ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-FRAR01

TARANTA, THE NO-CODE WEB DASHBOARD IN PRODUCTION

M. Eguiraun[*], V. Hardion, Y. Li, M. Saad, A. Amjad, J. Rosenqvist, L. Nguyen, J. Forsberg
MAX IV Laboratory, Lund, Sweden
M. Canzari, V. Alberti INAF-OAAB, Teramo, Italy
H. Ribeiro, Atlar Innovation, Portugal
V. Alberti, INAF-OATs, Trieste, Italy
A. Dubey,Persistent Systems, Pune, India

Abstract

The remote control and monitoring of accelerators and experimental setup has become essential when remote work has become the norm for the last two years. Unlike the desktop user interfaces which have been developed for the use from physical workstations, web application are naturally accessible remotely via the ubiquitous web browsers. On the other hand, Web technology development requires a specific knowledge which has yet to be disseminated in the control system engineering and desktop frameworks still have the benefit of rapid, and easy development even for the non-specialist. Taranta Suite is a collection of web applications jointly developed by MAX IV Laboratory and the SKA Observatory, for the Tango Control System. In line with the "no-code" trend for the users, truly little knowledge of web technologies is needed. An operator can create a graphical user interface on-the-fly and can share it instantly. Authentication and authorization ensures that the right access level is given to the user. This paper will describe the system, the details of its implementation, and the first usage at the different facilities.

INTRODUCTION

There is no doubt about the usability and the optimised user experience of web interfaces and applications in everyday life situations. The last decade has seen an explosion of these kind of developments. New tools, new frameworks and new wide-spread applications have gained fame and number of users and slowly displaced traditional desktop applications. However, scientific environments are usually built on traditional and well known infrastructure, lagging behind software innovation and trends. MAX IV and SKA facilities were pushing and promoting the usage of web applications in their respective communities and in 2019 they joined efforts and gave birth to Taranta, a web application for building user interfaces in a Tango ecosystem [1]. The tango community renamed it from the previous Webjive name [2].

This new application is profiting from recent years of development in User Experience (UX) and User Interface (UI) web frameworks. It provides an out of the box, modern and stylish environment for accessing the most important functionality of a Tango device, from what is called the Device View [2]. Simple and powerful enough for fast access, however, the most remarkable functionality is the Dashboard View. This element is where the user can create its own user interface by drag and drop components (which are called Taranta Widgets) and easily configure and link them to Tango devices. This is where the idea of No-Code plays a fundamental role, Taranta leverages the UI development to the end user.

NO-CODE PARADIGM

The lead time to get a new user interface is usually very long. The software development and UI design have to follow a number of established stages starting from the user requirement gathering to the usage of a final product. The no-code trend define a way for any end-user to develop their own software and then to make it available immediately to a larger audience of the same end-user group [3]. This way, user of a no-code system doesn't need to be knowledgeable in software development and how the software is deployed to be able to add value into the system. All the infrastructure is completely transparent and does not prevent fulfilling their will.

There is a lot of advantage for the users to bypass the traditional software development chain. First of all, as users themselves, they know exactly what the software should look like and which feature is expected. The users own the requirements like in any development method, although in a no-code system many filters are avoided since the users develop their own product. Writing down the requirements, understanding the specification and code development throughout different persons are some examples of filters which attenuate the original idea.

The concept of accelerating the software development has started years before the no-code trend arrived. Prototype generated by sketch of the Graphical User Interface (GUI) in the Rapid-Application Development (RAD) [4] made a step closer to the final product by developing only the functionalities. An extension of the same concept appears in the early 2000s for the creation of UI based on the definition of the domain i.e in the model development driven (MDD) all the structure of the software (Model) and part of its implementation is generated from the requirement. Visual programming language helps the power-users with enough software skills to design a system rapidly for their local applications i.e workbench like LabView can produce application with no manually written code code. The dissemination to a larger group of users can be delegated later to the software engineers.

