ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-THPDP023

EVOLUTION OF CONTROL SYSTEM AND PLC INTEGRATION AT THE EUROPEAN XFEL

A. Samadli*, T. Freyermuth, P. Gessler, G. Giovanetti, S. Hauf, D. Hickin, N. Mashayekh, A. Silenzi, European XFEL, Schenefeld, Germany

Abstract

The Karabo software framework is a pluggable, distributed control system that offers rapid control feedback to meet the complex requirements of the European X-ray Free Electron Laser facility. Programmable Logic Controllers (PLC) using Beckhoff technology are the main hardware control interface system within the Karabo Control System. The communication between Karabo and PLC currently uses an in-house developed TCP/IP protocol using the same port for operational-related communications and self-description (the description of all available devices sent by PLC). While this simplifies the interface, it creates a notable load on the client and lacks certain features, such as a textual description of each command, property names coherent with the rest of the control system as well as state-awareness of available commands and properties. To address these issues and to improve user experience, the new implementation will provide a comprehensive self-description, all delivered via a dedicated TCP port and serialized in a JSON format. A Python Asyncio implementation of the Karabo device responsible for message decoding, dispatching to and from the PLC, and establishing communication with relevant software devices in Karabo incorporates lessons learned from prior design decisions to support new updates and increase developer productivity.

INTRODUCTION

As one of the world's leading light sources, The European X-ray Free Electron Laser facility is opening up new research opportunities for scientists and industrial users by generating ultrashort X-ray flashes - 27 000 times per second and with a peak brilliance of 5×10^{33} photons / s / mm^2 / mrad^2 / 0.1% bandwidth [1]. Unique characteristics of the facility enable researchers to study tiny structures, ultrafast processes, extreme states, and small objects. Operating such a complex facility without a robust control system is impossible. Karabo [2] is the control system in use at European XFEL: a pluggable, distributed control system that offers rapid control feedback to meet the complex requirements of the facility. The main interface with the existing hardware infrastructure are Programmable Logic Controllers (PLC), which use Beckhoff technology. The primary objective of this project is to enhance the communication between Karabo and these Beckhoff PLCs [3].

* ayaz.samadli@xfel.eu

CURRENT COMMUNICATION BETWEEN KARABO AND PLC

The communication between Karabo and PLC currently uses an in-house developed protocol over TCP/IP using the same port for operational-related communications and a self-description of the PLC's configuration and functionality (the description of all devices available on the PLC) [4]. The self-description and operational-related data (e.g., Value updates) are in a custom binary format.

BeckhoffCom Device

The Human-Machine Interface relies on Karabo devices, enabling users to observe and control the connected hardware devices seamlessly. The BeckhoffCom device serves as a transparent interface between Karabo devices and the PLC, and is critical to this communication flow. It distributes the updates from the PLC to the connected Karabo devices and forwards the reconfigurations and commands from the Karabo devices to the PLC. The PLC terminals are structured within what we call *soft devices*, with each *soft device* being associated with a corresponding Karabo device. BeckhoffCom's core design is based on the Model-View-Presenter (MVP) pattern, with the model managing PLC communication (including TCP communication with a view adapter, view model, and TCP view) and the presenter enforcing functionality based on functional requirements. The system defines events and methods using abstracted interfaces, which increases flexibility and testability.

Production Environment

In the existing production environment at EuXFEL, there are approximately 15,000 Beckhoff devices spread across ten different scientific endstations and beam line installations. Within this infrastructure, approximately 1,000,000 Beckhoff properties are exposed to the control system, consisting of parameters such as temperature, velocity, flow, and more. The following figures offer visual insights to provide a clearer picture of the distribution of PLC devices throughout the infrastructure. Figure 1 illustrates the allocation of Beckhoff devices per instrument. The SASE2 (SA2) photon tunnel installations stand out with roughly 2,000 PLC devices, while Laser topics (LA) require fewer than 200 PLC devices.

We examine the SQS (Small Quantum Systems) to obtain a more detailed view of the distribution of various Beckhoff devices within a specific instrument. Figure 2 shows that the combination of Digital Input and Output devices makes up about 40% of the total devices, with MC2 Beckhoff Motors and Analog Inputs contributing a combined total of

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-THPDP023

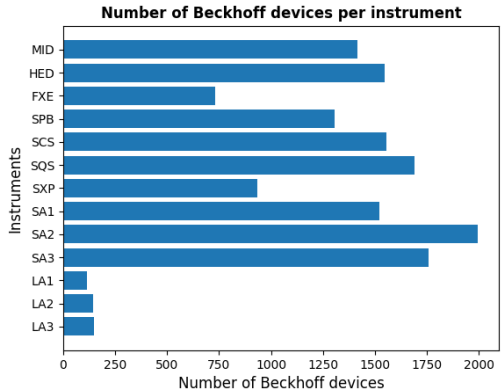

Figure 1: Number of Beckhoff devices per instrument in the infrastructure.

