ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEPHA048

MANAGEMENT OF IOCs AT ESS

R. Fernandes[†], S. Gysin, T. Korhonen, J. Persson, S. Regnell, ESS, Lund, Sweden
M. Pavleski, S. Sah, Cosylab, Ljubljana, Slovenia

Abstract

The European Spallation Source (ESS) is a neutron research facility based in Sweden that will be in operation in 2023. It is expected to have around 1500 IOCs controlling both the machine and end-station instruments. To manage the IOCs, an application called IOC Factory was developed at ESS. It provides a consistent and centralized approach on how IOCs are configured, generated, browsed and audited. The configuration allows users to select EPICS module versions of interest, and set EPICS environment variables and macros for IOCs. The generation automatically creates IOCs according to configurations. Browsing retrieves information on when, how and why IOCs were generated and by whom. Finally, auditing tracks changes of generated IOCs deployed locally.

To achieve these functionalities, the IOC Factory relies on two other applications: the Controls Configuration Database (CCDB) and the ESS EPICS Environment (E3). The first stores information about IOCs, devices controlled by these, and required EPICS modules and snippets, while the second stores snippets needed to generate IOCs (st.cmd files). Combined, these applications enable ESS to successfully manage IOCs with minimum effort.

INTRODUCTION

The Integrated Control System (ICS) Division at ESS is mandated to deliver a system to control both its machine (i.e. accelerator) and end-station instruments. To create the system, or more precisely (distributed) control system, an open-source framework called EPICS [1] was chosen. With worldwide usage and acceptance, EPICS allows the creation of Input/Output Controllers (IOCs) that high-level software applications (e.g. CS-Studio, Archiver Appliance) may consume (i.e. connect to) to tackle domain specific businesses (e.g. OPI designing, signals archiving).

Typically, an IOC is an executable (i.e. process) that utilizes resources from EPICS modules to interface (logical or physical) devices and exposes their input/output signals as Process Variables (PVs). Eventually, an IOC may also implement logic to control these devices.

A PV is a named piece of data, usually associated with devices to represent input and output signals (e.g. status, setpoint). It has a set of attributes (i.e. fields) that integrators configure according to the specificities of the domain to solve. A PV can be read, written or monitored by applications and tools using the Channel Access (CA) library.

The (distributed) control system being built by ICS will be composed of hundreds of IOCs interfacing and controlling a multitude of devices. To develop and maintain this amount of IOCs is a challenging task, which – without proper automation – puts a heavy burden on integrators.

To alleviate this burden, the IOC Factory was developed in recent years at ICS. This application not only allows authenticated and authorized users to execute well-defined functions – configuration, generation, browsing and auditing of IOCs – but also promotes a formal, standardized workflow to manage IOCs from a high-level perspective.

DESCRIPTION

The IOC Factory has been under development since mid-2015, and a first version was publicly released early 2016. During its development phase and first years in production, the application required around one FTE. Currently, it requires less than 0.5 FTE mainly for maintenance (i.e. implementation of minor functionalities and bug fixes), training and supporting users.

To date, several versions of the IOC Factory have been released for production, the latest (version 1.2.20) in August 2019. It currently manages (i.e. stores) around 260 configurations from 50 different IOCs. These configurations (and other information) are stored in an open-source RDBMS and total approximately 10 MB. Table 1 summarizes the most important metrics about the IOC Factory.

Table 1: Metrics about the IOC Factory

Description	Value
Tables (persistence tier)	10
Constraints (persistence tier)	12
Indexes (persistence tier)	14
Lines of code (persistence tier)	0
Classes in Java (business tier)	169
Lines of code (business tier)	10801
Web pages (presentation tier)	7
Dialogs (presentation tier)	22
Lines of code (presentation tier)	2598

The IOC Factory has been developed in the context of the DISCS collaboration [2], with the aim of becoming a useful tool for more sites than only ESS. This (international) collaboration is composed of several research facilities with the aim of developing databases, services and applications that any facility can easily configure, use and extend for its commissioning, operation and maintenance.

