF observing station on Mt. Etna, Italy. The next step is the development of the ASTRI mini-array composed of nine ASTRI SST-2M telescopes proposed to be installed at the final CTA southern site. The ASTRI mini-array is a collaborative and international effort carried on by Italy, Brazil and South-Africa and led by the Italian National Institute of Astrophysics, INAF. To control the ASTRI telescopes, a specific ASTRI Mini-Array Software System (MASS) was designed using a scalable and distributed architecture to monitor all the hardware devices for the telescopes. Using code generation we built automatically from the ASTRI Interface Control Documents a set of communication libraries and extensive Graphical User Interfaces that provide full access to the capabilities offered by the telescope hardware subsystems for testing and maintenance. Leveraging these generated libraries and components we then implemented a human designed, integrated, Engineering GUI for MASS to perform the verification of the whole prototype and test shared services such as the alarms, configurations, control systems, and scientific on-line outcomes. In our experience the use of code generation dramatically reduced the amount of effort in development, integration and testing of the more basic software components and resulted in a fast software release life cycle. This approach could be valuable for the whole CTA project, characterized by a large diversity of hardware components.

Keywords: Imaging Atmospheric Cherenkov Telescope, Telescope System Software, GUI, Code Generation, CTA, ASTRI,

Further author information:
C.T.: E-mail: claudio.tanci@brera.inaf.it, Telephone: +39 075 585 5934
G.T.: E-mail: tosti@pg.infn.it , Telephone: +39 075 585 5934

Software and Cyberinfrastructure for Astronomy IV, edited by Gianluca Chiozzi, Juan C. Guzman, Proc. of SPIE Vol. 9913, 99133X · © 2016 SPIE
CCC code: 0277-786X/16/$18 · doi: 10.1117/12.2232005

1. INTRODUCTION

The ASTRI mini-array will consist of nine ASTRI SST-2M telescopes at the southern site of the Cherenkov Telescope Array (CTA) compliant with CTA requirements and guidelines.[1] A first prototype telescope is operated at the INAF observing station in Serra La Nave on Mount Etna (Italy). The Mini-Array Software System (MASS) is used for the operation of this telescope, performing the observation and analyzing data, and supporting engineering operations.[2] MASS is built upon the ALMA Common Software (ACS)[3] framework and uses the Open Platform Communication Unified Architecture (OPC-UA)[4] protocol for the interface with the telescope components. We first started using code generation in the ASTRI project to build ACS components for the MASS, togheter with simplified engineering user interfaces, then we added the capability to produce a set of communication libraries that could be used by external applications to interface with and use the generated components. The ASTRI engineering GUI application employs these libraries to offer to the operator an integrated environment for monitoring and control of the telescope.

2. THE MINI-ARRAY SOFTWARE SYSTEM

The main MASS components are depicted in Fig. 1. ACS components, grouped by subsystem in the Control Devices layer, communicate with the hardware using the OPC-UA protocol.

