Electronic Communications of the EASST
Volume 85 Year 2025

deRSE25 - Selected Contributions of the 5th Conference for Research Software Engineering in Germany

Edited by: René Caspart, Florian Goth, Oliver Karras, Jan Linxweiler, Florian Thiery, Joachim Wuttke

Building Digital Twins for Particle Accelerators: The Role of Architecture and Patterns

Waheedullah Sulaiman Khail, Pierre Schnizer

DOI: 10.14279/eceasst.v85.2707

Electronic Communications of the EASST (https://eceasst.org).
Published by **Berlin Universities Publishing**
(https://www.berlin-universities-publishing.de/)

Building Digital Twins for Particle Accelerators: The Role of Architecture and Patterns

Waheedullah Sulaiman Khail, Pierre Schnizer

Helmholtz-Zentrum Berlin für Materialien und Energie GmbH

Abstract: Particle accelerators are complex machines consisting of hundreds of interdependent devices. Control systems and commissioning applications are used to steer, control, and optimize them. Online models—now increasingly implemented as digital twins—enable real-time derivation of critical beam parameters during operation.

This paper presents our experience building reusable and configurable digital twins for particle accelerators using architectural layering and software design patterns. By separating domain logic from control infrastructure and applying both established and custom patterns, we have developed a flexible twin framework capable of supporting multiple backends, simulation engines, and control systems.

We demonstrate how this architecture has enabled rapid deployment across two accelerator facilities—BESSY II and MLS—both operated by Helmholtz-Zentrum Berlin, where our team develops and maintains the digital twin framework for operational modeling and commissioning support, while also improving testing and packaging through containerized delivery. Our findings highlight that intentional architecture and pattern-based design are essential for sustainable, adaptable scientific software in complex engineering domains.

Keywords: Digital Twin, Particle Accelerator, Design Patterns, Software Architecture

1 Introduction

Particle accelerators are complex scientific devices consisting of hundreds of interdependent components such as magnets, radio frequency (RF) cavities, and diagnostics operating in tightly synchronized feedback loops [PR14]. They are used worldwide for applications ranging from fundamental physics research to materials science, medicine, and industry. Operating and optimizing such systems requires precise configuration and control, continuous monitoring, and adaptive software-supported feedback to maintain stability, reliability, and safety.

Digital twins, virtual representations of physical systems that remain continuously synchronized with their real-world counterparts, have emerged as a powerful tool to improve operational efficiency, enable predictive maintenance, and accelerate commissioning. As described in recent literature and industrial applications [Mat24, Gel91], a digital twin does more than simulate behavior: it maintains a live link to the physical asset, enabling diagnostic, prognostic, and predictive functions. In the context of predictive maintenance, digital twins can detect performance drift, estimate remaining useful life, and schedule interventions before

critical failures occur. These capabilities depend on the ability to synchronize digital models with physical measurements through a bi-directional flow of data.

Implementing an effective digital twin requires an ecosystem of interoperable components: physics-based models, data-driven analytics, control interfaces, and execution engines. This complexity makes software architecture a decisive factor in long-term success. A well-structured architecture formalizes component roles, clarifies dependencies, and supports adaptation across facilities. Without a proper architecture, systems quickly become brittle, making reuse and evolution difficult.

In this context, design patterns offer proven, reusable solutions for structuring complex systems, making them an effective means to realize and maintain such an architecture. Introduced by the *Gang of Four* (GoF) [VHJG95], they are reusable solutions to recurring design problems. They offer a shared vocabulary for structuring systems and addressing cross-cutting concerns such as modularity, coupling, and dependency management [Fow02, BHS07]. Our earlier work [SS24] presented several domain-specific patterns for accelerator digital twins; in this paper, we focus on their integration into a coherent, facility-independent architecture. Patterns have a rich history: originally formalized in the field of architecture by Christopher Alexander [Ale77], they have since been applied to domains as varied as organizational behavior [HC04], user interface design, and algorithmic strategies [Ski08]. In software architecture, they can be combined into a *pattern language*, a coherent set of interrelated solutions that guide the construction of robust, adaptable systems.

