USAGE AND DEVELOPMENT OF WEB SERVICES AT MAX IV

A. Milan-Otero[*], J. Forsberg[†],
F. Bolmsten, J. Brudvik, M. Eguiraun, V. Hardion, L. Kjellsson,
D. P. Spruce, L. Zytniak, MAX IV Laboratory, Lund University, Sweden

Abstract

The web continues to grow as an application platform, with accessibility and platform independence as major benefits. It also makes it possible to tie services together in new ways through simple APIs.

At MAX IV we are using web services for various purposes related to the control system. For example, monitoring servers and services, accessing alarm history, viewing control system status, managing system and users logs and running recurring jobs. Furthermore, all user management is also accessed via web applications, and even data analysis and experiment control can now be performed via web based interfaces.

We make an effort to use existing tools whenever possible (e.g. Kibana, Prometheus), and otherwise develop systems in-house, based on current well established libraries and standards, such as JavaScript, Python, Apache, etc.

This paper presents an overview of our activities in the field and describes different architectural decisions taken.

MONITORING AND STATUS

In a large and complex control system environment it is often very useful to collect information about many different elements into a single place, to quickly look at some specific aspect of them. We have investigated and implemented several ways of accomplishing this for various purposes, and this section presents the resulting applications.

State Grid

In a TANGO [1] control system, the State attribute of any device gives a high level summary of its current mode of operation and health. For example, if a device can be either "on" or "off", this is reflected by TANGO states ON/OFF, and if something is preventing it from normal operation, it should enter the FAULT state.

To make better use of this information, we have developed an application called the "state grid", see Fig. 1. It's a hierarchical 2D grid of devices, where each grid cell represents either the state of a single device, or the "worst" state of another state grid. This way, we can show a top level grid that represents the collective states of an entire system, for example a storage ring. Bad states will "bubble" up from the lower levels. The user can drill down to get further detail by clicking grid cells, in order to identify any problematic device(s). It's also possible to open control panels for individual devices from the grid, in order to further diagnose problems or directly interact with devices. The main usage of this application is to get a "bird's eye" view of the current operation and health of the entire control system.

The state grid is mainly implemented as a HTML5 application using JavaScript. It is backed by a specifically developed TANGO device that collects the current state from a configured set of devices via event subscriptions and makes this information available as an attribute.

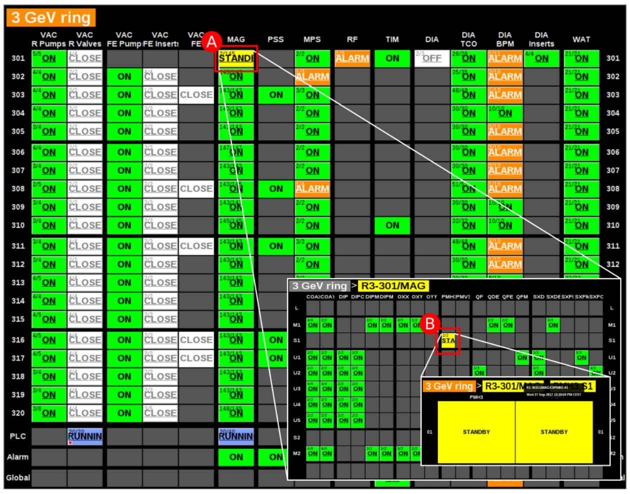


Figure 1: State grid for the large storage ring at MAX IV, showing different subsystems (columns) in each achromat (rows). Clicking the cell marked as "A" changes the grid to displaying the "subgrid" for that part of the machine (in this case the magnet subsystem for achromat 1). The subgrid in turn shows different types of magnet systems (columns) in the parts of the achromat (vertical). Clicking cell "B" here finally reveals individual TANGO devices. Note that the STANDBY state of these devices "bubbles" up to the top level grid since it's considered more important than the ON state.

