ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-MOAR03

LOFAR2.0: STATION CONTROL UPGRADE

T. Juerges[1,2*], J. D. Mol[2†], T. Snijder[2‡]
[1]Square Kilometre Array Observatory (SKAO),
Jodrell Bank, Macclesfield, SK11 9FT, United Kingdom
[2]Netherlands Institute for Radio Astronomy (ASTRON),
Oude Hoogeveensedijk 4, 7991 PD Dwingeloo, Netherlands

Abstract

After 10 years of operation, the LOw Frequency ARray (LOFAR) telescope is undergoing a significant hardware upgrade towards LOFAR2.0. The hardware upgrade will enable the phased array telescope to observe at 10-90 MHz and at 120-240 MHz frequencies at the same time. With the upgrade comes also the chance to review LOFAR's Control System and to make it ready for the next 10 years of operation at the forefront of low-frequency astronomy. In this work we will give a brief overview over the LOFAR telescope with its more than 50 geographically distributed receiver locations (LOFAR Stations), and the software that is necessary to monitor and control every single one of them. We will then describe the Station Control architecture, with its software design and how it is implemented in Python 3 with Tango Controls, OPC-UA clients and deployed as Docker containers. Lastly we will report on the successful use of open stack software like ELK and, Grafana.

LOFAR TELESCOPE OVERVIEW

LOFAR [1] is a geographically distributed radio telescope array, consisting of around 60,000 dipole antennas. The antennas are grouped into 56 *stations*, 38 of which are deployed in the Netherlands, and the remaining 14 in other countries across Europe. The scientific data from these stations are streamed to our real-time GPU correlator [2] in Groningen, the Netherlands. Thusly correlated (and beamformed) data products are subsequently post processed. We send the end result to our tape archives in the Netherlands, Germany, and Poland. There the data products are made available for download by the scientists.

LOFAR2.0 Station Upgrade

A LOFAR2.0 station will, like the current LOFAR stations, consist of up to 96 high-band dipole tiles (110–250 MHz), and 96 low-band dipole antennas (10–80 MHz). The tiles and antennas are connected to 64 Receiver Control Units (RCUs), which apply a configurable analog filter.

LOFAR2.0 will redesign these RCUs to have improved filters. The new RCUs will also have the ability to process data from all antennas simultaneously [3].

The RCU output is sent to station signal-processing boards, to be beamformed and converted into UDP packets. These packets are streamed over 10 GbE fibres to the correlator.

Station Signal Processing

A station will contain up to 8 Uniboard^2 processing boards [4]. The boards use 32 FPGAs in total to sample and digitise the signal at 200 MHz and calibrate, exchange, beam form, and correlate their input. The end result is a 3–9 GBit/s data stream to the correlator per station, as well as up to 300 Mbit/s of statistical information.

STATION MONITORING AND CONTROL

The hardware in each station exposes tens of thousands of monitoring and control points through various interfaces and protocols. Basically the Monitoring and Control of a LOFAR2.0 station can be condensed into two simple operations at a station:

- Modify the behaviour of our hardware over time, e.g. point at different sources in the sky.
- Verify that the dynamic behaviour has been successfully modified.

In addition to the basic concepts of station operation, the nature of the distributed telescope requires that we also keep track of the system health and let the station autonomously act on extreme scenarios such as overheating of the equipment.

Finally, we are interested in monitoring the quality of the data recorded through our antennas and produced by our processing boards. To this purpose, the signal-processing boards continuously emit statistical information from several points in the signal chain.

OPC UA as a Common Hardware Interface

The hardware that is to be monitored and controlled in a LOFAR2.0 station comes in various shapes and forms. This could imply that a station's Monitor and Control system would have to support a variety of different hardware interfaces and protocols. We have, for example:

- Uniboard^2 processing boards: I^{2}C
- FPGAs on Uniboard^2: UCP (Uniboard Control Protocol) over IP
- RCUs: I^{2}C
- Power supplies: PLC interface
- Temperature sensors: PLC interface
- Network switch: SNMP

From prior experience in LOFAR1, as well as in other telescope monitor and control systems (ALMA, WSRT), we

* thomas.juerges@skao.int
† mol@astron.nl
‡ snijder@astron.nl

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-MOAR03

learned that a common hardware communications protocol minimises software complexity. Engineering and maintenance personnel also benefits from a common hardware interface on site as well as remote. The tooling can be unified and be kept simple to use, yet powerful enough to debug hardware at an acceptable low enough level.

