D3.2
Monitorable and trustworthy verification loops
WP3

Public Document

Grant Agreement	101133807
Project	RoboSapiens
Deliverable Number	D3.2
Version	1.2
Due Month	M18
Date	June 2025

http://robosapiens-eu.tech/

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or HADEA. Neither the European Union nor the granting authority can be held responsible for them.

Contributors:

Roland Behrens, IFF
Robert Scharping, IFF
Yannick Bollmann, IFF
Morten Haahr Kristensen, AU
Thomas Wright, AU
Claudio Gomes, AU
Lukas Esterle, AU
Carlos Isasa, AU
Avramelou Loukia, AUTH
Akrivousis Dimitrios, AUTH
Katsikas Dimitrios, AUTH
Moustakidis Vasileios, AUTH
Tosidis Pavlos-Apostolos, AUTH
Tefas Anastasios, AUTH
Nikolaidis Nikolaos, AUTH
Passalis Nikolaos, AUTH

Editors:

Yannick Bollmann, IFF
Claudio Gomes, AU

Internal Reviewers:

Guoyuan Li, NTNU
Thomas Peyrucain, PAL
Peter Gorm Larsen, AU

Consortium:

Aarhus University	**AU**	University of Antwerp	**UA**
Aristotle University of Thessaloniki	**AUTH**	Norwegian University of Science and Technology	**NTNU**
Danish Technological Institute	**DTI**	PAL Robotics	**PAL**
Fraunhofer IFF	**IFF**	ISDI Accelerator	**ISDI**
University of York	**UoY**	Simula Research Lab	**SRL**

Funded by the European Union

Document Revision History:

Ver	Date	Author	Description
0.1	29-01-2025	Yannick Bollmann	Document created
0.2	03-03-2025	All contributors	Finished strucutre and subsection titles
0.3	19-03-2025	Thomas Wright	Added content on dynamic and distributed monitoring.
0.4	19-03-2025	Morten Haahr Kristensen	Added and revised content on dynamic and distributed monitoring.
0.5	31-03-2025	Carlos Isasa	Added survey section.
0.6	01-05-2025	Claudio Gomes	Added static and dynamic properties for the case studies.
0.7	09-05-2025	All contributors	Revision of content; added introduction and conclusion.
0.8	23-05-2025	Claudio Gomes	Revision of conclusion and introduction.
0.9	30-05-2025	Yannick Bollmann	Document ready for internal review.
1.0	16-06-2025	Claudio Gomes, Yannick Bollmann	Added appendix, addressed first review feedback.
1.1	20-06-2025	Claudio Gomes	Addressed additional review comments.
1.2	27-06-2025	Yannick Bollmann	Addressed last review comments and final edits.

Funded by the European Union

Abstract

Deliverable D3.2 reports the progress of RoboSAPIENS work package 3 until the project milestone M18.

In section 1 an overview is given of the findings of the systematic survey on the state of art of self adaptive system with a particular focus on MAPE-K loops (defined in previous deliverables).

Section 3 describes the implementation of monitoring for Trustworthiness Checkers as dynamic property monitoring (section 3.2) and distributed monitoring (section 3.3).

Lastly, in section 4 the implementation of monitoring for properties of the RoboSAPIENS case studies are presented.

The appendix contains three papers that are related to the content of this deliverable:

- A.1 - *State of the Art of the MAPE-K Loop: Architecture, Implementation and Verification* (summarized in section 2)
- A.2 - *Runtime Verification of Autonomous Systems utilizing Digital Twins as a Service* (summarized in section 3.1)
- A.3 - *DynSRV: Dynamically Updated Properties for Stream Runtime Verification* (summarized in section 3.2)

Funded by
the European Union

Abbreviations

CPS	cyber-physical system
DSU	Dynamic Software Updating
DUP	Dynamically Updated Property
DT	Digital Twin
LiDAR	Light Detection And Ranging
LTL	Linear Temporal Logic
MAPE-K	Monitor, Analyse, Plan, Execute - Knowledge
MAPLE-K	Monitor, Analyse, Plan, Legitimate, Execute - Knowledge
MTL	Metric Temporal Logic
past-CTL	past Computation Tree Logic
FMU	Functional Mock-up Unit
PT-DTL	Past-Time Distributed Temporal Logic
RoboSAPIENS	Robotic Safe Adaptation In Unprecedented Situations
ROS2	Robot Operating System 2
RV	Runtime Verification
SAS	Self-Adaptive Systems
STL	Signal Temporal Logic
SRV	Stream Runtime Verification
SwarmSTL	Swarm Signal Temporal Logic
TC	Trustworthiness Checker

Funded by
the European Union

Contents

Funded by the European Union

1 Introduction

This deliverable reports on the progress in RoboSAPIENS work package 3 until month 18. Starting with section 2, this introduction will briefly present the findings of the systematic survey regarding the state of the art of the MAPE-K (Monitor, Analyze, Plan, Execute, Knowledge) loop. The survey was first reported in Deliverable D3.1 [GIK$^+$24], but has since been expanded.

The main part of this deliverable focuses on the development of monitorable and trustworthy verification loops. To fulfill this task, the RoboSAPIENS project will implement Trustworthiness Checkers (TCs) in the proposed MAPLE-K architecture (see [ANM24]). In section 3 we describe the implementation of monitoring techniques for these Trustworthiness Checkers, more specifically dynamic property monitoring (section 3.2) and distributed monitoring (section 3.3).

Section 4 outlines how these techniques can be applied to the RoboSAPIENS case studies and specific use case properties, that have been developed together with the case study owners.

This deliverable mainly advances the following RoboSAPIENS overarching objectives:

O2 *Advance safety engineering techniques to assure robotic safety not only before but also during adaptation and after adaptation has taken place*

Section 2 presents the insights that justify the work on section 3 that promotes this objective.

O4 *Assure trustworthiness of systems that use both deep-learning and computational architectures for robotic self-adaptation*

Section 4 discusses preliminary integration of TCs in the RoboSAPIENS architecture, which is a key step towards achieving this objective. In particular, different safety properties are formalized for each case study, which is a necessary step towards ensuring trustworthiness of the systems.

Funded by
the European Union

2 Main Findings of Systematic Survey

In the continuation of the preliminary work introduced in D3.1 [GIK+24], we have surveyed the state of the art of the MAPE-K loop, classifying papers in one of three categories: architecture, implementation or verification.

Our survey highlights a significant gap in current research on self-adaptive systems, particularly those involving physical dynamics. Existing surveys predominantly focus on systems that rely on passive data collection for anomaly detection, neglecting scenarios where active data gathering is necessary to reduce uncertainty. This oversight hampers the formal verification of self-adaptive systems, especially within MAPE-K (Monitor, Analyze, Plan, Execute, Knowledge) loops. The RoboSAPIENS team recognizes this issue and addresses it in more detail in the concluding sections of the report.

Notably, the team reviewed 213 scientific articles to identify innovative concepts that exceed current capabilities. The resulting insights have informed the project team on a custom MAPLE-K loop architecture, aligning closely with project objectives, particularly in enabling robots to adapt to unpredictable structural and environmental changes.

