ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-THPDP065

UNIFIED SOFTWARE PRODUCTION PROCESS FOR CERN'S CRYOGENIC CONTROL APPLICATIONS

M. Pezzetti, T. Barbe, C. Fluder, T. Kubla, A. Tovar-Gonzalez
CERN, Meyrin, Switzerland
Sebastian Jan Rog, AGH, Cracow, Poland

Content from this work may be used under the terms of the CC BY 4.0 licence (© 2023). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI

Abstract

The software engineering of process control system for CERN cryogenic installations is based on an automatic code production methodology and continuous integration practice. This solution was initially developed for the LHC Accelerator applications, then adapted to LHC Detectors, test facilities and non-LHC cryogenic facilities. Over the years, this approach allowed the successful implementation of many control system upgrades, as well as the development of new applications, while improving quality assurance and minimizing manpower resources. The overall complexity of automatic software production chains, their challenging maintenance, deviation between software production methods for different cryogenic domains and frequent evolution of CERN frameworks led to the system's complete review. A new unified software production system was designed for all cryogenic domains and industrial technologies used. All previously employed frameworks, tools, libraries, code templates were classified, homogenized and implemented as common submodules, while projects specific configuration were grouped in custom application files. This publication presents the new unified software production solution, benefits from shared methodology between different cryogenics domains, as well as a summary of two years of experience with several cryogenic applications from different PLCs technologies.

INTRODUCTION

Large cryogenic systems are an integral part of CERN's accelerator complex and experimental facilities, especially the Large Hadron Collider (LHC) with its detectors [1]. The 24h a day continuous cryogenic operation during the physics campaign is mandatory to the collider's stable beam operation. As any large and complex technological system, the cryogenic installations process control programs running on an industrial control PLC must be as robust and error-free as possible. At CERN, the design and the development of the process control software is performed through the CERN UNICOS framework [2], which defines a set of conceptual objects such as valves, heaters, IOs, alarms, PID controllers, etc. Instead of writing PLC code manually, the UCPC (UNICOS Continuous Process Control) code generator creates the code based on a specification file and Python templates. For several years, CERN has developed various solutions to help with the whole lifecycle of a cryogenic process control system [3]. Nowadays, CERN's systematic use of version control (with git) and continuous integration (CI) in the engineering workflow has significantly reduced the necessary time (from months to weeks) dedicated to develop, test and deploy a new application or an upgrade. This has allowed the production of quality and highly reliable cryogenic control and electrical systems [4].

CERN CRYO CI SYSTEM HISTORY

The first control system using continuous integration was the 18 PLCs LHC Cryogenic Tunnel Applications, followed by cryogenic magnet test benches using the Siemens PLCs technology [3]. Additionally, a database-oriented information system was chosen for the LHC tunnel Cryogenic process control in the project's early development [5]. This choice has enabled the successful use of the continuous integration system toward a higher automatization level. It allowed to improve development speed and reduce the time in feedback loops. After a successful use of the CI in these two cases, it was progressively rolled out to all cryogenic process control systems at CERN. This includes LHC Detectors ATLAS and CMS [6] which are controlled by Schneider PLCs technology. The last category consists of several non-LHC and test facilities installations that use a mix of Siemens and Schneider PLCs. Each of these domains had their own special requirements that lead to the development of slightly different CI solutions (cf. Fig. 1). After some time, this became very hard to maintain and the system had to be redesigned.

