



SMART GUIDED COMMISSIONING OF INDUSTRIAL CONTROL SYSTEMS

W. Niinimäki*, Z. A. Morrissey-Ralevic, M. Marin, H. B. Dennis, J. M. de Paco
CERN, Geneva, Switzerland

Abstract

This paper proposes a web application to assist in the commissioning of industrial control systems with guided functional tests. The tests verify that trigger conditions produce expected responses within the control system by automatically reading process states from the controllers (e.g. PLCs). The commissioning engineer will receive step-by-step guidance according to the test scripts, real-time feedback, and a final report at the end of the test. The reproducible tests aim to reduce human error and speed up commissioning by automating the tedious task of process state verification.

COMMISSIONING TESTING OF INDUSTRIAL CONTROL SYSTEMS

Commissioning testing refers to the functional testing of a control system in production before the machine is handed over to the operators. During commissioning testing the commissioning engineers verify the full functionality of the machine.

Commissioning testing procedure in the Converter Controls Software section at CERN

The commissioning testing procedure is dependent on the project at hand, but it at least covers the following points:

- Hardware IO tests.
- Triggering every alarm and confirming their effects and interlocks.
- Traversing the state machine(s) of the industrial control system.
- Confirming the most important conditions for actuations or transitions in the state machine(s).
- Operational mode tests (if applicable).

AUTOMATED FUNCTIONAL TESTING

The main idea of automated testing is to catch any issues and bugs in software before they make it to the production environment, and it's been the industry standard practice for a long time in IT. Automated tests are scripts used to verify that a software component behaves as expected. These tests vary in their scopes all the way from testing an isolated function to full integration tests between interfaced systems.

In contrast to IT, Operational Technology (OT) has traditionally relied on manual testing methods, such as Factory Acceptance Testing (FAT) and Site Acceptance Testing

(SAT). However, in the past few years, automated software testing has started to emerge in OT development cycles. For example, Siemens provides a way to unit test TIA Portal applications with Test Suite add-on, and more recently integrated function tests in the Simatic AX development environment. Other vendors also provide similar tools for their ecosystems. This paper explores the use of functional testing of a full system, including the control system, actuators and sensor readings, most closely aligned with end-to-end testing.

APPLYING AUTOMATED TESTING TO IMPROVE MANUAL COMMISSIONING

Automated software testing can support and enhance manual commissioning through a method we call *smart guided commissioning*. The goal is not to replace manual commissioning entirely, but to speed it up, reduce errors, and ensure consistency by automating repetitive tasks and enforcing correct test procedures.

In smart guided commissioning, test scripts guide the commissioning engineer step-by-step. They indicate the required actions, automatically verify system responses by monitoring live process values, log state changes with timestamps, and generate test reports automatically. This approach offers several benefits:

- Faster commissioning in production environments.
- Traceable and repeatable test results.
- Reduce the chance for human error by automating the verification of live process values.
- Suite of reproducible tests that can be used for example for regression testing during maintenance of the system to confirm that everything is functioning as expected.

However, preparing high-quality test scripts takes time. Mistakes in these scripts can disrupt commissioning or result in false positives.

SMARTCO

SmartCo is a web application that was developed to prove the concept of smart guided commissioning. It allows users to create and run tests, as well as store and view test reports. A test in SmartCo connects to all relevant controllers, provides real-time feedback and instructions, verifies process states and generates test reports with time stamps. SmartCo also manages and stores all the test files and reports centrally.

* walter.niinimaki@cern.ch

Tests

A test is a composition of sequentially executed steps. Currently there are three types of steps that are available for composing a test.

Assert immediately verifies that the process states are as expected. This can be used to verify that the process has reacted in a correct way to an event.

Trigger waits for the process states to match the expected states before carrying on with the test. Triggers can also be configured to expire after a specified duration if the process states were not reached, which will lead to the test timing out and failing. For example, a trigger step can be used to begin a test by waiting for the commissioning engineer to turn a switch to a specific position.

Sleep waits a specified amount of time before proceeding with the test. This can be used to allow the process to react before the test continues to the next step.

Additionally, constants can be defined, for example, for sleep durations.

Tests are defined in YAML files, which are parsed by SmartCo's backend during execution. An example from the beginning of a test script can be seen in Figure 1. It shows a combination of a trigger and an assert step to confirm initial states in the system, a delay step to wait for event propagation in the control system, and an assert to verify that the system has reacted in a correct way. The orange banner indicates that a production environment is being used for the connection configuration.