In terms of safety and trustworthiness, the team conducted thorough reviews of industrial standards and legal requirements to ensure compliance during robotic adaptation. These findings will support the development of Trustworthiness Checkers within the MAPLE-K loop. Additionally, the team examined how to verify safety-critical, non-functional requirements using formal methods. A focused discussion on trust in robotics further refines the understanding and implementation of trustworthy adaptation mechanisms. Collectively, these insights enable the RoboSAPIENS project to push the boundaries of current research and deliver robust, adaptable, and cross-domain robotic systems.

In order to differentiate our work from other surveys, we have followed a systematic approach, detailed in fig. 1. The study followed a structured multi-phase approach, beginning with the identification of gaps in existing literature related to the MAPE-K loop in self-adaptive systems. Phase 1 involved defining research goals and formulating five key research questions, which focused on implementation techniques, their suitability, verification methods, interrelationships between MAPE-K components, and common architectural patterns. A research protocol was developed and refined through internal review. In Phase 2, a keyword-based search using Scopus identified the initial set papers, which were narrowed down after excluding non-relevant studies. Snowballing methods added more papers, and a cutoff was applied to exclude works published before 2010, resulting in a final dataset of 347 relevant studies. Phase 3 focused on extracting relevant data from these studies to address the research questions, involving a thematic analysis that led to a preliminary classification of implementation, verification, and architecture-related papers. In Phase 4, content analysis helped create taxonomies for these categories, while narrative synthesis identified trends and knowledge gaps. These findings were aligned with the research questions to generate practical insights. The final phase, Phase 5, compiled all results into a comprehensive manuscript, accompanied by a publicly accessible replication package to support transparency

Funded by the European Union

and future research replication.

We summarize here the main findings of the survey, which is available in full in Appendix A.1.

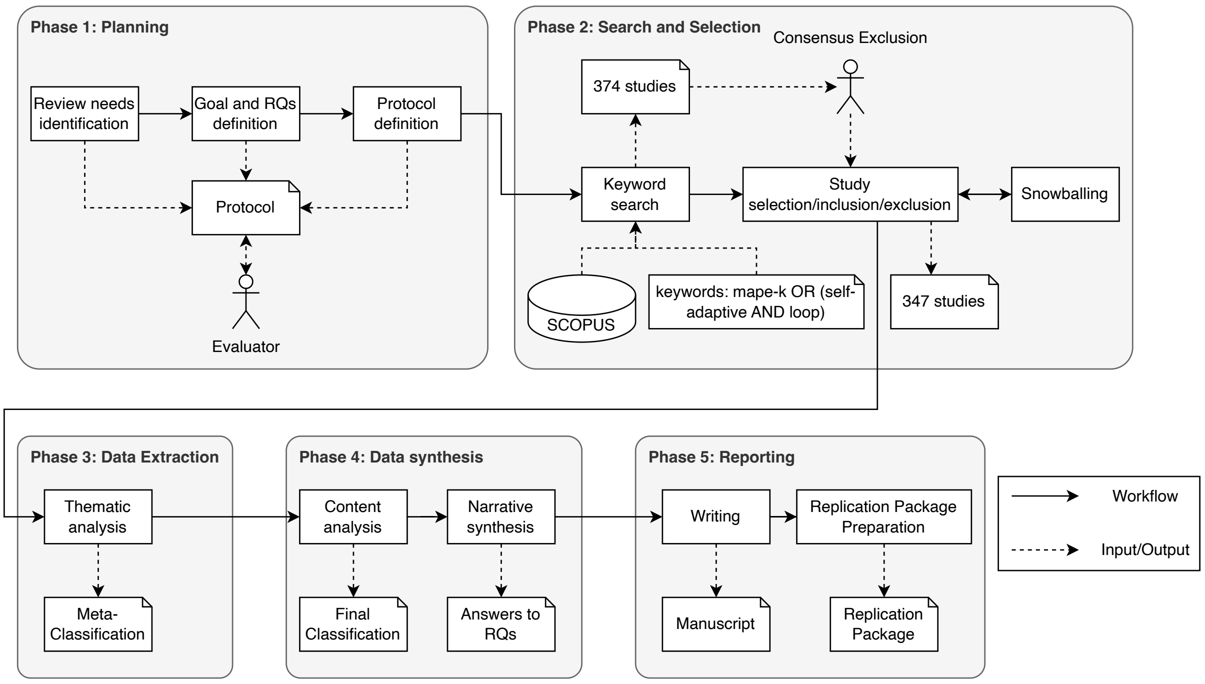


Figure 1: Study Design.

2.1 Architecture Findings

We can classify MAPE-K architectures with respect to 3 different axis, introduced in D3.1 [GIK+24]. component cardinality, component distribution and communication flows, detailed next.

Component Cardinality Based on how many software components implement each phase of the MAPE-K loop. We differentiate between **n:1** (monolotic, e.g., all all phases implemented in one software component), **n:n** (one to one) and **n:m** (one to many).

Component Distribution Based on how many devices realize the implementation. We distinguish between **single/monolitic**, **single/component-based** (single system but multiple components) and **distributed/component-based**

Communication Flows Based on how a system handles communication between them MAPE-K phases. We have identified the following categories: **sequential/unidirectional** where several MAPE-K loops are connected but communication is unidirectional, **parallel/bidirectional** multiple loops in parallel that communicate with each other, **parallel/independent** running in parallel but no communication, **hierarchical** multiple MAPE-K loops form a hierarchical communication structure.

Funded by the European Union

Architectural Extension We have identified a subset of papers that extend the loop with additional phases, running in parallel of sequentially to existing phases. We have classified them into the following categories:

- **Informative Phases**: Results of this phase are used to inform or assist the operation of an existing phase by performing other functionality.
- **Recording Phase**: Records and analyses information about the MAPE-K loop itself.
- **Reflexive Action Phase**: This phase acts as a shortcut between the Monitor and Execute phases, bypassing the rest of the loop if some conditions are met during the monitoring phase.
- **Rule Augmentation Phase**: Adds new rules to the Plan component adaptively.

2.2 Implementation

Monitoring We have identified the following categories in our taxonomy:

- **Direct Feed**: Data is send to analyze untreated.
- **Filtered Feed**: Anomalies are detected and data is filtered.
- **Batch**: Data is send in batches instead of a real-time approach.
- **Decentralized**: Distributed monitoring happens accross different nodes.

Analyse We have identified the following categories in our taxonomy:

- **Anomaly Detection**: Either via a rule-based approach or a machine-learning approach, deviations from the expected data are detected.
- **Feature Extraction**: Data can be treated to extract desired features from it, via machine-learning techniques or semantic modelling.
- **Data Preprocessing**: When the Monitor phase produces raw data, filters or semantic procesing can be used to generate cleaner data.
- **Adaptation Plan Selection**: The Analyse phase can be used to pre-select an adaptation plan for the Planning phase to study.

Plan We have identified the following categories in our taxonomy:

- **Rule and Policy**: Logical rules and system policies are used to guide the plan selection.
- **Reasoning Based**: Semantic reasoning can be used to select a way to deal with the unexpected behaviour.
- **Optimization Based**: The plan is created via solving an optimaztion problem.
- **Offline Machine-Learning**: A machine-learning algorithm, trained before the system is deployed, is used in order to make the planning decisions.
- **Online Machine-Learning**: A modified version of the offline machine-learning algorithms that is able to learn during system execution.