Thus we expect a common hardware protocol to have the following properties:

- Open tooling available for remote hardware and interface debugging
- No need for specialised equipment to communicate with the hardware (JTAG dongles, I^2C cards, etc.)
- Protocol client implementation available in Python 3 to integrate seamlessly with the Station Control software
- Single server supports multiple clients
- Support for a direct mapping of Monitor and Control points from hardware to software.

The widely adopted OPC UA protocol [5] supports all of the properties above. In addition, OPC UA supports the following aspects in a client-server system:

- OPC UA protocol is based on TCP/IP: Connection reliability, simplicity of software integration
- Protocol insensitive to specific hardware timing: We chose to implement timing sensitive behaviour server side.
- Ability to browse self-describing attributes and methods: Enables introspection of the available Monitor and Control points (name, type, dimensionality)
- Server implementations in C/C++ and Python 3: Allows hardware engineers to implement the servers observing timing specific behaviour and other details of the hardware.

OPC UA is a feature-rich protocol, but does not require the user to make use of the full set of features. For LOFAR2.0 we chose to limit the use of OPC UA features to:

- Browsing of attributes and methods
- Reading and writing of attributes
- Calling of methods
- Use of OPC UA native data types only

OPC UA Hardware Translators

We run a dozen of OPC UA servers on Raspberry Pis in the LOFAR2.0 station as interfaces to the station's hardware. These servers „translate" the hardware-specific communications protocols[1] into OPC UA attributes and methods. Thus the name Hardware Translator, or Translator for short, for the OPC UA server.

HIGHER-LEVEL MONITOR AND CONTROL WITH TANGO CONTROLS

For the LOFAR2.0 Station Control software Tango Controls [6], an open source and distributed object-oriented Monitor and Control system, has been chosen as the core software framework. It enables the software engineers to focus on designing and implementing feature rich Monitor and Control application hierarchies, that consist of an arbitrary number of abstraction levels. The software engineer does not have to implement low level functionality, like for example automatic polling of an attribute's value from a source, attribute value on-change events, attribute value alarm subscriptions or a system wide logging system. It also provides a transparent and configuration-independent component discovery through its underlying CORBA [7] foundation. Tango Controls also provides a lot of tooling for rapid prototyping and supports native Python 3 bindings.

Devices and Device Servers

A LOFAR2.0 station has no moving parts — the station is controlled through Station Control, the software stack hosted on a computer located in each station. It manages the thousands of control and monitoring points exposed by the Hardware Translators.

In order to map the Translators into the Tango Controls realm, we represent each Translator in Tango Controls as one or more Devices. These Devices are implemented in Python 3 with PyTango [8], the high-level Python 3 API for Tango Controls.

Devices are executed in Device Servers, which can run one or more Devices using multithreading. In order to avoid a shared process space and thus shared crashes and performance bottlenecks, we chose to run each Device in a dedicated Device Server.

The Station Control Devices

We designed a hierarchy of Devices that together form the Station Control Devices landscape, as shown in Fig. 1. The lowest layer Devices communicate with the Hardware Translators, while the highest layer exposes business-logic functionality to our central Telescope Manager.

Especially our signal-processing boards expose a lot of functionality through numeric properties that can be set. In order to keep the responsibilities of each Device manageable, we divide the functionality exposed by some of the Hardware Translators into multiple Devices. OPC UA makes this a seamless experience because it does not enforce and one-to-one mapping between clients and servers.

To further limit the number of attributes per Device, we group most hardware Monitor points into arrays. For example, in one Translator, an array of 16 booleans can flag communication issues with the 16 FPGAs that the Translator monitors and controls. About a hundred arrays remain, ranging from tens to tens of thousands of elements per array.

Tango Controls Attribute Wrapper

The multitude of Tango Controls attributes forced us to look at representing them as efficiently as possible in our source code. For that we developed a generic „attribute wrapper" framework. It allows us to map almost any source of information to a set of Tango Controls attributes. The Attribute Wrapper takes care of the interaction and management of the data source, while the software engineer provides

[1] See section "OPC UA as a common hardware interface".