In this paper, we show how integrating architectural thinking with design pattern-based strategies can significantly enhance the sustainability, extensibility, and operability of digital twins for particle accelerators. Rather than proposing new methodology or mining additional patterns (i.e., identifying existing design patterns in software [DZP09]), our focus is on the *application* of proven architectural and design solutions—both established and domain-specific—to construct a coherent, facility-independent architecture. This perspective highlights how layered architectures, standardized interfaces, and well-defined mappings between abstract and physical domains support sustainable development, while patterns enable modularity and reuse across multiple facilities.

We validate our approach using two accelerator facilities, BESSY II and the Metrology Light Source (MLS), which serve as test benches to demonstrate portability and adaptability. The remainder of this paper is structured as follows. Section 2 discusses the current lack of architectural thinking in many scientific software projects and motivates the need for disciplined design in the context of digital twins. Section 3 presents the foundational architecture of our digital twin system, highlighting its modular layering and separation of concerns. Section 4 describes the architectural evolution from a static, facility-specific implementation to a reusable, pattern-driven framework, detailing the design patterns employed and their interactions. Finally, Section 5 concludes with reflections on the impact of these architectural decisions and outlines directions for future work.

2 The Missing Architectural Lens in Scientific Software

Software architecture is often defined not merely by structural components, but by the critical decisions that are hardest to change [Fow02]. Martin Fowler emphasizes that architecture comprises those choices which fundamentally shape a system's flexibility, scalability, and resilience. Despite its importance, many scientific software projects evolve without a clearly defined architectural vision, resulting in systems that are difficult to extend, test, or integrate.

In domains such as accelerator physics, the focus is traditionally placed on achieving scientific outcomes rather than long-term software sustainability. Consequently, systems often emerge through rapid prototyping, hardcoded logic, and facility-specific assumptions. Over time, these characteristics lead to significant technical debt. Without architectural separation of concerns, even seemingly minor adaptations—such as switching simulation backends or modifying measurement logic—can require disproportionate effort. This not only slows development but also hinders collaboration, reproducibility, and reuse.

In modern accelerator facilities, device control and data acquisition are typically managed through dedicated control system frameworks. Examples include EPICS (Experimental Physics and Industrial Control System) [DKK], deployed at Helmholtz-Zentrum Berlin (HZB) for facilities such as BESSY II and the Metrology Light Source (MLS), and TANGO Controls [CGK+01], used at the European Synchrotron Radiation Facility (ESRF) and other laboratories. These toolkits provide standardized mechanisms for device communication, configuration, and monitoring, but differ significantly in their protocols, APIs, and deployment models. Other control system frameworks also exist, and the diversity of these environments means that a reusable digital twin framework needs to be able to integrate with at least these two major paradigms while remaining adaptable to others.

Our experience in evolving from an ad-hoc, lattice-specific digital twin [Cor94] (where a *lattice* refers to the ordered sequence of magnets and beamline elements defining the particle beam path and focusing) to a configurable, architecture-driven system highlights the value of making architectural decisions explicit. Applying design patterns has allowed our framework to scale across facilities, simulation engines, and operational contexts. As software becomes increasingly central to accelerator operation and automation, the adoption of professional architectural practices in scientific codebases transitions from best practice to necessity.

Beyond individual design patterns, the concept of a pattern language articulated in Buschmann et al.'s *Pattern-Oriented Software Architecture, Volume 5* (POSA5) [BHS07] offers a systematic approach for informing architectural decisions throughout the software lifecycle. Rather than applying isolated patterns ad hoc, a pattern language ensures coherence and contextual relevance, aligning multiple patterns toward architectural goals such as modifiability, scalability, and testability. Just as individual words convey meaning but gain expressive power when combined into sentences, patterns in sequences (being applied in combination) can address broader and more complex design challenges than any single pattern could alone [SV17]. In our digital twin framework, this thinking influenced the composition of Facade, Repository, and Update patterns to support runtime configurability and device abstraction. Such a language-oriented approach is especially crucial in scientific domains where evolving requirements often outpace formal development processes.