Funded by
the European Union

Execute We have identified the following categories in our taxonomy:

- **Path Through**: The Execution phase is considered a passive phase that just deploys the adaptation generated in the Plan phase.
- **Controlled**: A decision unit that checks if the plan is applicable is used.

Knowledge We have classified the contents of the Knowledge Base (KB) according to the following categories:

- **Data**: Represents all forms of monitored data, including sensor readings, system state information, historical performance data, and real-time observations.
- **Logs**: Encompasses all recorded logs, including event histories, error logs, execution traces, and runtime records.
- **Configurations**: Covers system settings, configuration files, mapping definitions, and parameter descriptions.
- **Strategies**: Includes high-level plans, policies, decision strategies, goal definitions, and adaptation tactics.
- **Rules**: Refers to policies, decision rules, security constraints, and predefined conditions that dictate system behavior.
- **Metrics**: Comprises performance indicators, quality attributes, and quantitative measures used for system evaluation.
- **Models**: Encompasses a wide range of modeling elements, including system topology, architectural schemas, behavior patterns, and ontologies.
- **Resources**: Includes all necessary resources, parameters, dependencies, and supporting data required for MAPE-K operations.

Note that a KB may contain several categories.

Results The distribution of categories per phase is found in fig. 2 (acronyms are defined in the survey). The value n defined in the caption of each graph defines how many papers discuss the corresponding phase. The distribution of papers across the MAPE-K phases reflects varying complexity and roles of each component. In the Monitor phase, the dominant "Direct-feed" category highlights a preference for simple, straightforward data collection methods. The Analyse phase is primarily used for anomaly detection using either rule-based or machine learning techniques, occasionally also serving in feature extraction and preprocessing. The Plan phase shows a more balanced distribution, reflecting its complexity, with "Rules and policy" being the most common approach, followed by optimization-based methods. Execution is typically the simplest phase, often acting as a pass-through, though some papers detail controlled execution strategies. Lastly, the Knowledge component mainly stores monitored data, system models, or adaptation rules.

Additionally, we have studied how associated are the taxonomies on each phase, i.e. given a phase is implemented in a certain way, how are the other phases imple-

Funded by the European Union

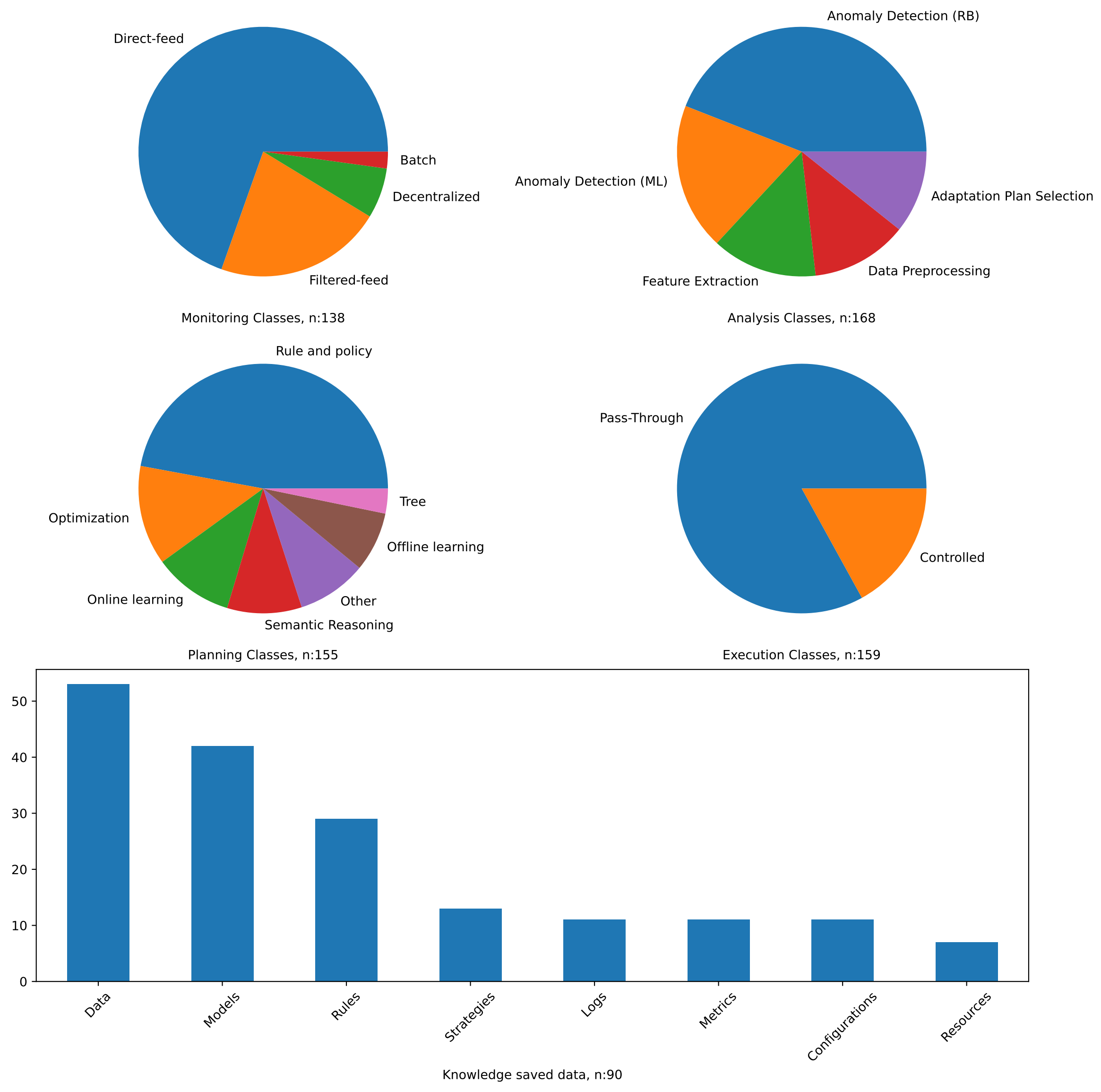


Figure 2: Distribution of classes of the MAPE phases and objects stored in K.

mented and if there is any correlation. Contingency tables showing these relations can be found in fig. 3. The knowledge component K is presented as a bar chart.

The tables reveal key correlations between MAPE-K phases and their implementation categories. A strong link exists between Rule and Policy planning and Direct Feed monitoring, particularly in applications that simplify both Monitor and Plan phases. A secondary correlation appears between Batch monitoring and Semantic Reasoning planning, though the limited use of Batch monitoring makes this connection unreliable. In the A-P table, Rule-Based Anomaly Detection aligns with Rule and Policy planning due to the simplicity of empirical rules, often used in exploratory application studies. Learning-based planning also correlates with Analyse methods: ML Anomaly Detection often pairs with Online Learning, using shared neural networks, while Feature Extraction aligns with Offline Learning in data-focused studies like Ortiz et al. Additionally, the M-A table highlights that Direct Feed monitoring often negates the need for separate Data Preprocessing in the Analyse phase. Associations involving the Execute phase are considered insignificant due to a sparse and imbalanced category distribution.