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-MOAR03

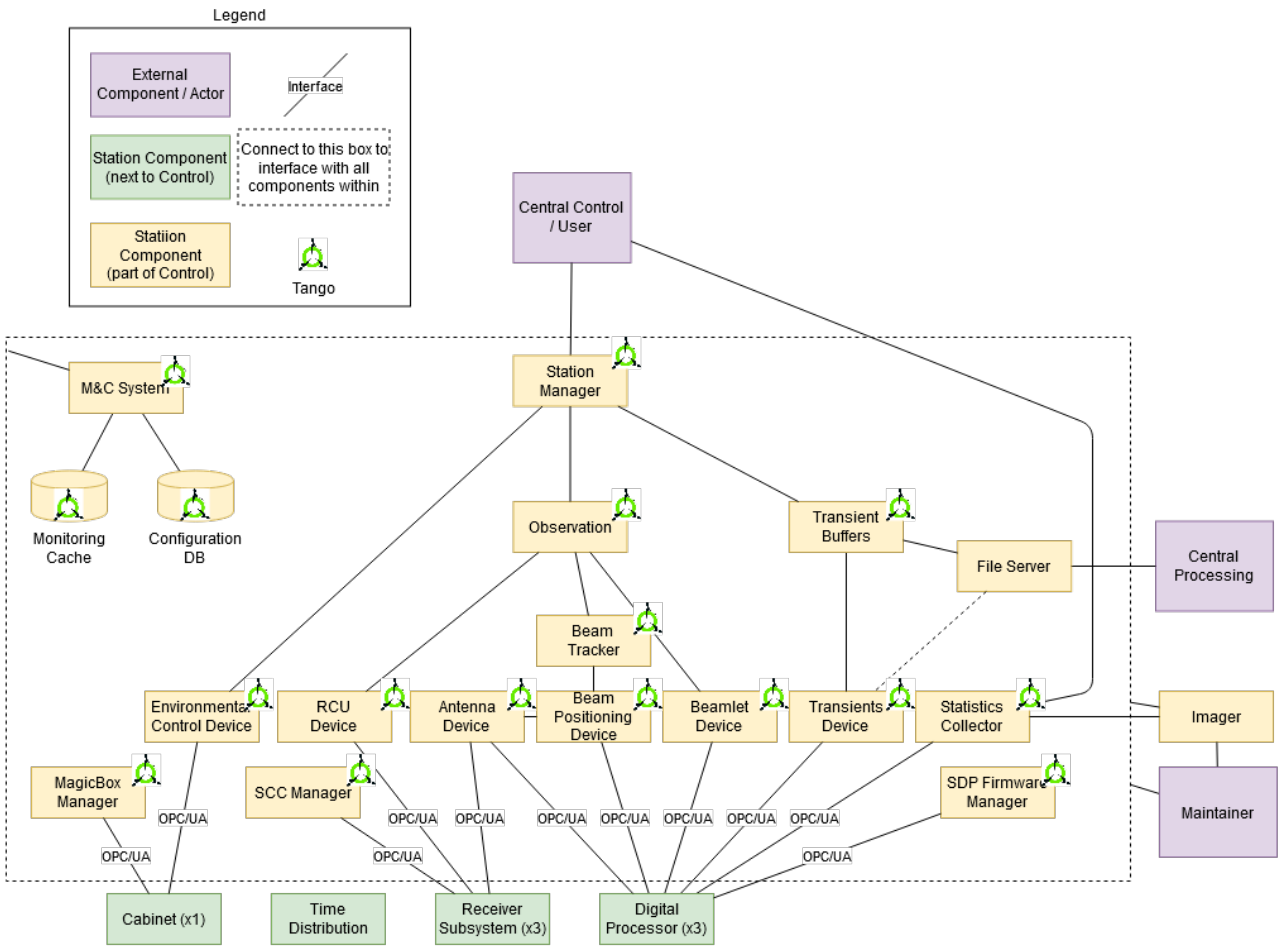

Figure 1: Station Control Devices landscape. The yellow boxes represent software components. Components belonging to Tango Controls, as well as our Tango Devices, are annotated with the Tango logo. The green boxes represent collections of Hardware Translators. Purple boxes represent LOFAR2.0 components external to the station.

just a minimum amount of information when constructing an attribute. The example source code in Fig. 2 shows just two lines of source code that fully constructs two complex Tango Controls attributes. One attribute just reads its value from an OPC UA server, the other one reads from and writes to an OPC UA server.

