Available online at www.sciencedirect.com

ScienceDirect

Physics Procedia 67 (2015) 1134 – 1140

Physics Procedia

25th International Cryogenic Engineering Conference and the International Cryogenic Materials Conference in 2014, ICEC 25–ICMC 2014

Improved software production for the LHC tunnel cryogenics control system

C. Fluder[a,*], T. Wolak[b], A. Drozd[b], M. Dudek[a], F. Frassinelli[c], M. Pezzetti[a], A. Tovar-Gonzalez[a], M. Zapolski[b]

[a] *CERN, CH-1211 Geneva 23, Switzerland*
[b] *AGH UST, 30 Mickiewicza Av, 30-059 Krakow, Poland*
[c] *Politecnico di Milano, Piazza Leonardo da Vinci 32, 20133 Milan, Italy*

Abstract

The software development for the control system of the cryogenics in the LHC is partially automatized. However, every single modification requires a sequence of consecutive and interdependent tasks to be executed manually by software developers. A large number of control system consolidations and the evolution of the used IT technologies lead to reviewing the software production methodology. As a result, an open-source continuous integration server has been employed integrating all development tasks, tools and technologies. This paper describes the main improvements that have been made to fully automate the process of software production and the achieved results.

Peer-review under responsibility of the organizing committee of ICEC 25-ICMC 2014

Keywords: CERN; LHC; cryogenics; industrial; control; software; development; continuous integration

* Corresponding author. Tel.: +41-76-487-49-78; fax: +41-22-767-88-85.
E-mail address: Czeslaw.Fluder@cern.ch

Peer-review under responsibility of the organizing committee of ICEC 25-ICMC 2014
doi:10.1016/j.phpro.2015.06.176

1. Introduction

The control system for the cryogenics in the LHC tunnel maintains the LHC accelerator superconducting magnets, radio frequency cavities and electrical distribution field boxes at cryogenic nominal condition. All systems are distributed along the 27 km tunnel and split into eight independent sectors, each 3.3 km long.

From the very beginning, the large scale of the control system forced the use of automatic code production in development processes. The existing CERN code generation tools were adapted to cover the requirements of the control system, which became fully operational for the first time in 2008. Experience after months of operation led to the review and optimization of the process functional analysis (Gomes et al. 2009). As a result the second major release was successfully deployed in 2010, ensuring the operability of the cryogenic infrastructure during the first LHC run. In 2013, the LHC entered the first Long Shutdown (LS1), a 2-year consolidation and maintenance work period. Driven by the technical requirements and by the feedbacks from the first LHC run period, numerous hardware consolidations and software improvements were scheduled to be executed during the LS1. The evolution of related IT technologies, including hardware platforms, operating systems, CERN core frameworks and commercial software made old systems unsupported and unprotected. In consequence, rebuilding the control system applications became necessary.

2. The architecture of the control system for the cryogenics in the LHC tunnel

The highly distributed control system architecture follows the structure of the LHC cryogenics. The readout of the 26500 instrumentation channels distributed along the 27-km tunnel is performed through two main types of field-buses: Profibus and WorldFIP. Front-End Computers (FEC) are used as gateways between the WorldFIP I/O modules and the SIMATIC S7 Programmable Logic Controllers (PLC), while the Profibus remote I/O is accessed directly by the PLCs. Each of the eight sectors is controlled by a set of two PLCs (arc section (ARC) and long straight section (LSS)) with two human-machine interfaces (HMI) using a WinCC OA system: the Supervisory Control and Data Acquisition (SCADA) and the Cryogenic Instrumentation Expert Tool (CIET).

3. Control system software development process

The control system software is developed using the UNIfied Industrial COntrol System framework (UNICOS) and its Continuous Process Control package (UCPC), providing a library of standard device types (objects), a methodology and a set of tools to design and implement industrial control applications (Fernandez et al. 2011). Developing control system software within UNICOS CPC6 environment is a complex process, composed of the tasks presented on Fig. 1a.

The first step of the process (1), the generation of the control system configuration, consists of two parts: generating the objects' specification and configuring the PLC hardware. All the data describing hardware and computed objects is stored and maintained in the LHC Layout Database (Layout DB) (Tovar et al. 2013). Dedicated software tools are then used to generate the specification in the format required by the UCPC framework and to produce a file with the PLC hardware configuration in a format accepted by the SIMATIC environment.

