Archives available at journals.mriindia.com

International Journal on Advanced Computer Theory and Engineering

ISSN: 2319 - 2526

Volume 15 Issue 01s, 2026

Software System Adaptability: A Middleware for Orchestrating Legacy Component Behavior

Marisha Dhimar
Email: marishadhimar499@gmail.com

Peer Review Information

Submission: 08 Dec 2025

Revision: 25 Dec 2025

Acceptance: 10 Jan 2026

Keywords

Self-Adaptive Software Systems, Legacy Software Integration, Middleware Frameworks, MAPE-K Feedback Loop, Dynamic Runtime Reconfiguration

Abstract

The increasing need for software systems to adapt dynamically to changing operational contexts has highlighted the challenge of integrating legacy components, which were often designed without adaptability in mind. This paper introduces Tekio, a lightweight middleware framework built on the OSGi specification that enables legacy software libraries to function effectively within self-adaptive systems. Tekio incorporates the MAPE-K feedback loop to support dynamic reconfiguration through parameter adjustment, component replacement, context-event response, and quality-of-service adaptation. A case study in computer vision using the OpenCV library demonstrates that Tekio achieves near-native performance while enabling frequent runtime adaptations. Empirical evaluation shows minimal overhead in throughput, CPU utilization, and memory consumption, with settling times in the millisecond range for low-to-medium complexity adaptations. The results indicate that Tekio provides a viable pathway for extending the operational lifespan of legacy software in modern adaptive environments, balancing flexibility with performance efficiency.

Introduction and Background

The rapid growth of computing technologies has driven the emergence of self-adaptive systems, which are capable of adjusting their structure and behavior in response to contextual changes and feedback from internal operations. These systems demonstrate resilience by continuing to function even under fluctuating environmental conditions, hardware faults, or user interventions [1, 2]. Modern software platforms such as large-scale web services and social networks provide well-known examples of adaptability, dynamically scaling and reconfiguring themselves to maintain quality of service (QoS) in the face of unpredictable demand.

In contrast to contemporary adaptive applications, most legacy software libraries and systems were not originally designed with runtime reconfiguration or fault tolerance in mind. Their architecture assumes static deployment, limited flexibility, and minimal interaction with dynamic operating contexts. As organizations continue to rely heavily on these legacy components, the question arises: can they be effectively reused within a self-adaptive framework? Addressing this challenge is critical, as discarding proven libraries outright would entail both economic and technical losses.

Middleware has emerged as a promising strategy for bridging the gap between rigid legacy software and modern self-adaptive paradigms. Middleware frameworks can encapsulate existing libraries, providing dynamic configura-tion, runtime monitoring, and controlled adaptation mechanisms. In this regard, the OSGi (Open Services Gateway Initiative) specification has gained traction due to its modular, service-oriented architecture that supports component lifecycle management and runtime substitution. OSGi has found widespread use in domains ranging from mobile computing and enterprise applications to embedded automotive systems [3].

The Tekio middleware framework builds on OSGi to provide a lightweight yet flexible solution for orchestrating the behavior of legacy components. Tekio introduces self-adaptive features that allow components to be loaded, replaced, or reconfigured at runtime without disrupting the overall system. Its architecture emphasizes three pillars: component management, instance management, and self-adaptation management. Together, these mechanisms enable legacy modules to be transformed into adaptable units capable of participating in dynamic runtime environments.

To demonstrate the practicality of Tekio, this study presents a case study in computer vision, a domain that demands high throughput, low latency, and adaptability to diverse operational contexts. Leveraging the OpenCV library as a legacy component, Tekio manages dynamic reconfigurations of vision pipelines, enabling tasks such as segmentation, intrusion detection, and face recognition to adapt in real time. This use case highlights the efficiency of Tekio in balancing performance with adaptability, illustrating its minimal overhead compared to native implementations.

Another critical contribution of Tekio lies in its empirical evaluation. While many self-adaptive frameworks propose architectural models, fewer provide rigorous experimental validation of their performance impact. Tekio is evaluated with respect to throughput, CPU utilization, memory consumption, and settling time—the period required for the system to stabilize after adaptation. Results indicate that Tekio achieves near-native performance while supporting high-frequency adaptations, though certain trade-offs emerge at higher video resolutions.

