ISBN: 978-3-95450-248-6 ISSN: 2673-5490 doi: 10.18429/JACoW-IPAC2025-THPS050

DEVELOPMENT OF A FLEXIBLE DIGITAL TWIN FRAMEWORK FOR ACCELERATORS USING DESIGN PATTERNS

W. Sulaiman Khail*, P. Schnizer,
Helmholtz-Zentrum Berlin für Materialien und Energie Gmbh, Berlin, Germany

Abstract

Modern accelerator design increasingly relies on prototyping and validating commissioning software through digital twins. Digital twins serve as natural test benches for validating and monitoring the required physics software stack. These twins must align with the current design state of the accelerator from the project's inception to the machine's commissioning. The authors have developed a modern digital twin framework based on software design patterns. Its architecture emphasizes clean design principles with minimal coupling between components. Its setup requires only lattice and device configuration data. Thanks to its design, it seamlessly integrates into prototyping environments or control system infrastructures. In this paper, we briefly describe the design patterns underlying this architecture, highlight the flexibility and advantages of the infrastructure, and outline the steps needed to implement it for a machine currently lacking a digital twin.

INTRODUCTION

Particle accelerators are intricate systems composed of hundreds of tightly coupled devices. Steering and optimizing these machines during design and commissioning phases requires not only physical understanding but also sophisticated software infrastructure. Recent advancements have introduced *digital twins*—virtual representations of these machines that are synchronized with the real-world system and enable predictive analysis, configuration testing, and operational insight.

Digital twins for accelerators must manage real-time data exchange, maintain synchrony between models and hardware, and remain adaptable to evolving hardware configurations. To tackle these challenges, we have developed a modular, pattern-based digital twin framework. This framework decouples simulation logic, control system integration, and state management using software design patterns. By leveraging patterns such as *Simulation System Facade*, *Simulation System Update*, and *Twin State Synchronization* [1], as well as more recent work including the *Facilitator/Liaison Pattern*, *Object Translation Pattern*, *Measurement Plan Pattern*, and *Command Execution Pattern*, our framework achieves a high degree of flexibility and reuse across facilities.

This paper describes the core design decisions and patterns behind the framework and outlines how they were applied to deploy digital twins for synchrotron light source facility. The remainder of this paper is structured as follows: Section describes the core architectural patterns of the framework. Section outlines deployment across different facilities, and Section summarizes lessons learned and future directions.

PATTERN-DRIVEN ARCHITECTURE OF THE DIGITAL TWIN FRAMEWORK

Our digital twin framework adopts a layered architecture that separates simulation core logic from facility-specific control interfaces. The simulation core is implemented using the *dependency injection pattern* [2], which decouples engine instantiation from usage and allows the dynamic selection of physics engines such as *PyAT* [3], *MAD-NG* [4], or *THOR-SCsi* [5]. This enables the accelerator model to remain independent of the chosen backend and easily switchable during deployment or testing.

Facility-specific control system logic is separated into a customizable extension layer. This layer supports both *EPICS* [6] and *TANGO* [7] infrastructures and handles initialization of process variables (PVs) in EPICS or device attributes in TANGO. This modular separation provides the flexibility needed to support heterogeneous facilities using a common software foundation.

At the heart of this architecture is a set of reusable software design patterns that structure how simulation data, control commands, and physical measurements are exchanged and maintained.

Simulation System Facade This pattern abstracts the complexity of the virtual accelerator and exposes a unified interface for interacting with device models and simulation backends. It hides control-specific details and provides a consistent setup mechanism regardless of the underlying infrastructure.

In our EPICS-based implementation, the facade initializes PVs for each magnet and power converter, including properties such as magnetic field strength, current, and position setpoints. Each PV is linked to an update handler, enabling automatic propagation of changes to the simulation backend whenever a device parameter is modified.

The TANGO implementation follows the same architectural principles. Each power converter is represented as a device server exposing attributes like current, voltage, and status. These attributes are initialized at runtime using calibration data and device associations. They remain synchronized with the simulation core, enabling bidirectional interaction between the control system and the virtual machine.

Both implementations encapsulate simulation initialization and decouple control-specific concerns, supporting easy reuse and modular adaptation to new facilities.

* waheedullah.sulaiman_khail@helmholtz-berlin.de

ISBN: 978-3-95450-248-6 ISSN: 2673-5490 doi: 10.18429/JACoW-IPAC2025-THPS050

Simulation System Update The *Simulation System Update* pattern governs how updates to device parameters—such as power supply currents or magnet strengths—are propagated through the system. Each PV or device attribute is connected to a change handler that triggers recalculation of affected lattice elements and propagates simulation results back to the views. This event-driven design ensures tight synchronization between user inputs, hardware models, and computed outputs.

Twin State Synchronization The digital twin supports three operational modes: *model*, *shadow*, and *twin*. This pattern enables controlled switching between these modes using a centralized state controller. It ensures that updates to the twin can either remain internal (model), follow physical measurements passively (shadow), or actively drive and synchronize with the hardware (twin).