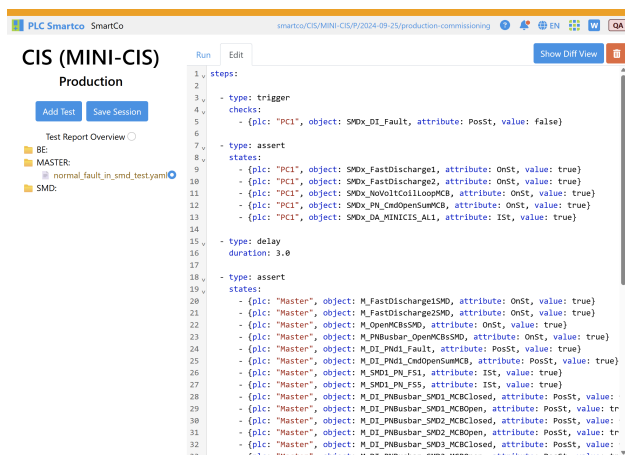


Figure 1: YAML-definition for a SmartCo test.

The run view of a test gives a rendered view of the YAML as can be seen in Figure 2. It shows the live state for each step in the test. Following the conditions and possible comments will guide the commissioning engineer through the test.

The steps are constantly being updated and the currently active step is highlighted as seen in Figure 3. In case a condition is lost before the step is completed, the states are also updated and timestamped.

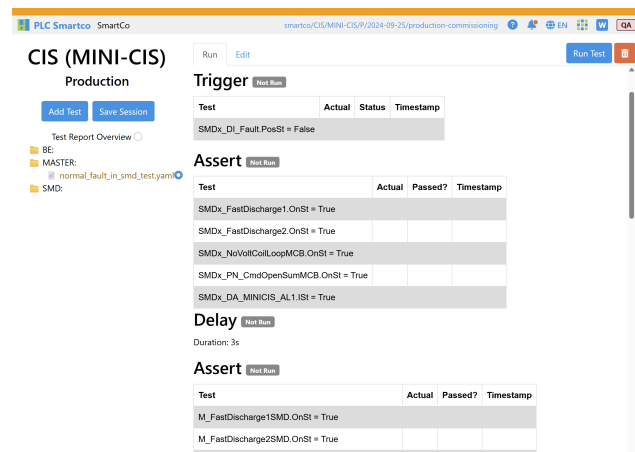


Figure 2: Running a test renders a live feedback view of the YAML-definition.

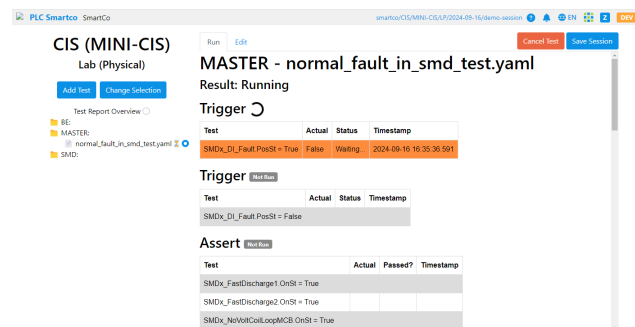


Figure 3: Currently active step is highlighted in orange in the run view.

Test reports

SmartCo keeps track of process value states and changes. It timestamps any verification that is done during a test. Regardless of whether a test has passed or not, a test report is generated afterward. The test report is stored in a JSON file that contains the process values of each completed step together with the time stamps. A serialized version of the YAML test file is also included in the JSON. These test reports are stored in GitLab and they can then be viewed through the SmartCo web application.

Sessions

Creating a SmartCo session corresponds to creating a branch in Git. When a commissioning engineer starts a testing session, SmartCo creates a branch in the remote GitLab repository. Any modifications to the tests, along with test reports, are saved to the remote branch. After the testing session, the branch can be merged into the main branch through a merge request. This approach facilitates version control of tests and ensures traceability of test reports.

Environments

An environment refers to the network configuration that is used to connect to the controllers in the tests. Three separate environments can be created for each project.

Production refers to the final environment where the commissioning testing will take place.