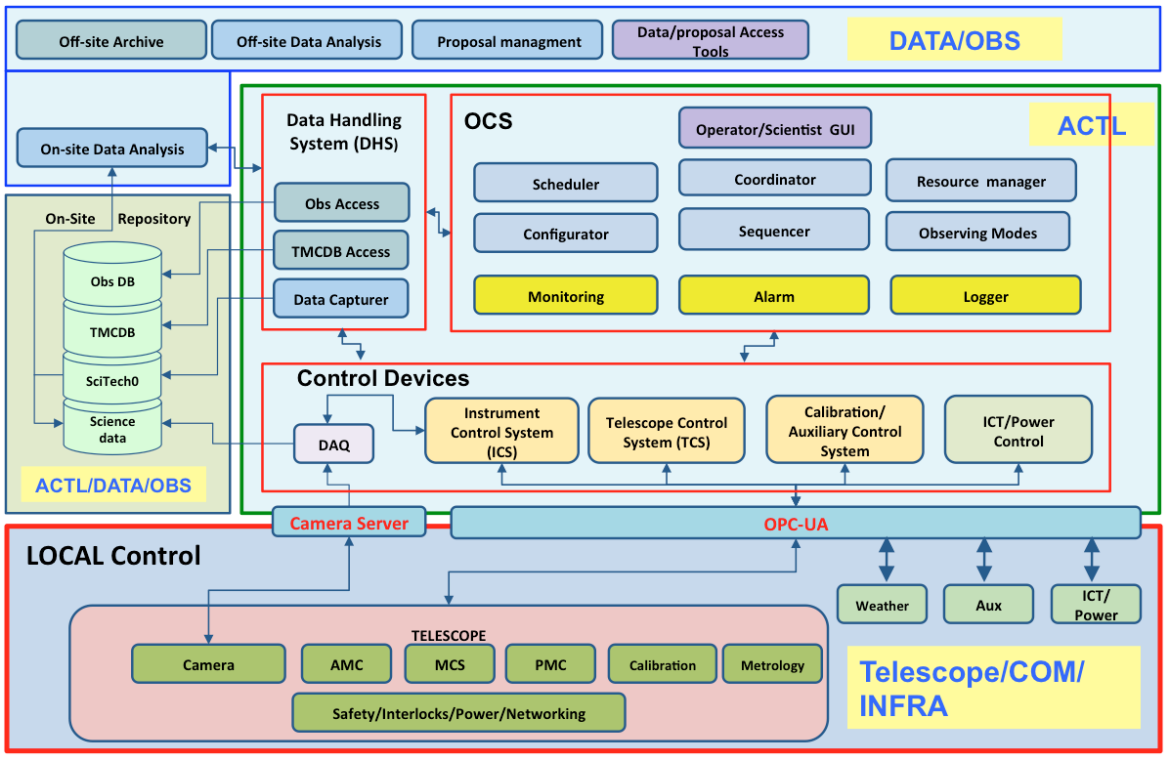


Figure 1: The main components of MASS

The OPC-UA protocol was chosen for the whole CTA project as the interface between the higher level control software and the hardware assemblies. In the ASTRI SST-2M prototype the low-level functions of the Telescope Control Unit (TCU), the Telescope Health Control Unit (THCU), the Active Mirror Control Unit (AMCU) are implemented via software Programmable Logic Controllers (PLCs) that use the Beckhoff TwinCAT platform.* While Beckhoff provides a native OPC-UA server other OPC-UA servers were developed using the Prosys OPC

*Beckhoff TwinCAT software allows the use of standard compatible PCs as hosts of real-time controller systems. More details can be fond on https://www.beckhoff.com/

UA Java SDK to give access to the Camera, the Weather Station and other assemblies that were not using the Beckhoff platform.