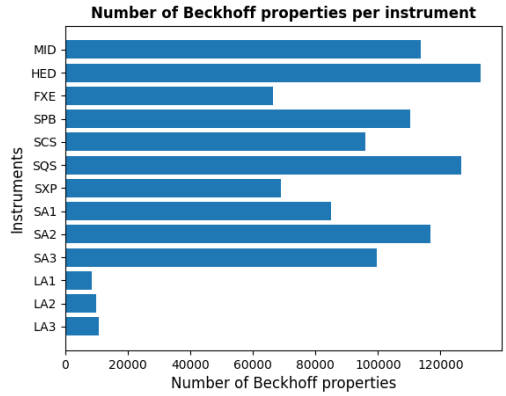

Figure 3: Total number of Beckhoff properties per instrument in the infrastructure.

about 30%. Valves, drive units, various types of motors, and other components constitute the remaining 30% of the total devices within the SQS instrument. The infrastructure comprises 39 distinct Beckhoff devices, most commonly utilized as digital inputs/outputs, analog inputs/outputs, various motor types, valves, pumps, encoders, gauges, and more.

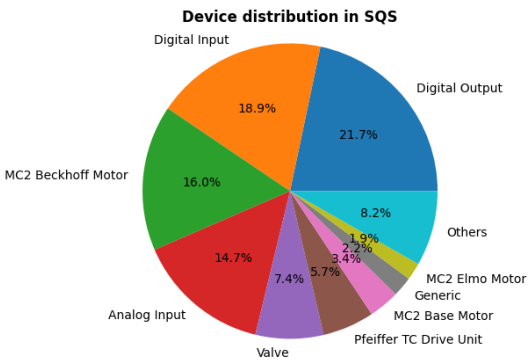

Figure 2: Distribution of the Beckhoff devices in the SQS instrument.

Each device contains numerous device-specific properties, including attributes like actual/target position, actual/target velocity, software/hardware limits, interlocks, and various others. These properties can either be read-only or reconfigurable, and each device may have anywhere from a few to up to 200 distinct properties.

The total count of devices alone does not provide a comprehensive view of the data rate these devices produce within the control system, as some devices may have many properties. Therefore, the total number of properties within a topic can significantly increase if an instrument has numerous such devices. A comparison between Figs. (1 and 3) shows a semi-strong correlation between the number of devices and the number of properties. For instance, despite ranking fourth in the number of devices, the HED instrument has the highest number of properties. This is attributed to HED using approximately 450 Beckhoff MC2 motors, each with approximately 160 parameters. It is important to note that specific properties remain static or change infrequently, while others may experience frequent updates, such as actual position. To gain a comprehensive understanding, assessing the average update rate for each topic is valuable, as this provides a more complete picture of the data dynamics across various properties and parameters.

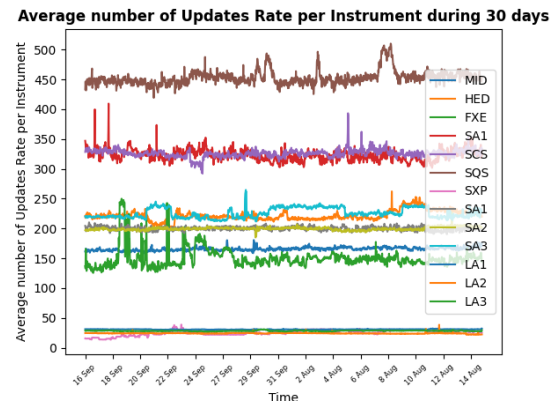

Figure 4: Average number of Beckhoff update rates per instrument during 30 days.

Figure 4 illustrates that update rates typically range from 100 to 500 updates per second across various topics where SQS takes the lead, except for the laser topics. On a regular day, BeckhoffCom is expected to handle updates at a rate of 500 Hz. It is important to note that this number can increase significantly, especially if the connection is restarted and a BeckhoffCom receives the self-description of the PLC it is connected to.