[*] mikel.eguiraun@maxiv.lu.se

In our Scientific Facilities and in particular in the Control System, the software development has followed the same trend all the long to the industry standard evolution. Application like Taurus Designer [5] for the Tango Control System and Control System Studio for EPICS [6] allows to leverage the skills of the scientist to create GUI on top of their Control System. No-code introduces another level of independence for the user and rely on them to also publish their software to an entire group of people without involving any IT staff for the deployment, which is a large cost of a software. Not only can the end-user benefit from this possibility but also the they can leverage the use of technologies traditionally separated, e.g. web technologies and control system frameworks.

The No-code platform are designed from the web technology for its low-cost of deployment of application. Only one computer server running a web application is necessary to allow many clients to access the program. On the contrary, a desktop based application has to be deployed on every single workstation which is usually a long and not user-friendly process.

Although No-code is completely oriented to simplify the end-user development, the software developers are still needed to program the elementary bricks of the application.

TARANTA

A Taranta application is composed of two main elements: the backend and the frontend application. The backend provides a GraphQL API to a Tango control system. The client is a web application that provides a generic tango device view, similar in functionality to the well known Jive desktop application, but it also provides a dashboard. A blank canvas for the users to build their User Interfaces.

Figure 1 displays how Taranta components are linked together. It is composed of several applications, or micro-services, this way each component is developed and tested independently. The next sections will describe the most relevant aspects of the main elements.

TangoGQL

GraphQL is a modern query protocol for the application layer developed by Facebook [7]. It provides a unified interface between the client and the server for performing data oriented actions e.g fetching and manipulation. It removes the need of having multiple endpoints for data manipulation to/from the server and instead returns the data what the client asks for over a single API endpoint. The advantage of this approach is that the client asks explicitly for what it needs. In addition, due to its schema-based implementation, the extension of the API does not change the API interface so that compatibility between versions is simplified.

TangoGQL [8] is an implementation of GraphQL over Tango. It provides a communication based on web-socket [9] for subscribing to asynchronous attribute value events, and a GraphQL interface to the Tango database. The standard operations over a Tango device are supported, for example

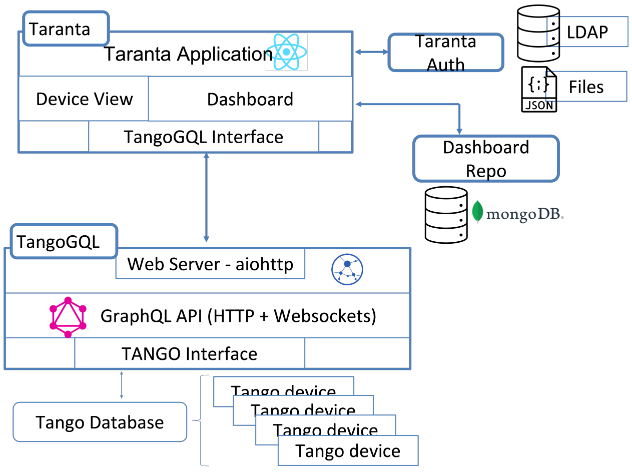
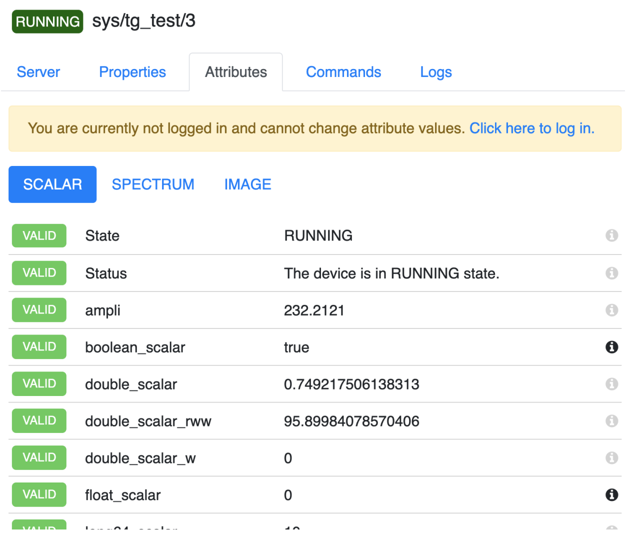