Dependencies

To implement the aforementioned functionalities and, consequently, manage IOCs in an efficient manner, the IOC Factory relies on two other applications actively developed at ICS: the Controls Configuration Database (CCDB) [3] and the ESS EPICS Environment (E3) [4].

[†] ricardo.fernandes@esss.se

Controls Configuration Database (CCDB) is an application that enables the collection, storage and distribution of (static) controls configuration data needed to install, commission, operate and maintain the ESS control system.

Specifically, the CCDB manages the information of thousands of (physical and logical) devices such as racks, power supplies, motors, pumps, PLCs and IOCs, that are in operation at ESS by defining their properties and relationships from a controls point of view. This information is then consumed both by end-users and other ICS applications (e.g. IOC Factory) to enable successful performance of domain specific businesses.

From an IOC context, the CCDB stores the information of several entities namely: the device that runs the IOC, the IOC itself, the devices controlled by the IOC, and the relationships between the IOC and devices. Concretely, for the IOC Factory to work correctly and as expected, the information (stored in the CCDB) needs to adhere to the following rules (or model):

- The device type that runs an IOC (e.g. CPU, IPC) has a slot (placeholder in the control system hierarchy to install devices) property named “OperatingSystem”. This property is of type enumeration and pre-defined with all the operating systems (or platforms/architectures) supported by ICS (e.g. linux-x86_64, linux-ppc64e6500).
- The device that runs an IOC is installed in an appropriate slot and its property “OperatingSystem” is set with the concrete operating system installed in the device (e.g. linux-x86_64).
- The installed slot device that runs an IOC has a relationship of type “Contains” with the IOC.
- The IOC is installed in an appropriate slot and has a relationship of type “Controls” with each device that it interfaces.
- Each device that the IOC interfaces has two properties named “EPICSModule” and “EPICSSnippet” – both are of type strings list. The property “EPICSModule” enumerates all EPICS modules (e.g. StreamDevice) needed to interface the device, while “EPICSSnippet” enumerates all EPICS snippets (also known as iocsh files) that compose the IOC (i.e. st.cmd file).

ESS EPICS Environment (E3) is an application that manages EPICS bases and modules, additionally providing a runtime engine for IOCs. It can be seen as a tailored package of pre-compiled EPICS bases and modules for several operating systems (or platforms/architectures) – e.g. linux-x86_64, linux-ppc64e6500 – to (more) easily build the ESS control system, enabling IOCs to dynamically load EPICS modules thanks to the *require* module developed at PSI (and further customized at ESS).

From a file system structure point of view, E3 is organized in function of the EPICS base versions, *require* module versions, EPICS modules’ names and versions, and operating systems that it manages. In other words:

```
/epics/base-<base_version>/require/<require_version>/siteMods/<module_name>/<module_version>/lib|bin/<operating_system>
```

Each EPICS module managed by E3 is compiled for every operating system supported by ICS. Amongst other resources, a module has snippets that are loaded by IOCs using the *iocshLoad* function. These snippets contain instructions that are executed upon launching the IOC. A typical instruction is to load and expand an EPICS database file containing (PV) records declaration. Example:

```
dbLoadRecords("test.db", "IOC=$(IOCNAME)")
```

In general, an IOC based on E3 is structured in three sections: 1) EPICS modules to load dynamically, 2) EPICS environment variables and macros to set with specific values, and 3) EPICS snippets to load. The following illustrates a typical IOC (i.e. st.cmd file) based on E3:

```
# load (dynamically) module "test" version 2.1.3
require test, 2.1.3

# set EPICS environment variables and macros
epicsEnvSet("EPICS_CA_SERVER_PORT", "5100")
epicsEnvSet("P", "SEC-SUB01:")
epicsEnvSet("R", "DIS-DEV-01")
epicsEnvSet("IOCNAME", "$(P)$(R)")

# load iocsh file "test.iocsh"
iocshLoad("test.iocsh", "IOCNAME=$(IOCNAME)")

# initialize IOC
iocInit()
```

Functionalities

Thanks to helpful discussions with stakeholders, fundamental use-cases were identified. The IOC Factory was subsequently built to solve these via well-defined functionalities that users may expect from a tool that aims to manage IOCs in a high-level way. These functionalities are the configuration, generation, browsing and auditing of IOCs.