The consistency of the generated specification is verified in step 2. It is done using the UNICOS Application Builder (UAB) with generic and user defined validation rules. It checks whether all the objects' properties are correct and whether the interdependencies are satisfied and it also verifies system-specific requirements. In the UCPC environment passing data validation procedure is mandatory before executing code generation.

Step 3 is the main part of the development process with the UCPC, in which the UAB takes a previously generated specification as an input and generates a source code for a complete PLC application and an object database for SCADA. Custom functionality of the system can be implemented in user templates (Jython) executed by the UAB. The generated database can then be imported (4) to a configured SCADA project hosted on the WinCC OA server. At the same time the complete PLC project can be built (5) in the Siemens SIMATIC Step7 by importing and compiling the generated hardware configuration and all the generated and static software components. This task, i.e. importation of many source code files (over 1000 for larger ARCs) and their compilation, is the most complex and time-consuming part for the developer.

The successfully built project is deployed (6) in a test environment, which only partially imitates the production system since PLCs are not connected to any real hardware, such as sensors or actuators. The testing (7) has to be performed manually by checking if the system reacts properly on the specific conditions of the system. The objects can be manipulated and observed through an HMI of the SCADA system or directly in the PLC using the debugging functions of SIMATIC. After passing the tests the system is ready for release into production.

4. Development challenges

The large scale and complexity of the discussed control system makes many of the tasks quite complicated, error-prone, time-consuming (Fig. 1a) and special tools and environment are required for their execution. Moreover, despite strict procedures and supporting software tools, many tasks require a lot of attention and time for a developer, since many actions have to be triggered or done manually.

Software development is an iterative process: it requires many repetitions of a part or of the whole build chain to produce working applications. For instance, modifications in the Layout DB may imply changes in the semantics of the generated code and therefore they must be followed by going through all the steps of the project building process. It may last around 2 hours if we consider the most frequent development loop until steps 4-5, when most of the errors appear. Considering that the control system is composed of 16 applications and that the process needs to be repeated for each error or modification, it makes the development process very time-consuming. Completing the whole task of rebuilding all 16 applications could take up to 32 hours.

Building the project is followed by deployment and testing. For testing an important limitation is the lack of real hardware connected to the system - it significantly reduces testing possibilities. Also, due to the scale of the system, it is difficult to test the entire system manually.