The Attribute Wrapper requires a protocol or interface-specific implementation. This is as simple as implementing the following functions once per type of interface:

- Connect to the data source
- Disconnect from the data source
- Generic read function that reads a value from a location of the data source
- Generic write function that writes a value to a location of the data source

As seen, in the case of an OPC UA data source for attributes, the attribute construction with the Attribute Wrapper requires just a simple string annotation that tells the built-into the Attribute Wrapper OPC UA client where it can find the respective OPC UA attribute in the server. Once the OPC UA connection is established in the device's initialisation phase, the attributes are automatically linked to

```
RCU_temperature_R = attribute_wrapper(
    comms_annotation = ["RECV", "RCU_temperature_R"],
    datatype = numpy.float64, dims = (32,))
HBA_element_beamformer_delays_RW = attribute_wrapper(
    comms_annotation = ["RECV", "HBA_element_beamformer_delays_RW"],
    datatype = numpy.int64, dims = (32, 96), access = AttrWriteType.READ_WRITE)
```

Figure 2: Example code showing the definition of two attributes in a Tango Controls Device. The `comms_annotation` field encodes the OPC UA specific parameters the OPC UA connection class needs to access this attribute.

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-MOAR03

the actual OPC UA connection and the relevant OPC UA attribute (and unlinked on de-initialisation).

The result is a very efficient and simple to use framework which we extended to cover the management of, among others, SNMP devices, Docker containers and INI-format files.

Statistical Data Streams

The FPGAs produce streams of statistical data about their main 3–9 Gbps data streams flowing out of the station: crosslet statistics (XSTs) revealing phase and amplitude differences between the inputs, frequency & amplitude plots of each individual antenna (SSTs), and power spectra of the output data stream (BSTs). These statistical data are divided over UDP packets, sent collectively by all FPGAs.

The FPGAs require to be configured with the MAC and IP address of the receiver, of which there can be only one, as UDP Multicast is tricky to set up, especially across switch boundaries. Yet multiple users are interested in these statistics: telescope operators and maintainers use them to verify the data quality and expert users record the stream to produce additional scientific data products.

In LOFAR1, we allow the UDP stream to be configured by any interested user, but learned that such a solution gives an awful user experience. Receiving and recording UDP streams tends to require low-level programming, easily leads to data loss and demands that the user does not forget to turn off the stream explicitly.

The conclusion from past experiences is that we designed a more elegant solution. Each type of statistical data is configured to be sent to a dedicated Device Server. The Statistics Device, that runs in that Device Server, collects the UDP streams from all FPGAs. With the help of our Attribute Wrapper framework, we easily expose the resulting statistical data matrices, the metadata, such as timestamps and other properties, and meta-statistics, such as packet and error counters.

These statistical data matrices represent the most recent statistical values as computed by the processing board. The Statistics Devices expose the matrices as arrays of f.e. 192x512 floats, or 192x192x8 complex values, which typically get updated every second or faster.

Statistics Replicator Monitoring these matrices automatically is rather computing expensive, so we added another interface to record these statistics over time: We expose a TCP port per type of statistic. Any connecting user receives the raw UDP packet stream over this TCP connection until the user disconnects or the Device is shut down. The Device allows as many connections as the available bandwidth allows. Monitoring points for this „Statistics Replicator“ are exposed through our Attribute Wrapper as well.

Statistics Writer Furthermore, we provide the user with a Statistics Writer. It reuses the same classes to convert the Statistical Data Stream into matrices on the client side. Then it just writes them as an HDF5-file to disk. The HDF5-file contains proper data structures and attributes of the Statistical Data Stream.

Observation and Tracking Devices

A station's main task is to partake in observations performed by one to all stations in concert. Observations carry many configuration settings and require parts of the hardware to be reconfigured every second. The adjustments of hardware parameters are necessary to track the observed source across the sky as Earth rotates. We are in the process of managing this behaviour by implementing a Tracking Device to perform the necessary computations.

For each observation, dedicated Observation and Tracking Devices are dynamically created and started. Both work together to maintain the station in a state such that it fulfills the observational requirements throughout an observation. This is done without interfering with observations that are executed in parallel on the same station. The Tango Controls system seamlessly allows this dynamism.

DEPLOYMENT

The core of our Station Control software stack is based on the Docker image framework for Tango Controls, originally provided by the Square Kilometre Array Observatory (SKAO) [9]. A part of the SKAO framework provides a basic setup of Docker containers that run the core Tango processes and an example Device Server. We have significantly extended this part of the framework, for example, by:

- Making the Device Servers that run inside Docker externally reachable, by adjusting the underlying CORBA [7] parameters
- Running each Device Server in a dedicated Docker container
- Added a rich set of integrations with modern tool chains, such as ELK [10], Prometheus [11], Grafana [12], and Jupyter Notebooks [13] (see below)
- Added an integration and unit-test framework, run from Gitlab CI/CD

Stacking Open Interfaces

Modern toolchains make it easier to attain high levels of integration between complex software stacks. We added a significant number of web interfaces by adding off-the-shelf Docker images that needed only little configuration. The separation of services into Docker containers and linking them through open interfaces proved to be a powerful combination.