Figure 1: Structure of Taranta, at the bottom and connected to the tango ecosystem there is the TangoGql interface, to which the Taranta web application talks to. Together with the authentication and repository database systems, Taranta suite is complete.

read/write access to device properties and attributes, and command executions. It is developed in Python 3.9.

Front-End

The front-end of Taranta is a React [10] application that is used both to browse, inspect and control Tango devices and to create and run dashboards, each composed of widgets. It accesses the Tango Control System through the TangoGQL API, the communication between Taranta and TangoGQL is managed by an appropriate frontend component. The first developed component was the Device Viewer, Figure 2,


RUNNING sys/tg_test/3

Server Properties Attributes Commands Logs

You are currently not logged in and cannot change attribute values. Click here to log in.

SCALAR SPECTRUM IMAGE

VALID	State	RUNNING
VALID	Status	The device is in RUNNING state.
VALID	ampli	232.2121
VALID	boolean_scalar	true
VALID	double_scalar	0.749217506138313
VALID	double_scalar_rww	95.89984078570406
VALID	double_scalar_w	0
VALID	float_scalar	0

Figure 2: Generic Tango device viewer. Several tabs are available for displaying different types of information, device properties, attributes, commands and logs. The attribute view is further split into the different data types present in the given attribute.

which provides a generic view for any tango device running in the system. It has a search section, which aims at helping the user navigating the full tango device tree hierarchy. Once the desired device is selected, the right section of the page will display several navigation tabs for accessing device properties, attributes (further split based on the data format), commands and logs. Any user can see everything, but only logged in users can modify attributes or execute commands. The values of the attributes are updated at real time. In addition, the selected device name is appended to the page url, thus, it can be shared with others colleagues to access the same view.

As monitoring tool the Device Viewer is a very fast way to access a Tango device, but it lacks of any kind of customization. In order to provide a configurable user interface the Dashboard View was developed. It belongs to the Taranta front-end but it is a very different user interface. As opposed to the static device view, here the user can create their User Interface starting from a blank page and simply dragging and dropping widgets into it. Each widget comes with its own configuration, depending on the purpose of the widget, but they all provide a Tango device (or attribute) selection.

Taranta widgets A widget is a dashboard component to serve the function of interacting with Tango devices. There are different types of widgets. A command can be sent to tango device through command related widgets. An attribute can be read/write through specific attribute widgets. Using widgets, users are able to monitor attribute changes and take remote control operations.

Available widgets The available widgets fall into different categories according to their functions: labels, attributes, commands and plotting. The label widget is mainly used to display static information. It is basic and also flexible with a hyperlink feature, that can be configured by the user to switch between different dashboards. It enables well-structured dashboards.

The attribute widget family is used to display the value of interested attributes, which also includes the device status. The displayed information and text size can be customized to user settings. Several different widgets are available depending on the data type of the attribute (numeric, boolean, etc.), but also some widgets provides extra functionality for example the SimpleMotor and SardanaMotor widgets, where movement steps can be defined and move commands issued.

A command widget enables users to send commands with parameters to tango devices. It also enables users to customize widget settings based on their needs. Again, several widgets are available for different command arguments and data types.

The array type attributes have its own set of widget. For example, a spectrum widget is a special attribute widget, which allows plotting of 2-dimensional attributes. Multiple spectrum attributes can be plotted on the y-axis against one spectrum attribute on the x-axis . The plots are updated when a new attribute value is pushed in by its associated tango device. The rendering of plot is achieved through Plotly [11], which enables user interaction with the plots including zooming, extracting specific graph parts, and also axes resetting. Several widgets are available providing different visualization options for example heatmaps, scatter plots, table like view, a few others.