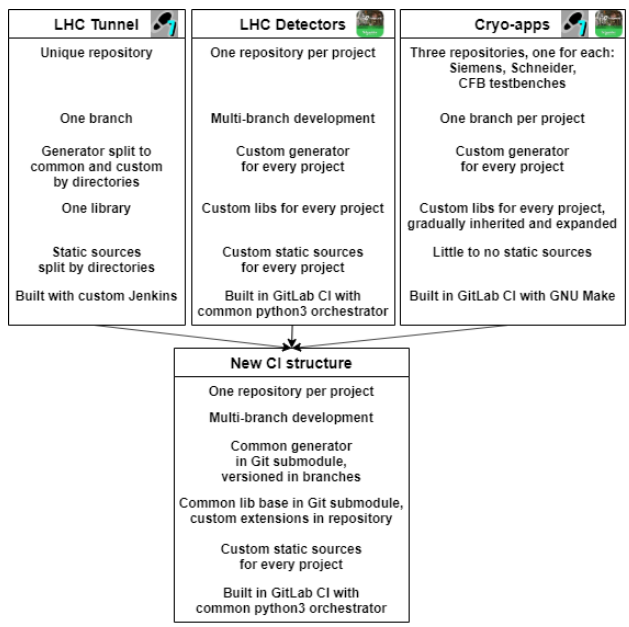

Figure 1: Evolution of CI systems.

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-THPDP065

Table 1: Size of Our Cryogenic Installations

	# of PLCs	# of I/Os
LHC Tunnel	18	55000
LHC Detectors	12	10000
non-LHC	70	28000

CI System Design Principles

The CI system(s) developed presented some limitations (e.g. being heavily application-focused) and did not scale effectively. The challenge to maintain more than a hundred applications (see Table 1 for details) with limited team forced the need for an improved continuous integration system, more efficient and multi PLC technology. Based on the return of experiences with the first version, we were able to point out weaknesses and address them. The key lessons learned are as follows:

- AVOID GROUPING of applications: non-LHC applications were grouped into a single git project and organized into branches. As these projects were relatively independent, this resulted in a large repository that slowed down operations such as git checkout and clone, with no real benefits.
- SHARE common parts: Although each application is unique, they often share a common structure and a significant amount of common files (e.g. the CI configuration). When one of the common files had to be modified/improved/fixed, it was done directly in the project, causing a slow but certain divergence of this shared part. Eventually, no one could ascertain the correct version or why differences existed.
- MINIMIZE rule variations: Many applications were considered specific, justifying slight variations in how they were approached (e.g. the 18 LHC-tunnel applications), bringing additional complexity. However, after due consideration, these specificities could be accommodated in a unique, consistent approach to project construction.

NEW STRUCTURE CI SYSTEM

Git Repository Structure

The existing continuous integration framework was completely revised, redesigned and the following new structure was proposed (Fig. 2). Now every application has its own git repository separated into three main parts:

- CUSTOM: Contains all project-specific data, this is the "real" content of the git repository.
- COMMON: Contains links to git submodules (see next section).
- TEMP: Not present in the repository itself, contains build artifacts from the GitLab® CI.

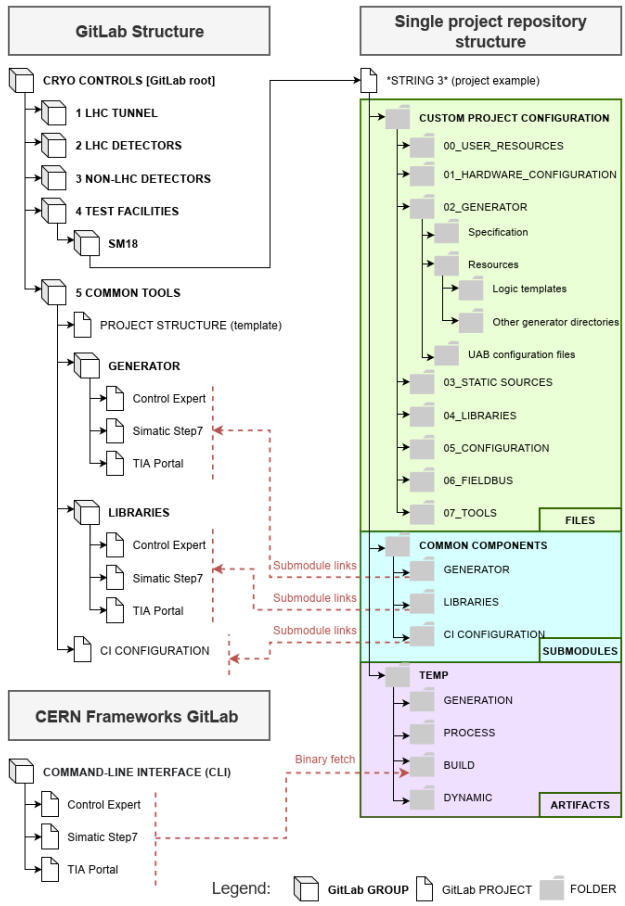

Figure 2: Git repositories structure.