Liaising/Facilitator and Object Translation To bridge the gap between engineering units and physical quantities, we employ a liaison-translation mechanism. The Facilitator maintains associations between simulation components and their hardware counterparts. The Translation performs unit conversions using calibration data—e.g., translating between current and magnetic field strength—ensuring consistency between the control layer and simulation engine.

Command and Measurement Execution Measurement routines are expressed as serialized command plans. These are interpreted and executed by the Command Execution Engine, which coordinates hardware interaction, logging, and synchronization with detectors. Command and Measurement Execution Engines extend this pattern by coupling each step with simulation feedback, enabling detailed comparison between measured and expected behavior. Plans can be rewritten at runtime, logged, and reused, facilitating reproducible diagnostics.

For example, an orbit response measurement plan may consist of sequential steerer excitations followed by BPM readouts, with each step linked to both hardware execution and simulation validation.

Together, these patterns define a coherent architectural language that supports scalability, testability, and adaptability of digital twins across different facilities and simulation infrastructures.

Architectural Layers and Interaction Flow

The architecture is structured into three main layers: the application layer, the middle layer, and the backend layer.

Figure 1 illustrates this design. The application layer includes user-facing services, such as Orbit Response Measurement (ORM), LOCO, or Beam-Based Alignment (BBA) and many other high level user applications. The middle layer orchestrates measurement logic, update propagation, state management, and pattern execution. The backend comprises interchangeable physics engines (e.g., PyAT, THOR-SCsi, MAD-NG), the simulation model built from lattice data, and control system interfaces.

This modular organization enables clean deployment, simplifies facility onboarding, and supports consistent behavior across various control system protocols.

APPLICATION OF PATTERNS IN ACCELERATOR DEPLOYMENT

The pattern-based digital twin framework has been successfully instantiated and validated using lattice data at BESSY II and the Metrology Light Source (MLS). These deployments demonstrate how a clear architectural structure enables reuse and adaptability across different operational environments.

From Static to Configurable Twins

The original digital twin prototype was tightly coupled to a static instance of the BESSY II lattice and simulation engine. This made adaptation to other facilities cumbersome and error-prone. Through architectural refactoring using the patterns described in Section 2, we transformed the system into a reusable, configurable framework.

Using the *Simulation System Facade*, the internal logic was decoupled from the control interfaces, allowing both EPICS and TANGO bindings to be introduced modularly. The *Dependency Injection* approach enables selection between simulation engines such as PyAT, thor-scsi-lib, or MAD-NG at runtime without modifying core logic.

Portability and Deployment

Finally, the entire system is packaged using container-based deployment. Thanks to the modular architecture, site-specific configuration files are sufficient to instantiate a complete digital twin, dramatically reducing setup overhead for new facilities or experimental contexts.

The architectural modularity of our digital twin framework is also reflected in its high-level code organization. Figure 2 illustrates the top-level folder structure, which highlights the clear separation between core functionality and platform-specific implementations.

The `core` directory contains simulation logic, model definitions, orchestration patterns, and backend engine integration. Platform-specific bindings to control systems such as EPICS or TANGO are isolated in dedicated modules like `custom_epics` and `custom_tango`, enabling clean substitution or extension for different facilities.

This structure ensures that extending the framework to a new site or swapping out simulation backends requires minimal impact on the rest of the codebase. [1] We have also provided support for container-based deployment, as facility-specific configurations and interfaces are modularized.

CONCLUSION AND OUTLOOK

We have presented a flexible and reusable digital twin framework for particle accelerators built around modern