The variable selector widget is a new feature which enables the interaction between a widget and a dashboard variable. A dashboard variable allows users to create a parametric dashboard where a running device can be replaced by a variable in widget configuration. Users can associate a device to a dashboard variable and change device in run mode. All widgets having this device which subscribed to this variable will be updated with the new device value.

Currently 25 different widgets are available, and more coming regularly due to requests from our user community. Moreover, development is ongoing to allow users to create new widgets from the existing ones.

Creating a new widget A widget is composed of a definition (also called a widget definition) and a React component. The definition is a declarative object describing the characteristics of the widget and the inputs that it receives.

A widget definition starts with a JSON object. An example of a definition is as follows:

```
const definition = {
    type: "OUR_NEW_WIDGET",
    name: "Our New Widget",
    defaultWidth: "10",
    defaultHeight: "15",
    inputs: {
        device: {
         type: "device",
         publish: "$device",
       },
       position: {
         type: "attribute",
         device: "$device",
         attribute: "Position"
       },
       ....
    }
}
```

In this example a new widget is defined with some styling options as well as the required input fields, in this particular case a tango device must be defined and which attribute of that device should be considered position. In addition, Table 1 describes a brief description about required widget definition keys.

Different types of inputs can be defined through their corresponding input definition interfaces. For example, an attribute can be defined with *AttributeInputDefinition* where the bounded device and attribute name as well as other characteristic keys are set. Different from other inputs, the device name can be published to a variable which is available for other inputs. For TypeScript implementation, the widget inputs can be obtained through *WidgetProps* type mapping, which makes the process much easier to develop a new widget and avoid type inaccuracy [12].

Table 1: Description of Widget Definition Keys

Key	Type	Description
type	string	Type identifier for the widget. Must be unique (e.g. "ATTRIBUTE_PLOT").
name	string	The name of the widget shown to the user (e.g. “Attribute Plot”).
defaultWidth	number	Default width (in number of tiles)
defaultHeight	number	Default height (in number of tiles)
inputs	object	An object where the keys are input names and the values are corresponding to their input definition.

In the corresponding React component, the declared inputs are made available through a prop named *input*. A minimal example of a React component for our new widget would as follows:

```
class OurNewWidget extends Component{
    render() {
        const {
            position
        } = this.props.inputs;

        return (
           <div>
             Position: {position.value}
           </div>
        );
    }
}
export default {
    component: OurNewWidget,
    definition
}
```

This minimal example describes how to create a basic widget. All the rest of the required interaction with the dashboard page as well all the communication with the tango device is already handle by Taranta, no additional development is required. Figure 3 shows the newly created widget in action, in both editing mode for selecting a tango device as well as in running mode displaying an attribute value.

Taranta Dashboard Repository

The dashboard repository is an API for fetching and storing the dashboards. It is crafted from ExpressJS and communicates with MongoDB to store and fetch dashboards. Few endpoints are open to provide data to anonymous users, for example, fetching the dashboards. Whereas other endpoints are restricted and require a valid JWT (JSON Web Token) to perform the respective function.

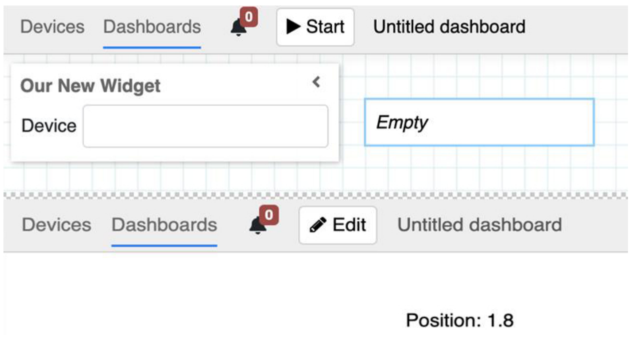


Figure 3: A minimal widget example. The top view displays the widget in editing mode where the users selects the tango device. The bottom view shows the widget in action.