Funded by the European Union

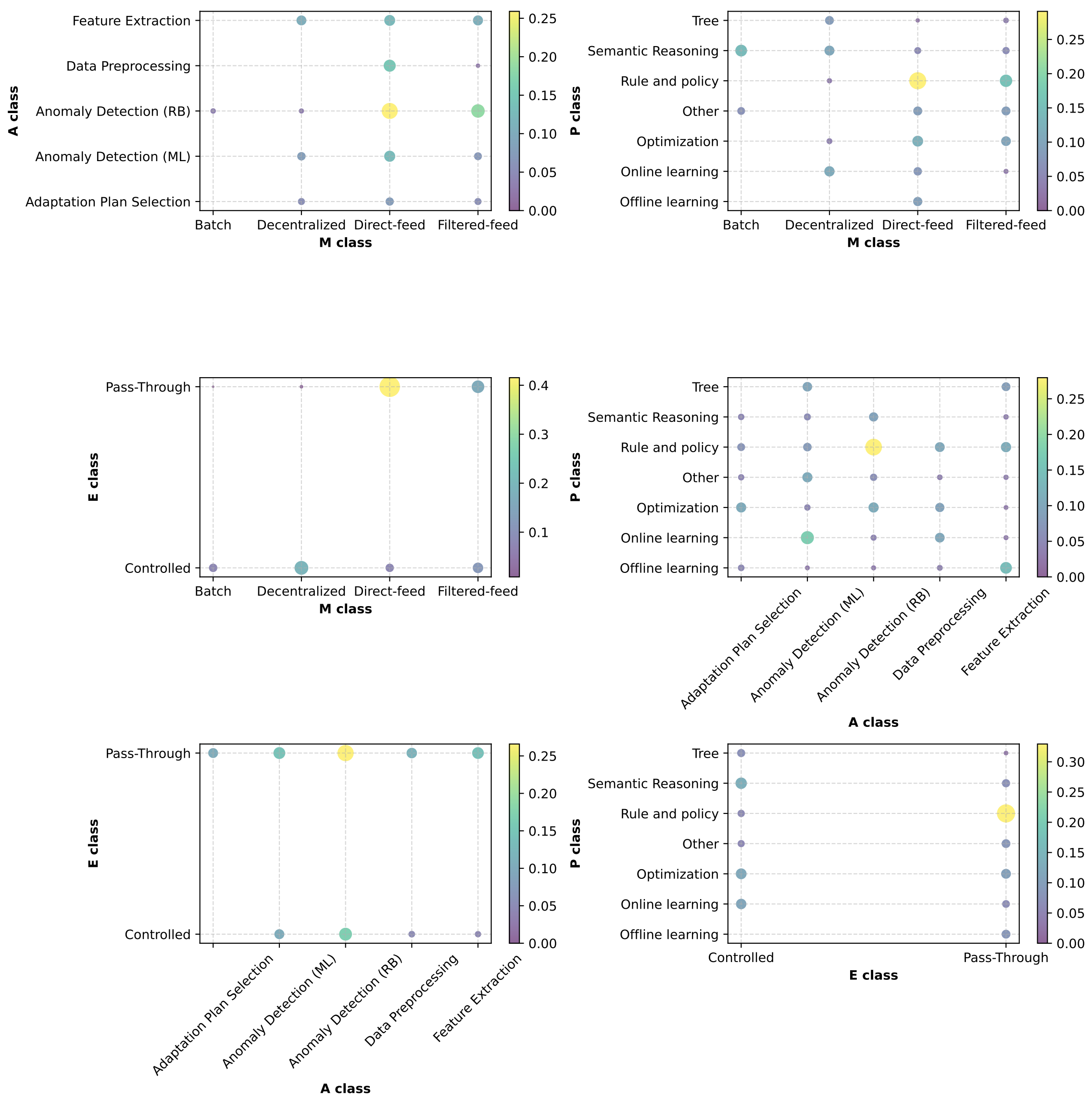


Figure 3: Visualization of contingency tables between MAPE phases (normalized).

2.3 Verification

From the papers that study ways to verify the behaviour of MAPE-K loops, we studied 3 aspects: formalisms, including the modelling and specification languages; verification time, defined as the time to use the verification approach and self-adaptation approach which describes the mathematical abstraction used to model self-adaptation. Results can be seen in fig. 4

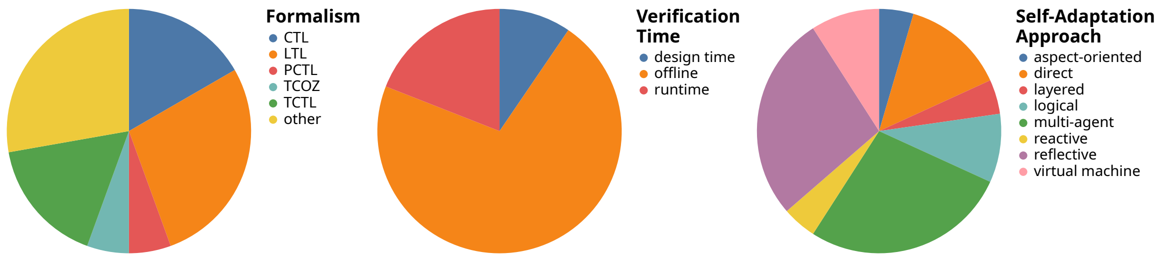


Figure 4: An overview of the papers considered by requirement specification language, time of verification, and the approach taken to modelling self-adaptation.

Funded by the European Union

The majority of works have focused on using the MAPE-K loop to monitor its own behavior, or as a verified monitor for the managed system. Only few works have considered runtime verification of the whole self-adaptive system as a means to provide additional safety guarantees.

The complete survey can be found in Appendix A.1:

- Carlos Isasa, Ziggy Attala, Morten H. Kristensen, Sahar Nasimi Nezhad, Thomas M. Roehr, Robert Scharping, Pavlos Tosidis, Claudio Gomes, Vasileious Moustakidis, Theodoris Manousis, Lukas Esterle, Ana Cavalcanti, Peter Gorm Larsen. "*State of the Art of the MAPE-K Loop: Architecture, Implementation and Verification*", ACM Computing Surveys Submitted 2025

Funded by
the European Union

3 Monitoring for Trustworthiness Checkers in the MAPLE-K loop

To ensure the reliability and safety of self-adaptive systems within the RoboSAPIENS architecture, it is essential to monitor their behavior continuously—even as the systems evolve during operation. Various techniques in runtime monitoring and runtime enforcement have been developed and employed [BFFR18, PFJM14]. These methods allow operators to define properties of the system, that are then monitored at runtime. If a property is violated, the system can either be repaired or the operator can be notified. This allows for the detection of anomalies and the enforcement of safety properties in real-time.

However, these techniques are often limited to a single system hindering application in distributed systems and systems-of-systems. Dynamically changing distributed systems, where new components or systems may be added or removed at runtime, require a centralised component to monitor all relevant elements and components. However, this also requires all systems to register with the central entity as well as the central entity to keep track of the state of all systems, leading to a massive overhead in terms of communication and processing. Additionally, adequate communication networks need to be available. Given the potential dynamics of self-adaptive and self-organising systems, this might not always be given. In turn, this might compromise the adequate intervention in case of property violations due to latency and response times. In addition, the centralised entity may become a single point of failure, leading to a complete breakdown of the monitoring system if it fails or becomes unreachable.