Configure IOC allows users to create configurations which are subsequently used to generate IOCs. A configuration is stored in the IOC Factory persistence layer and it is composed of the following information (specified by users):

- Version of the EPICS base to use (from a set of EPICS base versions found by the IOC Factory when dynamically scanning E3 file system structure).
- Version of the *require* module to use (from a set of *require* module versions found by the IOC Factory when dynamically scanning E3 file system structure).
- Port number to assign to *procServ*.
- Description (i.e. brief explanation) about the purpose of the configuration.
- EPICS environment variables to use and values assigned to these.
- Versions of the EPICS modules to use (from a set of EPICS modules’ versions found by the IOC Factory when dynamically scanning E3 file system structure).
- Values to assign to EPICS macros (found by the IOC Factory when parsing database record files belonging to the specified EPICS modules’ versions).

Other information composing a configuration are its revision number (which is incremented by one unit every time a configuration is created or pasted) and the identification of who (i.e. LDAP username used to log in the IOC Factory) has created or pasted a configuration. Both pieces of information are automatically calculated/retrieved by the application (i.e. users do not need to specify these).

Operationally speaking, the IOC Factory retrieves all existing IOCs from the control system hierarchy (stored in the CCDB) and presents these to users. When users select a certain IOC, all existing configurations associated with the IOC are shown. Users may edit an existing configuration or create a new one for the selected IOC. Editing a configuration is only possible if it has never been used to generate an IOC for production. Otherwise, the configuration is "frozen" (i.e. not editable) to allow redeploying it for production in case of need – e.g. downgrade the IOC to a known working state. The functionality 'Configure IOC' also enables the comparison of two configurations (even from distinct IOCs). Further, it can display their differences graphically to ease understanding. In addition, it allows users to copy a certain configuration and paste it either in the same or another IOC. The pasted configuration will be identical to the original, except that: 1) its revision number is increased one unit (counting from the last revision number), 2) it becomes editable (in case the original configuration is "frozen"), and 3) its creator field is set with the information of the logged user (i.e. LDAP username).

Finally, every time the topology of the IOC evolves (in other words, when changes are made to the layout of devices controlled by the IOC and/or the IOC itself), users create a new configuration to cope with this evolution. In case of discrepancies between the current IOC topology (stored in the CCDB) and a configuration (stored in the IOC Factory) based on an outdated IOC topology, users may view a list of inconsistencies (e.g. in the CCDB, a certain device is not controlled by the IOC anymore, while a configuration – based on a previous topology – still has a reference that the IOC controls the device). The list is automatically calculated by the IOC Factory to enable users to solve discrepancy issues. This feature is also available when copying and pasting a configuration from an IOC to another IOC that is different (topologically speaking).

Generate IOC allows users to generate (i.e. create) IOCs (i.e. st.cmd files) from scratch according to configurations selected by them. The generated IOCs, structurally similar to the one illustrated in the ESS EPICS Environment (E3) (see subsection 'Dependencies'), are deployed by the IOC Factory in an infrastructure designed to store these. Depending on users' selection, generation of an IOC may either be for development or production. In case for production, the configuration used to generate the IOC is "frozen" (i.e. not editable anymore). Users may preview a generated IOC to check that its logic is correct before deploying it in the infrastructure. Moreover, the IOC Factory also checks that certain rules are respected to help users guarantee a high quality IOC – e.g. ensuring that: 1) a certain port number assigned to *procServ* is used only once across IOCs running in the same device (e.g. CPU, IPC); 2) there are no conflicting dependencies in the EPICS modules dependency tree; 3) values assigned to EPICS macros are unique across IOCs running in the same device. Immediately after an IOC is generated, information about the operation – e.g. who made the generation, timestamp, configuration used, directory where the IOC was deployed – is stored in the IOC Factory persistence layer to enable browsing and auditing IOCs functionalities afterwards.