Taranta Auth

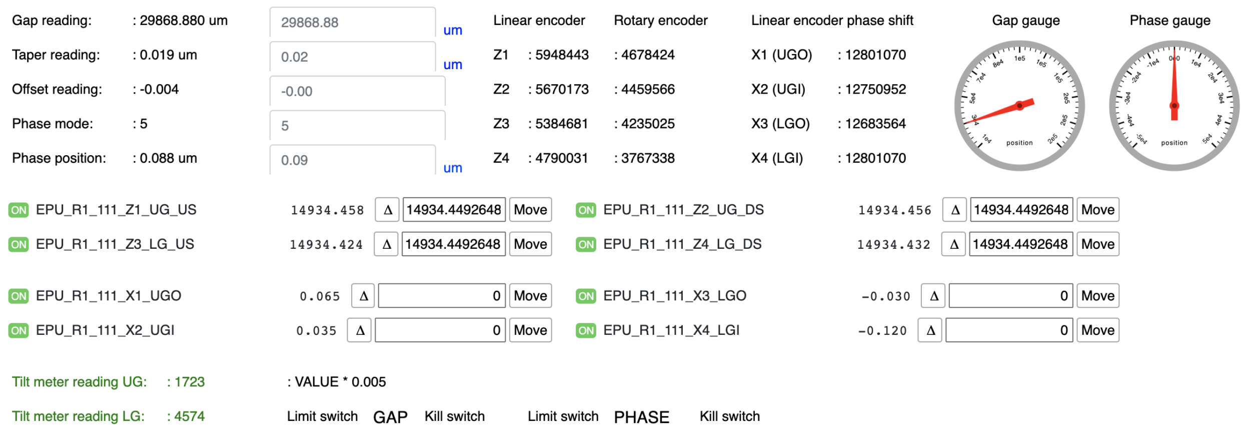
Security of web applications has been recently gaining enormous popularity. This could be due to its nature of being accessible from various end-points and therefore of potentially being an entry point to malicious attackers. Furthermore, it could also contain sensitive data and years of progress and their loss can be catastrophic. Several industry standard protocols could be followed at application-level as well as organization-level to enhance the security of web applications and to ensure the data integrity. The Taranta Auth is a micro-service based on NodeJS and provides an authentication service to be use by the Taranta suite for authenticating and authorising users and identifying their shared groups. It uses JSON Web Tokens [13], an open standard to digitally sign information for being transmitted between services. Taranta Auth access an LDAP repository or a JSON file to manage users or to retrieve user information. An anonymous user can run dashboards and can browse and inspect devices. However, to be able to send commands to devices, to change their attributes and to create or modify dashboards, an authentication is required.

USAGE

All facilities involved in the development have a very different experience using Taranta. This is partly due to the stage in which each facility is. MAX IV is in a stable user operation phase while SKA is in building stage. Despite these differences on the particular situation on each facility the development has benefit for both experiences.

MAXIV

Almost every system in MAX IV uses Taranta in one way or another, from accelerator operators to the beamlines, including the whole range of support groups. The operators use dozens of dashboards to monitor the insertion devices, RF and water cooling systems in the LINAC, and also for the status of the personal safety system. These dashboards are primarily focused on monitoring usage, but they also provide control functionalities, e.g. set-point adjustment of the motor movements.