Moreover, since the systems change, it is reasonable to assume that the properties of the system also change. This means that the properties need to be updated at runtime, which is not possible with static monitoring techniques. In addition, the properties may be defined in a way that they are not applicable to all components of the system. This means that the properties need to be adapted to the specific components and their context.

This section introduces the ongoing implementation of Trustworthiness Checkers (TCs) in the MAPLE-K loop, focusing on two core aspects: dynamic property monitoring and distributed monitoring. These approaches address the limitations of static verification methods by allowing monitors to adapt in response to runtime changes and by distributing monitoring tasks across system components. Together, these mechanisms form a foundation for verifying dynamic and decentralized systems, enabling robust and trustworthy self-adaptive behaviors. As part of this effort, we presented a hands-on tutorial on Runtime Verification (RV) for Self-Adaptive Systems (SAS) at the ACSOS conference. The tutorial demonstrated how RV tools can be practically integrated, and its contents are summarised below.

3.1 Tutorial on Runtime Verification for Self-Adaptive Systems

In this section, we summarise the tutorial on RV for SAS presented at the ACSOS 2024 conference. The work consisted of a tutorial paper and a hands-on session during the conference. The tutorial focused on practical techniques for verifying autonomous systems using RV technologies within the context of Digital Twins (DTs). The full tutorial paper is provided in section A.2, and a summary is provided below.

Summary

SAS are increasingly capable of adapting to complex and dynamic environments, but this autonomy complicates reasoning about their correctness. RV offers a lightweight verification approach that can ensure system properties are upheld even during autonomous behavior.

The tutorial demonstrated the integration of RV tools into the *Digital Twins as a Service* (DTaaS) platform, showcasing how monitoring components can be deployed alongside or within DT-based SAS. The use case centered around an *Incubator* DT, a SAS that maintains a stable internal temperature. The system includes anomaly detection and energy-saving services, which respond to events such as the lid of an insulated Incubator box being opened.

Five integration strategies were presented:

- **NuRV FMU Monitor:** A monitor generated as a Functional Mock-up Unit (FMU) for early co-simulation-based validation of system behavior.
- **NuRV FMU Service Monitor:** Reuses the FMU in a service-oriented manner, integrated into the live system via RabbitMQ.
- **NuRV ORBit2 Monitor:** Deploys NuRV as a remote monitoring server accessed via CORBA for fully decoupled verification.
- **TeSSLa Passive Monitor:** Uses the TeSSLa language for passive monitoring based on event streams, with integration via Telegraf.
- **TeSSLa Active Monitor:** Extends passive monitoring with runtime enforcement, allowing the system to adapt in response to detected anomalies.

Attendees were guided through the hands-on deployment of these techniques using the DTaaS platform. By engaging with both NuRV and TeSSLa, participants learned how to practically apply RV to their own systems and gained insight into the trade-offs of different deployment architectures.

The tutorial emphasized cross-community knowledge exchange between researchers in SAS, DTs, and RV. For RoboSAPIENS, the tutorial provided valuable knowledge about exisitng RV tools and their practical application in SAS, which helped with development of the TC.

3.2 Dynamic runtime monitoring

The work presented below is an extended abstract of the paper [KWG$^+$25], which introduces DynSRV, a framework that allows stream runtime verification properties to be dynamically updated. The full paper, which is currently in submission for the Runtime Verification 2025 conference, is included in section A.3.

Funded by the European Union

3.2.1 Introduction

Modern software systems, such as self-adaptive robots, frequently evolve during runtime. Traditional runtime verification struggles in such settings because specifications are assumed to be static. When systems adapt via DSU or autonomously, specifications must also evolve to reflect changing requirements.

To address this, we introduce DynSRV, a Stream Runtime Verification (SRV) language with first-class support for dynamically updating RV properties. It enables properties to be defined, replaced, or refined at runtime, without requiring a complete system restart. This is especially crucial in systems where certain properties rely on extensive historical context, where restarting the runtime verification service would result in the loss of that accumulated state, leading to incorrect verdicts or requiring costly recomputation. DynSRV allows monitors to evolve while preserving internal history, ensuring continuity and correctness in long-running adaptive systems.

In the paper, we provide a motivational example for a scenario where a robot in a manufacturing scenario may need DynSRV.

3.2.2 Language Features and Semantics

DynSRV extends standard SRV with two dynamic primitives, which we call Dynamically Updated Properties (DUPs):

- `defer`(ϕ): delays the specification of property ϕ to a later point, allowing a single update.
- `dynamic`(ϕ): allows ϕ to be updated repeatedly during execution.

In the semantics section of the paper, we define the formal denotational semantics of DUPs using a fixed-point approach over stream contexts. This includes a formal definition of `defer` and `dynamic` in relation to stream namespaces and contexts. It also defines the semantics of our DUP helper functions, `update`(ϕ_1, ϕ_2), `when`(ϕ), and `default`(ϕ, c).

3.2.3 Design Patterns for Adaptation

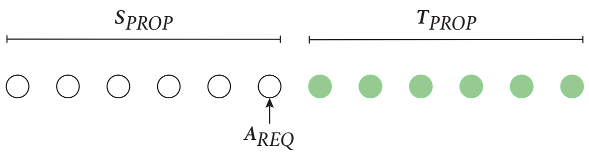
In the paper, we present design patterns for writing specifications that evolve:

- **Open properties:** Unrestricted use of DUPs. Following this pattern, the sender is responsible for ensuring the safety and validity of the provided property.
- **Weaken:** Decrease the restrictions on a requirement using DUPs.
- **Strengthen:** Increase the restrictions on a requirement using DUPs.
- **Refinement:** Replace old rules while ensuring backward compatibility.

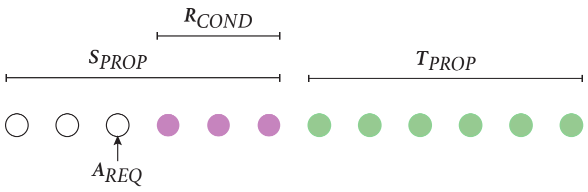
Furthermore, we define design patterns inspired by well-known adaptation semantics from the Dynamic Software Updating (DSU) literature, illustrated in fig. 5. In DynSRV, *one-point adaptation* is the default behavior of DUPs. We provide concrete specification examples for expressing both *guided adaptation* and *over-*

Funded by
the European Union

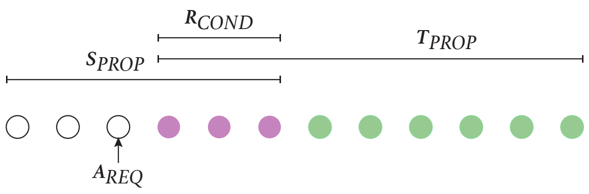
lap adaptation, demonstrating how these semantics can be encoded using DynSRV.



(a) One-point adaptation, where T_{PROP} is used immediately upon arrival.



(b) Guided adaptation, where T_{PROP} is used when R_{COND} is satisfied.



(c) Overlap adaptation, where T_{PROP} is used alongside S_{PROP} until the R_{COND} is satisfied, whereafter only T_{PROP} is used.

