ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-MOPV039

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2022). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI

UCAP: A FRAMEWORK FOR ACCELERATOR CONTROLS DATA PROCESSING @ CERN

L. Cseppentő*, M. Büttner, CERN, Geneva, Switzerland

Abstract

The Unified Controls Acquisition and Processing (UCAP) framework provides a means to facilitate and streamline data processing in the CERN Accelerator Control System. UCAP's generic structure is capable of tackling classic "Acquisition - Transformation - Publishing/Presentation" use cases, ranging from simple aggregations to complex machine reports and pre-processing of software interlock conditions.

In addition to enabling end-users to develop data transformations in Java or Python and maximising integration with other controls sub-systems, UCAP puts an emphasis on offering self-service capabilities for deployment, operation and monitoring. This ensures that accelerator operators and equipment experts can focus on developing domain-specific transformation algorithms, without having to pay attention to typical IT tasks, such as process management and system monitoring.

UCAP is already used by Linac4, PSB and SPS operations and will be used by most CERN accelerators, including LHC by the end of 2021.

This contribution presents the UCAP framework and gives an insight into how we have productively combined modern agile development with conservative technical choices.

INTRODUCTION

The Unified Controls Acquisition and Processing (UCAP) Framework is a recent product in the CERN Controls Software & Services (CSS) group's portfolio. This generic, self-service online Controls data processing platform, enables clients to easily implement and run Controls Device data acquisition and processing in Java or Python.

The main objective of the project is to provide a common approach to solve problems where:

1. data is acquired and grouped from several sources (*Acquisition*),
2. based on these inputs, and optionally internal state, a result is calculated (*Transformation*),
3. the result is made available to clients (*Publishing/Presentation*).

Such "Acquisition - Transformation - Publishing/Presentation" problems are regularly featured in many Controls products, such as data concentrators, software interlocks, logging adaptations and autopilot-style Controls software.

At CERN, Controls software development responsibilities are often split between the CSS group (mainly computing engineers providing frameworks and generic services), and equipment and operations groups (typically experts in their specialised domains). Experience showed that in order to effectively tackle certain problems, collaboration of teams with different backgrounds is essential, as the expertise of Controls framework libraries and accelerator domain knowledge are distributed. The UCAP service aims to streamline development and maintenance by splitting responsibilities:

- the UCAP team provides infrastructure, tools, training and support for development, testing and deployment of transformations,
- while the transformation code and configuration stays under the responsibility of the end-user (typically a domain expert).

This *self-service* model enables end-users to focus on realising their business logic, while the service takes care of secondary tasks, such as process management and monitoring. It is also *scalable*: as service usage increases, only new machines and UCAP nodes need to be added to the system.

Nevertheless, this set up brings other challenges, such as isolating user groups on shared hardware, ensuring dynamic server-side code loading, providing substantial documentation and user-friendly access. At this scale, a high level of *automation* is essential

The "UCAP-idea" originates from 2016, with a first prototype started in 2018. During this phase, the first use case was satisfied, providing transformations on approximately 1000 data streams. The following year, in the beginning of Long Shutdown 2, development started on the operational product – in close collaboration with stakeholders. A few months later, still in early 2019, the system went to production while iterative development continued. As of 2021 Q3, the UCAP service is composed of 105 nodes (isolated deployment units), running on 6 physical servers, performing around 20 000 data transformations.

UCAP IN A NUTSHELL

The UCAP service is a *multitenant* system. *Nodes* are assigned to client groups or use cases for isolation purposes. All nodes are fully functional data processing services, differing only in basic configuration parameters, such as name, description and responsible. The layered architecture, presented in Fig. 1, mirrors the "Acquisition - Transformation - Publishing/Presentation" chain – each layer being responsible for dealing with one aspect of data processing. UCAP complies with this model on the architecture level, avoiding the introduction of any new structural elements in the data representation.