A common challenge in scientific software development is the tendency to prioritize

immediate experimental outcomes over long-term architectural sustainability. Numerous studies have noted that such systems often suffer from tight coupling, monolithic designs, and insufficient modularization [WZX+23, SLV+24]. This makes them difficult to test, extend, or integrate with emerging technologies. In many cases, critical functionality becomes tied to specific hardware configurations or legacy dependencies, limiting portability and reuse. These limitations are especially detrimental in evolving domains such as accelerator operations, where new measurement strategies and control workflows frequently arise. Without architectural foresight, adapting scientific software to such changes can become prohibitively expensive and error-prone.

3 Architectural Layering and Modularity

The architecture of our digital twin framework follows a clear separation between domain logic, integration mechanisms, and facility-specific extensions. At its center lies a simulation-agnostic representation of the accelerator, supporting interchangeable physics engines such as PyAT [RCFN17], thor-scsi-lib [SSB+23], or MAD-NG [Den24]. Backend selection occurs at runtime, allowing the same framework to adapt to different operational contexts without code changes.

Facility-specific control system interactions—such as those required for EPICS or TANGO environments—are encapsulated in dedicated extension layers. By keeping these concerns separate from the core logic, the framework can be reused across facilities, deployed in simulation-only contexts, or adapted to new control paradigms with minimal effort. This modularity also improves testability and supports the use of containerized environments for development and validation.

The system is organized into five conceptual layers, shown in Figure 1:

- **Application Layer** – user-specific applications and scripts for commissioning, pre-processing, or analysis.
- **Middle Layer** – orchestration components for measurement execution, liaison/translation, and high-level command handling. Pattern mining for this layer is ongoing, as it holds key domain-specific coordination logic.
- **Core Layer** – the simulation-agnostic accelerator model and its orchestration logic.
- **Integration Layer** – interfaces to simulation engines and control systems.
- **Data Layer** – persistent lattice and device configuration storage.

Within each layer, suitable architectural and design patterns are applied to promote loose coupling, runtime configurability, and maintainability. For example, the *Integration Layer* employs the Simulation System Facade to isolate control-system specifics, while the *Core Layer* uses the Update and Twin State Synchronization patterns to ensure that the digital object and the real world object stay properly synchronized. These and other patterns are discussed in detail in Section 4.