Browse IOC allows users to retrieve and display information about historical (i.e. past) generation of IOCs. It gives a broad view and deep understanding of when, how and why a certain IOC was generated and by whom. In detail, for each generated IOC, it displays the IOC name, operating system, hostname, configuration used, username, timestamp, generation type (development or production), directory (where the IOC was deployed in the infrastructure), and a user-defined description (i.e. brief explanation about the generation).

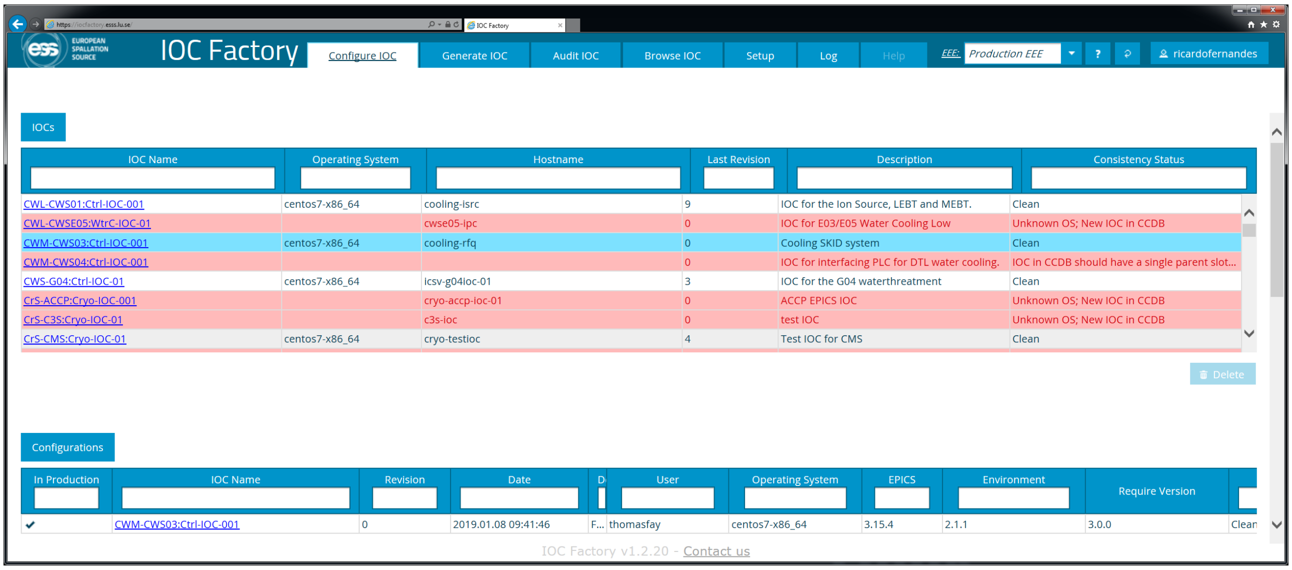
Audit IOC allows users to track changes which generated IOCs may have suffered – i.e. st.cmd files edited manually by integrators – in the infrastructure where these are deployed (i.e. stored). Specifically, the functionality provides users with a list of all generated IOCs along with information about whether they have been modified or not. Users can subsequently select an IOC to either 1) see the modifications made to the IOC by (graphically) displaying the differences between the IOC originally deployed (and snapshotted in the IOC Factory persistence layer) and the version currently deployed with changes, or 2) revert the IOC to its original state (i.e. discard local changes) in case the modifications are incorrect/not relevant anymore.