The CERN Control System uses the *Device-Property model* [1], meaning that all endpoints providing data are

* lajos.cseppento@cern.ch

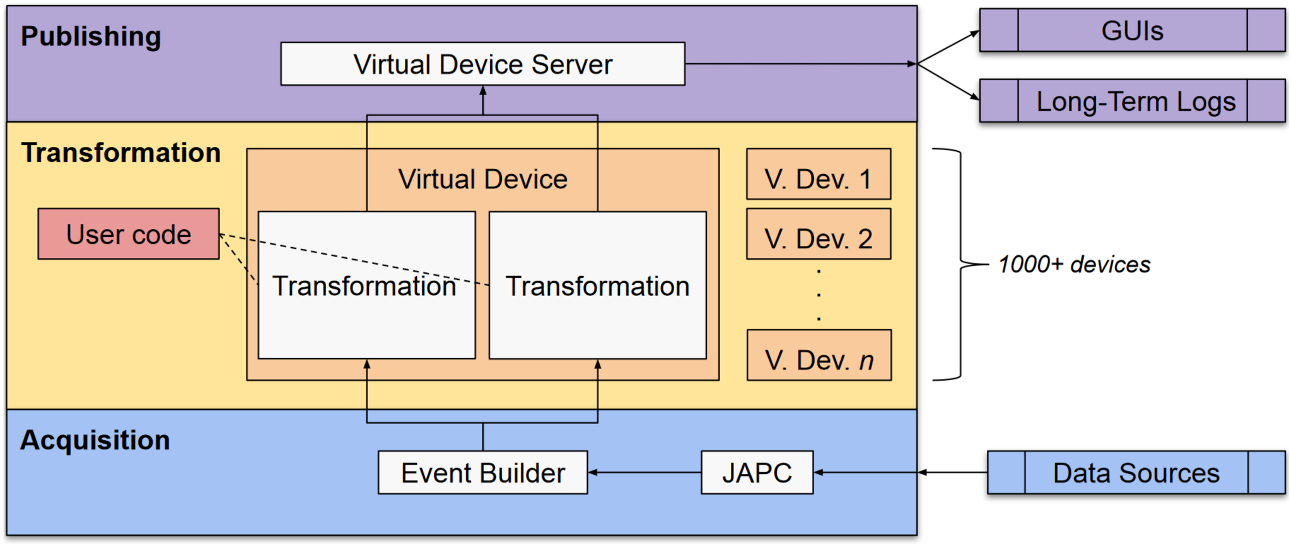


Figure 1: UCAP Node Architecture.

expressed in a `DEVICE_NAME/PropertyName` format, to which data is published as a map of key-value pairs.

Acquisition Layer

This layer is responsible for *acquiring inputs* from data sources. This is carried out using JAPC[1] [2], a CERN standard library for reading inputs from individual Controls Devices. As JAPC is an abstraction layer over communication protocols, a wide range of inputs are supported, including UCAP itself (i.e., chaining of transformations).

In many cases, it is required to group inputs from several Controls Devices, for which *UCAP Event Builders* are provided. These data acquisition algorithms are the driving force for transformations. At the time of writing, 10 event builder implementations are provided, ready to be selected and configured by end-users.

A basic example of an event builder definition is shown in Listing 1, expressed in JSON format.

Listing 1: Basic Event Builder Definition

```
{
  "type":
    "GroupTriggeredCycleStampGrouped",
  "triggerGroup": {
    "subscriptions": [
      { "parameter":"MAGNET_1/Acq" },
      { "parameter":"MAGNET_2/Acq" }
    ],
    "timeoutMs": 1000
  }
}
```

This code snippet instructs UCAP to instantiate an event builder, grouping data from two sources based on their time stamps. The `type` field identifies the name of the acquisition logic, which also decides the rest of the configuration layout.

[1] Java API for Parameter Controls

In this case the `triggerGroup` declares how event creation shall be handled: the `subscriptions` list contains the name of the inputs and the `timeoutMs` field specifies how long to wait for them – this is notably useful when one of the sources is not available, since processing can move with the inputs that were available when the timeout was reached.

Community feedback highlighted how powerful event builders are. End-users can express acquisition logic in a declarative manner, instead of they themselves having to create subscription handles, to start monitoring threads and to implement combination logic.

Each event created contains the data received from all sources and it is passed to the Transformation Layer for processing.

Transformation Layer

The Transformation Layer is responsible for carrying out the actual data processing. Transformations are instances of *converter* algorithms, which are configurable and can be implemented in Java or Python (Python with C/C++ bindings is also supported and used by the community). UCAP currently provides and maintains 20+ so called *standard converters*, which implement commonly required algorithms in a generic manner. Examples include merging homogeneous data from several sources; splitting multiplexed data prior to long-term logging. This enables a part of our user community to benefit from UCAP without having to write any code.

