ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-TUMG010

SOLARIS SYNCHROTRON CONTROL SYSTEM UPGRADE: ADDRESSING CHALLENGES AND IMPLEMENTING SOLUTIONS

M. Fałowski, M. Piekarski, I. Zadworny, M. Mleczko, M. Floras
NCPS SOLARIS, Kraków, Poland

Abstract

The National Synchrotron Radiation Centre SOLARIS, a 3rd Generation Synchrotron Light Source, stands as the most advanced research infrastructure in Poland. Since its commencement of operation in 2015, SOLARIS has undergone significant expansions. Initially, system upgrades were straightforward to implement. However, as the facility matured, new beamlines were created, and the number of equipment increased significantly. This led to a rise in the complexity of upgrades, prompting the SOLARIS team to focus on creating automation tools for deployments and maintaining up-to-date libraries and software. During this period, many versions of libraries, such as Python and PyQt, as well as the CentOS operating system, became obsolete, leading to increased maintenance costs. To address these challenges, a comprehensive strategy was developed. This strategy includes transitioning from CentOS 6 and 7 to AlmaLinux 9, upgrading older versions of Python to version 3.9, and updating automation tools such as Ansible and GitLab CI/CD. This paper presents the methodology employed for the control system upgrade, detailing the architecture of the new system, the upgrade process, and the challenges encountered.

INTRODUCTION

The control system at the National Synchrotron Radiation Centre SOLARIS has evolved significantly since its inception. Initially, system upgrades were straightforward, but the increasing number of beamlines and devices introduced complexity. This complexity, coupled with the obsolescence of software components such as Python 2, Python 3.6, PyQt4, and CentOS 6/7, necessitated a comprehensive modernization strategy. The goal was to ensure scalability, maintainability, and security while minimizing downtime and operational disruptions.

MOTIVATION FOR UPGRADE

The end of support for Python 2.7 and 3.6 and the growing incompatibility with modern tools prompted the migration to newer Python version. Additionally, the control system relied on outdated versions of Tango and Taurus, which hindered development and integration efforts. Separately, CentOS 7 was a significant source of hardware compatibility issues due to its aging platform and lack of support, especially for new drivers. Maintenance challenges were exacerbated by legacy code and the absence of automated testing, underlining the need for a more resilient, future-proof control system. These challenges underscored the need for a robust and future-proof control system architecture.

TECHNOLOGICAL STACK AND ARCHITECTURE

SOLARIS operates more than 270 virtual machines and 20 workstations running CentOS 6, CentOS 7, and Alma 9. The control system uses multiple versions of Python (2.7, 3.6, 3.9), PyQt4 and PyQt5 for graphical interfaces, Taurus (3.7 to 5.1), and Tango (mainly 9.2 and 9.3) for device management [1]. All system packaging was based entirely on the rpm format, ensuring consistency and compatibility across the diverse infrastructure.

Automation and deployment processes use GitLab CI/CD, Ansible, and AWX for configuration management and regular software updates. Kubernetes is applied for web application management to achieve flexibility and reliability.

System and service monitoring is conducted with Zabbix and CheckMK. FortiAnalyzer and Flowmon are used for network analysis and security alerts. These tools are selected to support system stability, ongoing maintenance, and protection against cyber threats and help in recognising control system downtimes.

UPGRADE STRATEGY

The upgrade process was a blend of strategic planning and timely adaptation. While major changes—such as workstation replacements—were scheduled for shutdown periods to maintain seamless operations, frequent failures of aging workstations led to earlier-than-planned migrations. Most notably, the operating system upgrade for workstations, shifting from CentOS 7 to Alma 9, was completed during this year's summer shutdown, addressing hardware compatibility issues and ensuring long-term support.

Similarly, migration to newer Tango versions was mapped out well in advance, with preparations starting long before the scheduled shutdown. For Taurus, PyQt, and Python, updates were executed together and, depending on the application, either during shutdowns or as needed throughout the year. The introduction of new features or substantial changes to existing software often provided opportunities for these upgrades, enabling smoother transitions to the latest versions. In addition, some applications were migrated from traditional GUIs to web-based interfaces, leveraging technologies such as Vue.js and Tango GraphQL to improve flexibility, maintainability, and user experience.

A significant benefit of graphical application upgrades and refactoring was the ability to transfer logic—such as state calculation—from the GUIs into Tango devices themselves. By relocating core logic out of the graphical layer,

ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-TUMG010

the GUI codebase became smaller and easier to maintain, while the extracted logic could now be reused by other applications, further enhancing the modularity and scalability of the control system.

REQUIRED PREPARATION

A critical aspect of the preparation phase involved updating both Ansible and AWX to newer versions. The legacy deployment tools lacked support for several essential functions and suffered from persistent bugs, making them unsuitable for managing a modernised infrastructure. The introduction of AWX Execution Environments (AWX EE) in the latest release was particularly transformative, as it enabled the use of multiple, isolated environments with different Ansible versions tailored to specific deployment needs. This flexibility proved vital for orchestrating complex upgrades and maintaining compatibility across diverse systems.

In parallel, it became necessary to build and package new versions of Tango and PyTango, alongside many other Python dependencies. These packages were not natively available in the standard RHEL repositories, requiring custom builds to ensure the control system could leverage the latest features and security updates. This preparatory work laid a robust foundation for the subsequent upgrade steps, mitigating risks associated with dependency management and ensuring all components would integrate seamlessly within the updated architecture.

ENCOUNTERED CHALLENGES

A variety of significant challenges emerged throughout the upgrade and maintenance process. Chief among these was a persistent shortage of skilled personnel, a problem amplified by the sheer scale of the infrastructure and the multitude of applications, hardware, and users it supports. Frequent rotation among staff meant that expertise, coding styles, and depth of knowledge varied greatly, making it difficult to transfer comprehensive understanding of programs, applications, and codebases. This inevitably contributed to issues with legacy applications—many of which languished without maintenance for extended periods.

The lack of well-defined standards, or the failure to rigorously enforce them in the system's early days, further complicated matters. Insufficient scheduled shutdowns forced upgrades and migrations to occur piecemeal, leaving several older software components unaddressed for years. As a result, the environment became fragmented, with numerous versions of libraries and dependencies coexisting, rather than being unified under a streamlined set of versions.

Migrating to newer versions of libraries occasionally led to dissatisfaction among users, especially when features were deprecated or modified. For example, the transition to newer releases of Taurus resulted in the removal of obsolete trends, altering both functionality and appearance in ways that users found frustrating. Meanwhile, the relentless pace of development for core technologies—such as Python and Tango—meant that stability was always a moving target. While Python 3.9 had already reached end-of-life, the effort required to rebuild and validate applications for Python 3.11 available on Alma9, as well as to switch to newer Tango releases (which introduced breaking ABI changes), proved too great to accomplish within restricted upgrade windows.

Package management introduced its own unique complications. The fpm tool, which was previously used to generate RPM packages from Python projects, relies primarily on the presence of a setup.py file and zipped source code. However, as many modern Python projects have transitioned to using pyproject.toml and a variety of new build backends instead of setuptools, fpm's ability to package them has become limited [2].

Additionally, as Python advanced through its versions, package versions evolved in tandem, resulting in the frequent necessity to track down the correct version of each package to satisfy shifting dependencies. This ongoing search for compatibility further complicated the already demanding upgrade process, forcing teams to continually rethink and adjust their packaging strategies.

In sum, the convergence of stretched human resources, fragmented standards, aging applications, rapid shifts in library versions, and the obsolescence of familiar tooling created a landscape where upgrades were an intricate balancing act—one that demanded constant adaptation and careful prioritization.

IMPLEMENTED SOLUTIONS

A cornerstone of the implemented solutions was the expanded adoption of automation for deployment tasks, harnessing the flexibility of AWX Execution Environments (EE) together with Ansible. This strategy enabled reliable orchestration of upgrades and configuration management across isolated, reproducible environments—minimising manual intervention and reducing the likelihood of human error. Additionally, software that had originally been developed in SOLARIS was simultaneously built and maintained for both CentOS 7 and Alma9 versions. This parallel build process ensured broader compatibility across operating systems and simplified the transition between legacy and modern platforms.

Equally vital was the collaborative approach to software refactoring and testing. Contributions from various departments—notably operators and scientists—proved invaluable. Their domain expertise and practical feedback facilitated the identification of edge cases and the refinement of both core functionality and usability, ensuring that system upgrades served actual user needs.

In tandem, there was a concerted push toward stricter standardisation of the codebase. This included the enforcement of type hinting and the integration of linters into development workflows, driving improvements in code quality, readability, and long-term maintainability. Package naming conventions were also standardised across both operating systems, further simplifying maintenance and reducing ambiguity during deployment. Alongside these efforts, a unified approach to the graphical user interface was introduced, establishing consistent standards for the appearance and behaviour of GUIs across applications. Such

ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-TUMG010

measures helped reduce technical debt, enhanced user experience, and streamlined onboarding for new team members.

Prior to each upgrade, comprehensive backups were created to safeguard against potential issues or the loss of critical features—ensuring there was always a path to recovery should unforeseen complications arise. Nonetheless, emphasis shifted toward proactive communication with users, encouraging them to accept that some missing or altered features—at least temporarily—were a necessary part of the process. Users were thoughtfully encouraged to acknowledge the realities of maintaining legacy software and operating systems, recognising the increasing risks and costs, and the point at which continued support simply becomes impractical. By fostering a sense of understanding and acceptance, the team helped set realistic expectations and paved the way for smoother transitions.

CONCLUSION

Upgrading operational infrastructure at facilities such as the SOLARIS synchrotron is a demanding, ongoing process. This work involves balancing legacy needs with future requirements, establishing and consistently following clear standards, and accepting some unavoidable sacrifices along the way.

Effective progress was achieved through rigorous communication and coordinated teamwork, ensuring that each phase of the upgrade was managed systematically and that all stakeholders remained informed and engaged throughout the process.

REFERENCES

[1] M. Piekarski and M. Fałowski, "SOLARIS Status Update", presented at 39th Tango Community Meeting, Giulianova, Italy, May 2025, unpublished. indico.tango-controls.org/event/422/contributions/908/.

[2] jordansissel /fpm, github.com/jordansissel/fpm/issues/2040