Monitoring and Alerting

The "state grid" described above gives operators an immediate grasp of the current situation, but it's also very simplified and provides little help when trying to find out what previous events were the root causes of a problem. Conversely, a logging system is usually only configured to store logs for devices that are either new, or suspected to have some issue, since logging usually also means some performance overhead. It also only stores information that the developer has deemed to be of interest. Part of the missing piece is what's usually called an archiving system, but that in turn usually focuses on end user relevant information such as equipment readings.

[*] antonio.milan_otero@maxiv.lu.se
[†] johan.forsberg@maxiv.lu.se

For the more general case, it's useful to also have a monitoring tool that continuously collects various system information and stores it over time. This allows for "post mortem" investigation and discovery of things like slow memory leaks and other unexpected resource usage patterns. Prometheus [2] is such a tool, which is currently in use at MAX IV. Apart from the usual system metrics (CPU, memory, network, etc) for each server in the system, we have also developed an "exporter" that monitors TANGO servers. Exporters of health information for specific systems such as the Beam Position Monitors (BPMs) and the HDB++ archiving system have also been developed. This information is mostly useful for developers and engineers, since it typically requires technical knowledge to be interpreted.

Prometheus can be configured to send out alerts if some arbitrary conditions are fulfilled, based on recent data. We use this to get emails when e.g. a host is running low on disk, or when a TANGO server suffers from problems that prevent normal operation. This kind of service, if used properly, can increase the confidence in the control system since it allows the engineers to know about and potentially even solve issues before they impact the users.

Grafana [3] is another service that focuses on displaying "dashboards" composed of plotted data from other services like Prometheus. We use Grafana to configure convenient overviews.

Logging

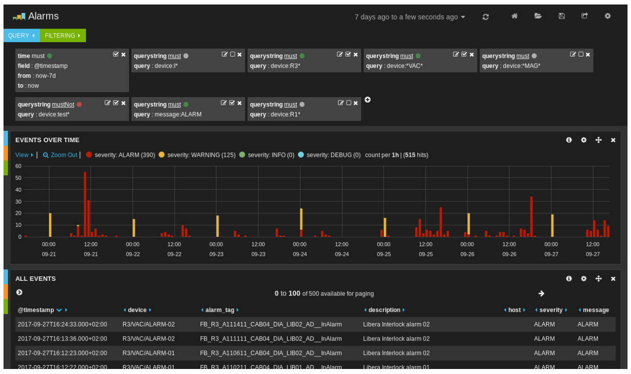
Logging is an essential tool in software development and deployment. The usual method is to log errors and other information to the local filesystem, in some text format. While it's easy to access file based logs in a local development environment, it quickly becomes very tedious to look through multiple files spread out across networked hosts in a distributed environment. There are many existing solutions to this problem, usually implemented as some form of agent running on each host, periodically checking local logfiles for new lines, and pushing those to a centralized database (e.g. Logstash, Splunk) for convenient querying.

Specifically for use with TANGO we have developed a "logger" device [4], which uses Elasticsearch [5] as a storage backend and acts as a standard TANGO logging target. This means that any other TANGO device can be configured to push its log messages to the logger device. It is also able to store alarm events from PyAlarm devices.

On top of the database, we use Kibana [6] to provide a flexible web interface to the data for use by operators and end users. Predefined queries can be configured to give a quick overview of, for example, the number of vacuum related alarms over some specified period of time, but also allows navigating down to individual events, see Fig. 2. This has become a valuable tool for tracking down problems.