In addition, *custom converters* are also supported. For each event created, a transformation may create one or more results or choose to skip the input. Plus, if there is a need for a stateful transformation, the internal state of the transformation is preserved between events. Apart from these aspects, it is up to the end-user's creativity how they implement their business logic. An example of a Python converter structure can be seen in Listing 2: this particular code re-publishes the sum of two incoming values.

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2022). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-MOPV039

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2022). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI

Listing 2: Example Python Converter Structure

```
def convert(event):
    log = logging.getLogger(__name__)

    result = AcquiredParameterValue(
        published_parameter_name,
        event.trigger_value.header
    )

    v1 = event.get_plain_value('BLM1')
    v2 = event.get_plain_value('BLM2')

    result.update_value(
        "result",
        v1 + v2
    )

    log.info("Input␣:␣%s", event)
    log.info("Output:␣%s", result)

    return FailSafeParameterValue(
        result
    )
```

For converter development we recommend using our in-house standards, CBNG [3] for Java and Acc-Py [4] for Python, however, it is also possible without those.

Publishing Layer

In order to make the transformation results visible to consumers, UCAP nodes encapsulate an RDA3[2] Device Server [5] [6] (the same communication protocol used by FESA[3] [7]). The user interface of the Publishing Layer is a simple `return` statement. Internally, the UCAP RDA3 Virtual Device Server takes care of managing client subscriptions. The results can be consumed with C++, Java and Python clients. Results can also be directly logged in the NXCALS[4] [8] time-series data logging system.

Other Notable Features

Apart from "doing the job", UCAP provides several other Controls features, notably:

- FESA-style alarms, which can be integrated e. g., with LASER[5] [9],
- access to previously logged data from NXCALS,
- transparent rolling updates of user code without data loss,
- language-agnostic, RBAC[6]-protected [10] REST API for inspection and management and
- development, testing, monitoring and troubleshooting utilities.

[2] Remote Device Access
[3] Front-End Software Architecture
[4] Next CERN's Accelerator Logging Service
[5] LHC Alarm SERvice
[6] Role-Based Access Control

USER EXPERIENCE

User experience is crucial to ensure customer productivity and satisfaction. Good user experience also reduces the support load of the service provider team.

To facilitate development, all Java public API are annotated with javadoc and all UCAP Python public API are decorated with docstrings and MyPy type hints. Sample definitions and converter code are provided in both languages. UCAP includes testing utilities to aid users in writing unit test for converters, inspecting event builders or even running UCAP nodes locally either, directly from Java or using the UCAP Docker image.

When a user wants to start using UCAP, the UCAP team prepares and assigns two nodes for the use case:

- a PRO node for operational purposes,
- a TEST node for verification prior to deployment of transformations in operation.