Jupyter Notebooks A Jupyter Notebook server [13] allows users to access all control and monitoring points in a station. It comes with PyTango [8] pre-installed and pre-configured for easy access to all Station Control Devices. We provide example Notebooks (see Fig. 3) and the user can save and load their own. Plotting libraries are pre-installed as well which makes the Jupyter Notebooks a powerful engineering interface.

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-MOAR03

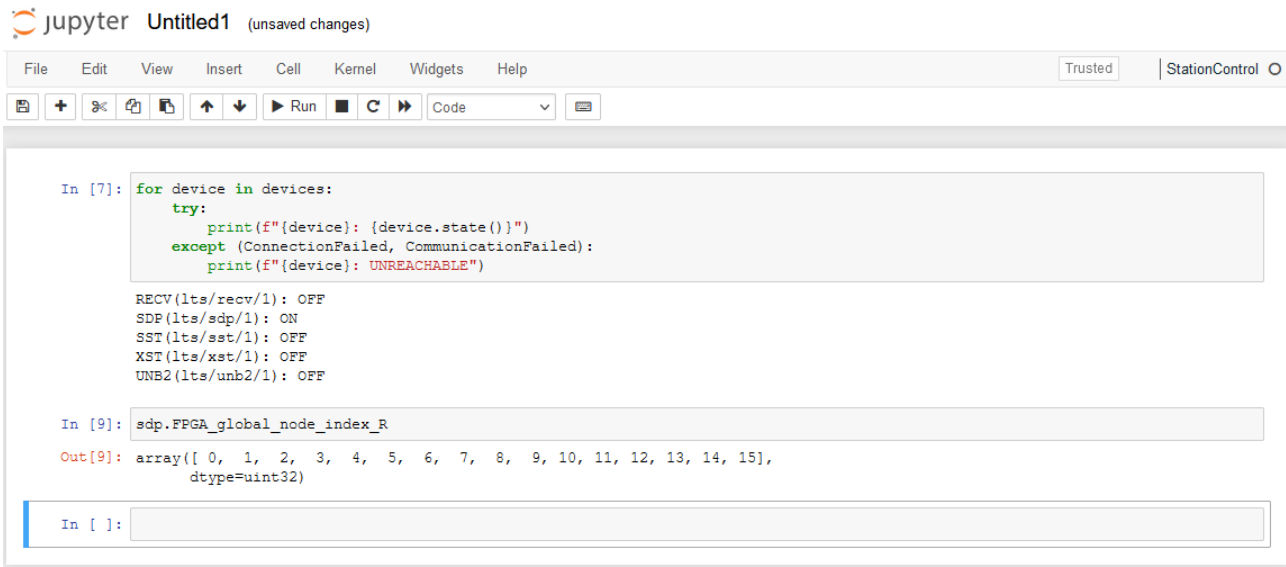

Figure 3: Example of using the station's Jupyter Notebook to read the state of the Devices and one of the Device attributes.

ELK stack An ELK stack [10] collects logs from all Devices, Device Servers and several of the other services that we run. We added Python LogHandlers and Filters to automatically route all Python log output to Tango's log system, our ELK stack and stdout. Each log line is annotated with the name of the Tango Device that generated it. This both leads to richer logs and alleviates the need of having to forward the Tango log streams to our utility classes, as all classes can now just use Python's native logging interface yet still log to Tango.

Grafana We added and adjusted the SKA's experimental TANGO-Grafana exporter [14] to allow all but the biggest arrays of monitoring points to be periodically scraped by a Prometheus time-series server [11]. A Grafana [12] dashboard system (see Fig. 4), also installed on the station, has this Prometheus server configured as its data source, along with the Tango Archiver database, Tango database, and our station's ELK stack. Pre-configured dashboards provide a rich overview of the station's state and state history, from temperature sensors to current hardware settings and active software and firmware versions and the state of all Docker containers.

Sphinx & Read the Docs Finally, we generate user documentation through a Sphinx [15] integration and a web hook from our Gitlab server to Read the Docs [16]. Having our source code publicly available made this trivial to setup.

Figure 4: An excerpt from one of our Grafana dashboards.