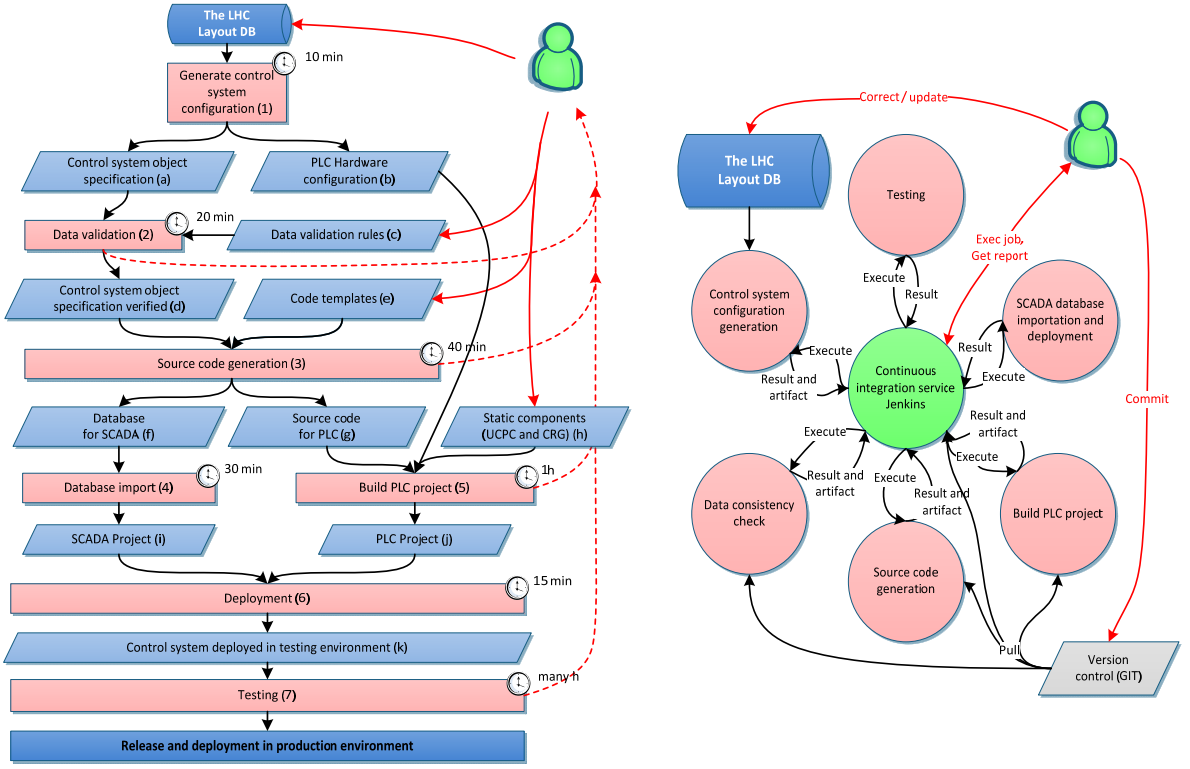

Fig. 1. (a) Software development process; (b) Development process with the Continuous Integration service.

Taking into account limited time and resources, achieving the goal of delivering upgraded, high-quality control system applications within given constraints was very challenging, and we were pushed towards seeking methods for improving and optimizing the development process.

5. Task automation and improvements

Clearly, the way to optimize the process and to spare the developer's time is to transfer as much work as possible to machines by automating each of the tasks and then, if possible, the whole build process. Unfortunately, while some of the steps could be automated very easily (e.g. with a dedicated software tool having as option a command-line interface or at least a programming interface), not all of the tasks allowed for easy automation and in some cases dedicated software tools had to be developed.

5.1. Improvements in generation and validation of control system configuration data

Generating both inputs with the system configuration (Fig. 1a (a) and (b)) is done by dedicated software tools. One of them, the one producing hardware configuration for PLCs, had to be re-implemented. It was prepared for the legacy UCPC5 systems and only had a graphical user interface, which makes task automation difficult. It was replaced with a simple and portable program developed in Python with `cx_Oracle` library.

Concerning the data validation process, the discussed applications have additional requirements regarding data consistency (in addition to generic and user-defined semantic checks supported by the UCPC environment). For instance, corresponding communication objects defined in both of the sector's PLCs must be defined in a coherent way (concerning order, names and memory addressing). Since such a check requires accessing data for two different UAB applications it cannot be performed in the UCPC environment and therefore additional external data validation tools, along with a library for accessing data of UCPC objects, had to be developed (in Python).

5.2. Automating data validation and code generation with the UNICOS CPC6

The UNICOS Application Builder, core software tool for the UCPC development process, is an application with a typical graphical user interface (initially dedicated to MS Windows users), which is very well suited for small and medium size projects (or partial generation of the code), for which the time required for executing data validation and code generation is not very long. In the case of large scale applications for the cryogenics in the LHC tunnel, producing a complete source code and a SCADA database for a single application can last up to an hour.

Automating graphical user interface is possible, but unnecessarily complex and not very robust - therefore it should be avoided whenever possible. Fortunately the UAB is a Java application and after a support request the UCPC developers provided a way for executing the process using Apache Maven making it possible to execute a complete generation using a simple command-line interface. Moreover, our tests revealed that it also allowed the execution of a generation process using different platforms (i.e. Scientific Linux CERN 6 with Java 1.7). After setting the environment properly it was also possible to execute multiple generation processes in parallel on the same machine and to execute the process on many machines at the same time easily, with a single command.

As a result, the generation of all control applications for cryogenics in the LHC tunnel could be executed at the same time, reducing the time required for producing the code for all applications by a factor equal to the number of applications (16 in our case), and to do it in a fully automated way.