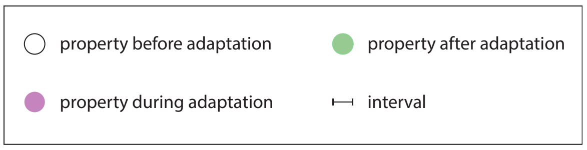

○ property before adaptation
🟢 property after adaptation
🟣 property during adaptation
⊢⊣ interval

Figure 5: Adaptation semantics proposed by Zhang and Cheng, figure adapted from [ZC06] with minor modifications for SRV.

3.2.4 Memory Management Strategies

DUPs introduces unique requirements on the implementation to ensure efficient memory management, as there exists a trade-off between allowing dynamic properties to access historical data and garbage collecting unneeded data. To balance memory usage and trace availability, we explore three memory management strategies with regards to this trade-off:

- **Full history retention:** Ensures solvability but loses bounded memory (BM).
- **Static dependency declarations:** Requires users to annotate expected dependencies.
- **Dynamic dependency graphs:** Dynamically tracks dependencies to preserve BM and expressiveness.

3.2.5 Implementation and Evaluation

DynSRV is implemented in Rust as part of the TC, using a modular architecture supporting both constraint-based and actor-based backends. The actor model handles dynamic dependencies via pub-sub channels, allowing asynchronous updates with minimal runtime overhead.

Performance evaluations demonstrate the ability to process over 100,000 events with property updates in under 500ms. Static properties incur negligible overhead compared to direct monitoring.

Funded by the European Union

3.2.6 Future Work

Future work includes extending DynSRV to support timed, asynchronous, and distributed specifications and evaluating it on RoboSAPIENS case studies.

3.3 Distributed monitoring

Traditional centralised runtime monitors face a number of challenges:

- **Auditing Overhead:** The centralised monitor may introduce a signification bottleneck in the overall system since large quantities of data may need be sent to the centralised monitor for auditing. For example, in the case of handling LiDAR occlusions from a collection of robots with a centralised monitor, all the LiDAR data from all the robots would need to be sent to the centralised monitor alongside their control decisions.
- **Locality of Response and Latency** Many types of failures are localised to a single component or a single subsystem (e.g. a single stage of an individual robot's MAPE-K loop). In these cases, a centralised monitor may not be able to respond to the failure in a timely manner since it must wait for all relevant data to be sent to it before it can make a decision. For example, if an individual robot's planning phase breaks its deadline, it may be appropriate to reject the current adaptation cycle. However, if the robot must wait for the centralised monitor to make a decision, it may be unable to adapt to other anomalies until it has received a verdict.
- **Reliability:** If the only monitor of individual system components is in the form of a centralised monitor, then it is impossible to check reliability of components in the absence of a connection to the centralised monitor. For example, if a robot is monitored by a centralised checker there is no way of specifying that the robot should stop moving if it loses network connectivity, since the checker can no longer communicate with the robot.

We propose a distributed runtime monitoring framework that allows for the specification of distributed runtime monitors. These monitors can be deployed on different nodes of a distributed system. This allows for the monitoring of the system as a whole in the absence of a central entity to keep track of all components. Utilising a stream-based monitoring language, we can express the monitors in a more flexible and expressive way. This allows for the specification of distribution constraints, which define the requirements and constraints on the monitors. Furthermore, stream-based runtime monitors can be programmed, allowing for more complex monitors. This is particularly useful in the context of distributed systems, where the behaviour of the system may change over time and the monitors need to adapt accordingly. To ensure monitors are deployed and operated continuously and without interruption, we also propose a self-adaptive scheduler for migrating monitors to nodes at runtime. These local monitors not only get rid of potential bottlenecks and reduce the overhead in communication but also introduce an additional layer of security as sensitive data can remain local and does not need to be shared across the network.

To achieve this, we present the following ongoing work: First, we propose a stream-

Funded by the European Union

based language to express distributed runtime monitors. This language also allows for the specification of distribution constraints, which define constraints and requirements on the monitors. This can range from minimum or maximum distances, redundancies, or even the number of monitors per node. Second, we provide a monitoring framework that allows for the verification of distribution constraints at runtime using our novel stream-based approach. This ensures continuous monitoring of the system. As nodes and respective monitors might fail, choose to leave the system, or are required by the operator otherwise, our third contribution introduces a self-adaptive scheduler for migrating monitors to nodes. This scheduler takes into account the distribution constraints as well as the current state of the system and its individual nodes. Finally, we present a case study to demonstrate the applicability and feasibility our approach in a multi-robot application focusing on navigation tasks.

3.3.1 Related Work

Formalisms to describe synchronous systems, such as SIGNAL [BLGJ91], and formalisms to specify properties to be monitored of synchronous systems, such as LTL [Pnu77] and LOLA [DSS+05] have evolved in tandem. In the same fashion, distributed systems, and their techniques, have driven the need and development of distributed monitoring. A dedicated scheduling strategy and supporting tools for the distributed implementation of SIGNAL [MLG94] became the basis for a plethora of stream-based runtime verification languages such as LOLA.

In the following, we classify the state of the art in the following three axis, defined in the following subsections: System Under Study; Types of Properties; Monitoring and Distribution Method.

System Under Study The system under study is the system that is being monitored. In this axis we use the adjective adaptive to refer to systems that can change their structure and/or behaviour at runtime.

A distributed system is a system that consists of multiple components that are located on different networked computers, which communicate and coordinate their actions by passing messages to one another. Bauer et al. [BF16] present one of the first algorithms to distribute a Linear Temporal Logic (LTL) formula into subformulas monitored on different components without a central observer. The local monitors collaboratively decide satisfaction or violation of a global LTL property. Mostafa et al. [MB15] employs a three-valued semantics for LTL (true/false/unknown) to handle partial knowledge at each monitor until enough information arrives. The approach ensures that even without a global clock, monitors can collaboratively detect property satisfaction or violation.

Other works consider the spatial location of the components in the system. Example works, in the category of spatially distributed systems, are [BBLN] that introduces STREL (Spatio-Temporal Reach and Escape Logic), a novel logic designed to specify and monitor spatio-temporal properties of mobile and distributed cyber-physical systems (CPS).

Ganguly et al. [GXJ+22] leverages Metric Temporal Logic (MTL) in partially syn-

Funded by the European Union

chronous distributed systems. The authors introduce a progression-based formula rewriting technique: each MTL formula is progressively evaluated over time and reduced to simpler formulas, with checks delegated to an SMT solver. This approach yields a generalized distributed RV algorithm for time-sensitive properties. Yan et al. [YJ22] defines a Swarm Signal Temporal Logic (SwarmSTL) for properties of robot swarms, and develops a decentralized monitoring framework. Each robot runs a local monitor that evaluates the SwarmSTL formula relative to its own observations and those received via consensus from neighbors.

Types of Properties Another axis of our classification is the type of properties that are monitored. The most common properties are temporal properties, such as LTL (as implemented in [CF16]), STL (as implemented in Bonakdarpour et al. [BMNS25]), and MTL (implemented in Ganguly et al. [GXJ+22]), which are used to specify the behavior of systems over time. However, there are also other types of properties that are subsets of temporal languages such as PT-DTL (subset of LTL focused on safety), introduced in Sen et al. [SVAR04], and languages that give more control over what to monitor and the distribution of the monitors, such as LOLA [DS19]. [DS19] introduces a formal model including routing, lazy vs. eager communication strategies, and a novel monitorability condition called decentralized efficient monitorability, ensuring bounded memory use regardless of input trace length. The authors also provide correctness proofs, complexity bounds, and a prototype implementation, dLola, which they evaluate empirically.