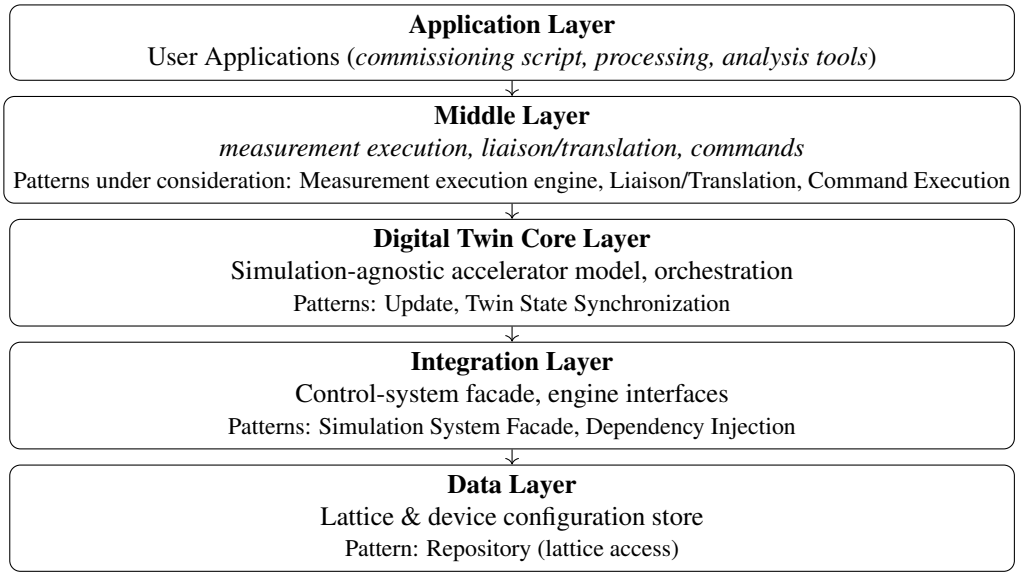

Figure 1: Layered, abstract view of the digital twin framework. Section 4 expands these layers into a concrete architecture, showing different components (Fig. 2).

The overall behavior is event-driven: changes initiated at the Application Layer propagate downward through the layers to update the model and, when required, trigger simulation runs. Results then flow upward toward the user through the configured integration and presentation pathways. This separation of concerns ensures that changes in one layer—such as replacing a simulation backend or modifying the data source—do not ripple unnecessarily through the rest of the system.

4 Architecting Reusable Digital Twins Using Design Patterns

While Section 3 described the layered structure of our framework, here we focus on how specific design patterns—both established and domain-specific—enable this structure to be reusable, configurable, and sustainable across facilities.

In the early stages of our work, the digital twin was implemented for a fixed instance of the BESSY II lattice. The system tightly coupled simulation logic, device setup, and measurement routines to this single facility configuration. Any extension to new hardware, beamlines, or calculation engines required code modifications and extensive manual reconfiguration. While functional, this "static twin" architecture lacked portability and scalability.

This limitation prompted a shift from implementation-first thinking to architecture-first design. The static twin was refactored into a modular framework guided by architectural principles and implemented through a pattern language. The result is a facility-independent, simulation-agnostic core with well-defined extension points for control systems, simulation engines, and measurement workflows.