Lab (Physical) refers to a development environment where the controllers are installed in a laboratory. For example, this configuration can be used for virtual commissioning with a Hardware-in-The-Loop (HIL) simulator.

Lab (Virtual) refers to a development environment where the controllers are simulated. For example, this configuration can be used for test script development.

Technologies and application design

SmartCo was developed using the technology stack of the Tools team in the Converter Controls Software section as well as with their assistance [1]. The architecture can be seen in Figure 4. SmartCo's backend is built on Python, utilising FastAPI [2]. The frontend was implemented using the JavaScript based Vue.js [3] framework, utilising the Tools Team's web application template and Javascript component library. The application operates within the Ethernet Technical Network, an internal network for operational devices at CERN. As the application is used by only a limited number of people, a GitLab repository was chosen as an alternative solution for a database for storing and managing the YAML based tests and JSON reports. Using Git provides built-in version control, while GitLab provides an API that is easily integrated into the backend and accessed from both the office network and the technical network. The application is deployed in a control's software Kubernetes [4] cluster that is maintained by the Tools team.

A diagram of the aforementioned infrastructure and how the different parts of it are interfaced can be seen in Figure 4.

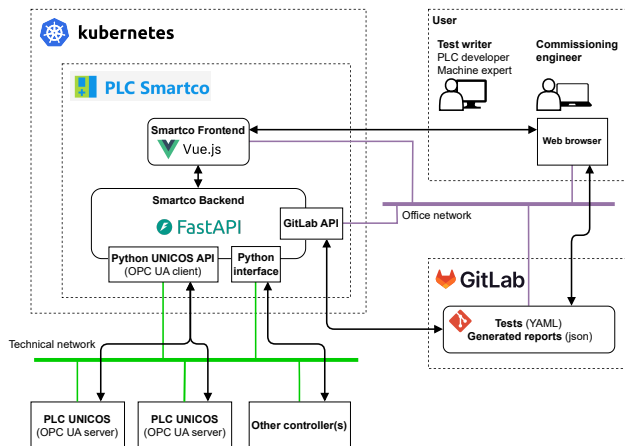


Figure 4: Software architecture of SmartCo.

Currently, SmartCo can interface only with PLCs running UNICOS-CPC [5] based applications, using a Python

interface built on Pypcltest. Pypcltest is a framework developed by CERN's central Industrial Control Systems section (BE-ICS). It provides an OPC UA interface specifically for UNICOS-CPC PLCs [6]. Expanding SmartCo's compatibility to support additional controller types requires:

1. Implementing a Python interface for communication with the new controller over the Ethernet Technical Network.
2. Incorporating the new controller's data model in the YAML parser.

Implementing support for Function Generator/Controllers (FGC) [7] is planned for more comprehensive testing of the consolidated POPS (Power for Proton Synchrotron) power converter [8], which integrates both a central PLC and distributed FGCs. Incorporating a new type of controller in SmartCo also serves as a foundation for future projects involving more complex and heterogeneous control systems.

Development Life Cycle

SmartCo adopted the development infrastructure and methodology of the Tools team, and their GitLab repository. When code is pushed to the repository, automated quality checks are performed through GitLab Continuous Integration (CI) pipelines. The frontend code is linted with ESLint [9] and tested with Cypress [10], while the backend code is linted with Ruff [11] and tested with pytest [12].

Both development and stable versions of SmartCo can be released at any stage of the development process. A deployment CI pipeline packages the software into a Docker container image, which is published to an internal image registry and mirrored to the Technical Network image registry. Once available in the registry, the image can be deployed by updating the configuration of the Tools team's Kubernetes cluster.

Using SmartCo for verifying changes in PLC programs of POPS-B

During the End of Year Technical Stop 24-25 (YETS), POPS-B (POWer for Proton Synchrotron Booster) [13] underwent changes to the alarms and interlocks related to a switch that connects the power supply to one of three distinct loads. SmartCo was used to verify the changes in production. A test for verifying functionality of dummy load permissions followed a sequence:

1. Trigger: Verify initial conditions.
2. Trigger: Remove the permission key for connecting the dummy load in order to start the test.
3. Sleep: Wait for the propagation of states in the system.
4. Assert: Verify correct reaction in the system.
5. Trigger: Re-insert the dummy load permission key and reset the system of any active alarms.

6. Sleep: Wait for the propagation of states in the system.
7. Assert: Verify that the initial conditions have been restored.