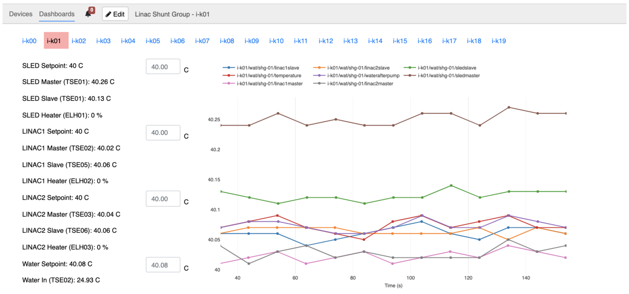


Gap reading:	: 29868.880 um	29868.88 um	Linear encoder	Rotary encoder	Linear encoder phase shift
Taper reading:	: 0.019 um	0.02 um	Z1 : 5948443	: 4678424	X1 (UGO) : 12801070
Offset reading:	: -0.004	-0.00	Z2 : 5670173	: 4459566	X2 (UGI) : 12750952
Phase mode:	: 5	5	Z3 : 5384681	: 4235025	X3 (LGO) : 12683564
Phase position:	: 0.088 um	0.09 um	Z4 : 4790031	: 3767338	X4 (LGI) : 12801070

ON EPU_R1_111_Z1_UG_US	14934.458	14934.4492648	ON EPU_R1_111_Z2_UG_DS	14934.456	14934.4492648
ON EPU_R1_111_Z3_LG_US	14934.424	14934.4492648	ON EPU_R1_111_Z4_LG_DS	14934.432	14934.4492648
ON EPU_R1_111_X1_UGO	0.065	0	ON EPU_R1_111_X3_LGO	-0.030	0
ON EPU_R1_111_X2_UGI	0.035	0	ON EPU_R1_111_X4_LGI	-0.120	0

Tilt meter reading UG: : 1723 : VALUE * 0.005
Tilt meter reading LG: : 4574

Figure 4: Example of a dashboard for monitorization and control of MAX IV insertion device for a beamline.

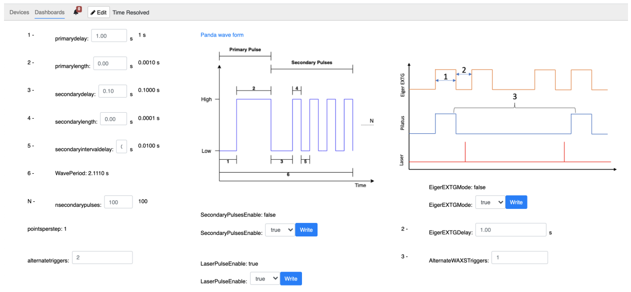


Linac Shunt Group - i-k01

SLED Setpoint: 40 C 40.00 C
SLED Master (TSE01): 40.26 C
SLED Slave (TSE01): 40.13 C
SLED Heater (ELH01): 0 %
LINAC1 Setpoint: 40 C 40.00 C
LINAC1 Master (TSE02): 40.02 C
LINAC1 Slave (TSE05): 40.06 C
LINAC1 Heater (ELH02): 0 %
LINAC2 Setpoint: 40 C 40.00 C
LINAC2 Master (TSE03): 40.04 C
LINAC2 Slave (TSE06): 40.06 C
LINAC2 Heater (ELH03): 0 %
Water Setpoint: 40.08 C 40.08 C
Water In (TSE02): 24.93 C

Devices
Dashboards
Edit
i-k00
i-k01
i-k02
i-k03
i-k04
i-k05
i-k06
i-k07
i-k08
i-k09
i-k10
i-k11
i-k12
i-k13
i-k14
i-k15
i-k16
i-k17
i-k18
i-k19
i-k01/wat/shg-01/linac1slave
i-k01/wat/shg-01/temperature
i-k01/wat/shg-01/linac1master
i-k01/wat/shg-01/linac2slave
i-k01/wat/shg-01/waterafterpump
i-k01/wat/shg-01/linac2master
i-k01/wat/shg-01/sledslave
i-k01/wat/shg-01/sledmaster
40.25
40.2
40.15
40.1
40.05
40
40
60
80
100
120
140
Time (s)

Figure 5: Example of a dashboard for monitorization of cooling systems in the Linac. There is one like this for every klystron.