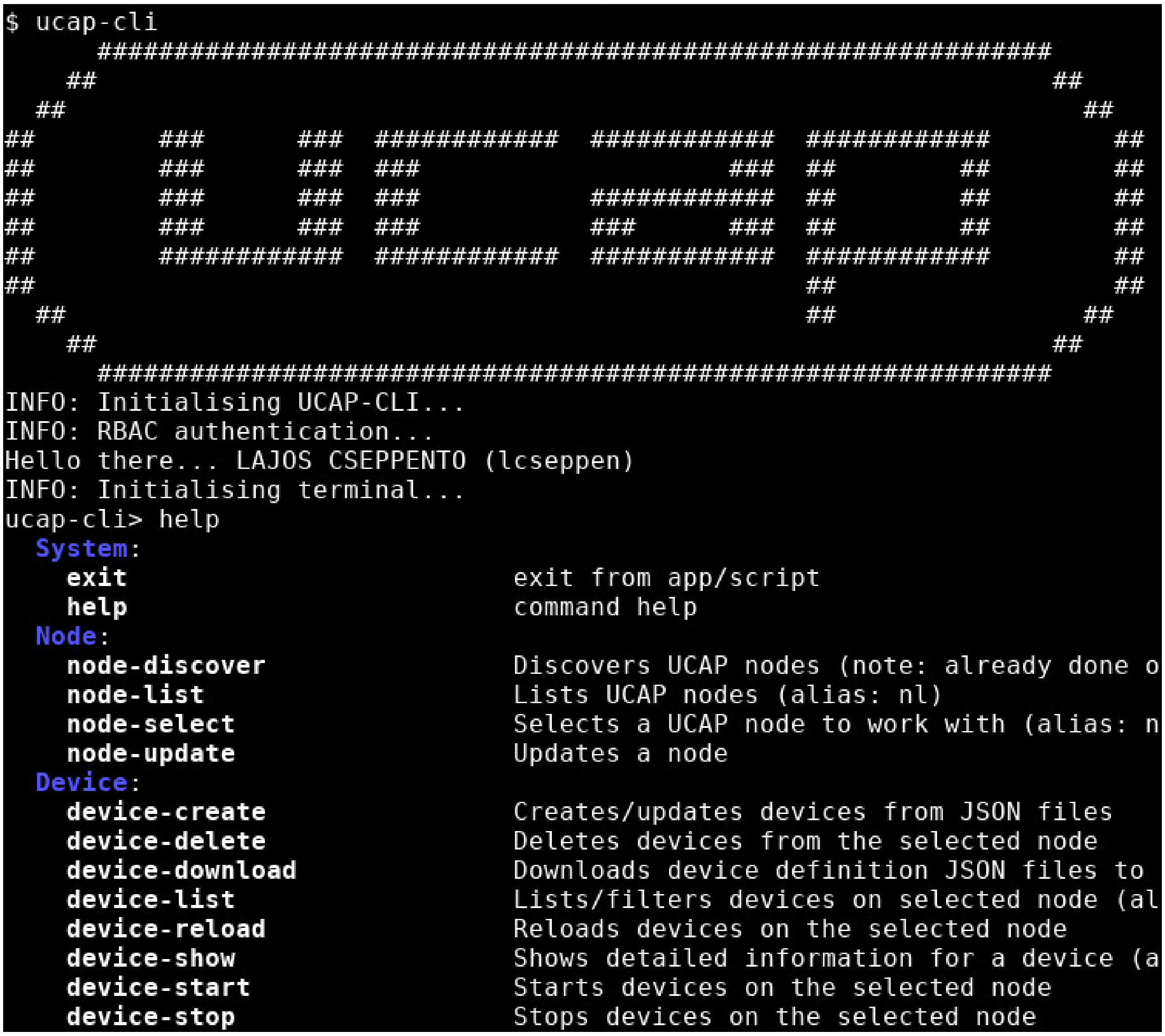
A rich command line tool, *UCAP-CLI* (see Fig. 2), is provided to facilitate node management and introspection. It has an extensive `TAB`-based auto-completion feature, which aids users with displaying commands, Device, transformation and package names on pressing the `TAB` key. The CLI is also used extensively by the UCAP team for maintenance and support purposes. UCAP-CLI is available on all developer and control room machines, and it can connect to PRO, TEST and local UCAP nodes. The development and maintenance costs of this tool are also low compared to a fully featured GUI.

Out-of-the-box, service monitoring is provided to end-users in the form of Grafana dashboards [11] (example in Fig. 3). These dashboards are automatically created for each node and Device added to the UCAP service, and include subscription (data input) status, conversion errors and input/output rates. These dashboards are quite convenient as in case of failures (e. g., control room display is blank) they make *fault isolation* quick (e .g., inputs are down).

TRAINING AND SUPPORT

Training for end-users is an essential element of the UCAP service. In the early stages of the service, focus was given to close stakeholder collaboration, frequent pair-programming sessions and an emphasis on written documentation. As the user base expanded, effort was put into public presentations (training lectures and product demos) which were recorded and made available online. Since publishing such materials, a correlation with a drop in the number of support requests from new users has been observed.

Communication channels comprise a traditional e-mail list and a Mattermost Channel for instant messaging. Mattermost has shown to be efficient in tackling quick questions. These channels are completed with periodic UCAP Technical Meetings which include demos and hands-on training.



```
$ ucap-cli
      ##############################################################
    ##                                                              ##
  ##                                                                  ##
##        ###      ###  ############  ############  ############        ##
##        ###      ###  ###                    ###  ##        ##        ##
##        ###      ###  ###           ############  ##        ##        ##
##        ###      ###  ###           ###      ###  ##        ##        ##
##        ############  ############  ############  ############        ##
##                                                  ##                  ##
  ##                                                ##                ##
    ##                                                              ##
      ##############################################################
INFO: Initialising UCAP-CLI...
INFO: RBAC authentication...
Hello there... LAJOS CSEPPENTO (lcseppen)
INFO: Initialising terminal...
ucap-cli> help
  System:
    exit                         exit from app/script
    help                         command help
  Node:
    node-discover                Discovers UCAP nodes (note: already done o
    node-list                    Lists UCAP nodes (alias: nl)
    node-select                  Selects a UCAP node to work with (alias: n
    node-update                  Updates a node
  Device:
    device-create                Creates/updates devices from JSON files
    device-delete                Deletes devices from the selected node
    device-download              Downloads device definition JSON files to
    device-list                  Lists/filters devices on selected node (al
    device-reload                Reloads devices on the selected node
    device-show                  Shows detailed information for a device (a
    device-start                 Starts devices on the selected node
    device-stop                  Stops devices on the selected node
```