Machine Status

In a synchrotron, it is important to know the status of the accelerator in each moment. This information is crucial



@timestamp	device	alarm_tag	description	host	severity	message
2017-09-27T16:24:33.000+02:00	R3/VAC/ALARM-02	FB_R3_A111411_CAB04_DIA_LIB02_AD__InAlarm	Libera Interlock alarm 02		ALARM	ALARM
2017-09-27T16:13:36.000+02:00	R3/VAC/ALARM-02	FB_R3_A111111_CAB04_DIA_LIB02_AD__InAlarm	Libera Interlock alarm 02		ALARM	ALARM
2017-09-27T16:12:23.000+02:00	R3/VAC/ALARM-01	FB_R3_A110611_CAB04_DIA_LIB02_AD__InAlarm	Libera Interlock alarm 02		ALARM	ALARM
2017-09-27T16:12:22.000+02:00	R3/VAC/ALARM-01	FB_R3_A110211_CAB04_DIA_LIB01_AD__InAlarm	Libera Interlock alarm 01		ALARM	ALARM

Figure 2: Kibana 3 user interface to the alarm data stored in Elasticsearch. The histogram shows the number of alarms over time, and the list at the bottom shows each individual alarm events. The dashboard is configured with some preset filters that can be activated to restrict the information shown in various ways. Custom searches on the various fields are also possible.

and could affect the decisions of different subsystems or beamlines.

At MAX IV we have our systems split in different networks, which pushed us to find a reliable design based on distributed systems. In order to fulfill this requirements we have adopted a web service based on events.

As shown in Fig. 3, this service is composed by four parts working together to forward events from TANGO devices to the web. First, we have a TANGO Device Server (StatusPublisher) subscribed to events in the devices to be monitored and pushing these events in the form of an HTTP POST request. At least one instance of this device is running in each network with devices to be monitored. The second part is an Apache reverse proxy that receives these POST requests and forwards them to another subnetwork that exposes only limited information to others subnetworks inside MAX IV. This subnetwork is usually called Demilitarized Zone (DMZ). Next, we have a Python Flask [7] web server running in the DMZ, receiving the HTTP POST from the reverse proxy and forwarding them as Server Send Events (SSE) to the clients to be updated. The fourth part is the client, Fig. 4, a web application that is updated based on the SSE events received. By utilizing SSE the clients do not request data but instead listens for incoming events.

This solution was deployed in 2016 and is used daily at MAX IV.

BEAMLINE CONTROL AND ACQUISITION

The software applications used for direct control and data acquisition of a beamline have requirements that were difficult to fulfill with web applications until few years ago. Any usable user interface needs to react to user interactions quickly, also a web environment demands new interface designs instead o mimic desktop applications. In addition, the performance of these kind of applications has to be guaran-

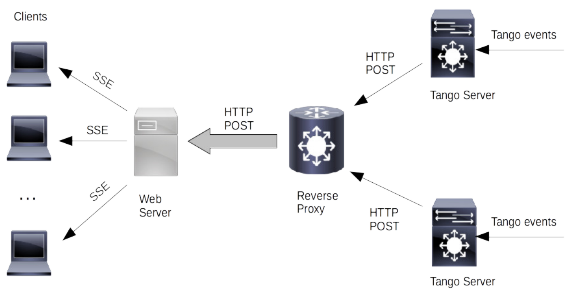
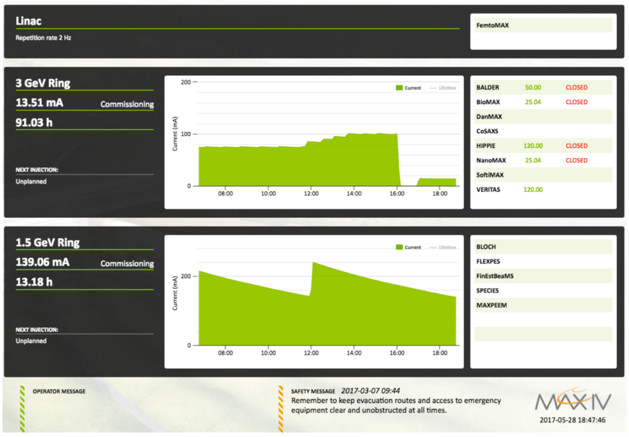


Figure 3: Machine Status Architecture. On the left, the TANGO devices in charge of gathering TANGO events and forwarding to the Revese Proxy, which in turns makes HTTP POST calls to the web server that faces clients



Remember to keep evacuation routes and access to emergency equipment clear and unobstructed at all times.