In summary, the introduction of Tekio middleware demonstrates the feasibility of reusing legacy components within a modern self-adaptive framework. By leveraging OSGi for modularity and extending it with mechanisms for dynamic reconfiguration, Tekio enables conventional software to remain relevant in dynamic computing environments. The following sections expand on related work, the architecture of Tekio, its adaptive mechanisms, experimental validation, and future directions for enhancing middleware-driven adaptability.

Related Work and Research Context

The design of self-adaptive middleware has been a growing area of research in software engineering, motivated by the need to manage variability and ensure system resilience in dynamic environments. Early efforts focused primar-ily on creating frameworks that allowed modular composition and runtime reconfiguration of software components. While these systems demonstrated the feasibility of dynamic adaptation, they often lacked mechanisms for empirical evaluation or integration of legacy software libraries. Consequently, much of the earlier research did not address the practical challenges of balancing adaptability with performance efficiency [2].

One of the notable projects in this area is DIVA (Dynamic Variability in Complex, Adaptive systems), a European initiative aimed at addressing the complexity of managing system configurations. DIVA leverages model-driven and aspect-oriented techniques to represent dynamic variability, providing developers with tools to handle context changes systematically [4]. While effective in modeling a wide range of configurations, DIVA is considered heavy-weight in practice, as its adaptations occur on the scale of seconds rather than milliseconds. This temporal limitation restricts its applicability in domains such as computer vision, where rapid responsiveness is essential.

Another significant initiative is the MUSIC (Self-Adapting Applications for Mobile Users in Ubiquitous Com-puting Environments) project, which provides an open-source platform for building context-aware, adaptive mobile applications. MUSIC incorporates sensing mechanisms and QoS monitoring to determine when adaptations should occur [5]. By maintaining a separation between business logic and adaptation logic, MUSIC achieves a high degree of modularity and flexibility. However, it does not adequately address the problem of adaptation frequency and does not explore how legacy libraries can be incorporated within its framework.

Both DIVA and MUSIC illustrate the broader challenges in self-adaptive middleware: the complexity of handling a vast number of possible system variations, ensuring seamless migration between configurations, and maintaining predictable performance under frequent adaptations. Importantly, these frameworks provide limited empirical analysis of their performance characteristics. Without rigorous validation, it remains unclear how well they perform under stress conditions such as rapid reconfiguration or high data throughput.

The Tekio middleware distinguishes itself by explicitly addressing these gaps. Unlike earlier frameworks, Tekio emphasizes the reuse of legacy libraries within an OSGi-based adaptive infrastructure. By building on the OSGi standard, Tekio inherits the advantages of modular component management and runtime lifecycle control while extending these features to support empirical validation of adaptation costs. This design choice makes Tekio lightweight and suitable for domains that demand both flexibility and high performance.

Furthermore, Tekio's evaluation methodology provides insights into the trade-offs between adaptability and re-source consumption. By measuring throughput, CPU usage, memory utilization, and settling time, Tekio offers a more comprehensive perspective on how adaptive middleware behaves under realistic workloads. This focus on empirical evidence is particularly valuable for practitioners, as it helps establish confidence in deploying middleware-based adaptive systems in production environments.

In summary, while prior research in adaptive middleware has made significant contributions in modeling variability and providing modular frameworks, Tekio advances the field by targeting legacy software reuse, lightweight design, and empirical validation. Its development positions it as a practical complement to earlier frameworks, offering both theoretical contributions and operational insights into middleware-supported self-adaptation.

Tekio Middleware: Concept and Architecture

Tekio is designed as a middleware framework that enables the reuse of legacy software components within self-adaptive systems. Its architecture is grounded in the principles of modularity, separation of concerns, and runtime flexibility. At its core, Tekio leverages the OSGi specification, a widely adopted standard for modular software platforms that supports dynamic component lifecycle management. By adopting OSGi, Tekio provides a foundation for loading, unloading, and replacing components while the system remains operational [3]. This feature is crucial for supporting uninterrupted adaptability in dynamic environments.

The architecture of Tekio can be conceptualized as a layered system. The lowest layer consists of the OSGi runtime and the Java Virtual Machine (JVM), which together manage the execution of services and ensure modular separation. On top of this, domain-specific components encapsulate legacy libraries, such as the OpenCV computer vision toolkit. These components are designed to provide well-defined interfaces in Java while internally invoking native C/C++ implementations via Java Native Access (JNA). This design ensures portability while preserving the performance advantages of native code [2].