Each project's custom part contains:

- A hardware configuration, a minimalistic PLC project containing only the CPU (with the IP address) with configured additional hardware used in the project.
- Data required by the UCPC code generator to create the logic files. This includes configuration files, the specification file and Python templates.
- Static sources and libraries specific for this project that are imported as-is during the build stage.
- Additional data not required for the build but nevertheless required to form a logically complete project.

The distinction between static sources and libraries is subtle. In the Siemens environment, source files are compiled into blocks, which collectively constitute the program. While sources can exist independently of the project, blocks cannot. Libraries in this context refer to specialized projects that encompass blocks imported during the build process.

In contrast, in the Schneider environment, there is more flexibility in importing and exporting any data as XML files. Here, static sources typically denote files integrated into the master task, while libraries often contain data and function block (FB) types, or variables. It's crucial to arrange static files with care because their importation order also dictates their execution sequence within the master task.

Content from this work may be used under the terms of the CC BY 4.0 licence (© 2023). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-THPDP065

Content from this work may be used under the terms of the CC BY 4.0 licence (© 2023). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI

Use of Submodules

The key feature of this improved CI system is its intensive utilization of submodules. A significant effort was invested in identifying common parts in all our applications and extracting them as git submodules. This approach not only reduces code duplication but also standardizes the execution of certain functions in our software, such as communication, alarm handling, and bit manipulation, among others. Furthermore, in the event of a bug discovery or the requirement for an essential update, these changes are implemented in a separate git repository, ensuring that every project depending on it receives the necessary updates. The custom part of any application is made minimalistic (one specification file, a couple of object-specific templates and a few static sources), thus simplifying future maintenance.

Among the submodules, the most critical one is the generator submodule. For nearly all projects, 90% of the code templates are identical, with most of them originating directly from the UCPC tool. These templates are housed in a single repository (per PLC technology) and are linked by every project. During this process, the common templates are merged with the custom ones present in the project repository creating a comprehensive generator that can be executed. The generator submodule makes heavy use of git branches. Applications use different UCPC versions, defined by what version was the most recent (or stable) at the time of commissioning. To maintain a single generator submodule (for one PLC technology), the different versions are separated in branches. In case one application has to be redeployed with a new UCPC version, it suffices in the easiest case to change the submodule reference.

Additionally, there is an optional submodule dedicated to shared templates. In case where a group of projects exhibit significant similarity (such as the SM18 horizontal test benches), they share a lot of project-specific generator templates. Instead of duplicating these templates within each project's repository, an additional middle layer is created between the project's custom and common part. These projects then source their templates from three different places (common generator, shared template's repository, custom templates in the project itself) which are combined before running the generator. This system also facilitates making logic changes that should apply to all subprojects; only one repository needs to be modified in such cases. Currently, it is used by the LHC tunnel projects and superconducting test benches.

Another submodule contains libraries common for a PLC technology. They consist of either standard functionality such as communication or custom-made implementations of functions used in most of our projects such as bit manipulation. During build, these are imported together with the custom libraries present in the repository.

The CI system itself is also a submodule, allowing to detach the CI development from the application development. The CERN cryo CI system is implemented in Python3® (besides yaml files required by the GitLab® CI) as opposed to GNU Make. Using a fully-featured programming language allows to better orchestrate all the complexity around the git submodules, collecting files, etc.