Figure 2: UCAP-CLI, the rich command-line tool.

DEVELOPMENT METHODOLOGY

An agile workflow was used to balance development and maintenance of UCAP with other ongoing projects. This comprised weekly planning meetings, daily stand-ups, frequent pair-programming, and code reviews using iterative development practices. A JIRA Scrum Board was helpful to keep focus and GitFlow practices to keep a high velocity. With limited resources and challenging deadlines, runtime verification favoured integration and system tests over unit tests, as experience showed that they were more efficient for discovering bugs and regressions during development.

TECHNOLOGICAL CHOICES

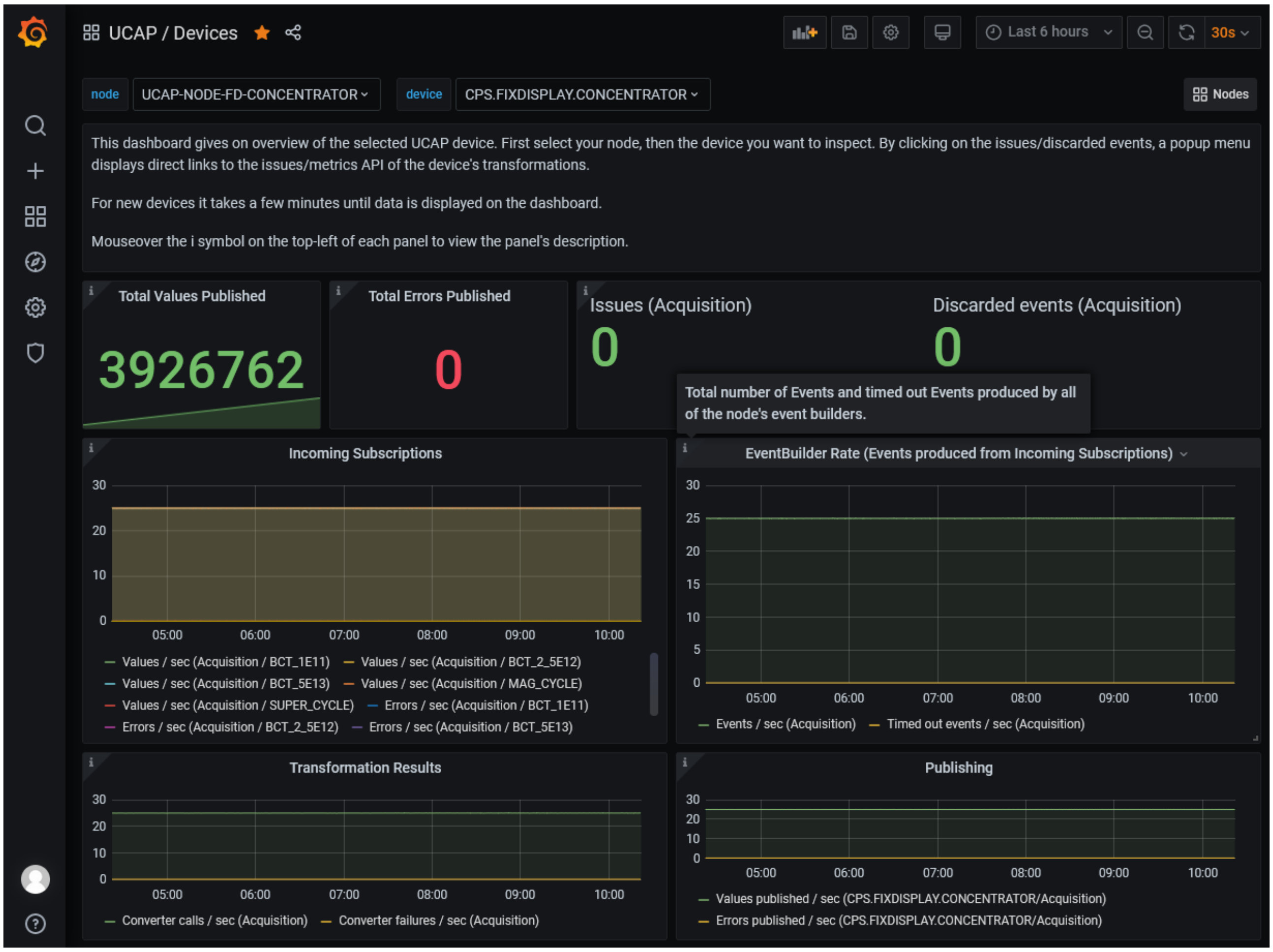
In 2018 the UCAP prototype was implemented with recent technologies (Java, Spring Boot, OpenFeign, Project Reactor and protobuf/gRPC). Several FaaS[7] solutions with Kubernetes were evaluated, however, at the end of 2018, Kubernetes-based solutions were considered as too risky to deliver within the strict deadlines, due to the significant involvement and long-term commitment required from multiple teams.