Above the domain-specific components lie Tekio's self-adaptation mechanisms. These modules are responsible for monitoring runtime conditions, analyzing context, and triggering reconfiguration decisions. By organizing the architecture into distinct layers, Tekio maintains a clear separation of concerns: the OSGi runtime ensures modularity, the domain-specific layer integrates legacy functionality, and the self-adaptation layer manages dynamic behavior. This design enables the middleware to balance adaptability with performance stability.

Tekio's architecture incorporates the Monitor-Analyze-Plan-Execute-Knowledge (MAPE-K) loop as a conceptual model for self-adaptation. The context manager serves as the monitoring component, gathering runtime data and triggering adaptation requests when conditions change. The adaptation planner generates suitable strategies for re-configuration, while the executor applies these changes by reconfiguring the component chain. Finally, the knowledge base retains historical data about previous adaptations, improving decision quality over time [1]. This loop ensures that Tekio not only adapts reactively but also evolves its strategies based on accumulated experience.

Another important architectural feature of Tekio is its support for different levels of adaptation. Parameter adapta-tion allows fine-tuned changes to running components, such as adjusting segmentation thresholds in image processing tasks. Component adaptation enables runtime replacement of modules with alternatives offering similar functionality. Context-event adaptation reacts to external or internal events that trigger reconfiguration. Finally, QoS adaptation ensures that the system maintains acceptable performance even under resource constraints or fluctuating workloads. Together, these mechanisms provide Tekio with a versatile adaptation toolkit capable of handling diverse scenarios.

Tekio also incorporates the notion of processing chains, which represent the active configuration of interconnected components at any given time. A processing chain can consist of acquisition modules, segmentation algorithms, object detection modules, and event generation components. By supporting multiple alternative configurations, Tekio allows the system to dynamically switch between processing chains depending on environmental conditions or application requirements. This flexibility is particularly important for domains such as computer vision, where algorithms may need to change frequently to accommodate different input resolutions, lighting conditions, or computational budgets. In summary, the architecture of Tekio combines the modularity of OSGi, the performance of legacy libraries, and the intelligence of self-adaptive control loops.

Its layered design and support for multiple adaptation mechanisms make it well suited for integrating legacy components into modern adaptive systems. By providing runtime flexibility with minimal performance penalties, Tekio demonstrates a practical pathway for extending the lifespan and functionality of conventional software within dynamic operating environments.

Adaptive Mechanisms in Tekio

A defining feature of Tekio middleware lies in its adaptive mechanisms, which are designed to balance flexibility with performance efficiency. These mechanisms are informed by the Monitor-Analyze-Plan-Execute-Knowledge (MAPE-K) loop, a conceptual model widely recognized in self-adaptive systems research [1, 2]. Tekio adopts this model to ensure continuous monitoring of runtime conditions, context-aware analysis, planning of suitable adaptations, and ex-ecution of configuration changes. By embedding this loop within the middleware, Tekio achieves both responsiveness to immediate events and the ability to improve decision-making over time.

The *context manager* serves as the monitoring component, responsible for gathering runtime data such as through-put, error rates, or QoS indicators. This functionality aligns with findings in adaptive middleware literature, where effective monitoring has been identified as essential for ensuring timely adaptations [6]. Tekio extends this concept by enabling the context manager to interact directly with domain-specific components, ensuring that both system-level and application-level contexts influence adaptation decisions. This dual focus enhances Tekio's ability to detect subtle performance degradations before they become critical.

Once monitoring data is collected, the *adaptation planner* evaluates potential reconfiguration strategies. Plan-ning involves mapping contextual changes to suitable responses, whether through parameter adjustment, component replacement, or QoS-driven reallocation. The importance of robust planning has been emphasized in prior work, par-ticularly in frameworks such as DIVA and MUSIC, where the lack of fast and efficient planning limited adaptability [4, 5]. Tekio addresses this by prioritizing lightweight planning mechanisms that minimize latency, making it suitable for domains like real-time computer vision.

The *executor* component applies planned adaptations by dynamically reconfiguring the processing chain. Using OSGi's lifecycle management, Tekio can load or unload components at runtime, replace them with alternatives, and apply parameter adjustments without halting the system [3]. This seamless reconfiguration capability is crucial for environments where downtime is unacceptable. Furthermore, Tekio ensures that the executor maintains consistency during transitions, thereby reducing the likelihood of transient errors or system instability.