Every OPC-UA interface is described by an Interface Control Document (ICD). These hardware ICDs take the form of Microsoft Excel spreadsheet files with a list of all the supported commands and their attributes (e.g. OPC-UA node name and data type, sampling interval, alarms thresholds, description). As an example Fig. 2 shows some of the commands supported by the Weather Station.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Name of command	Actionee	Short name	OPC_UA node	OPC UA Data type	Sampling Interval (s)	Alarm low	Alarm high	Withdraw alarm low	Withdraw alarm high	Unit	Operation modes	Expected execution time (s)	Maximum execution time (s)	Description
2	GET_WS_EXTTMP	WS	EXTTMP	ns=2;s=exttmp	Int32	2	-15	25			°C				Value of external temperature, expressed in degrees Celsius.
3	GET_WS_DEWPOINT	WS	DEWPOINT	ns=2;s=dewpoint	Double	2					°C				Value of the dewpoint, expressed in degrees Celsius
4	GET_WS_WINDDIR	WS	WINDDIR	ns=2;s=winddir	Int32	2					deg				wind direction value (0°= in no wind data)
5	GET_WS_WINDIR10M	WS	WINDIR10M	ns=2;s=windir10m	Int32	2					deg				wind direction for the 10-min wind guest (0°= in no wind data)
6	GET_WS_WINDSPD	WS	WINDSPD	ns=2;s=windspd	Double	2		60			Km/h				wind speed value
7	GET_WS_WINDGUST	WS	WINDGUST10M	ns=2;s=windgust10m	Double	2					km/h				Max value of the wind gust in the last 10 minutes
8	GET_WS_WND10AVG	WS	WND10AVG	ns=2;s=win10avg	Double	2		36			KM/h				mean wind speed in the last 10 minutes
9	GET_WS_SOLARRAD	WS	SOLARRAD	ns=2;s=solarrad	Int32	2					W/m2				value of the solar radiance
10	GET_WS_EXTUMDY	WS	EXTUMDY	ns=2;s=extumdy	Int32	2	2	90			%				external relative humidity
11	GET_WS_RAINALRM	WS	RAINALRM	ns=2;s=rainalrm	Double	2					mm				rain alarm
12	GET_WS_RAINRATE	WS	RAINRATE	ns=2;s=rainrate	Double	2	0				mm/h				rainfall rate
13	GET_WS_RAINDAILY	WS	RAINDAILY	ns=2;s=dailyrain	Double	2					mm				day rain
14	GET_WS_RAIN1H	WS	RAIN1H	ns=2;s=rain1h	Double	2					mm				last hour rain
15	GET_WS_RAIN15M	WS	RAIN15M	ns=2;s=rain15m	Double	2	0				mm				last 15-min rain
16	GET_WS_BAROMTR	WS	BAROMTR	ns=2;s=baromtr	Double	2					hPa				value of the atmospheric pressure
17	GET_WS_BARTREND	WS	BARTREND	ns=2;s=bartrend	String	2									Current 3-hour barometer trend: 1 - Falling Rapidly 2 - Falling Slowly 3 - Steady 4 - Rising Slowly 5 - Rising Rapidly 6 - No trend info is available 7 - Insufficient data to determine Bar-Trend
															state of the weather station and any error code reading: 0 = no error – device ok 1 = RS232 error –

Figure 2: An excerpt of the ASTRI Weather Station ICD

Four different types of commands are supported:

GET commands represent sensors monitoring points.

SET commands are R/W variables for configuration and parameters.

CMD commands are proper commands.

MODE commands are used to request state machine transitions.

3. ACS COMPONENTS AND GUI GENERATION WORKFLOW

ACS characteristic components are a specific class of ACS components that represents monitor and control points with properties having uniform access, independently of the peculiarities of the communications with the hardware.[5] Characteristic components have usually many repetitions of similar code snippets and in the configuration files to implement the properties and their configuration. As an example the ASTRI Telescope Control Unit interface has more than 350 monitoring and control points, this entails 9000 lines of implementation code for the ACS component, 4700 lines for the associated configuration file, and 1100 lines for its schema file. All without business logic. We resort to code generation to ease the task of writing this repetitive code, with the aim to reduce the number of lines of code that a developer needs to craft and to concentrate resources on behavior implementation in higher levels of abstraction while reducing time from interface specification to testing.

In this context the ICD file not only documents the interface, but also allows us to leverage code generation to help the developers in the repetitive task of producing all the support code that depends on no device-specific logic.

The ASTRI code generator is implemented in a few Python scripts. We are using XLRD[†], a library to extract

[†] http://www.python-excel.org/

data from Microsoft Excel files, to read the ICD files and CHEETAH[‡] as template system.