NEW CI SYSTEM FEATURES

Spec Version Control

The input specification utilized by the UCPC code generator is an Excel spreadsheet. This choice offers advantages in terms of working with tabular data, allowing easy mass-edits, filtering and formatting as well as easier navigation.

However, there is a drawback to using Excel: it does not harmonize well with version control systems like Git due to it fundamentally being a binary format.

Since the specification undergoes frequent edits, tracking changes, especially when attempting to trace them backward using Git's blame feature, becomes an arduous task. To address this issue, the repository contains a set of JSON files, representing the different CERN UNICOS objects, systematically updated with every change to the specification. This approach allows us to track changes effectively on these files. It's worth noting that these JSON files are not utilized in the program generation itself, instead they serve as a static snapshot of the current specification in the commit action in text format. This enables us to leverage standard Git tracking features and mitigate the version control challenges associated with the Excel-based specification.

Comparisons of Code Version: Quality Assurance Tests

Besides the typical generation and build tasks, the pipelines also contain verification test jobs. These tests play a vital role in enhancing overall quality assurance by identifying and rectifying common yet significant errors, such as incorrectly configured IP addresses, missing spare objects in the specification, or unused code in the program. This is very helpful during an installation upgrade as it forces us to deal with accumulated legacy code and ensure the project is in a pristine state before deploying the upgrade.

Furthermore, two comparison jobs were implemented. One of these jobs compares the currently built project to the previous build, while the other compares it to the version currently running in production. These comparisons serve as double-check mechanism for changes introduced in code. Any disparities detected are meticulously processed to eliminate non-essential information like timestamps and specification versions, presenting only the pertinent alterations. Automatic comparison to production environment also helps in reverse-engineering tasks. Often, when modifications have to be done during the operation of the installation, these changes are initially performed manually with the PLC software connected to the production PLC. Subsequently, these changes are backported to the Git project. Having an automatic comparison to the production environment aids in this transition, ensuring that the operational adjustments are accurately reflected in the version-controlled codebase.

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-THPDP065

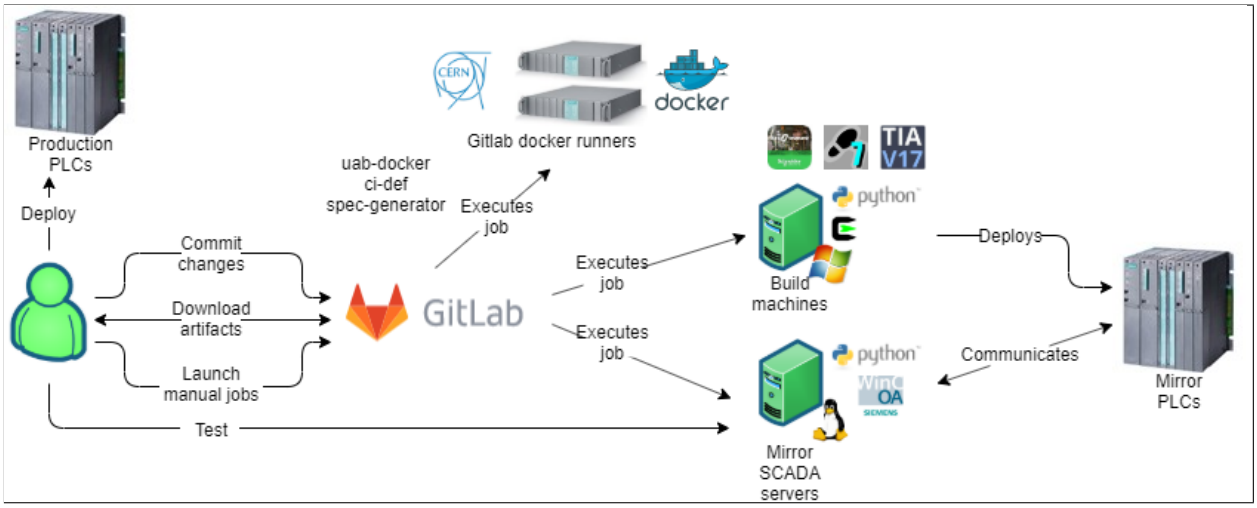

Figure 3: CI structure.