Tekio supports four primary adaptation types. *Parameter adaptation* involves fine-tuning configuration values, such as threshold levels in image segmentation algorithms. *Component adaptation* replaces entire modules at runtime, ensuring functional continuity while optimizing performance. *Context-event adaptation* is triggered by specific events from the environment or application, such as detection of anomalies in sensor data. Finally, *QoS adaptation* maintains acceptable levels of service quality under varying workloads or environmental conditions [1, 6]. Together, these mechanisms provide Tekio with the versatility to respond effectively to a wide range of operational challenges.

A key innovation of Tekio is its use of *processing chains*, which represent current configurations of interconnected components. Each chain consists of acquisition modules, segmentation algorithms, object detection modules, and event construction units. By maintaining multiple alternative chains, Tekio allows the system to switch dynami-cally between configurations. For instance, when integrated with the OpenCV library, Tekio can alternate between segmentation-based motion detection and HAAR-based face detection depending on input resolution or workload. This modular approach ensures adaptability without sacrificing the performance benefits of specialized legacy li-braries [7].

In practice, Tekio demonstrates that adaptive middleware can achieve near-native performance while supporting frequent reconfigurations. This is particularly significant when compared with earlier frameworks such as DIVA, which performed adaptations at slower timescales, or MUSIC, which lacked robust integration of legacy software [4, 5]. By adopting lightweight planning, efficient execution, and multi-level adaptation strategies, Tekio provides an op-erationally viable solution for embedding self-adaptation into legacy-driven systems. These mechanisms underscore the potential of middleware as a unifying platform for adaptability across diverse software domains.

Experimental Validation and Findings

To evaluate Tekio, an empirical study was conducted with a focus on computer vision, a domain well suited to stress-testing adaptive middleware due to its high data throughput and sensitivity to latency. The experiments aimed to address two central research questions: (1) how much performance overhead is introduced by embedding legacy components within Tekio's adaptive middleware, and (2) how frequently the system can adapt while maintaining acceptable quality of service (QoS). These questions are critical, as prior studies of self-adaptive middleware frameworks have often neglected empirical validation, limiting confidence in their operational deployment [2, 6].

The experimental setup implemented six different configurations of a vision system built on the OpenCV library, including segmentation, intrusion detection, and face recognition tasks. Tekio dynamically managed these configurations as processing chains, with each chain representing a distinct arrangement of acquisition, segmentation, object detection, and event construction components. Input consisted of 1020 frames of video processed at three resolu-tions—low, medium, and high. Performance was measured across four metrics: frames per second (FPS) throughput, CPU usage, memory utilization, and settling time (the time required to stabilize after reconfiguration) [7]. This design ensured a comprehensive assessment of Tekio's capabilities under varied conditions.

In the first set of experiments, Tekio's performance was compared directly with native C implementations of OpenCV components. Results showed that FPS for Tekio was marginally lower than the native version across all resolutions, indicating minimal throughput loss due to the middleware layer. CPU usage between Tekio and native implementations remained comparable, while memory usage was slightly higher in Tekio due to the overhead of the OSGi framework. However, the additional memory consumption did not exceed 2.5% of the system's total capacity, suggesting that the trade-off is acceptable for most practical applications [1, 3].

The second set of experiments evaluated Tekio's ability to handle frequent reconfigurations. By switching between all 38 possible pairs of the six configurations at varying time intervals (from two minutes down to one second), the system was subjected to increasing stress. Results revealed that Tekio could perform up to 30 adaptations within two seconds for low- and medium-resolution videos while maintaining meaningful output. For high-resolution input, adaptation frequency had to be reduced to around 90 seconds to sustain performance. These findings highlight the trade-offs between adaptation frequency and input complexity, a challenge also noted in earlier adaptive middleware studies [4, 5].

Settling time was another key measure of adaptability. Tekio consistently exhibited low settling times, stabilizing within milliseconds for low- and medium-resolution inputs. However, when adaptation frequency increased beyond practical limits (approximately 0.25 seconds per adaptation), Tekio stopped producing meaningful outputs rather than crashing. This resilience is significant, as it suggests that Tekio prioritizes system continuity even under extreme loads. Such behavior contrasts with earlier frameworks like DIVA, which exhibited instability when confronted with frequent adaptations [4].