Figure 4: The main interface of the machine status application. Separated panels for both rings displaying the historic and instantaneous current, and status of the beamlines on each ring. The bottom part is devoted to status messages

teed. Thanks to the latests developments and technologies appearing in the web development, such single page applications, modern browsers, efficient communication protocols, etc., the negative barriers have been overcoming, thus, bringing modern and extensively used technologies into beamline control.

This section presents two examples of how modern web applications can be used to control and acquire data in a beamline.

MXCuBE 3

MXCuBE is an application developed originally at ESRF to control and automate routines in MX beamlines. After a second version developed in collaboration with other facilities in Europe, a third version of MXCuBE [8] has been created as a collaborative effort between ESRF and MAX IV.

During the design of this third version, different architectural solutions were evaluated, ending with the decision of moving this software from a desktop application to a web application. Due to this movement, the software originated would benefit from several advantages that comes with a single page application, like for instance, the decoupling of code and services or the better support of remote operations and cross platform requirements among others.

As a low level layer, MXCuBE 3 is using the same libraries implemented for the previous versions of the software, keeping in that way the same core in terms of hardware control software.

As a back-end we are using a Python Flask web server, implementing a REST API that provides the endpoints of the application. It uses SocketIO for bidirectional communication between server and client, and Python Gevent for asynchronous tasks. The client, Fig. 5, is developed using mainly JavaScript React library [9].

Even being still under development, MXCuBE 3 has accomplished a grade of maturity that has made it available for its usage in the BioMAX beamline at MAX IV with successful results.

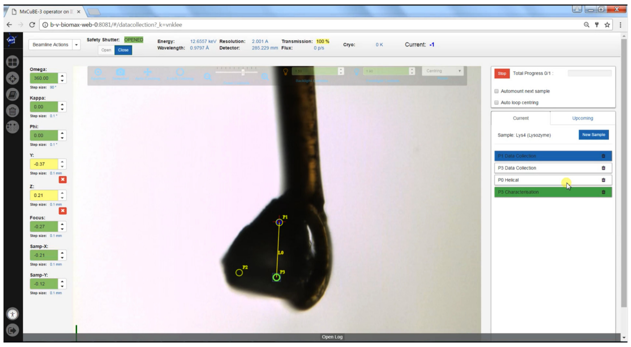


Figure 5: MXCuBE v3 user interface displaying a crystal, a successful data collection and a new collection already started.

TangoJS

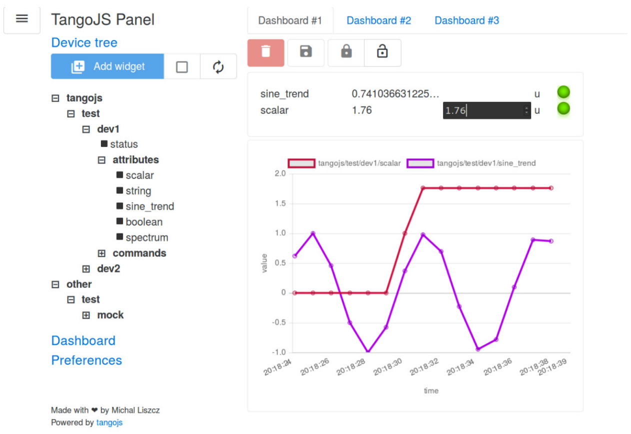
The graphical user interfaces are key components that impact in the beamline exploitation, usability and users satisfaction. Recently, at MAX IV, we received the requirement from our Bloch beamline of improving the interfaces that will be used, making a special emphasis in the cross platform and portability capabilities. For that purpose, we started the evaluation of the usage of web services for the beamline control and we decided to use TangoJS.

TangoJS [10] is a complete solution for creating TANGO clients in a web application. Developed at Solaris Synchrotron, is based on web components and has been designed to be a modular, extensible and framework-agnostic front-end stack. It has been built using modern web standards like JavaScript ECMAScript 2015 (ES2015) and CSS3.