In early 2019, technological choices were finalised: Java, Spring Boot, OpenFeign, REST, JAPC and Controls Middleware (CMW). This eliminated partial duplication of communication features realised by gRPC in the prototype. Classic Java executor pools were favoured over reactive programming practices. The Open Services Gateway initiative (OSGi) framework [12] is used to support dynamic code loading (e.g., when users want to replace the transformation code on a running node). This resulted in simple and clear server-side code, while the OSGi layer is hidden from the user via a Gradle Plugin [13]. For UCAP-CLI we used the JLine library [14]. Java-Python communication is carried out over TCP and regarded as an internal protocol.

[7] Function-as-a-Service

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2022). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2022). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI


UCAP / Devices
Last 6 hours
30s
node
UCAP-NODE-FD-CONCENTRATOR
device
CPS.FIXDISPLAY.CONCENTRATOR
Nodes
Total Values Published
3926762
Total Errors Published
0
Issues (Acquisition)
0
Discarded events (Acquisition)
0
Incoming Subscriptions
EventBuilder Rate (Events produced from Incoming Subscriptions)
Transformation Results
Publishing
Values / sec (Acquisition / BCT_1E11)
Values / sec (Acquisition / BCT_2_5E12)
Values / sec (Acquisition / BCT_5E13)
Values / sec (Acquisition / MAG_CYCLE)
Values / sec (Acquisition / SUPER_CYCLE)
Errors / sec (Acquisition / BCT_1E11)
Errors / sec (Acquisition / BCT_2_5E12)
Errors / sec (Acquisition / BCT_5E13)
Events / sec (Acquisition)
Timed out events / sec (Acquisition)
Converter calls / sec (Acquisition)
Converter failures / sec (Acquisition)
Values published / sec (CPS.FIXDISPLAY.CONCENTRATOR/Acquisition)
Errors published / sec (CPS.FIXDISPLAY.CONCENTRATOR/Acquisition)

This dashboard gives on overview of the selected UCAP device. First select your node, then the device you want to inspect. By clicking on the issues/discarded events, a popup menu displays direct links to the issues/metrics API of the device's transformations.

For new devices it takes a few minutes until data is displayed on the dashboard.

Mouseover the i symbol on the top-left of each panel to view the panel's description.

Total number of Events and timed out Events produced by all of the node's event builders.

Figure 3: Grafana Dashboard of an Operational Device.

For code collaboration we leveraged GitLab. Highly automated testing is executed by a CI/CD pipeline on unit, integration and system test levels. Overall monitoring is based on the well-integrated Prometheus/Alertmanager/-Grafana [15] [11] stack, with JMX used for some expert interventions. Node maintenance is carried out based on UCAP-specific YAML files, while UCAP tooling takes care of tackling deployment and process management. This has proven to be a cost-effective solution.

The documentation is based on *Mkdocs Material* [16], adhering to the CERN design guide, encapsulating both Java and Python API documentations. As the last step of the development tasks and one of the least rewarding engineering tasks, focus was given to making documentation easy to write: the documentation source is hosted with the UCAP source code, thus documentation can be reviewed in the same Git merge request.