The empirical findings confirm that Tekio introduces only a negligible performance overhead while enabling dy-namic reconfiguration of legacy components. Its lightweight design, based on OSGi and JNA, ensures that perfor-mance remains close to native implementations even during frequent adaptations. More importantly, Tekio demonstrates robustness by sustaining continuous operation under stress, an attribute that aligns with the broader goals of engineering dependable self-adaptive systems [6].

In summary, the validation of Tekio illustrates the practical viability of middleware-driven self-adaptation. By balancing performance with flexibility, Tekio extends the operational life of legacy libraries and provides an adapt-able platform for modern computing environments. The experimental results strengthen the case for middleware approaches that combine modularity, empirical validation, and runtime adaptability, bridging the gap between theoretical frameworks and real-world application demands.

Conclusion and Future Directions

The development of Tekio middleware demonstrates the feasibility of enabling legacy software components to operate effectively within self-adaptive environments. By building on the OSGi specification, Tekio delivers modularity and runtime flexibility while introducing lightweight mechanisms for dynamic adaptation. The experimental evaluation confirmed that Tekio introduces minimal overhead, with throughput, CPU usage, and memory consumption remaining comparable to native implementations. These findings place Tekio among the few frameworks that not only propose conceptual architectures but also provide empirical validation of their performance [8, 9].

One of the central contributions of Tekio lies in its ability to reuse proven legacy libraries, such as OpenCV, within modern adaptive contexts. This approach addresses both economic and technical challenges faced by organizations that depend heavily on mature but static software assets. Prior research has emphasized the high cost of discarding legacy systems, noting that reuse strategies significantly extend system longevity and reduce technical debt [10, 11]. Tekio aligns with this perspective by providing middleware mechanisms that transform legacy assets into adaptable, service-oriented components.

The adoption of the MAPE-K loop within Tekio highlights the continued relevance of feedback-driven adaptation in software engineering. Feedback loops are widely regarded as a cornerstone of autonomic computing, ensuring that systems remain robust under uncertainty [12]. By tailoring MAPE-K to handle parameter, component, context-event, and QoS adaptations, Tekio achieves a balance between adaptability and efficiency. This focus complements broader research advocating for layered control loops as a strategy for managing trade-offs in self-adaptive systems [13].

Tekio's validation in computer vision provides a representative case study, but its design principles are generalizable across domains. For example, self-adaptive middleware has been applied successfully in cloud computing, where scalability and cost efficiency are critical [14]. Similarly, in cyber-physical systems, adaptive middleware supports dynamic responses to sensor-driven events while maintaining real-time constraints [15]. By demonstrating robust adaptation with low overhead, Tekio strengthens the case for applying middleware solutions in diverse application areas.

Nevertheless, the experimental results also revealed limitations that suggest directions for improvement. At high video resolutions, Tekio's adaptation frequency must be reduced to sustain meaningful outputs, highlighting the re-source trade-offs inherent in middleware-driven adaptation. Addressing such scalability issues requires more sophis-ticated resource management techniques, such as predictive models for workload estimation and runtime optimization [16]. Incorporating these strategies would allow Tekio to maintain responsiveness even under extreme computational demands.

Future work on Tekio could also benefit from integrating self-healing capabilities. Self-healing has been identified as a critical extension of self-adaptive systems, enabling software to recover automatically from faults without external intervention [17]. Middleware frameworks with built-in self-healing could significantly increase reliability in mission-critical systems. Extending Tekio with mechanisms for error detection, automated repair, and fault-tolerant adaptation would broaden its applicability to safety-critical domains.

Another promising direction involves enriching Tekio with runtime models. Model-driven adaptation has gained attention for its ability to reason about system states at higher levels of abstraction and guide adaptation decisions more effectively [18]. Integrating runtime models into Tekio could enable predictive adaptation, where the middleware anticipates future conditions rather than merely reacting to observed changes. Such predictive capabilities align with ongoing trends in adaptive systems research toward proactive, intelligence-driven adaptation.

In conclusion, Tekio contributes to the evolving body of work on self-adaptive middleware by combining legacy component reuse, lightweight architecture, and empirical validation. Its demonstrated ability to balance adaptabil-ity with performance efficiency makes it a strong candidate for adoption in diverse software ecosystems. Looking forward, augmenting Tekio with predictive adaptation, self-healing, and advanced resource management will fur-ther strengthen its role as a robust enabler of software adaptability in an era of increasing system complexity and dynamism.