5.3. Automating building Siemens PLC project

Building a complete PLC project from all components (static and produced by UAB) is one of the biggest, most complex and time-consuming tasks for the developer. All source code modules, binary program blocks from libraries and program symbols (all coming from a number of places) have to be manually imported by the developer to the base PLC project and compiled. PLC projects for Siemens PLCs are built within an integrated development environment (IDE) SIMATIC Step7 V5.5, a complete solution for developing control applications for PLCs. Being

well designed for small and medium size projects it has some weaknesses when working with large scale projects: both importation and compilation can be slow and absorb developers' time.

Unfortunately, SIMATIC does not provide a simple solution to execute these time-consuming actions in an automated way. Testing GUI automation proved the approach to be difficult and not robust enough in the case of environment changes. On the other hand, SIMATIC includes two libraries (Windows DLL: `SimaticLib` and `S7HCOM_XLib`) with a programming interface sufficient to develop a custom command-line tool. The program `s7cli`, written in C#, has a functionality allowing to script and to automate building PLC projects for our applications.

5.4. Automatic testing with hardware emulation device and the UNICOS simulation and testing framework

The longest part of the whole development process is testing the system. Thorough testing of only the most critical functionality of the system, alarms and interlocks, could last many hours (Fig. 1a): there are over 700 alarms in each of the ARCs and over 400 in LSS. This step was also very absorbing for developers since no supporting tools were initially available.

The availability of equipment simulating Profibus network with devices (SIMBA Profibus) and libraries allowed to develop software to control and to check the state of simulated devices. This, alongside an open-source library `libnodave` which gives access to on-line data of the program running in PLCs, permitted to develop the UNICOS simulation and testing framework. The framework, developed using Python programming language, provides an abstraction layer for manipulating and checking the state of UNICOS objects in a common way while, depending whether the object exists only internally (in the PLC) or if it is a Profibus device, a proper operation is executed. The framework also allows the implementation of custom testing scenarios. Several were implemented to test the most important functionalities of the discussed control system (alarms and interlocks, communication, calculated objects)

Automated testing permitted to test each of the most critical objects in the developed system, and to do it without using the time of developers. More detailed testing allowed to detect and correct many issues - for instance, affecting a single object (which otherwise probably would not be detected) - and to improve quality of produced software.

6. Automating the development process with Jenkins-based Continuous Integration system

The possibility to execute the vast majority, and (more importantly) the most time consuming and the most frequently repeated tasks automatically provides a way to integrate all the tasks of the development process into a Continuous Integration (CI) system. The system, after proper setup and configuration, completely frees developers from executing the tasks manually (Fig. 1b) and also provides many additional useful features.

All automated tasks are defined and configured in the CI system as jobs. Their execution can be requested by a developer or, what is more intensively used in practice, by user-configured triggers scheduling execution, for instance, at a specified time (e.g. every night) or when an event such as a code update in the version control system is detected.

The status of a user-defined number of job executions, a detailed log from the console output of the executed programs and the build artifacts (i.e. generated data or code, also defined in the configuration) are automatically stored in the CI system and they are available for later download and review.

The jobs configured in the CI system implement the same development process (Fig. 1a): the jobs are linked to be executed in the same sequence. The successful execution of a task may trigger the execution of the following ones in the build chain, passing build artifacts (i.e. generated data or code) automatically as inputs to the next tasks – this way all jobs in the build chain may be executed without any human supervision. It usually happens after updating the code in GIT repository and (partially) every night.

A failing job in an earlier stage is not a blocking problem - the latest artifact from a successful execution can be used to execute the following tasks, so the development can progress on different stages in parallel.

Additionally, our task configuration in the CI system also reflects the development team's workflow, allowing to execute the same task (e.g. data consistency check) with different inputs, isolating development stages. It is important, because, for instance, a newly implemented functionality may require additional arguments

in the system object definition. It is often necessary to test such a modification before applying permanent changes in the Layout DB.

Developers having a CI system available can allocate their time to work on solving issues or preparing new code, which is automatically tested after committing changes to the revision control system (GIT), while results can be reviewed and build projects downloaded accessing CI system with any common Web browser.