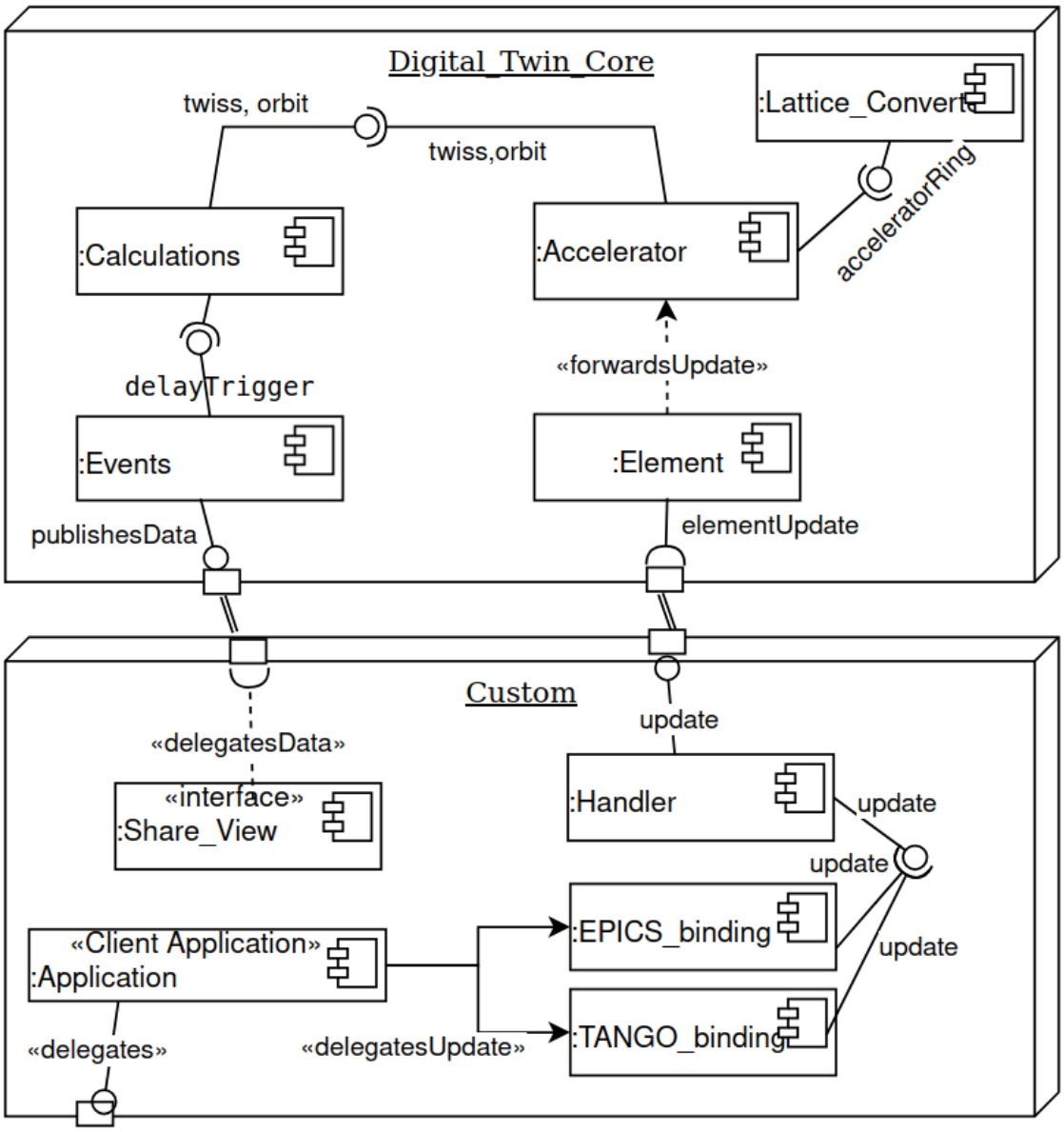

Figure 2: Component-level architecture of the digital twin framework, showing how pattern-derived components integrate across the layered structure

Established software patterns such as Dependency Injection [SV19], Model–View–Controller (MVC), and Repository [Fow02] provide the backbone for runtime flexibility and testability. MVC separates simulation logic (model), orchestration and control (controller), and user-facing interfaces (view), allowing each to evolve independently. Dependency Injection supports dynamic binding to different simulation backends—such as PyAT [RCFN17], thor-scsi-lib [SSB+23], or MAD-NG [Den24]—without architectural rewrites. The Repository

pattern decouples configuration and device state management from the computational core, enabling central validation and streamlined data flow.

In addition to these established solutions, we mined domain-specific patterns tailored to accelerator digital twins [SS24]. The *Simulation System Facade* hides setting up a virtual accelerator and its control-system specifics behind a unified interface. The *Update Pattern* structures consistent propagation of parameter changes through middle layer into the model and back to outputs. The *Twin State Synchronization* pattern manages transitions between *model*, *shadow*, and *live* states, ensuring operational safety during commissioning and live operation. The resulting architecture integrates both established and domain-specific patterns into a cohesive component framework, as depicted in Figure 2.

Figure 2 expands the abstract layering shown in Figure 1 into a concrete component-level architecture. Rather than depicting the patterns themselves, it illustrates the principal components and their relationships as shaped by the applied design patterns. For instance, the Accelerator Model embodies the Facade and Repository concepts, the Update Manager (update, elementUpdate, forwardUpdate) realizes the coordination logic introduced by the Update pattern, and the Virtual Accelerator reflects the synchronization principles of the Twin State Synchronization pattern. Together, these components form a cohesive structure that isolates control-specific logic, supports interchangeable backends, and maintains consistency between the virtual and physical accelerator systems.

