Self-Adaptive Systems: Methodologies for Reusability and Security

Mutasem Salloum (ms227fm)
Rashed Qazizada (rq222ah)

Abstract

This report investigates two core research questions: (1) What are the current methodologies and approaches for achieving reusability in self-adaptive software systems (SAS)? and (2) What are the benefits and challenges of implementing reusable components in these systems? We examine frameworks such as Autonomic Software Product Line Engineering (ASPLe) and code-level adaptation methods to understand how SAS can be made adaptable and secure. ASPLe supports modularity and flexibility, while code-level strategies enhance adaptability at a granular level. Our findings indicate that reusability reduces development costs and increases resilience. However, the lack of integrated security measures presents risks, especially in critical environments. This study highlights the need for adaptable, security-conscious frameworks that evolve with system requirements, providing a foundation for future research to enhance the security and effectiveness of reusable SAS components.

1. Introduction

As modern software systems grow in complexity, there is a rising demand for systems that can adapt autonomously to dynamic conditions. Self-adaptive systems (SAS) are designed to meet this need, allowing software to monitor, analyse, and adjust its behaviour based on environmental changes without human intervention. In critical areas such as healthcare, cloud computing, and autonomous vehicles, this capability is essential to maintain continuous functionality despite shifting requirements or unexpected conditions. Achieving this adaptability efficiently requires the reuse of modular components that can be applied across various contexts, reducing both development time and costs.

This report investigates two research questions central to the advancement of SAS:

- **RQ1:** What are the current methodologies and approaches used to achieve reusability in self-adaptive software systems?
- **RQ2:** What are the benefits and challenges of implementing reusable components in self-adaptive software systems?

To address these questions, we need to understand a few concepts:

- **Self-Adaptive Software System (SASS):** A system that can modify its behaviour and structure in response to changes in its environment, its own internal state, or its goals.
- **ASPLe (Autonomic Software Product Line Engineering):** ASPLe organizes SAS development into domain engineering, specialization, and integration stages, promoting systematic reuse of components across different applications. This modular approach saves time and supports consistent quality by allowing components to be adapted with minimal effort.
- **Code-level adaptation methods:** This strategy focuses on reusability at the coding level, allowing adaptation within specific modules. Code-level adaptation enables flexibility on a smaller scale, complementing the higher-level reusability of frameworks like ASPLe, though it may encounter scalability challenges in larger systems.

The motivation behind these research questions stems from the need to balance adaptability and security in SAS. While reusability can enhance efficiency and resilience, it can also expose systems to potential vulnerabilities if security measures are not integrated within the reusable components. This report provides insights into these challenges and suggests future directions for enhancing SAS with more secure and flexible reusable components, offering a foundation for further research on building robust adaptive systems that can evolve with changing conditions.

2. Approach or Method

To explore and answer the research questions, we conducted a systematic literature review (SLR), focusing on recent studies related to self-adaptive systems (SAS), particularly those emphasizing modularity, reuse, and adaptive security. In alignment with the project's requirements, all articles were selected from Norwegian List Level 2 journals, ensuring a high standard of peer-reviewed quality and relevance in the field.

2.1. Systematic Literature Review

The SLR involved a structured search across academic databases, including IEEE Xplore, ScienceDirect, and SpringerLink. The search terms used were "self-adaptive systems," "systematic reuse," and "security." Articles were chosen based on their alignment with the research questions, publication date (2020–2024), and relevance to SAS reusability. This selection process resulted in three primary sources:

1. **ASPLe:** A methodology for reuse in self-adaptive systems, detailing a structured approach for developing reusable components [1].
2. **Systematic review on security and safety in SAS:** This article highlights the security challenges in adaptive systems, especially in high-risk environments [2].
3. **Code-level adaptation for SAS:** This source explores code-level techniques to enhance flexibility and adaptability within SAS, complementing broader reuse strategies [3].