Mirror PLC Validation Test Platform

Additionally, subpipeline scripts offer the capability to automatically build and deploy a validation testing platform known as "mirror PLCs". This platform serves both developers and cryogenic operators. Developers use it to check the correctness of their implementations, while operators employ it either to review the correctness and relevance of the planned process control or for training purposes.

The mirror PLCs platform is composed of the same PLC CPU and a supervision SCADA system installed on an OpenStack® virtual machine (VM). The pipeline automatically adjusts the IP addresses, builds the project and deploys both the program to the PLC and the corresponding communication configuration to the SCADA. During the generation phase, an additional script is executed, providing basic simulation features, such as simulating circuit breaker status or copying the command of the actuator to the feedback. This approach enables the operator to easily connect to the SCADA VM and start testing. While the platform doesn't simulate the actual process dynamics (e.g., pressure or temperature evolution), the behaviour of actuators like valves and heaters mirrors that of the real plant. Previously, this setup demanded a manual update of the generator, downloading build artifacts, connecting the mirror PLC, SCADA, and executing program and communication uploads manually.

NEW CI SYSTEM INFRASTRUCTURE

Delegation of Responsibility

A comprehensive consolidation initiative has been implemented to streamline our processes and minimize the complexity of maintaining additional infrastructure. Here are the key changes and improvements (see Fig. 3):

1. CI Build System Transition: The custom Jenkins build system was swapped in favor of the CERN-provided GitLab CI. Instead of custom runners in OpenStack®, most jobs run in Docker containers on the CERN-provided shared runners' infrastructure.

2. Docker Images for CI: Our CI pipeline requires three distinct Docker images: one with the UCPC generator for the generation job, one with an Oracle client to build the database-backed specification file for tunnel applications [5] and one minimalistic with just Python3® for all other jobs.

3. Outsourcing CLI Tool Development: The responsibility for developing and maintaining Command Line Interface (CLI) tools required to control the various PLC "IDEs" (Schneider Control Expert®, Siemens Simatic Step7® and Siemens TIA Portal®) are now under the responsibility of the CERN central control group at CERN. We do no longer maintain our own versions of these tools, the provided compiled binaries are automatically pulled in our pipelines.

GitLab Runners

Maintaining custom runners is however still required for certain tasks, primarily Microsoft Windows® runners for PLC build jobs. This necessity arises from the fact that Siemens and Schneider software tools are not compatible with Linux environments.

The runners' architecture is similar to [3], employing Cygwin to provide a Linux-like environment for the execution of the CI and calls .exe CLI programs to communicate with the build tool.

These runners are registered as group runners in GitLab and are differentiated by tags according to what PLC software it has installed (Control Expert®, Simatic Step7®, TIA Portal®) so that the correct one is picked for the build job of an application. At present, we utilize the Windows Server 2019® version on most runners due to its perceived stability and reliability concerning automatic updates.

Additionally, the mirror SCADA VMs are registered as GitLab runners so that a new mirror build can be directly deployed to them. In this setup, each project specifies in its configuration which mirror SCADA environment it should be deployed to. GitLab then dispatches the deployment job based on runner tags, ensuring that the build is deployed to the mirror SCADA system associated with the project.

Content from this work may be used under the terms of the CC BY 4.0 licence (© 2023). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-THPDP065

Content from this work may be used under the terms of the CC BY 4.0 licence (© 2023). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI

CERN CI REALIZED PROJECTS

This new CI was used in the two extensive upgrades of cryogenic installations:

- HIE-ISOLDE: The process control logic of this system was outdated and unnecessarily complex. We took profit of the installation shutdown to completely re-implement it. After receiving the logic specification, implementation and testing took three weeks, instead of the previous months. The automated mirror generation feature was crucial during the development of this application and was used several times. It allowed iterative progress between the software developer and the operator, with a systematic validation of new feature of the cryogenics process along with the software coding. After deployment, the installation successfully operated according to specification.