Built with Node.js, TangoJS is available in npm, making easier its integrations into any Node.js project.

The stack provided by TangoJS is composed by several elements. The first one to be mentioned is the tangojs-web-component, which is a widget toolkit used for the creation of front-ends. The core API used to built this components, or any new component, is called tangojs-core. As a last element, tangojs support the usage of different pluggable back-ends called connectors. The connectors used in MAX IV were tangojs-connector-local and tangojs-connector-mtango, providing the first one an in-memory mock and the second one an mTango integration.

So far, at MAX IV, we have being using TangoJS to build web applications to help during the commissioning of the Bloch beamline, Fig. 6, receiving positive feedback from the personnel involved.



sine_trend	0.741036631225...	u
scalar	1.76	u

Figure 6: An example of a Tangojs interface that displays data extracted from the control system.

DATA ACCESS

The access and analysis of the data collected is a clear example of an area that can benefit from the movement to a web environment, due to the improvements in accessibility and usability of that data.

In this section, two applications demonstrate the capabilities and possibilities of web services applied to evaluation of the data generated.

HDF5 Viewer

Once an experiment has been completed, it's important to have a quick and easy access to the generated data. In order to improve the access to the tools and the analysis of the data, at MAX IV we have developed an HDF5 viewer in a form of a web application, Fig. 7.

Based on a REST API web service, this application provides a quick inspection of the data, without the need to download or install any software, and allows in an easy way, the remote access and analysis of the available data.

Although it's still under development, it provides the basic functionalities expected in an HDF5 viewer such folder exploration, data inspection, 2D and 3D plots, step through image series, etc. And it will continue to receive new features like dynamic zooming, Z-cutoffs in images and mobile interface improvements among others.

HDB++ Viewer

The HDB++ archiving system was developed originally at Elettra and is now also used and developed at ESRF. We

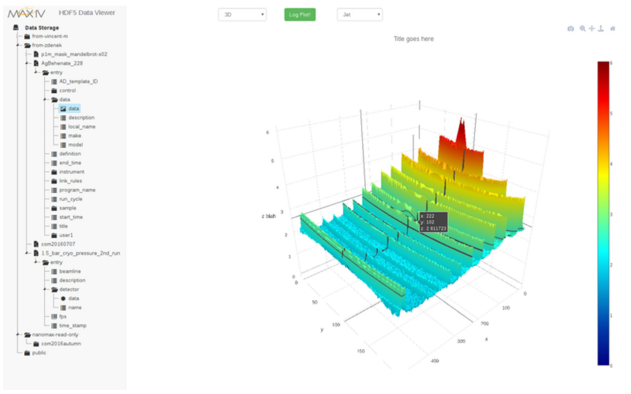


Figure 7: HDF5 Viewer plotting acquired 3D data from an HDF5 file. On the left, the HDF data structure is presented and the user can interact with it.

are taking this system into production at MAX IV with the Apache Cassandra [11] database. The Java based viewer which is part of HDB++ works well but it requires installation on the local computer and direct network access to the database cluster. In order to make the data more easily available to users, we have started development of a simple web based user interface.

Since the timeseries data received for each attribute may go into the millions of points if the time window is large enough, it's not feasible to send it in raw form to the browser for plotting. Instead, with the use of the datashader [12] library, we reduce the data into a single bitmap image, where each attribute is represented as a colored line. This has a major benefit in comparison with other ways of downsampling (such as averaging) in that it does not lose outliers in the data, no matter how large the dataset. This can be very important, for example when looking for short pressure spikes in long term vacuum data. The images are encoded as PNG and even at high resolution typically only occupy a few 10s of kB regardless of the number of data points it shows.