2.2. Analyzing Methodologies

Each methodology was analyzed for its approach to modularity, adaptability, and security:

- **ASPLe (Autonomic Software Product Line Engineering):** Detailed in the work on ASPLe, this methodology divides SAS development into domain engineering, specialization, and integration stages, creating modular artifacts for reuse. This approach supports consistent quality and adaptability across different SAS applications [1].
- **Code-Level Adaptation Techniques:** Described in research on code-level approaches, this method focuses on adaptable coding practices to enable flexibility at a more granular level. Code-level adaptation is particularly beneficial for implementing fine-grained adjustments within specific modules, complementing the broader reuse strategy provided by frameworks like ASPLe [3].

This approach is suitable as it provides a comprehensive view of modularity, adaptability, and security concerns essential to effectively answering RQ1 and RQ2.

3. Results

This section presents the findings related to the research questions on reusable methodologies in self-adaptive systems (SAS). The results focus on the ASPL strategy and ASPLe methodology, the proposed algorithm for adaptive component development, and the component metrics used for evaluation.

3.1. ASPL Strategy and ASPLe Methodology

The **Autonomic Software Product Lines (ASPL)** strategy addresses key challenges in software development related to variability, reuse, and uncertainty. It is based on two fundamental principles: (1) strict separation of managed and managing system concerns, and (2) stepwise specialization of reusable assets into product-specific assets. These principles reduce complexity and mitigate uncertainty in the development process [1].

The ASPL strategy consists of three steps:

1. **Establish a Horizontal ASPL Platform**: Create a horizontal platform with domain-independent artifacts reusable across multiple managing subsystems. These artifacts cover all stages of the development lifecycle and are designed for reuse across various domains.

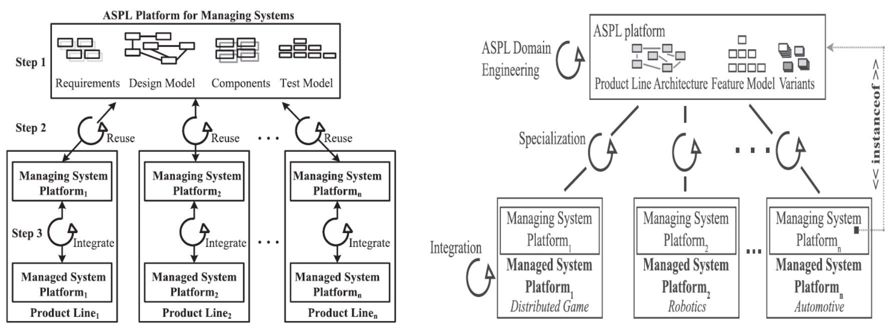
2. **Derive a Vertical Managing System Platform**: Transform the horizontal platform into vertical managing system platforms tailored to the adaptation logic of specific application domains.

3. **Integrate Platforms**: Integrate the managing system platform with a separately developed managed system platform containing the application logic, developed using approaches like Software Product Line Engineering (SPLE).

While the ASPL strategy is critical for developing SAS, it faces a significant research gap due to the lack of detailed process support for engineers to apply the strategy consistently in real-world projects.

To address this gap, the **ASPLe (Autonomic Software Product Line Engineering)** methodology was developed. It provides process support for systematic reuse at the managing system level, aligned with the ASPL strategy. ASPLe is organized into three primary processes: *ASPL Domain Engineering*, *Specialization*, and *Integration*. A key element of ASPLe is the **Extended Architectural Reasoning Framework (eARF)**, which incorporates design support for self-adaptive properties like self-healing and self-optimization. The eARF guides domain analysts and designers through a structured process of identifying and specifying self-adaptation requirements using domain Quality Attribute Scenarios (dQAS) and mapping these requirements to design alternatives. It leverages reusable architectural tactics and patterns to ensure that design decisions are well-founded and aligned with quality attributes. This facilitates the development of flexible, adaptable systems capable of accommodating changes in dynamic environments [1].