Figure 5 shows a partial view of the test report generated after the test, specifically steps 2, 3 and 4 that have passed.

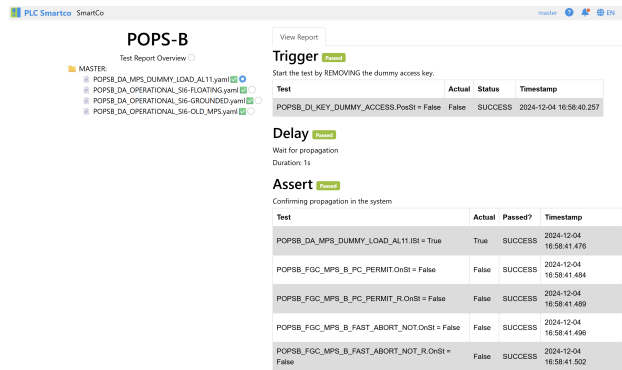


Figure 5: Partial rest report for the POPS-B dummy load permission alarm.

CONCLUSION

The concept was deemed useful for the team and power converter experts demonstrated interest in developing their own tests. SmartCo facilitates near end-to-end testing with increased reliability and reduced effort during the commissioning process. The tool will be used to support the Converter Controls Software section's commissioning process in the LS3 (Long Shutdown 3), where the team plans to commission systems including a new control system for Static VAR Compensators (SVC), the consolidation of POPS, among other systems.

ACKNOWLEDGEMENTS

The authors would like to thank the SY-EPC-HPC section at CERN for assisting in the testing of SmartCo on their machines, and the Tools Team of the SY-EPC-CCS section at CERN for assisting in the development of SmartCo.

REFERENCES

- [1] Tools Team, SY-EPC-CCS section website, CERN, "Tools and Technologies," [Online]. Available: <https://epc-ccs.web.cern.ch/content/tools>
- [2] S. Ramírez, "FastAPI," [Computer software]. [Online]. Available: <https://fastapi.tiangolo.com/>, 2018–2025.
- [3] E. You and the Vue.js Community, "Vue.js," [Computer software]. [Online]. Available: <https://vuejs.org/>, 2014–2025.
- [4] T. Oulevey, B. Copy, F. Locci, S. Page, C. Roderick, M. Eynden, and J. Martel, "An update on the CERN journey from bare metal to orchestrated containerization for controls," in *Proc. ICALEPCS'23*, Cape Town, South Africa, Feb. 2024, pp. 1138–1144. doi:10.18429/JACoW-ICALEPCS2023-TH2AO03
- [5] BE-ICS, CERN, "UNICOS CPC," [Online]. Available: <https://epc-ccs.web.cern.ch/content/tools>
- [6] B. Schofield, E. Blanco, and J. Borrego, "Continuous Integration for PLC-based Control System Development," in *Proc. ICALEPCS'21*, Shanghai, China, Oct. 2021, pp. 478–483. doi:10.18429/JACoW-ICALEPCS2021-TUPV035
- [7] M. Cejp, D. Zielinski, and R. Murillo Garcia, "Software architecture of FGC4, CERN's next-generation power converter control platform," in *Proc. IPAC'23*, Venice, Italy, May 2023, WEPM080. doi:10.18429/JACoW-IPAC2023-WEPM080
- [8] M. Marin, J. M. de Paco, O. Minaiev, and W. Niinimaeki, "A new PLC based control system to orchestrate the PS main power converters," Geneva, Switzerland, Sept. 2025, this conference.
- [9] N. C. Zakas and the ESLint Team, "ESLint," [Computer software]. [Online]. Available: <https://eslint.org/>, 2013–2025.
- [10] Cypress.io Contributors, "Cypress," [Computer software]. [Online]. Available: <https://www.cypress.io/>, 2017–2025.
- [11] C. Marsh, "Ruff," [Computer software]. [Online]. Available: <https://docs.astral.sh/ruff/>, 2022–2025.
- [12] H. Krekel and the pytest Development Team, "pytest," [Computer software]. [Online]. Available: <https://docs.pytest.org/>, 2004–2025.
- [13] SY-EPC-HPC section website, CERN, "Booster," [Online]. Available: <https://sy-dep-epc-hpc.web.cern.ch/machines/booster/general.stm>