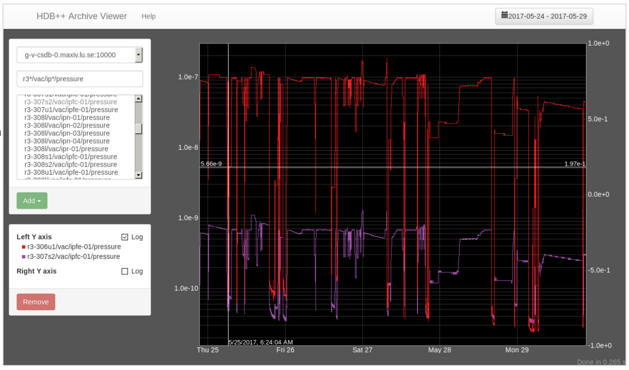
The web interface, Fig. 8, provides a quick way of filtering attributes, adding them to a plot and zooming/panning the plot using the mouse. There is also a date picker for more exact setting of time period. When the settings are changed, the front-end requests a new plot image via HTTP, and draws it.

This application is not yet ready for production as it still has some performance and reliability issues, but the source code is available [13].

CONCLUSION

In this paper we have collected various projects in use at MAX IV that show how modern web applications can substitute the traditional desktop software with at least equal performance, better usability and portability, among other advantages.

We have also given examples of how different architectures and tools can be used in distinct areas of the control system, allowing to find the best solution for each problem.



2017-05-24 - 2017-05-29

g-v-csdb-0.maxiv.lu.se:10000

r3*/vac/ip*/pressure

r3-307s2/vac/ipfc-01/pressure
r3-307u1/vac/ipfe-01/pressure
r3-308l/vac/ipn-01/pressure
r3-308l/vac/ipn-02/pressure
r3-308l/vac/ipn-03/pressure
r3-308l/vac/ipn-04/pressure
r3-308l/vac/ipr-01/pressure
r3-308s1/vac/ipfc-01/pressure
r3-308s2/vac/ipfc-01/pressure
r3-308u1/vac/ipfe-01/pressure

Left Y axis ☑ Log
🟥 r3-306u1/vac/ipfe-01/pressure
🟪 r3-307s2/vac/ipfc-01/pressure
Right Y axis ☐ Log

Figure 8: HDBPPViewer user interface. On the left, the user can filter within all the archived parameters and select the ones that will be plotted. The main view display the graphs, the user can zoom in and out and get basic information when hovering over the data.

We have found that adopting mature, community driven and open projects is also very beneficial in a resource constrained situation.

ACKNOWLEDGMENT

As this paper is a collection of several projects in different areas of the control system, MAX IV wants to express their gratitude to every single person and institute involved in any part of the development and deployment of the projects exposed here.

REFERENCES

[1] TANGO controls, an open-source device-oriented control toolkit. http://www.tango-controls.org/

[2] Prometheus, an open-source monitoring solution. https://prometheus.io/

[3] Grafana, open-source software for time series analytics. https://grafana.com

[4] TANGO Logger device, GitHub repository, https://github.com/MaxIV-KitsControls/dev-maxiv-logger

[5] Elasticsearch, a distributed, RESTful search and analytics engine, https://www.elastic.co/

[6] Kibana, https://www.elastic.co/products/kibana

[7] Flask, a microframework for Python, http://flask.pocoo.org/

[8] M. Oskarsson, A. Beteva, D. De Sanctis, M. Guijarro, G. Leonard, F. Bolmsten, M. Eguiraun, A. Milan-Otero, J. Nan, and M. Thunnissen, "MXCuBE3 Bringing MX Experiments to the WEB", presented at ICALEPCS2017, Barcelona, Spain, 2017, paper TUBPL05, this conference.

[9] JavaScript React library, https://reactjs.org/

[10] TangoJS, a complete solution for creating web-based TANGO Controls client applications, https://tangojs.github.io/

[11] Apache Cassandra, http://cassandra.apache.org/

[12] Datashader GitHub repository, https://github.com/bokeh/datashader

[13] HDBPPViewer GitHub repository, https://github.com/MaxIV-KitsControls/web-maxiv-hdbppviewer