N. Abbas, J. Andersson and D. Weyns / The Journal of Systems and Software 167 (2020) 110626


ASPL Platform for Managing Systems
Step 1
Requirements
Design Model
Components
Test Model
Step 2
Reuse
Managing System Platform$_1$
Managing System Platform$_2$
Managing System Platform$_n$
Step 3
Integrate
Managed System Platform$_1$
Managed System Platform$_2$
Managed System Platform$_n$
Product Line$_1$
Product Line$_2$
Product Line$_n$
ASPL Domain Engineering
ASPL platform
Product Line Architecture
Feature Model
Variants
Specialization
<<instanceof>>
Integration
Managing System Platform$_1$
Managed System Platform$_1$
Distributed Game
Managing System Platform$_2$
Managed System Platform$_2$
Robotics
Managing System Platform$_n$
Managed System Platform$_n$
Automotive

(a) ASPL Strategy (b) ASPLe Methodology

Figure 1: ASPLe Methodology Overview

Benefits and Challenges:

The ASPL strategy and ASPLe methodology offer several benefits. They **reduce complexity** by separating managed and managing system concerns, making systems easier to manage and evolve over time. **Systematic reuse** is promoted by reusing artifacts across multiple domains, which reduces development time and costs. **Scalability** is enhanced by deriving domain-specific platforms from a horizontal platform, enabling efficient expansion. **Flexibility in design** is achieved through stepwise specialization of reusable assets into product-specific assets, facilitating customization without starting from scratch.

However, challenges exist in their implementation. A significant **research gap in methodology** hinders consistent application in real-world projects due to the lack of detailed process support for engineers. **Complex integration** of managed and managing system platforms can be challenging, especially with diverse software components and technologies. Additionally, **domain-specific customization** may require significant effort, potentially affecting consistency and increasing redundancy across domains.

3.2. Proposed Algorithm for Adaptive Component Development

The algorithm proposed by Korra et al. [3] outlines a systematic approach for developing adaptive software components intended for reuse. The primary goal is to enhance efficiency and reduce development time by reusing components across different projects. The algorithm involves several key stages.

First, **Component Identification** entails analyzing the software system to identify potential components for reuse by isolating modules that perform distinct operations. Next, **Metric Evaluation** is conducted by applying specific component metrics to evaluate the identified components' suitability for reuse. Metrics such as CEM, CSEM, CRM, CFM, and CCSM assess attributes like efficiency, reliability, functionality, customer satisfaction, and cost.

Following evaluation, **Component Adaptation** involves modifying components as necessary to meet adaptability criteria, which may include refactoring code or enhancing interfaces to ensure compatibility across different systems. Then, **Component Documentation** is provided, offering comprehensive usage guidelines, interfaces, and dependencies to facilitate integration and reuse.

After documentation, **Repository Management** is performed by storing reusable components in a centralized repository with appropriate categorization and metadata for efficient retrieval and management. Finally, **Integration and Testing** are carried out by integrating components into new projects and conducting rigorous testing to verify their functionality and adaptability in different contexts.

Benefits and Challenges:

The proposed algorithm offers several benefits. It focuses on **component reusability**, promoting code reuse and reducing the need for new development, thereby speeding up project timelines. **Efficiency and cost savings** are achieved by leveraging pre-existing, well-tested components, which reduces overall development time and costs. The algorithm ensures **component compatibility** by using metrics like CEM, CSEM, and CRM to select components that meet specific adaptability criteria, enhancing system performance and reliability. Additionally, the use of a **centralized repository** simplifies component management and retrieval, facilitating easy access and integration into new projects.

However, there are challenges in implementing the algorithm. **Initial component identification** requires significant analysis and abstraction, particularly in large or legacy systems. **Adaptation complexity** arises when modifying components to fit different systems or contexts, often necessitating substantial refactoring or interface changes. The **evaluation of multiple metrics** may not always provide clear guidance, especially when components interact unexpectedly or when system requirements evolve rapidly. Ensuring components function correctly across different contexts requires rigorous testing, leading to **testing and integration overhead**. Finally, **dependency management** becomes increasingly complex as the number of components grows, making compatibility and version control challenging.