Workflow

Within the IOC Factory, the IOC development cycle (or workflow) of an IOC starts with the integrator creating a configuration for the IOC with proper values/settings. The integrator uses the configuration to generate the IOC and deploys it for development. He/she then launches the IOC and performs validation tests (e.g. checking that the IOC interfaces devices correctly, that input/output signals of devices are well mapped, or that the control logic is correct).

Every time validation tests reveal an issue with the IOC, the integrator may either edit the configuration (used to generate the IOC) or create a new configuration to solve the issue. Subsequently, he/she (re)generates the IOC with this edited/new configuration and perform the tests again.

When validation tests show no issues with the IOC, the integrator (re)generates the IOC and deploys it for production. The configuration used for the generation is then "frozen" and cannot be edited anymore (to secure incremental working baselines of the IOC that the integrator may always revert to in case of need). A new development cycle may start with the integrator creating a new configuration from scratch, or simply copying and pasting the configuration used for production (as it was thoroughly tested) and editing it with new values/settings to cope with the evolution of the (field) scenario that the IOC controls.



IOCs

IOC Name	Operating System	Hostname	Last Revision	Description	Consistency Status
CWL-CWS01:Ctrl-IOC-001	centos7-x86_64	cooling-isrc	9	IOC for the Ion Source, LEBT and MEBT.	Clean
CWL-CWSE05:WtrC-IOC-01		cwse05-ipc	0	IOC for E03/E05 Water Cooling Low	Unknown OS; New IOC in CCDB
CWM-CWS03:Ctrl-IOC-001	centos7-x86_64	cooling-rfq	0	Cooling SKID system	Clean
CWM-CWS04:Ctrl-IOC-001			0	IOC for interfacing PLC for DTL water cooling.	IOC in CCDB should have a single parent slot...
CWS-G04:Ctrl-IOC-01	centos7-x86_64	icsv-g04ioc-01	3	IOC for the G04 waterthreatment	Clean
CrS-ACCP:Cryo-IOC-001		cryo-accp-ioc-01	0	ACCP EPICS IOC	Unknown OS; New IOC in CCDB
CrS-C3S:Cryo-IOC-01		c3s-ioc	0	test IOC	Unknown OS; New IOC in CCDB
CrS-CMS:Cryo-IOC-01	centos7-x86_64	cryo-testioc	4	Test IOC for CMS	Clean

Configurations

In Production	IOC Name	Revision	Date	D	User	Operating System	EPICS	Environment	Require Version	
✓	CWM-CWS03:Ctrl-IOC-001	0	2019.01.08 09:41:46	F...	thomasfay	centos7-x86_64	3.15.4	2.1.1	3.0.0	Clean

IOC Factory v1.2.20 - Contact us

Figure 1: Graphical Interface of the IOC Factory.

Architecture and Technology Stack

The IOC Factory is a distributed system based on a classical client-server model where users access its functionalities remotely through a web-based graphical interface (Figure 1). This model – or architecture – is composed of three tiers, namely: Presentation (the layer which users interact with), Business (the layer which implements business logic) and Persistence (the layer in which data is stored/retrieved). Figure 2 illustrates the architecture of the IOC Factory (and the technology stack used to implement it).