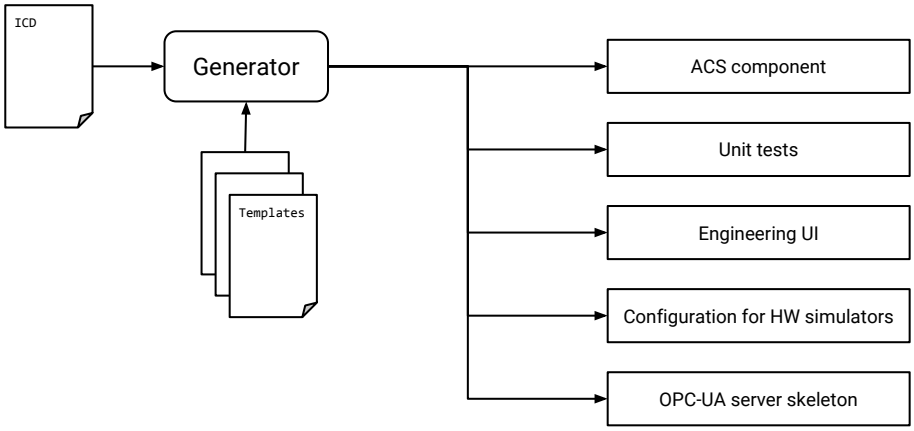


Figure 3: MASS code generation

Fig. 3 depicts the outputs of the generation process. ACS characteristic components are the main target, as they allows access to the hardware capabilities by the ACS based control system, translating each OPC-UA node as an ACS property and each command in the ICD in an ACS method accessible via the Common Object Request Broker Architecture (CORBA) standard by other components. For each assembly all the files required to specify a Java ACS component are generated: a CORBA Interface description language (IDL) file that describe the public interface, the main implementation class, an XML configuration file and the related schema file, an ACS compatible makefile and deployment configuration files.[6]

For each hardware assembly also a stand-alone engineering GUI is generated. The GUI uses JavaFX in a usual Model-View-Controller pattern. The *View* is defined in FXML, an XML-based language that provides the structure of the user interface, the *Controller* is a Java class implementing the necessary logic while the *Model* is a local stub for the remote ACS component. The generated GUI is a complete, although not refined, way to access all the hardware functions, locally or remote (Fig. 4).

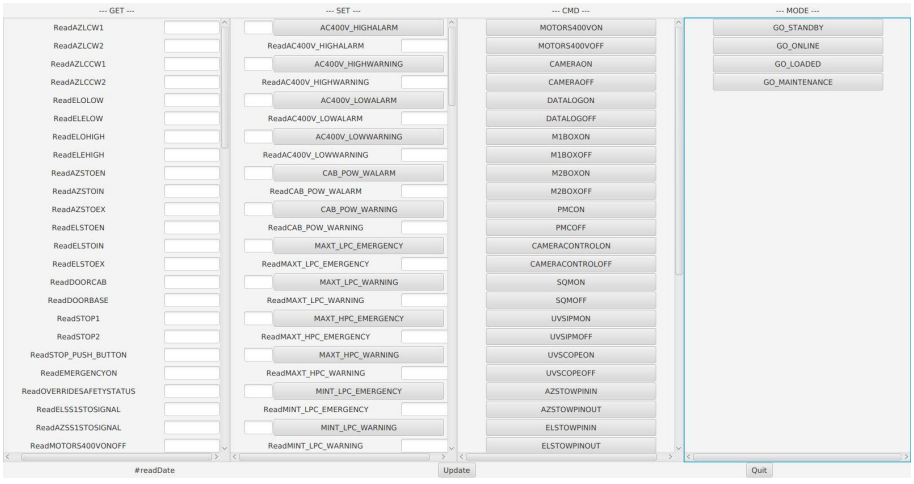


Figure 4: Generated GUI

To test the control software without hardware availability we developed an hardware simulator. The simulator takes as input a configuration file that describe all the monitoring and control points to be simulated and start

[‡] http://www.cheetahtemplate.org/

an OPC-UA server with the appropriated data model and node tree. The required configuration file is also generated from the assembly ICD.

The output from the generation process allows us to have a full test environment by simply providing the ICD: a complete ACS component, a simulator and a user interface are then produced.

Additional output templates are under development: all the software components need complementary unit testing suites, while a skeleton data model description file for an OPC-UA server could be a next possible candidate for the automatic generation process.

4. INTEGRATED ENGINEERING GUI

The use of the complete engineering UIs described in Section 3 could easily be much too cumbersome in some circumstances. While it is always possible to use Python scripting to interact with the generated ACS components we wanted to give an easy way for an operator to use the ASTRI SST-2M prototype. For this purpose we developed an integrated user interface that allows to point the telescope, monitor its operations, and keep track of alarms and faults of the system. We reused the code generated to give access to the telescope assemblies, while we tried to make its management as efficient and intuitive as possible following SCADA (Supervisory Control And Data Acquisition) industry standards and the best practices for electronic instrumentation.

Dedicated panels, their number increasing with the installation of instruments at the site, give access to the different functions available (Fig. 6). A main panel, visible in Fig. 5, allows the monitoring and control of the whole telescope at glance. This main panel shows the time and site related quantities, current telescope motion information, states of telescope assemblies, and safety and emergency information, and allows the pointing by providing a set of coordinates or by selecting a target from a list of objects.