ISSUES IN THE CURRENT IMPLEMENTATION

Although the BeckhoffCom implementation is generally robust and does not encounter frequent issues in operation,

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-THPDP023

it has limitations that could be removed by the development of new implementation. Maintenance often consumes more time than development, so addressing the limitations discussed in the following is a worthwhile endeavor with long-term benefits.

Limitation in Maintainability

European XFEL's control system requirements are dynamic and complex, necessitating the integration of new devices into the existing system and incorporating new features as requested by scientists. Therefore, despite having a robust system, recurrent maintenance is required to adapt to evolving needs. Karabo offers three primary application programming interfaces, with most devices implemented using the Python middle layer API since 2019. The current implementation of the Karabo-PLC interface is in C++, which contrasts the growing Python adoption within the group. A Python implementation of the PLC integration in Karabo could leverage the growing Python expertise throughout the group and facility. The existing C++ code is dense and has a multilevel model-view-presenter design, making maintenance and adding new features tedious if they were not foreseen in the original design. The MVP design results in the necessity of modifying many classes, even for simple additions. Compared to a more streamlined design, this could be considered boilerplate. Debugging and bug detection are made more challenging due to the time-consuming compilation, potentially leading to inefficiencies in the development process. The boost::asio [5] callbacks that are used in C++ can be less straightforward and intuitive than Python's asyncio [6]. Additionally, the current self-description lacks crucial aspects such as:

- Error Codes: Absence of error codes along with their explanatory details.
- State Machine: The need for a comprehensive state machine specifying allowable action progressions.
- Value Ranges and Options: Critical information such as minimum and maximum value ranges and available configuration options need to be included.

Finally, combining self-description (static) and operational communication data (dynamic) in a single protocol and data stream makes the protocol complex. Sharing a port and thus communication path for the self-description and operational communication creates a significant transient load when Karabo connects to a PLC, as the PLC transmits the self-description data immediately upon connection initiation, leading to increased traffic and potential operational data exchange delays.

PYTHON ASYNCIO BASED EVENT-DRIVEN PROPOSAL

Comprehensive Self-Description

The new implementation aims to enhance user interaction by providing an extended self-description delivered through a web service in JSON format [7]. The self-description is extended by the elements previously identified as missing: a list of error codes with detailed descriptions, a comprehensive state machine outlining permissible actions for a given state, defined value ranges such as minimum and maximum limits, and available configuration options. An example of the properties and commands description is shown in Tables 1 and 2, respectively.

Table 1: The Description of Properties in JSON Format

Name	Value
key	triggerDuration
alias	0x5011
displayedName	Trigger Duration
description	Duration of Trigger
dataType	Int16
unitSymbol	SECOND
metricPrefixSymbol	MILLI
allowedStates	[ON, PROCESSING]
requiredAccessLevel	EXPERT
accessMode	RECONFIGURABLE

Table 2: The Description of Commands in JSON Format

Name	Value
key	sendAll
alias	0x80100000
displayedName	Send All
description	Send all available values
allowedStates	[ON]
requiredAccessLevel	OPERATOR

Architectural Overview and Key Modules in the Implementation

The new implementation is deliberately organized into two main sections: *plc-binding* and *devices*. Notably, *plc-binding* is intentionally designed to be Karabo-agnostic, meaning it does not depend on or import any Karabo-specific components. This separation ensures that modifications to Karabo or Karabo devices won't require changes in *plc-bindings*. We achieve this by using abstractions in our methods, ensuring that devices remain independent of specific *plc-binding implementations*. Our design follows the principles of dependency inversion, aiming for a high degree of decoupling in our software architecture. This approach simplifies future updates, such as new protocols or enhancements, which can be primarily addressed within the *plc-binding* module, reducing the need for extensive code modifications.

The implementation is designed with complete asynchronicity in mind, leveraging Python's *asyncio* package to guarantee that data reading and writing operations do not impact execution of other code segments. This approach ensures non-blocking execution throughout the system. More-