Instead of implementing a new language to specify properties to be monitored, Henry et al. [HJMR25] leverages existing automata formalisms which have accepting states for properties that are violated or accepted. They focuses on real-time systems and considers properties specified as reachability conditions on timed automata in a distributed setting. The algorithm dynamically updates the set of possible states of the timed automaton as new events arrive, and computes verdicts as soon as they become conclusive.

Monitoring and Distribution Method It is important to distinguish works that focus on the monitoring of a system under study that is distributed, from works that distribute the monitor itself and "break" it into sub-monitors that are deployed along the distributed system under study components. Most of the works in the state of the art focus on decentralized monitoring, where the monitors are distributed along the components of the system under study. For instance, Audrito et al. [ADS+22] explores branching temporal logic in a distributed context, introducing past-CTL. Monitors for past-CTL formulas are automatically derived and deployed on each device in a network of IoT-like distributed nodes. The collective behavior of these monitors, coordinated through a novel use of the field calculus (an aggregate computing paradigm), ensures the property is checked across space and time without any central coordinator.

Within the decentralized monitoring category, we can distinguish manual distribution from automatic distribution. For instance, the work of Audrito, Colombo, and Yan, et al. [ADS+22, CF16, YJ22] are automated.

Finally, we have found no research publications that perform adaptive distribution,

Funded by the European Union

whereby monitors can migrate from one component to another, or change their distribution strategy at runtime. This is a key feature of our approach, and we believe it is a promising direction for future research.

Summary and Comparison with our Work Table 1 summarizes the state of the art in distributed runtime verification, highlighting the system under study, types of properties, monitoring method, and distribution method, as introduced above. Our work, driven by the need to monitor distributed adaptive systems, is unique in its focus on adaptive distributed runtime verification, where monitors can migrate and adapt their distribution strategy at runtime.

3.4 Planned Case Study

Inspired upon PAL's use case, we consider a scenario involving a team of N autonomous mobile robots operating in a static, enclosed environment. Each robot is initialized in one of the corners in the environment, and is assigned a continuous patrolling task (depicted in Figure 6).

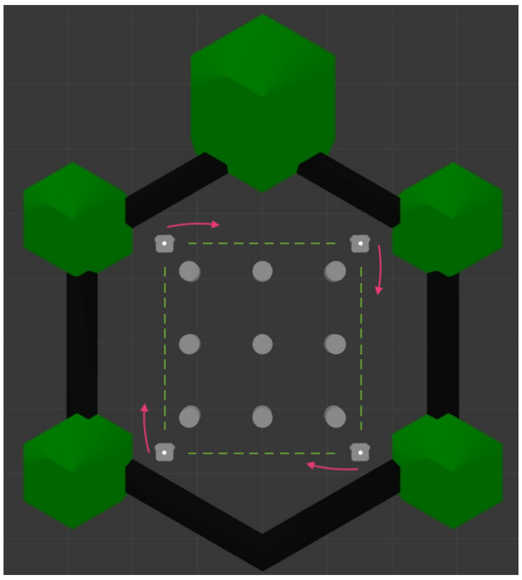


Figure 6: Four robots initialized in Gazebo to perform a patrolling task. The dashed green line indicates the path to be followed, whereas the magenta arrows the direction of movement.

Despite robots having the same navigation objectives, and assuming that they are of the same type, there can still be variability in their behaviour, as a result of network delays coupled with race conditions within the navigation stacks. In addition, motor and more generally hardware imperfections, can lead to different speeds achieved by the robots. Over time, this leads to desynchronization, where faster robots gradually catch up to slower ones, increasing the risk of unsafe situations, and even collisions. Due to the complexity of the underlying kinematic, con-

Funded by
the European Union

Table 1: Overview of monitoring approaches in distributed systems. The asterisk (*) indicates that the specification language is originally introduced in the reference. The last row shows our work.

Paper	**System Under Study**	**Types of Properties**	**Monitoring Method**	**Distribution Method**
[SVAR04]	Distributed Software System	PT-DTL*	Decentralized	Manual
[SS14]	Distributed Software System	PT-DTL	Decentralized	Manual
[MB15]	Distributed Software System	LTL	Decentralized	Manual
[BF16]	Distributed System	LTL	Decentralized	Manual
[CF16]	Distributed System	LTL	Decentralized	Automated
[DS19]	Distributed Systems (Synchronous)	Lola	Decentralized	Manual
[BBLN]	Spatially Distributed Systems	STREL*	Centralized	N/A
[GXJ+22]	Blockchain smart contracts	MTL	Centralized	N/A
[ADS+22]	Distributed Systems (Homogeneous)	past-CTL*	Decentralized	Automated
[YJ22]	Adaptive Kinematic Robot Swarms	SwarmSTL*	Decentralized	Automated
[MAB23]	Distributed System	STL (3-valued)	Centralized	N/A
[BMNS25]	Distributed System	STL	Centralized	N/A
[HJMR25]	Distributed System	Reachability timed automata	Decentralized	Manual
Our Work	Adaptive Distributed System	Lola + distribution	Decentralized, Dynamic	Automated, Adaptive

Funded by
the European Union

trol models, and inherent uncertainty of real world environments, using traditional methods to verify properties regarding safe distances and collision avoidance becomes infeasible. Instead, a distributed monitoring approach would be preferable, where monitors are deployed on the individual robots, enabling the latter to keep track of relevant properties such as safety distances to nearby robots. Moreover, these monitors can be updated at runtime depending on the context, e.g., if a robot enters a room with elderly people, it could be desired to maintain a higher safety distance, which can be reverted to a default lower one, once the robot leaves the room. Distributing the monitors would help in identifying possible issues early on, as the same property is monitored by several robots. Consider robots A and B operating near each other. Both A and B will monitor the distance between them d_{AB}. Inconsistent values for d_{AB} – bigger than an expected error[1] – will indicate potential issues with either A or B, triggering mechanisms for identifying which robot is the source of the problem. Note that, we focus on safety distance, as opposed to collisions, because should the safety distance be violated, there is still time for the system to enter a safe state (e.g., all robots nearby are instructed to slow down or stop).

To reflect realistic deployment conditions, each robot operates based solely on information obtainable through its onboard sensors, such as cameras or lidar, and a preloaded reference map of the environment. This includes knowledge of its own position and the ability to detect nearby robots within a limited visibility range. Crucially, no robot has access to the global state of the entire system; acquiring and maintaining such information would be both bandwidth-intensive and computationally impractical in a distributed setting. As a result, the monitoring approach must be inherently decentralized, relying only on local observations and partial, spatially bounded knowledge of the environment.

We are implementing this scenario in a multi-robot simulation built on ROS2 and the Nav2 stack, with Gazebo as the underlying physics simulator. The environment spawns N TurtleBot3 robots, each equipped with an individual navigation stack operating in localisation mode. A static occupancy map is provided beforehand, removing the need for online mapping.