Time Resolved

1 - primarydelay: 1.00 s 1 s
2 - primarylength: 0.00 s 0.0010 s
3 - secondarydelay: 0.10 s 0.1000 s
4 - secondarylength: 0.00 s 0.0001 s
5 - secondaryintervaldelay: (s 0.0100 s
6 - WavePeriod: 2.1110 s
N - nsecondarypulses: 100 100
pointsperstep: 1
alternatetriggers: 2

SecondaryPulsesEnable: false
SecondaryPulsesEnable: true Write
LaserPulseEnable: true
LaserPulseEnable: true Write

EigerEXTGMode: false
EigerEXTGMode: true Write
2 - EigerEXTGDelay: 1.00 s
3 - AlternateWAXSTriggers: 1

Figure 6: Dashboard for configuration of the timing strategy for the data acquisition at Cosaxs beamline.

Figures 4 and 5 show working examples of production dashboards. The usage approach is different, the first example is more focused on the moveable elements of an insertion device and thus, the user wanted to have the possibility of driving those axis. The second example is oriented towards cooling water trends over time, link widgets above the plot allows to switch to different dashboards.

On the beamline's side, the usage is focused on one hand to provide a simplified overview of equipment by adding into a dashboard only the most critical tango attributes of an equipment. On the other hand, it is also used to provide users with an easy to use interface to configure parts of the experiment. Figure 6 displays a dashboard to configure the timing pulse diagram for the data acquisition in Cosaxs beamline.

There was a slow start with the adoption of Taranta, but thanks to a few early adopters in the machine operators group that provided valuable feedback as well as bug reporting, Taranta is gaining popularity quickly. In addition, the knowledge of widgets development has been improved so we are able to provide new widgets faster than before, which helps gaining trust from our user community.

SKA

Taranta is seen as a tool with a lot of potential within SKA. At the current stage of the project's development, there are different categories of target users for Taranta. First, there are control system developers who create the dashboards and use them to inspect, debug and verify the devices they are developing; secondly, other teams who want to verify the behaviour of a component and use dashboards provided by the team who developed them; and lastly, managers or engineers who may want to check the current status of the system or are tasked with testing the Minimum Viable Product (MVP). Often, Taranta dashboards are used for showcasing the progress in the development of the control system during demos. Depending on their envisaged use, dashboards are expected to have different lifetimes and to require different levels of analysis before realising them.

Figures 7 and 8 show two examples of dashboards realised by SKA teams. The dashboard in 7 has been created by the team that is developing Taranta after some interactions with developers belonging to other teams. It allows to switch ON/OFF the Central Signal Processor (CSP) subsystems as well as to control them by configuring the set of resources to be used and for sending a set of commands for executing a scan (atomic part of an observation).. It also gives fast feedback on the state of subsystems. 8 shows a dashboard that is mainly used for checking the "state readiness" of the MVP for executing certain tasks (and allowing recovery of inconsistent states where possible) whilst conducting interactive end to end test development.

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-FRAR01

The close interaction with users of Taranta often promotes the development of new widgets or functionalities, such as the timeline widget, or triggers some usability improvements for the tool. Moreover using this dashboard helped developers in other teams to early detect needed changes in the control software and identify bugs.

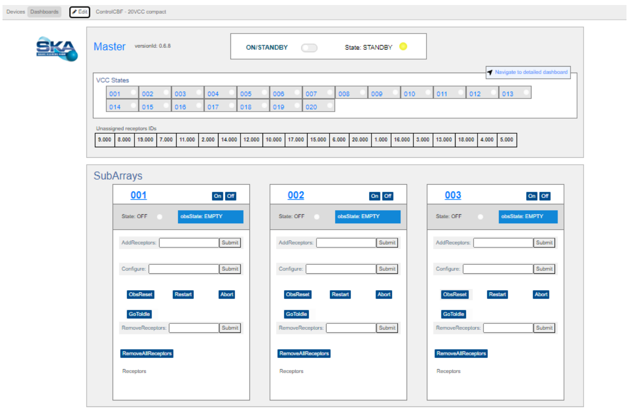


Figure 7: Example of a dashboard for monitoring and control SKA CSP.