Rather than representing data flow, the diagram emphasizes structural relationships and the placement of patterns within the architecture. These patterns do not operate in isolation. Together they form a *pattern language* in the sense of POSA5 [BHS07], where patterns are applied in combination to express an architectural intent rather than as disconnected solutions. This language supports configurability, testability, runtime substitution, and operational clarity—qualities that are essential in the accelerator domain.

The current framework follows an event-driven architectural style. Input changes—such as adjustments to a power converter current—trigger a chain of updates through liaison and translation layers, core model computation, and propagation of updated results back to the control interface. This modular event flow supports real-time responsiveness, measurement automation, and facility-specific workflows without affecting the core architecture.

The architectural transition is summarized in Table 1, highlighting how previously hardcoded, facility-bound logic was replaced with a reusable, testable, and facility-neutral design.

Importantly, this architectural clarity has improved both our development velocity and system transparency. Adding new measurement workflows, adapting to changes in control protocols, or switching simulation engines are now routine operations within a stable architectural shell. The experience reinforces a core tenet of software architecture: good structure enables change, while poor structure resists it.

5 Conclusion and Outlook

In this paper, we have shown how applying design patterns within a layered architectural framework enables the development of reusable, extensible digital twins for particle accelerators. Moving from a facility-specific, static implementation to a configurable, pattern-driven

Table 1: From Static Twin to Reusable Architecture: Key Transitions

Static Digital Twin (Initial)	**Reusable Digital Twin (Current)**
Hardcoded for a specific BESSY II lattice	Configurable for any facility with lattice and device data
Manual setup of PVs and control interfaces	Modularized control layer with plug-in extension support
Simulation tightly coupled to core logic	Dependency-injected simulation backends (e.g., PyAT, thor-scsi-lib)
Custom logic embedded per device	Unified handling via Update and Facade patterns
No separation between core and control logic	Layered architecture with clear boundaries
Difficult to scale, extend, or test	Reusable, testable, and facility-neutral design

architecture allowed us to support multiple backends, control systems, and operational workflows without code duplication or architectural erosion.

By combining established patterns—such as MVC, Dependency Injection, and Repository—with domain-specific ones like the Simulation System Facade, Update, and Twin State Synchronization, we created a flexible architecture that responds well to the unique demands of accelerator operations. This architecture supports real-time event-driven updates, modular composition, and clean separation of core simulation from control interfaces.

The resulting system has been deployed successfully for both BESSY II and MLS facilities, requiring only changes to configuration and machine-specific data. Its architectural modularity also simplifies testing and deployment: we provide containerized builds for internal users, eliminating the need for full application setup. This container-first approach will also support external collaborators, enabling lightweight usage and testing across institutions.

Future work includes extending this pattern language into operational tooling, generalizing measurement execution workflows, and abstracting liaison layers for physics-to-hardware transformations. We also aim to validate our architectural choices across a broader range of facilities, further refining the modularity, portability, and lifecycle support of digital twins in scientific infrastructure.

Our experience reaffirms that in complex scientific software, architecture is not a luxury—it is an enabler of adaptability, collaboration, and longevity. This paper demonstrates that the disciplined *application of proven patterns*, rather than ad-hoc implementations, is key to building sustainable scientific software in complex domains.

Bibliography

[Ale77] C. Alexander. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, 1977.

[BHS07] F. Buschmann, K. Henney, D. C. Schmidt. *Pattern-Oriented Software Architecture, Volume 5: On Patterns and Pattern Languages*. Wiley, 2007.

[CGK+01] J.-M. Chaize, A. Goetz, W.-D. Klotz, J. Meyer, M. Perez, E. Taurel, P. Verdier. The ESRF TANGO control system status. *arXiv preprint cs/0111028*, 2001. https://arxiv.org/abs/cs/0111028

[Cor94] M. Cornacchia. Lattices for synchrotron radiation sources. 1994.