[1] project repository: https://github.com/hz-b/dt4acc

ISBN: 978-3-95450-248-6 ISSN: 2673-5490 doi: 10.18429/JACoW-IPAC2025-THPS050

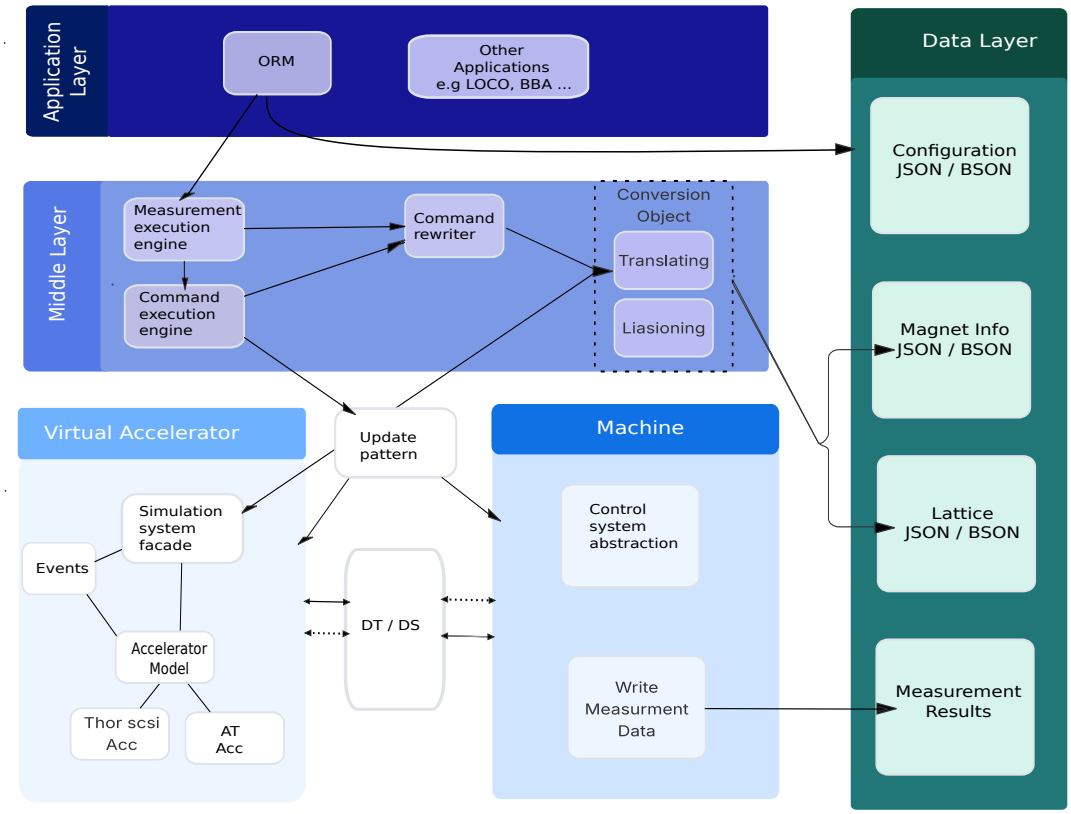

Figure 1: Layer architecture of the digital twin framework.

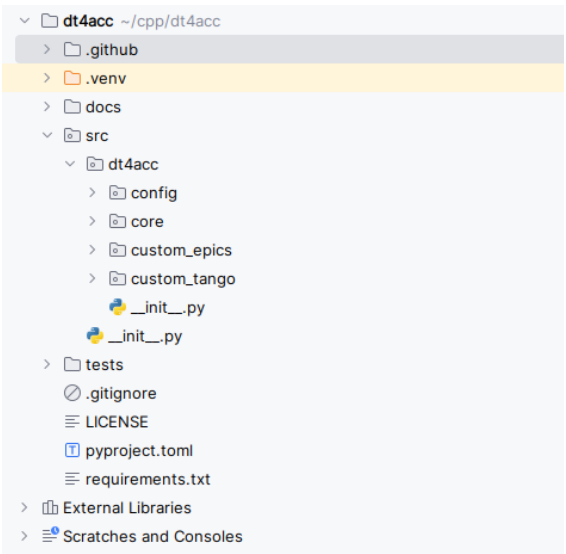

Figure 2: Top-level project structure illustrating modular separation of core logic and control-system-specific layers.

software design patterns. The architecture enables clean separation of concerns, supports multiple simulation engines, and facilitates deployment across different control system infrastructures. Ongoing work includes extending pattern usage to measurement plans, measurement execution and command execution.

REFERENCES

[1] W. Sulaiman Khail and P. Schnizer, "Patterns in digital twin development", in *Proceedings of the 29th European Conference on Pattern Languages of Programs, People, and Practices*, 2024. doi:10.1145/3698322.3698325

[2] J. Vlissides, R. Helm, R. Johnson, and E. Gamma, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[3] W. A. H. Rogers, N. Carmignani, L. Farvacque, and B. Nash, "pyAT: A python build of accelerator toolbox", in *Proc. IPAC'17*, Copenhagen, Denmark, May 2017, pp. 3855–3857. doi:10.18429/JACoW-IPAC2017-THPAB060

[4] L. Deniau, "MAD-NG, a standalone multiplatform tool for linear and non-linear optics design and optimisation", *arXiv:2412.16006*, 2024. doi:10.48550/arXiv.2412.16006

[5] P. Schnizer, W. S. Khail, J. Bengtsson, M. Ries, and L. Deniau, "Progress on Thor SCSI development", in *Proc. IPAC'23*, Venice, Italy, pp. 3413–3416, 2023. doi:10.18429/JACoW-IPAC2023-WEPL127

[6] L. R. Dalesio, M. R. Kraimer, and A. J. Kozubal, "EPICS architecture", in *Proc. ICALEPCS'91*, Tsukuba, Japan, pp. 278–282, 1991. doi:10.18429/JACoW-ICALEPCS1991-S09IC03

[7] J.-M. Chaize *et al.*, "The ESRF TANGO control system status", *arXiv:cs/0111028*, 2001. doi:10.48550/arXiv.cs/0111028