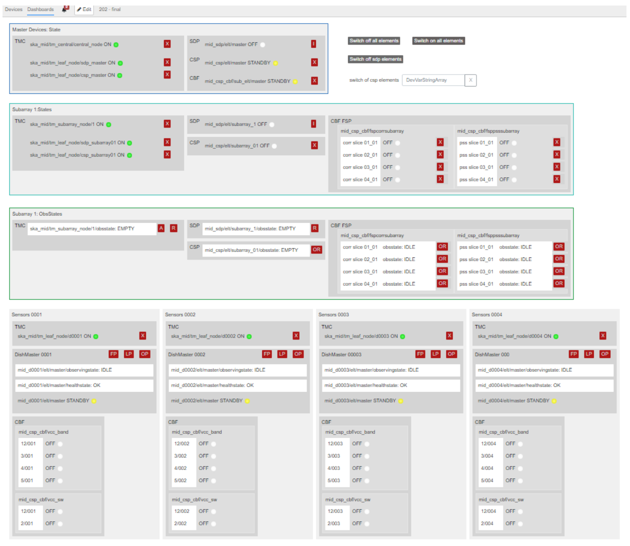


Figure 8: Example of a dashboard to monitor the status of various resources that are part of the SKA MVP.

FUTURE WORK

By the increase of user feedback a lot of new improvement can be done in Taranta. The increased number of available widgets requires a redesigned widget library element, possibly with searching and sorting by type functionality. This would make the life of our users a little bit easier. In addition, several new widgets are under development. For example, a grouping widget, onto which one can drop different widgets and then group it into one entity. This would make designing the interface layout smoother as well as give the users the possibility to create their own widgets. Another interesting widget under development is the synoptic, which aims at displaying svg images and linking svg elements to tango devices [14]. Moreover, optimisation on how and when the application sends updated values to the client, as well as data formatting improvements, specially for images, are on the list. Last but not least important, there will be new widgets coming to complement running the experiments, i.e. moving into web current functionality nowadays scattered over desktop applications and command line applications.

REFERENCES

[1] Taranta Suite home page, `https://gitlab.com/tango-controls/web`

[2] M. Eguiraun *et al.*, "Web Interface to Tango Control Systems at MAX IV", presented at 12th NOBUGS Conference, BNL, New York, 2018, unpublished.

[3] M. Woo, "The Rise of No/Low Code Software Development—No Experience Needed?", *Engineering*, vol. 6, no. 9, pp. 960-961, 2020. `doi:10.1016/j.eng.2020.07.007`

[4] J. Martin, *Rapid Application Development*, Indianapolis, IN, USA: Macmillan Publishing Co., Inc, 1991.

[5] Tarus designer, `https://taurus-scada.org/devel/designer_tutorial.html`

[6] Control System Studio, `https://controlsystemstudio.org/`

[7] GraphQL, A query language for your API, `https://graphql.org/`

[8] TangoGQL repository, `https://gitlab.com/tango-controls/web/tangogql`

[9] The Websocket Protocol, `https://datatracker.ietf.org/doc/html/rfc6455`

[10] React, A JavaScript library for building user interfaces, `https://reactjs.org/`

[11] Plotly JavaScript Open Source Graphing Library, `https://plotly.com/javascript/`

[12] MAXIV Taranta home page, How to create widgets, `https://webjive.readthedocs.io/en/latest/writing_a_widget.html`

[13] JSON Web Tokens, `https://jwt.io/`

[14] J. Forsberg *et al.*, "A Graphical Tool for Viewing and Interacting with a Control System", in *Proc. ICALEPCS'15*, Melbourne, Australia, Oct. 2015, pp. 681–684. `doi:10.18429/JACoW-ICALEPCS2015-WEM309`