- ATLAS Argon [7]: After 12 years, the control system of a critical part of the ATLAS detector had to be upgraded. This project spanning over multiple years required a complete logic review and extensive testing. The CI helped on focusing manpower on the design and testing, implementation was only a minor part of the effort. The feedback loop between bug discovery and the patched version being deployed for testing was shortened and almost automatic. Additionally, test jobs helped in putting the project in a fresh-like state before future improvement will be made.

- SM18 CFBs: The SM18 Cryogenic Feed Boxes (CFB) are a complex proximity cryogenic system dedicated to test of superconducting magnet. Some of the CFB test benches underwent significant mechanical modification to allow for the test of the new High Luminosity LHC magnets. The control system had to be upgraded to cope with those new requirements. The use of submodule allowed to guarantee fast and consistent updates of the modified benches optimizing our manpower allocation.

CONCLUSION

CERN's cryogenic process control system extensively uses continuous integration and automatic code generation. Our unified software production solution applies a shared methodology, enabling a robust and automatized global process control system generation from design to operation. The reorganization of our git repositories and the leveraging of submodules significantly improved code reuse and design consistency. By transitioning to GitLab CI and Docker, the need for custom runners was reduced. Critical software tools, including the UCPC code generator and PLC CLIs, are now maintained by the CERN central control group, alleviating our operational concerns. Moreover, with the integration of mirror SCADA system into our CI ecosystem, it is possible to automatically deploy mirror builds, which streamlines validation and testing. The addition of quality assurance tests into our CI pipeline also limits the risk of errors and deployment failures. These results highlight our commitment to finding smarter and more efficient ways to reach our scientific goals.

ACKNOWLEDGEMENTS

CERN's cryogenic process control responsibilities are carried out collaboratively with several other groups within the organization. We want to extend our thanks to all the groups that have contributed to this effort. Special appreciation goes to the BE-ICS group for their technical support in maintaining the CERN UNICOS framework. We also acknowledge the contributions of the BE-CEM and BE-CSS groups, as well as the IT infrastructure teams.

REFERENCES

[1] M. Pezzetti, "Control of large helium cryogenic systems: a case study on CERN LHC", in *EPJ Techn. Instrum.*, vol. 8, no. 6, Feb. 2021. doi:10.1140/epjti/s40485-021-00063-w

[2] CERN UNICOS Framework, https://unicos.web.cern.ch/

[3] C. Fluder *et al.*, "Automation of the software production process for multiple cryogenic control applications", in *Proc. ICALEPCS 2017*, Barcelona, Spain, Oct. 2017, pp. 375–379. doi:10.18429/JACoW-ICALEPCS2017-TUPHA006

[4] T. Barbe *et al.*, "An innovative approach for the design of cryogenic electrical and process control systems at CERN: the cryogenic Continuous Integration project", in *Proc. Cryog. Eng. Conf. 2021*, Virtual Conference, USA, Jul. 2021, paper 012042.

[5] A. Tovar *et al.*, "Validation of the Data Consolidation in Layout Database for the LHC Tunnel Cryogenics Controls Package", in *Proc. ICALEPCS'13*, San Francisco, CA, USA, Oct. 2013, paper THPPC057.

[6] C. Fluder *et al.*, "Status of the Process Control Systems Upgrade for the Cryogenic Installations of the LHC Based ATLAS and CMS Detectors", in *Proc. ICALEPCS'19*, New York, NY, USA, Oct. 2019, pp. 1215. doi:10.18429/JACoW-ICALEPCS2019-WEPHA050

[7] C. Fluder *et al.*, "Upgrade of the process control system for the cryogenic installation of the CERN LHC ATLAS liquid argon calorimeter", presented at the ICALEPCS'23, Cape Town, South Africa, Oct. 2023, paper TUPDP091, this conference.