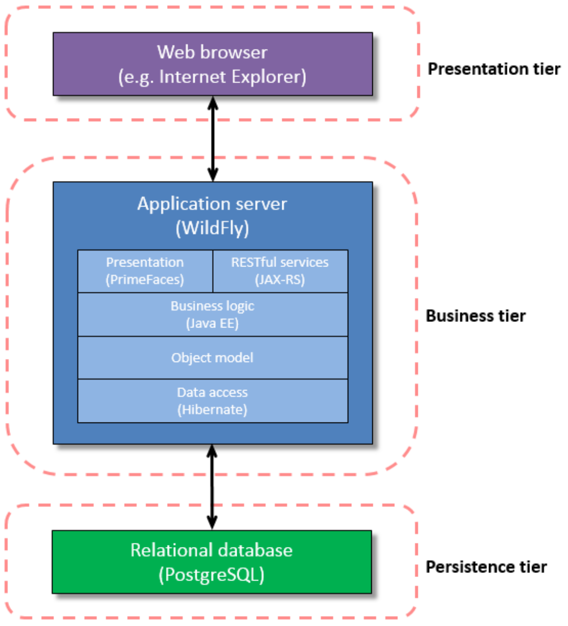


Figure 2: Architecture of the IOC Factory.

Several technologies are employed to implement this architecture guaranteeing that the IOC Factory is developed according to user requirements and expectations. Primordial criteria to select the technologies were that they had to be open-source, mature, well documented and actively maintained by the community. With these in mind, PostgreSQL, a relational database management system, is used to implement the persistence tier (i.e. database) of the IOC Factory. The business tier is implemented in Java Enterprise Edition (Java EE) running in an application server called WildFly. It uses Hibernate (a JPA implementation) to access data from the persistence tier and JAX-RS (a Java API) to consume data provided by external applications, namely RBAC [5] and CCDB, through RESTful services. Finally, the presentation tier (i.e. graphical interface) of the IOC Factory is based on PrimeFaces.

(Part of an) Ecosystem

Several (high-level) software applications have been developed (or are under development) in recent years to support both integration and controls efforts at the ICS Division. These are producers and/or consumers of services & data that form a rich (logical) ecosystem to solve a myriad of domain (i.e. integration/controls) specific issues such as management of IOCs, generation of PLC code, and calibration of devices.

At its core, the ecosystem possesses the CCDB with the main purpose of enabling the storage and distribution of (static) controls configuration data needed to operate the ESS control system efficiently. In this context, the IOC Factory consumes data stored in 1) RBAC to authenticate and authorize users to perform certain actions or not and 2) CCDB to retrieve information about IOCs topologies (in other words, the list of devices that a certain IOC controls as well as the list of EPICS modules and snippets needed to interface each device). This alleviates the IOC Factory to explicitly store information about IOCs and the devices controlled by these in its own persistence layer, consequently reducing data duplication and (potential) inconsistencies that could emerge across different applications. Figure 3 shows the ecosystem, as well as the IOC Factory as a consumer of both RBAC and CCDB.

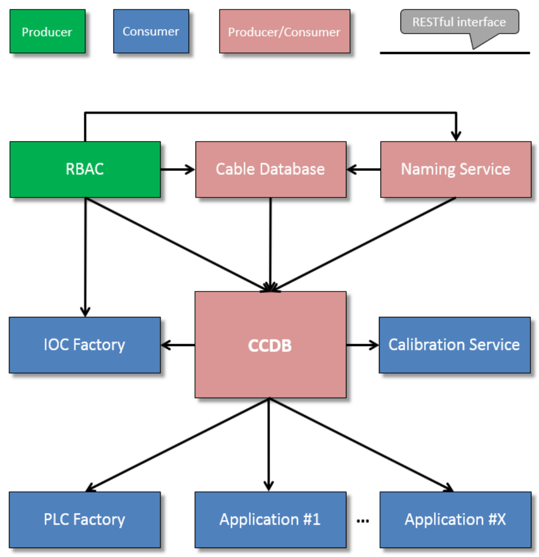


Figure 3: Overview of the ecosystem/IOC Factory.