3.3. Component Metrics for Evaluation

To effectively evaluate and select components for reuse, several metrics are introduced. **Component Efficiency Metrics (CEM)** measure performance in terms of execution time and resource utilization. **Component Semantic Efficiency Measurement (CSEM)** assesses efficiency from a semantic perspective, including code readability, maintainability, and adherence to coding standards. **Component Reliability Metrics (CRM)** estimate the probability of functioning without defects over a specified period, which is critical for safety-critical systems. **Component Functional Metrics (CFM)** evaluate functionality, precision, interoperability, and suitability for the intended purpose. **Component Customer Satisfaction Measurement (CCSM)** gauges the extent to which components meet user expectations, based on feedback and satisfaction surveys. Finally, **Component Cost Metrics (CCM)** account for costs associated with development, acquisition, and integration, aiding in assessing economic viability.

4. Discussion

The findings underscore the practical benefits and challenges associated with implementing reusable components in self-adaptive systems (SAS). This discussion interprets the results, reflects on their implications, and suggests areas for future research.

4.1. Interpretation of Results

The ASPL strategy, ASPLe methodology, and the proposed algorithm provide a comprehensive view of current methodologies for achieving reusability in SAS. However, the lack of integrated security highlights a significant challenge that must be addressed to ensure the safe deployment of SAS in sensitive environments.

The successful application of the ASPL strategy and the ASPLe methodology demonstrates the effectiveness of systematic reuse in SAS development. By separating managed and managing system concerns and employing stepwise specialization, organizations can reduce complexity and promote scalability and flexibility in design. The proposed algorithm for adaptive component development complements these methodologies by providing a practical approach to identifying, evaluating, and adapting components for reuse.

4.2. Implications for Practice

Implementing these methodologies and algorithms can lead to more efficient software development processes, particularly in domains where adaptability and reliability are critical. Organizations can benefit from reduced development time and costs by reusing well-tested components and combining real-time responsiveness with ASPLe's systematic reuse and the proposed algorithm's code-level adaptability.

However, the challenges identified, such as the initial effort required for component identification, adaptation complexity, and security concerns, suggest that organizations need to invest in developing expertise and tools to support these processes. The lack of detailed process support in the ASPL strategy indicates a need for further refinement of methodologies like ASPLe to facilitate their practical application.

4.3. Comparison with Existing Literature

These findings align with existing research emphasizing the importance of modularity and reuse in software engineering. The ASPLe methodology extends traditional software product line engineering by incorporating self-adaptive properties, addressing a gap noted in prior studies [1]. The proposed algorithm adds value by focusing on code-level adaptation, complementing higher-level strategies.

4.4. Limitations and Future Work

Despite the benefits, several limitations exist. Future research should focus on integrating security metrics and mechanisms to address potential vulnerabilities. Additionally, evaluating components using multiple metrics can be time-consuming, and the integration of components may present unforeseen challenges due to dependencies.

Developing automated tools to streamline the evaluation process could reduce the time and effort required. Customizing metrics for different domains would enhance their relevance and effectiveness, and enhancing documentation could mitigate integration issues.

5. Conclusion

This study explored methodologies for reusability in self-adaptive systems (SAS), focusing on the ASPL strategy, the ASPLe methodology, and a proposed algorithm for adaptive component development. By introducing systematic approaches and comprehensive metrics for component evaluation, the study demonstrated that careful selection and development of components enhance adaptability, cost efficiency, and reliability in SAS.

Key findings include that reusing component evaluated through specific metrics leads to significant reductions in development time and costs, and that combining different methodologies can enhance system flexibility.

However, challenges such as the lack of integrated security measures and scalability issues highlight the need for further research.

Future work should focus on integrating security considerations into these frameworks and developing tools to automate and streamline the adaptation processes. By addressing these challenges, the methodologies and algorithms can be refined to build more adaptable, secure, and efficient self-adaptive systems that meet the evolving demands of various industries.

References

[1] N. Abbas, J. Andersson, and D. Weyns, "A methodology to develop self-adaptive software systems with systematic reuse," *Journal of Systems and Software*, vol. 167, p. 110626, 2020.

[2] I. Pekaric, R. Groner, et al., "A systematic review on security and safety of self-adaptive systems," *Journal of Systems & Software*, vol. 203, p. 111716, 2023.

[3] S. Korra, V. Biksham, and T. Bhaskar, "Code-level self-adaptive approach for building reusable software components," in *Intelligent Computing and Applications*, pp. 49–58, 2022.