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-THPDP023

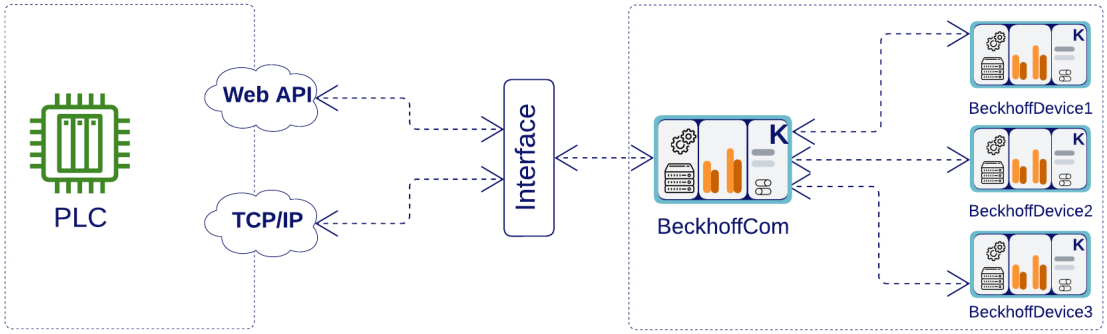

Figure 5: Karabo-agnostic Interface connects asynchronously to PLC/Web service, enabling communication between BeckhoffCom (leading) and BeckhoffDevices (subordinate) via event-driven signals.

over, the system operates fully event-driven, where value changes and updates are initiated in response to any alterations in the data. These changes are seamlessly propagated from *plc-binding* to *devices* using callback functions. This asynchronous and event-driven behavior eliminates the need for devices to wait on data or poll for data when no changes are pending, thereby enhancing overall efficiency of the design. The *plc-binding* package comprises three core components: *interface*, *model*, and *protocol* The *interface* handles tasks such as establishing and terminating connections with the PLC, as well as managing message communication and parsing incoming PLC messages. A simple illustration of the system design is depicted in Fig. 5. Within the *protocol* module, nested data classes are used to encapsulate self-descriptions, and encoding/decoding methods facilitate the translation of Python native data types into binary representations and vice versa. The primary objective of the *model* component is to maintain a comprehensive representation of the PLC structure. This class is designed to be populated with self-descriptions and maintains relationships between device classes and their parameters.

The *devices* package consists of the *bases*, *beckhoffCom*, *devices*, and *factory* modules. Within the *bases* module, the pivotal classes *beckhoffProperty* and *beckhoffSlot* are implemeted. These classes define the strategies for managing data updates from the hardware and ensure that changes made in the Karabo device are correctly applied to the hardware, thereby orchestrating seamless data synchronization. The *factory* module implements the methods for creating devices, properties, and commands. Within the *devices* module, the foundational *BeckhoffDevice* class is defined, serving as the base class for all Karabo Beckhoff devices, and specific Beckhoff devices are to be implemeted in this module. The *BeckhoffCom* device is the arbiter device that makes communication between hardware devices and Karabo devices possible. Upon instantiation, it establishes a connection to the PLC via the *interface* module. Once a successful connection is established, it retrieves the self-description of available PLC devices and generates *BeckhoffDevices* instances with properties and commands defined by the interface. Subsequently, BeckhoffCom requests the current configuration for each device and updates the devices when updates occur. The implementation is designed with testability in mind, using PyTest [8] for both unit and integration tests. The current test coverage is 92 percent.

OUTLOOK

The development process is still ongoing, The forthcoming implementation is expected to be deployed in production in 2025 during the long shutdown of the infrastructure.

REFERENCES

[1] M. Altarelli *et al.*, "The European X-ray free-electron laser facility in Hamburg", *Nucl. Instrum. Methods Sect. B*, vol. 269, no. 24, pp. 2845–2849, Dec. 2011. doi:10.1016/j.nimb.2011.04.034

[2] S. Hauf *et al.*, "The Karabo distributed control system", *J. Synchrotron Radiat.*, vol. 26, no. 5, pp. 1448–1461, 2019. doi:10.1107/S1600577519006696

[3] Beckhoff, https://www.beckhoff.com

[4] N. Coppola, J. Tolkiehn, and C. Youngman, "Control Using Beckhoff Distributed Rail Systems at the European XFEL", in *Proc. ICALEPCS'13*, San Francisco, CA, USA, Oct. 2013, paper TUPPC046, pp. 669–672.

[5] Boost::asio, https://www.boost.org/doc/libs/1_77_0/doc/html/boost_asio.html

[6] Python's asyncio, https://docs.python.org/3/library/asyncio.html

[7] T. Freyermuth *et al.*, "Progression Towards Adaptability in the PLC Library at the EuXFEL", in *Proc. PCaPAC'22*, Dolní Brežany, Czech Republic, Oct. 2022, pp. 102-106. doi:10.18429/JACoW-PCaPAC2022-FRO13

[8] Pytest, https://docs.pytest.org