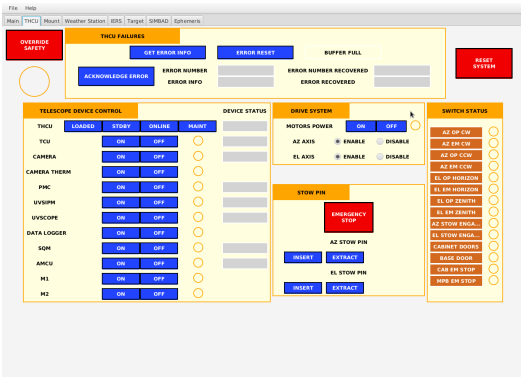
Targets can also be selected with an online research in the SIMBAD Astronomical Database[§] and checked for time visibility at the site with a call to the Staralt web site[¶] in the *SIMBAD* panel. In the *Ephemeris* panel local Sun and Moon ephemeris could be checked in preparation of an observation.

Figure 5: ASTRI Engineering GUI main view

Fig. 7 shows a part of the integrated GUI class diagram. A *SharedControl* class sits between the graphical view, again described by an FXML file with an associated style sheet file, and all the monitoring and control

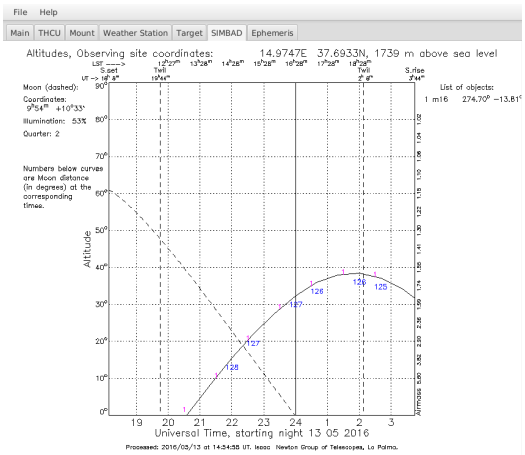
[§] http://simbad.u-strasbg.fr/simbad/
[¶] http://catserver.ing.iac.es/staralt/

(a) *Mount panel.*



(b) *Telescope Heal Control Unit panel.*

(c) *Weather Station panel.*



(d) *Target visibility panel.*

Figure 6: A few of the different GUI panels

functions to be provided. A *BridgeComponent* class, a class already automatically generated from the ICD, gives access to the hardware features. It acquires the related ACS component using an *ACSConnector* class (not depicted) to connect via network with the running ACS session. Monitoring classes (e.g. the *RefreshTCU* class) bind the values of the properties and constantly update in background the view for the operator. The use of independent and distinct threads for background value updating, operations that sometimes take a not negligible amount of time, safeguard the main UI thread responsiveness in all the circumstances. *Astro* is an utility package that take care of all the astronomical calculations, time and coordinate transformation. The *IERS* class retrieves information on Earth orientation and on the celestial reference frame constantly updated by the International Earth Rotation and Reference System Service (IERS)[‖]. The *Staralt* class takes care of the plot of target visibility previously described using an external library[**].

5. CONCLUSIONS

The use of strict ICDs allowed the ASTRI project to use code generation for all the glue and support code between the hardware, accessed via OPC-UA, and the high level control system implemented in ACS. Code generation dramatically reduced the amount of effort and time required for the integration of the assemblies with the control system while at the same time limiting errors and bugs in the code. This allowed to concentrate resources on designing and implementing an integrated engineering GUI built upon the generated components