FUTURE WORK

In the first half of 2021, an extensive review of UCAP was conducted in order to reflect on the development and user experience gained so far and identify the next objectives. A tighter integration with the Controls Configuration Service [17] and support for real-time processing were important aspects identified from the review. From a UCAP service provider perspective, redundant middleware and availability of a computing cluster are key aspects that could lead to an even better UCAP service.

CONCLUSION

Used in Linac4 [18], PSB, PS, SPS and LHC – UCAP has been rapidly adopted across the CERN accelerator complex. The user community continues to grow, which is a clear indication of the success of the product. The high-quality standards followed during development have contributed to scarce bug reports and no service outage thus far. At the same time, UCAP has facilitated the removal of a significant number of custom and complex standalone systems, thus fulfilling the objective of providing a truly “Unified” Controls Acquisition and Processing platform.

ACKNOWLEDGEMENTS

The authors would like to express special acknowledgements to early adopters of the service. Their feedback contributed a lot to the product.

REFERENCES

[1] V. Baggiolini, S. Jensen, K. Kostro, F. DiMaio, A. Risso, and N. Trofimov, "Remote Device Access in the New CERN Accelerator Controls Middleware", in *Proc. 8th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'01)*, San Jose, CA, USA, Nov. 2001, paper THAP003, pp. 496–498.

[2] V. Baggiolini, Lionel Mestre, "JAPC - the Java API for Parameter Control", in *Proc. 10th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'05)*, Geneva, Switzerland, Oct. 2005, oral TH.1.5-8O

[3] L. Cseppentő, V. Baggiolini, E. Fejes, Zs. Kővari, and N. Stapley, "CBNG - The New Build Tool Used to Build Millions of Lines of Java Code at CERN", in *Proc. 16th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'17)*, Barcelona, Spain, Oct. 2017, pp. 789–793. doi:10.18429/JACoW-ICALEPCS2017-TUPHA163

[4] P.Elson, I. Sinkarenko and C. Baldi, "Introducing Python as a Supported Language for Accelerator Controls at CERN", presented at the 18th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'21), Shanghai, China, Oct. 2021, paper MOPV040, this conference.

[5] W. Sliwinski, K. Kaczkowski, and W. Zadlo, "Fault Tolerant, Scalable Middleware Services Based on Spring Boot, REST, H2 and Infinispan", presented at the 17th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'19), New York, NY, USA, Oct. 2019, paper MOBPP03.

[6] J. Lauener and W. Sliwinski, "How to Design & Implement a Modern Communication Middleware Based on ZeroMQ", in *Proc. 16th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'17)*, Barcelona, Spain, Oct. 2017, pp. 45–51. doi:10.18429/JACoW-ICALEPCS2017-MOBPL05

[7] A. Schwinn *et al.*, "FESA3 – The New Front-End Software Framework at CERN and the FAIR Facility", in *Proc. 8th Int. Workshop on Personal Computers and Particle Accelerator Controls (PCaPAC'10)*, Saskatoon, Canada, Oct. 2010, paper WECOAA03, pp. 22–26.

[8] J. P. Wozniak and C. Roderick, "NXCALS - Architecture and Challenges of the Next CERN Accelerator Logging Service", presented at the 17th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'19), New York, NY, USA, Oct. 2019, paper WEPHA163.

[9] Z. Zaharieva and M. Büttner, "CERN Alarms Data Management: State & Improvements", in *Proc. 13th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'11)*, Grenoble, France, Oct. 2011, paper MOPKN011, pp. 110–113.

[10] P. Gajewski, S. R. Gysin, and K. Kostro, "Role-Based Authorization in Equipment Access at CERN", in *Proc. 11th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'07)*, Oak Ridge, TN, USA, Oct. 2007, paper WPPB08, pp. 415–417.

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

[12] OSGi https://www.osgi.org/

[13] Gradle Build Tool https://gradle.org/

[14] JLine https://github.com/jline/jline3

[15] Prometheus https://prometheus.io/

[16] Material for MkDocs https://squidfunk.github.io

[17] L. Burdzanowski and C. Roderick, "The Renovation of the CERN Controls Configuration Service", in *Proc. 15th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'15)*, Melbourne, Australia, Oct. 2015, pp. 103–106. doi:10.18429/JACoW-ICALEPCS2015-MOPGF006

[18] M. Hrabia, M. Peryt and R. Scrivens, "The Linac4 Source Autopilot", presented at the 18th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'21), Shanghai, China, Oct. 2021, paper WEPV018, this conference.

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2022). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI