BENEFITS OF A MICROKERNEL ARCHITECTURE FOR SLOW CONTROL SYSTEMS SOFTWARE IN RESEARCH ACTIVITIES: CASE STUDIES FROM LHEP, JINR

A. Kozlovsky[a], A. Bukharin[b], D. Ivliev[b], O. Melnikova[b], A. Noskov[b], Y. Smirnova[b], I. Shirikov[a], I. Zhabin[a]

[a]*Joint Insitute for Nuclear Research, 141982 Dubna, Russia,*

e-mail: kozlovsky@jinr.ru

[b]*Dubna State University, 141983 Dubna, Russia*

Received November 20, 2025

Abstract – This paper presents an approach to software development for slow control systems. It reports the results of applying a microkernel architecture and justifies its choice based on Neal Ford's classification and an analysis of the most critical quality attributes. The work is devoted to Maestro Software and its application in projects for the Synchronization System Segment for the Booster Injector and the Nuclotron-Collider Beam Transport Channel. The application of SOLID principles and GoF design patterns is considered to facilitate the maintenance of slow control system software under conditions of constantly changing requirements. The developed software tools have been verified for both technical completeness and conceptual clarity. The paper describes the experience of involving undergraduate students from the Dubna State University in tackling actual software development tasks for physical setup equipment control as a part of the International Engineering School activity. The developed architectural rules and technical tools lower the entry barrier for young specialists and students participating in the development of slow control system software.

INTRODUCTION

The presented work originates from the practical needs of the Laboratory of High Energy Physics (LHEP) at the Joint Institute for Nuclear Research (JINR). Our team is responsible for the development and maintenance of a wide range of hardware and software deployed across the NICA Complex [1]. These include synchronization systems installed at KRION [2], HILAC [3-4] and SPI [5], as well as the recently developed power supply control system for the Nuclotron Collider Beam Transport Channel.

Although the hardware in these systems often differs significantly in purpose and functionality, many modules share a common unified platform developed by the engineers. We arrived at the conclusion that the software should be unified as well. This decision was motivated not only by the need to reduce redundant development efforts, but also by the complexity of modern hardware systems used in research environments. Experimental physics frequently requires specialized hardware tailored by staff engineers, and such hardware inevitably leads to the need for bespoke software solutions.

Research activities also impose additional constraints: both hardware and software must remain flexible to accommodate constantly changing requirements from end users.

The article explores the development of a unified software platform based on a microkernel architecture and demonstrates its applicability to slow control systems within the framework of NICA complex.

SLOW CONTROL SYSTEMS IN THE RESEARCH CONTEXT

Slow control encompasses processes in which response times are on the order of milliseconds or higher [6] and where deterministic timing is not the primary requirement. Nevertheless, such systems are essential in accelerator operations – for example, for configuring hardware, setting process parameters, monitoring equipment state, and handling diagnostic data.

At LHEP, slow control tasks constitute a considerable portion of our software development activities. That is why the idea of unifying the software development process for slow control systems across various subsystems of NICA deserves the attention.

BACKGROUND

Despite the fact of our initial orientation on synchronization system software development we decided to come up with a new approach that could be systematically applied not only to such kind of systems, but also to other ones in the future.

To develop such a platform, we began by identifying the most essential quality attributes for software in this context:

1. **Integration**. The system must be capable of integrating with other software modules and controlling different types of hardware [7].
2. **Evolutionary**. The software must be able to grow functionally [7].
3. **Simplicity**. Since the laboratory often involves students in ongoing projects, the platform should be easy to learn and develop upon, enabling rapid onboarding and distributed development [7].

According to the Neal Ford and Mark Richards classification [7], the microkernel architectural style has been chosen. Within this paradigm, each software plugin corresponds to a specific hardware module or group of modules, while the core system provides common functionality.

ARCHITECTURE AND IMPLEMENTATION

Architectural Overview

A component diagram of the platform is shown in Figure 1. Each plugin implementation is a dynamic library. A plugin encapsulates all logic required to interact with a specific

hardware module. On the client environment, plugins are to simply placed into a designated directory, from which they are loaded at runtime.

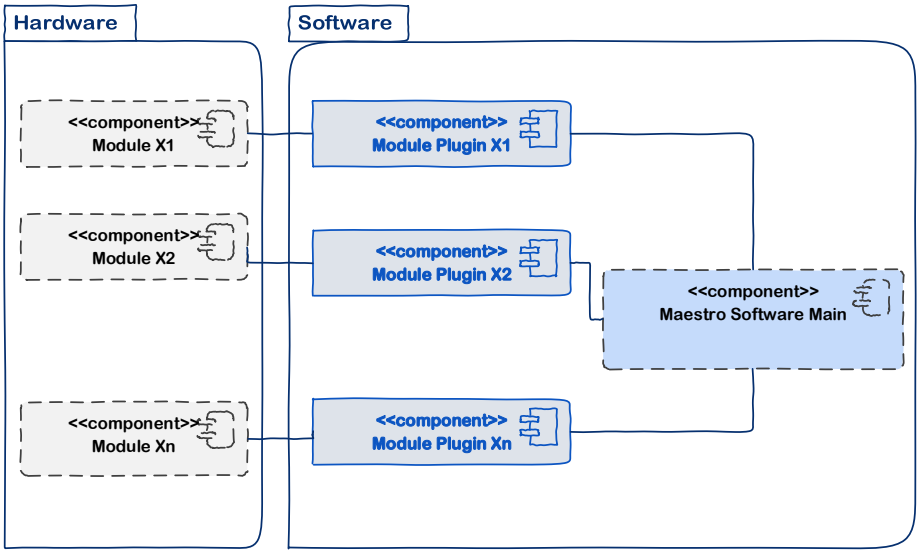

Figure 1. UML Component Diagram for the Software Platform architecture

The core system manages the plugin lifecycle, communication, configuration, and user interfaces. This design ensures that hardware-specific code remains isolated, while shared logic and UI elements remain reusable.

The platform is implemented in C++ using the Qt framework [8].

Development Toolkit

To support plugin development, we created a Software Development Kit (SDK) consisting of seven core libraries. As our team grew, maintaining these libraries became increasingly challenging. To address this issue, we deployed our self-hosted package manager conan [9].

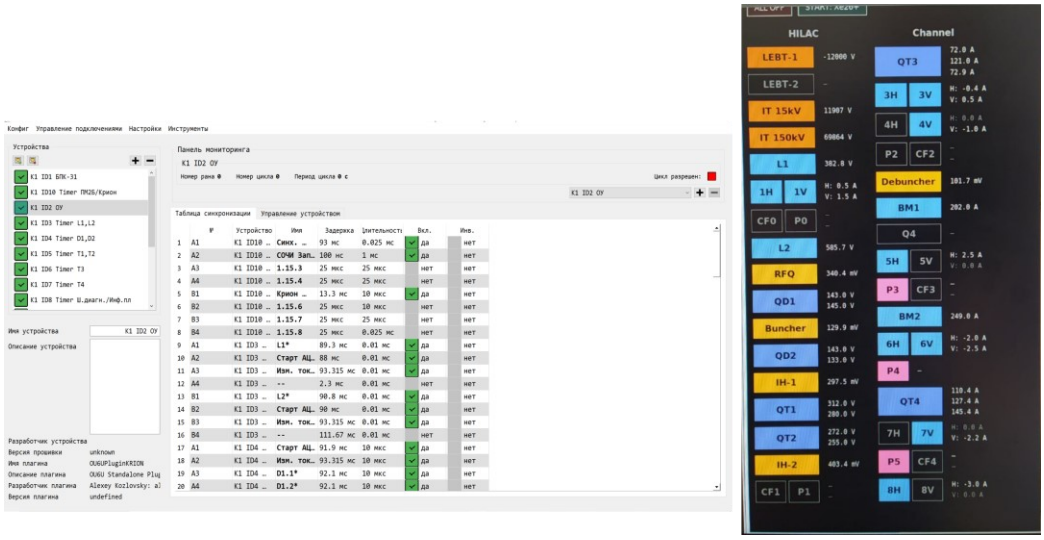

Figure 2. Maestro Software Tango Controls integration results

Integration with Tango Controls

While the platform initially supported standalone operation, we later integrated it with Tango Controls [10]. This integration expanded interoperability and made it possible to use our software as a part of larger distributed control environments (see Fig. 2)

EXTENDING THE PLATFORM BEYOND SYNCHRONIZATION SYSTEMS

Although the project began with synchronization systems, the platform has since proven capable of supporting other subsystems. For example, it has been successfully adapted for controlling the power supplies of the Nuclotron-Collider Beam Transport Channel (see Fig. 3).

These results demonstrate that the microkernel architecture indeed supports our evolutionary quality attribute.

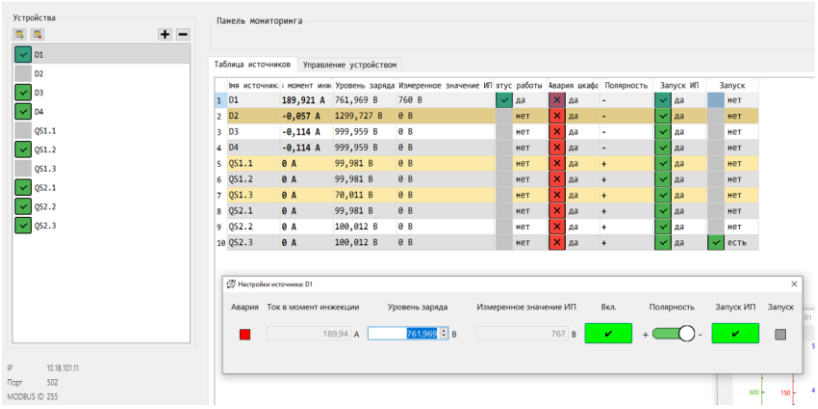

Figure 3. Maestro Software in Nuclotron-Collider Beam Transport Channel

EDUCATIONAL IMPACT AND SIMPLICITY OF DEVELOPMENT

One of the most satisfying outcomes of this project has been its impact on education. The School of Engineering [11] at Dubna State University has played an important role in involving students in laboratory software development activities.

This architectural clarity made it possible to delegate the development tasks – including plugin implementation – to students. This confirms that the simplicity attribute has been achieved in practice.

CONCLUSION

We have developed and deployed a unified software platform for both client (of distributive or client-server software) and standalone slow control system applications at various parts of the NICA Complex. It provides a flexible and extensible foundation for

hardware control tasks. It has been integrated with Tango Controls, has demonstrated applicability across multiple hardware subsystems, and has proven its ability to grow functionally without sacrificing maintainability.

Moreover, the platform has enabled successful involvement of students in real development activities, validating the simplicity and educational value of the architectural approach.

Taken together, these results show that the microkernel architecture offers clear benefits for software in slow control systems within research environments, particularly in terms of integration, evolution, and accessibility for new developers.

REFERENCES

[1] - Kekelidze, V., et al. “The NICA project at JINR Dubna.” EPJ Web of Conferences. Vol. 71. EDP Sciences, 2014.

[2] - Rassadov, D. N., et al. “Prospects for Using the Source of Krion-6T Multicharged Ions on the NICA Injection Complex: The Multiple Injection of Heavy Element Ions.” Physics of Particles and Nuclei Letters 21.3 (2024): 236-240.

[3] - Trubnikov, Grigory. “Heavy ion collider facility NICA at JINR (Dubna): status and development.” 36th International Conference on High Energy Physics. Vol. 174. SISSA Medialab, 2013.

[4] - Butenko, Andrey Valerievich, et al. “Development of NICA injection complex.” Proc. IPAC’14 (2014): 2103-2105.

[5] - Belov, A. S., et al. “Source of polarized ions for the JINR accelerator complex.” Journal of Physics: Conference Series. Vol. 938. No. 1. IOP Publishing, 2017.

[6] - Aprile, E., et al. "The distributed Slow Control System of the XENON100 experiment." Journal of Instrumentation 7.12 (2012): T12001.

[7] - Richards, Mark, and Neal Ford. Fundamentals of software architecture: an engineering approach. O'Reilly Media, 2020.

[8] - “Qt Widgets 6.7.3.” Doc.qt.io, doc.qt.io/qt-6/qtwidgets-index.html.

[9] - “Conan 2.0: C and C++ Open Source Package Manager.” Conan.io, 2024, conan.io/.

[10] - “Overview of Tango Controls — Tango Controls 9.3.4 Documentation.” Readthedocs.io, 2022, tango-controls.readthedocs.io/en/9.3.5/overview/overview.html. Accessed 20 Nov. 2025.

[11] - “Международная Инженерная Школа Университета Дубна.” Uni-Dubna.ru, 2018, ingenerka.uni-dubna.ru/. Accessed 20 Nov. 2025.