References

[1] P. Oreizy, M. M. Gorlick, R. N. Taylor, D. Heimbigner, G. Johnson, N. Medvidovic, A. Quilici, D. S. Rosenblum, and A. L. Wolf, "An architecture-based approach to self-adaptive software," *IEEE Intelligent Systems*, vol. 14, no. 3, pp. 54–62, 1999.

[2] Y. Brun, G. D. M. Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. A. Muller, M. Pezzè, and M. Shaw, "Engineering self-adaptive systems through feedback loops," in *Software Engineering for Self-Adaptive Systems*, ser. Lecture Notes in Computer Science. Springer, 2009, vol. 5525, pp. 48–70.

[3] S. Hallsteinsen, E. Stav, A. Solberg, and J. Floch, "Using product line techniques to build adaptive systems," in *10th International Software Product Line Conference (SPLC)*. IEEE, 2006, pp. 10–20.

[4] I. Romero, F. Juan, V. Chicote, B. Morin,

and O. Barais, "Using models@runtime for designing adaptive robotics software: An experience report," in *Proceedings of the International Conference on Context*. Springer, 2010. [Online]. Available: http://repositorio.bib.upct.es/dspace/handle/10317/1431

[5] B. Morin, T. Ledoux, M. B. Hassine, F. Chauvel, O. Barais, and J.-M. Jezequel, "Unifying runtime adaptation and design evolution," in *2009 Ninth IEEE International Conference on Computer and Information Technology*. IEEE, 2009, pp. 104–109.

[6] Gummadi, V. P. K. (2019). Microservices architecture with APIs: Design, implementation, and MuleSoft integration. Journal of Electrical Systems, 15(4), 130–134. https://doi.org/10.52783/jes.9328,

[7] A. Finkelstein, C. Gacek, K. Geihs, V. Grassi, G. Karsai, H. Kienle, J. Kramer, M. Litoiu, S. Malek, R. Mirandola, H. A. Muller, S. Park,

[8] M. Shaw, M. Tichy, M. Tivoli, D. Weyns, and J. Whittle, "Software engineering for self-adaptive systems: A research roadmap," in *Software Engineering for Self-Adaptive Systems*, ser. Lecture Notes in Computer Science. Springer, 2009, vol. 5525, pp. 1–26.

[9] OpenCV Community, *OpenCV: Open Source Computer Vision Library*, 2008, accessed: 2025-09-17. [Online]. Available: https://opencv.org/

[10] S. Dobson, R. Sterritt, P. Nixon, and M. Hinchey, "Fulfilling the vision of autonomic computing," in *Advances in Engineering Software*. Elsevier, 2006, vol. 37, no. 3, pp. 381–390.

[11] D. Weyns, M. U. Iftikhar, S. Malek, and J. Andersson, "Claims and evidence for architecture-based self-adaptation," *Proceedings of the International Conference on Software Engineering*, pp. 867–876, 2013.

[12] T. Mens and A. Serebrenik, "Challenges in software evolution," in *Software Evolution*. Springer, 2010, pp. 1–18.

[13] R. Seacord, D. Plakosh, and G. Lewis, *Modernizing Legacy Systems: Software Technologies, Engineering Processes, and Business Practices*. Addison-Wesley, 2003.

[14] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[15] D. Garlan, S.-W. Cheng, A. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: Architecture-based self-adaptation with reusable infrastructure," in *Proceedings of the International Conference on Automated Software Engineering (ASE)*, 2004, pp. 276–277.

[16] H. A. Müller, M. Pezzè, and M. Shaw, "Visibility of control in adaptive systems," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2009, pp. 23–32.

[17] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 4, no. 2, pp. 1–42, 2009.

[18] S. Shevtsov and D. Weyns, "Keep it simplex: Satisfying multiple goals with guarantees in control-based self-adaptive systems," in *Proceedings of the 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 2017, pp. 23–34.

[19] D. Ghosh, R. Sharman, H. R. Rao, and S. Upadhyaya, "Self-healing systems—survey and synthesis," *Decision Support Systems*, vol. 42, no. 4, pp. 2164–2185, 2007.

[20] N. Bencomo, R. France, B. H. C. Cheng, and U. Aßmann, "Models@runtime: Foundations, applications, and roadmaps," in *Models@Runtime*. Springer, 2014, pp. 1–18.