[‖] http://www.iers.org/

[**] https://github.com/vterron/simbad-resolver

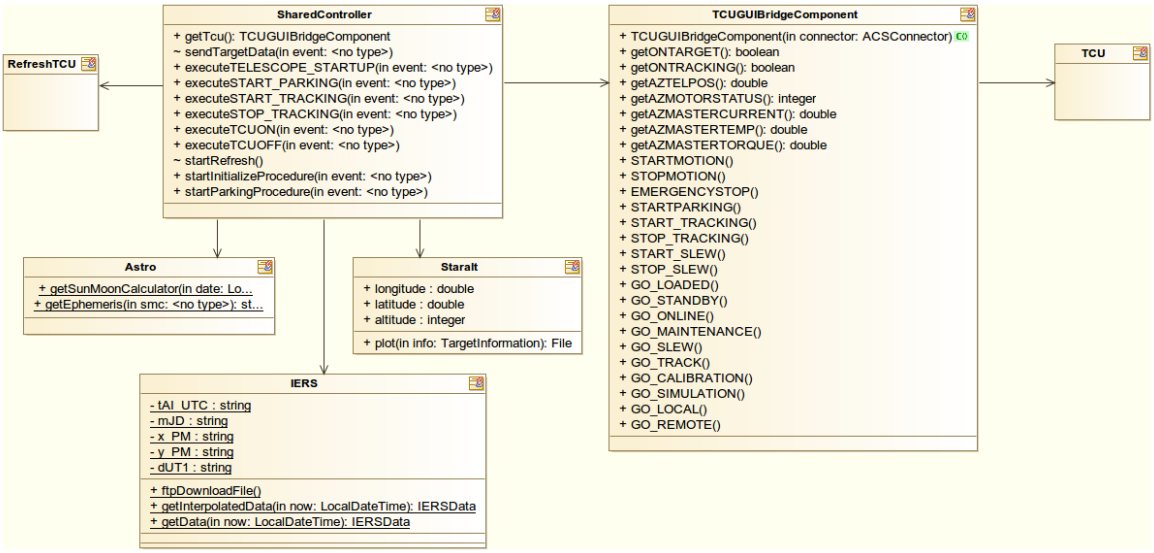


Figure 7: An excerpt from the GUI application class diagram

and libraries. We think that this approach could be valuable also for the CTA environment, where a large diversity of hardware, provided by many different teams, will exist. The integrated engineering GUI is now in active use at the ASTRI SST-2M prototype in Serra La Nave (Italy).

ACKNOWLEDGMENTS

This work is supported by the Italian Ministry of Education, University, and Research (MIUR) with funds specifically assigned to the Italian National Institute of Astrophysics (INAF) for the Cherenkov Telescope Array (CTA), and by the Italian Ministry of Economic Development (MISE) within the "Astronomia Industriale" program. We acknowledge support from the Brazilian Funding Agency FAPESP (Grant 2013/10559-5) and from the South African Department of Science and Technology through Funding Agreement 0227/2014 for the South African Gamma-Ray Astronomy Programme. We gratefully acknowledge support from the agencies and organizations listed under Funding Agencies at this website: http://www.cta-observatory.org/. This paper has gone through internal review by the CTA Consortium.

REFERENCES

[1] Pareschi, G. et al., The ASTRI prototype and mini-array: telescopes precursors for the Cherenkov Telescope Array (CTA) in [*Paper 9906-223 these proceeding*],

[2] Tanci, C. et al., The ASTRI mini-array software system (MASS) implementation: a proposal for the Cherenkov Telescope Array in [*Paper 9913-93 these proceeding*],

[3] Chiozzi, G., Gustafsson, B., Jeram, B., Plesko, M., Sekoranja, M., Tkacik, G., and Zagar, K., CORBA-based Common Software for the ALMA project in [*Astronomical Telescopes and Instrumentation*], 43–54, International Society for Optics and Photonics (2002).

[4] Mahnke, W. and Leitner, S.-H., OPC Unified Architecture-The future standard for communication and information modeling in automation *ABB Review* **3**, 2009 (2009).

[5] Jeram, B., Chiozzi, G., Ibsen, J., Cirami, R., Pokorny, M., Muders, D., and Wischolek, D., Generic abstraction of hardware control based on the ALMA Common Software in [*Astronomical Data Analysis Software and Systems (ADASS) XIII*], **314**, 748 (2004).

[6] Sommer, H., *ACS Java component programming tutorial*. ESO. http://www.eso.org/~almamgr/AlmaAcs/OnlineDocs/ACS_JAVA_Component_Tutorial.pdf.