7. Continuous integration system for the LHC tunnel cryogenics controls software today

The CI solution for developing controls software was built gradually, starting from automating tasks, finding a CI system that fulfills the needs (Jenkins) and initially allowing only partially built applications for just one sector. Then the solution evolved to cover all the needs and to fit the schedule of our developments, building all our applications and performing a number of additional validation tasks, for instance reporting changes in data coming from the Layout DB. The initial configuration composed of three virtual machines has grown to the configuration presented on Fig. 2, with additional useful services like JIRA or FishEye integrated with the system.

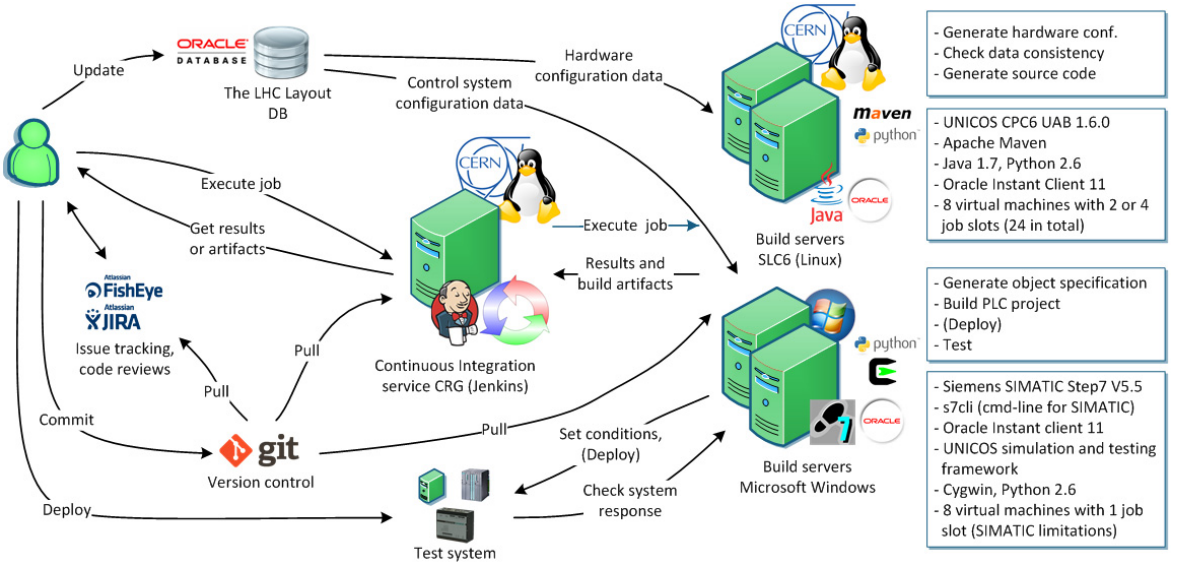

Fig. 2. The architecture of the Continuous Integration system for control system software development.

Concerning possible future improvements, the two missing tasks part of preparing the test system (deployment and SCADA setup / importation) may be implemented to fully complete the build chain. However, since these tasks were less frequent it was not necessary for us to optimize this part.

8. Conclusion

The solution allowed to produce and to successfully deploy control software in all the sectors of the cryogenics systems of the LHC tunnel and also to implement all requested last-minute changes. Currently two sectors are being cooled down while others are operational, in re-commissioning phase.

The Continuous Integration practice, including automated builds and tests, has been successfully used in software engineering for many years. Discussed developments, experiences and results of applying this approach to improve the process of producing control system software for the cryogenics in the LHC tunnel proved that the methodology can be equally useful in the field of industrial automation. It allows to optimize the development process significantly and (with the help of hardware simulation devices) to raise the quality of the software produced for large-scale control systems.

References

Fernandez Adiego B. et al., 2011, UNICOS CPC6: automated code generation for process control applications, ICALEPCS2011, Grenoble, France, WEPKS033.

Gomes, P. et al., 2009, The control system for the cryogenics in the LHC tunnel [first experience and improvements], ICALEPCS2009, Kobe, Japan, WEP061.

Jenkins, An extendable open source continuous integration server, http://jenkins-ci.org

Tovar-Gonzalez A. et al., 2013 Validation of the data consolidation in Layout database for the LHC tunnel cryogenics controls upgrade, ICALEPCS2013, San Francisco, CA, USA, THPPC057.