[Den24] L. Deniau. MAD-NG, a Standalone Multiplatform Tool for Linear and Non-linear Optics Design and Optimisation. *arXiv preprint arXiv:2412.16006*, 2024. https://arxiv.org/abs/2412.16006

[DKK] L. R. Dalesio, M. R. Kraimer, A. J. Kozubal. EPICS architecture. In *Proceedings of the International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS)*. Pp. 278–282. Los Alamos National Laboratory.

[DZP09] J. DONG, Y. ZHAO, T. PENG. A REVIEW OF DESIGN PATTERN MINING TECHNIQUES. *International Journal of Software Engineering and Knowledge Engineering* 19(06):823–855, 2009. doi:10.1142/S021819400900443X

[Fow02] M. Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2002.

[Gel91] D. H. Gelernter. *Mirror worlds, or, The day software puts the universe in a shoebox– : how it will happen and what it will mean*. New York : Oxford University Press, 1991.

[HC04] N. Harrison, J. O. Coplien. *Organizational Patterns of Agile Software Development*. Prentice Hall, 2004.

[Mat24] MathWorks. Digital Twins for Predictive Maintenance. https://de.mathworks.com/campaigns/offers/digital-twins-for-predictive-maintenance.html, 2024. ebook Accessed: 2025-08-18.

[PR14] X. Pang, L. Rybarcyk. GPU accelerated online multi-particle beam dynamics simulator for ion linear particle accelerators. *Computer Physics Communications* 185(3):744–753, 2014. https://www.sciencedirect.com/science/article/pii/S0010465513003780

[RCFN17] W. Rogers, N. Carmignani, L. Farvacque, B. Nash. pyAT: A Python Build of Accelerator Toolbox. In *Proceedings of the 8th International Particle Accelerator Conference (IPAC'17)*. Pp. 3799–3801. Copenhagen, Denmark, 2017. https://accelconf.web.cern.ch/ipac2017/papers/thpab060.pdf

[Ski08] S. S. Skiena. *The Algorithm Design Manual*. Springer, 2nd edition, 2008.

[SLV+24] P. Sowiński, I. Lacalle, R. Vaño, C. E. Palau, M. Ganzha, M. Paprzycki. Overview of Current Challenges in Multi-Architecture Software Engineering and a Vision for the Future. In *Big Data Analytics in Astronomy, Science, and Engineering (BDA 2024)*. Springer, 2024. https://link.springer.com/chapter/10.1007/978-3-031-86193-2_5

[SS24] W. Sulaiman Khail, P. Schnizer. Patterns in Digital Twin Development. In *Proceedings of the 29th European Conference on Pattern Languages of Programs, People, and Practices*. EuroPLoP '24. Association for Computing Machinery, New York, NY, USA, 2024.
https://doi.org/10.1145/3698322.3698325

[SSB+23] P. Schnizer, W. Sulaiman Khail, J. Bengtsson, M. Ries, L. Deniau. Progress on Thor SCSI development. In *Proceedings of the 14th International Particle Accelerator Conference (IPAC'23)*. Venice, Italy, 2023. Contribution ID: 2423, Contribution code: WEPL127.
https://indico.jacow.org/event/41/contributions/2085/contribution.pdf

[SV17] W. Sulaiman Khail, V. Vranić. Treating pattern sublanguages as patterns with an application to organizational patterns. In *Proceedings of the 22nd European Conference on Pattern Languages of Programs*. Pp. 1–12. 2017.

[SV19] M. Seemann, S. Van Deursen. *Dependency injection principles, practices, and patterns*. Simon and Schuster, 2019.

[VHJG95] J. Vlissides, R. Helm, R. Johnson, E. Gamma. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[WZX+23] Z. Wan, Y. Zhang, X. Xia, Y. Jiang, D. Lo. Software Architecture in Practice: Challenges and Opportunities. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE 2023, p. 1457–1469. Association for Computing Machinery, New York, NY, USA, 2023.
https://doi.org/10.1145/3611643.3616367