FUTURE DEVELOPMENTS

Many people at ESS – in particular integrators, controls and PLC engineers with disparate needs – use the IOC Factory. Its user base is expected to grow further in coming years, which will entail new requests for additional functionalities. The following missing functionalities have already been identified as candidates for development:

- EPICS macros setting: at present, the IOC Factory does not automatically set EPICS macros with default values when creating a new IOC configuration. To increase user productivity, the IOC Factory should be prepared to retrieve values from a (pre-agreed) property associated to devices (stored in the CCDB) and map the values with the corresponding EPICS macros found in database record files used to interface devices. For example, the device "PLC_3" (stored in the CCDB) has a property "EPICSMacro" storing values "ADEL=5" and "LOW=8"; when creating a new configuration of an IOC that controls device "PLC_3", the EPICS macros $(ADEL) and $(LOW) for the device in question are automatically set to 5 and 8 by the IOC Factory, respectively.
- PVs publication: the IOC Factory does not currently publish the list of PVs of IOCs that it generates. To increase the degree of automation delivered by the ecosystem, the IOC Factory could be extended to publish (i.e. write) the list of PVs in the CCDB where each device (controlled by the generated IOC) stores a subset of the list (i.e. the PVs belonging to the device) in a (pre-agreed) property associated to the device in question. This would not only 1) centralize controls-related information in the CCDB but also 2) pave the way for a new tool acting as a lean and effective PV "yellow pages" service to satisfy query requests about PVs, their location in the control system hierarchy, and related metadata (all stored in the CCDB).
- Automatic IOC configuration creation: when auditing a generated IOC for changes made locally, the IOC Factory should give the user the possibility to automatically create a new IOC configuration based on the modifications found in the generated IOC. This could alleviate users from having to manually create a new configuration that reflects all the modifications made to the IOC, which can be tedious and prone to error.
- Programmatic access: currently, the IOC Factory does not provide RESTful services which disable external applications to consume its functionalities (see subsection 'Functionalities'). Therefore, it could be beneficial to implement RESTful services to promote the development of new tools to help integrators in their activities. For example, it would be possible to implement a (software) daemon running once per day which (thanks to RESTful services provided by the IOC Factory) checks whether generated IOCs have suffered local changes and, if so, sends an email to people responsible for IOCs about modifications made.

CONCLUSION

The IOC Factory is a flexible (web-based) interface that leverages existing ICS applications (i.e. CCDB and E3) to manage IOCs in a high-level way. It can be seen as an effective graphical "frontend" for the E3, the "backend" that manages EPICS bases and modules at ESS, from an IOC development perspective.

Currently, the IOC Factory successfully manages (i.e. stores) around 260 configurations from 50 different IOCs. It provides well-defined functionalities – configuration, generation, browsing and auditing of IOCs – which can benefit users (i.e. integrators) by alleviating them from daily routine tasks.

Moreover, through these functionalities, the IOC Factory incentivizes a formal workflow with the main goal of promoting a standard approach to manage IOCs across a team of (numerous) integrators at ICS.

ACKNOWLEDGEMENT

The authors would like to thank everyone who contributed with suggestions to further improve the IOC Factory, in particular Nick Levchenko and David Brodrick, as well as Lars Johansson and Jeong Han Lee for providing material to prepare the present paper.

REFERENCES

[1] EPICS, `https://en.wikipedia.org/wiki/EPICS`

[2] V. Vuppala *et al.*, "Distributed Information Services for Control Systems", in *Proc. ICALEPCS'13*, San Francisco, CA, USA, Oct. 2013, paper WECOBA02, pp. 1000-1003.

[3] R. N. Fernandes, S. R. Gysin, S. Regnell, S. Sah, M. Vitorovic, and V. Vuppala, "Controls Configuration Database at ESS", in *Proc. ICALEPCS'17*, Barcelona, Spain, Oct. 2017, pp. 775-779. doi:10.18429/JACoW-ICALEPCS2017-TUPHA156

[4] J. Han Lee *et al.*, "The ESS EPICS Environment E3", EPICS Collaboration Meeting 2019, Cadarache, France, June 2019.

[5] Role Based Access Protocol, `http://openepics.sourceforge.net/security`