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-MOAR03

Benefits of Simulating the Hardware Interfaces

The LOFAR2.0 station contains custom hardware that the software engineers do not necessarily have access to. We discovered early the benefits of simulation of the interfaces that hardware exposes. Our Translator setup is such, that it allows us to run them stand-alone in Docker containers and simulate the hardware in a very basic and static fashion. While these instances do not have access to backing hardware, they do expose the OPC UA interface just like the actual equipment would. This allows us to write integration tests to verify the attribute names, types, dimensionalities, and other basic properties.

CONCLUSION

Over the past 18 months we found Tango Controls [6] to be a very powerful Monitor and Control framework that makes the life of our software engineers much easier. Development of software devices in Python 3 became a simple task. Rapid prototyping with quick cycles of testing, debugging, bug fixing showed to be extremely valuable not only for software-only devices but also for devices that represented parts or all of one of our Hardware Translators. Especially when hardware was involved, our Attribute Wrapper has shown its incredible power due to its built-in separation of Tango Controls on one side and interfacing with hardware on the other side.

Without Tango Controls, our Attribute Wrapper, OPC UA and the SKAO Docker images for Tango Controls, we would not have been able to deliver the wealth of functionality with the little manpower that we had at our disposal.

The entire source code of the LOFAR2.0 Station Control project [17] is made publicly available under the Apache, Version 2.0, Open Source license [18]. The SKAO Docker images that we use are publicly available at the SKAOś artefact repository [19].

REFERENCES

[1] M. van Haarlem, M. P. Wise *et al.*, "LOFAR: The LOw-Frequency ARray", in *Astronomy & Astrophysics*, vol. 556, 2013, p. A2. doi:10.1051/0004-6361/201220873

[2] P. C. Broekema, J. J. D. Mol, *et al.*, "Cobalt: A GPU-based correlator and beamformer for LOFAR", in *Astronomy and Computing*, vol. 23, 2018, pp. 180–192. doi:10.1016/j.ascom.2018.04.006

[3] H. W. Edler, F. de Gasperin, and D. Rafferty, "Investigating ionospheric calibration for LOFAR 2.0 with simulated observations", in *Astronomy & Astrophysics*, vol. 652, 2021, p. A37. doi:10.1051/0004-6361/202140465

[4] G. W. Schoonderbeek, A. Szomoru, *et al.*, "UniBoard^2, A Generic Scalable High-Performance Computing Platform for Radio Astronomy", in *J. Astron. Instrum.*, vol. 08, no. 02, 1950003, 2019. doi:10.1142/S225117171950003X

[5] W. Mahnke and S. H. Leitner, "OPC Unified Architecture - The future standard for communication and information modeling in automation", in *ABB Review*, vol. 3, 2009, pp. 56–61.

[6] J. M. Chaize, A. Götz, W. D. Klotz, *et al.*, "TANGO - An Object Oriented Control System Based on CORBA", in *Proceedings of the 7th ICALEPCS (ICAPELCS'99)*, 1999, paper WA2I01, pp. 475–479.

[7] Object Management Group, "CORBA: Common Object Request Broker Architecture", https://www.omg.org/spec/CORBA/

[8] S. Rubio-Manrique, T. Coutinho, and R. Suñé, "Dynamic Attributes and Other Functional Flexbilities of PyTango", in *Proc. ICALEPCS'09*, 2009, paper THP079, pp. 824–826.

[9] Square Kilometre Array Observatory, https://www.skao.int/

[10] ELK stack, https://elastic.co/what-is/elk-stack

[11] Prometheus, https://prometheus.io

[12] Grafana, https://grafana.com

[13] Jupyter Notebook, https://jupyter.org

[14] M. Di Carlo, P. Harding, *et al.*, "TANGO-grafana: an online diagnostic tool to assist in the analysis of interconnected problems difficult to debug in the context of the Square Kilometre Array (SKA) telescope project", in *Proceedings of the SPIE*, vol. 11452, Software and Cyberinfrastructure for Astronomy VI, 2020. doi:10.1117/12.2576297

[15] Sphinx Python Documentation Generator, https://www.sphinx-doc.org

[16] Read the Docs, https://readthedocs.org

[17] Git repository for ASTRONś LOFAR2.0 Station Control source code,https://git.astron.nl/lofar2.0/tango

[18] Apache Licence, Version 2.0, https://www.apache.org/licenses/LICENSE-2.0.html

[19] Square Kilometre Array Observatory: Docker images for Tango Controls, https://artefact.skao.int/#browse/browse:docker-all:v2