[1]There will always be some error, due to for example different accuracies of the distance measurement sensors. We assume in this paper that the bounds of this error are known or have been estimated before deployment.

Funded by
the European Union

4 Preliminary Integration of Trustworthiness Checkers

While the previous section covered the preliminary implementation of the TC, with a particular focus on its functionalities, in this section we cover the integration of it with the industrial-sized different case studies.

In this section, we formalize the static and dynamic properties that are monitored by the TC in the different case studies. We first cover traditional static properties, and then we cover dynamic ones. These come from the case studies, in D4.1 [HHHL+24], and the requirements presented in D3.1 [GIK+24].

4.1 Robotic laptop refurbishment case from DTI

This case study consists of a robotic cell that is removing stickers from laptop cases and unscrewing the laptop screen. The stickers are removed by scraping them off with a plastic scraper. They are supposed to be lifted by the plastic scraper and the robot needs to identify the edges of the sticker. For this task it uses machine learning to identify the boundaries of the sticker with the help of a camera.

One of the requirements of the robot's motion is that it should not go beyond the detected boundaries of the sticker. Doing so would potentially collide with nearby objects or people. Formally this static property is expressed as:

$$(\mathit{calcPos}(\mathit{pose}) < sAreaUpper + \delta) \wedge (\mathit{calcPos}(\mathit{pose}) > sAreaLower - \delta)$$

where:

- *pose* is the position and orientation of the robot
- *calcPos* is a function calculating the position of the robot relative to the sticker
- $sAreaUpper$ and $sAreaLower$ are the upper and lower areas of the sticker
- δ is a small tolerance value
- $<$ holds if both the x- and y-values of the left-hand side are less than those of the right-hand side

Another requirement focuses on the unscrew operation of the robot. Here, we wish to ensure that an acceleration limit is not exceeded, but leave this rule open for special cases, e.g., when the robot has failed to unscrew the screw and is trying again. This is a dynamic property that can be expressed as:

$$\alpha < \mathit{screw\_limit} \vee \mathit{dynamic}(\mathit{new\_rules})$$

where:

- α is the current angular acceleration of the screwing tool measured at the robot
- *screw_limit* is the acceleration limit for a typical unscrewing operation

Funded by
the European Union

- *dynamic*(*new_rules*) is a function that evaluates if new rules are in place that allow for a higher acceleration to be applied.

This property can be further reformulated enforce that new rules can only be applied if an attempt has failed. Such formulation looks like:

$$\alpha < \mathit{screw\_limit} \lor (\mathit{dynamic}(\mathit{new\_rules}) \land \mathit{failed\_attempt})$$

where:

- *failed_attempt* is a Boolean variable that indicates if an attempt to unscrew the screw has failed.

4.2 The Robot Navigation Case from PAL Robotics

This case study consists of a robot that is moving in an environment that is shared with other humans.

One of the requirements that is immutable and therefore specified as a static property is that the stopping distance of the robot should always be smaller than the distance kept to nearby obstacles or humans.

$$d - S_O > 0 \;\Rightarrow\; v_R \leq \frac{d - S_O}{T} - v_O$$

where:

- d is the distance to the obstacle or human
- S_O is the stopping distance of the robot
- v_R is the velocity of the robot
- T is the robot response time
- v_O is the approaching speed of the obstacle or human in relation to the robot.

An example of a dynamic property for this case study is that the robot should only accept goals if it has enough battery or during emergencies. For instance if the robot does not have enough battery but it is blocking a crucial passage way then it should accept goals to move elsewhere. Formally this dynamic property is specified as:

$$\mathit{goalAccepted} \Rightarrow (\mathit{battery} > 30\% \lor \mathit{dynamic}(\mathit{emergency}))$$

where:

- *goalAccepted* is a Boolean variable that indicates if the robot has accepted a goal
- *battery* is the current battery level of the robot
- *dynamic*(*emergency*) is a function that allows accepting new emergency rules

Similarly to the laptop refurbishment case, the property can be reformulated to enforce that new rules can only be specified during emergencies.

Funded by the European Union

4.3 Ship Motion Prediction Case from NTNU

This case study consists of moving ship that is making use of a predictive model of the shifts dynamics for future motion planning. A predictive model can be dynamically reconfigured or swapped by a presumably better predictive model. The focus is on the performance of the predictive models, in particular, its accuracy.

An example of static requirement in this case study is that whenever a planning occurs then it must always translate to a new predictive model:

$$\textit{phase} = P \Rightarrow \textit{model} \neq \textit{default}(\textit{model}[-1], \text{"BasicModel"})$$

where:

- *phase* is the current phase of the ship.
- P is the planning phase.
- *model* is the current predictive model.
- *model*$[-1]$ is the previous predictive model.
- *default*(*model*$[-1]$, "BasicModel") uses a basic model initially when no prior model is available.

One dynamic requirement is that the model accuracy must be above a threshold, and can potentially be made stricter[2] based on new knowledge from the MAPLE-K loop or a human:

$$\textit{acc} > \delta \wedge \textit{defer}(\textit{new\_acc})$$

where:

- *acc* is the accuracy of the model.
- δ is a small tolerance value.
- *defer*(*new_acc*) is a function that allows introducing a new stricter accuracy threshold.

4.4 Dynamic Risk Model Case Study from Fraunhofer IFF

This case study consists of a robotic manipulator operating on a cell. Contrary to the case study from DTI, here the focus is on dynamically adapting the risk assessment model of collision of the robot with nearby humans with the help of tracking the position and pose of those humans using a multi camera vision system.

Because the vision system is using machine learning one static property is that human "teleportation" should never occur:

$$\textit{abs}(\textit{humPos}, \textit{humPos}[-1]) < \textit{timeBetween}(\textit{humPos}, \textit{humPos}[-1]) \cdot \delta$$

where:

- *humPos* denotes the current centroid position of the human.

[2]Note that if a less strict threshold is provided, the original threshold dominates the expression

Funded by the European Union

- $\mathit{humPos}[-1]$ is the previous centroid position of the human.
- $\mathit{timeBetween}(\mathit{humPos}, \mathit{humPos}[-1])$ is the time between the two positions.
- δ is a small tolerance value, relating to how fast the average human can move in that workspace.

The dynamic property below demonstrates how Dynamic Risk Models integrate seamlessly into the TC. The property states that when the robot arm is active, the quantified risk must be below a predefined threshold. Initially, this risk is estimated using a simple model based on the robot arm's speed and distance to the human. As more information becomes available, the model can be refined based on information from the MAPLE-K loop or a human expert. Formally, this is expressed as:

$$\mathit{baseModel} = -\mathit{armSpeed} \cdot \mathit{humanDistance}$$

$$\mathit{calcRisk} = \mathit{update}(\mathit{baseModel}, \mathit{dynamic}(\mathit{riskModel}))$$

$$\mathit{armOn} \Rightarrow \mathit{calcRisk} < \mathit{riskThreshold}$$

where:

- $\mathit{baseModel}$ is the base model of the risk assessment.
- $\mathit{armSpeed}$ is the speed of the robot arm.
- $\mathit{humanDistance}$ is the distance to the human.
- $\mathit{calcRisk}$ is the calculated risk.
- $\mathit{update}(\mathit{baseModel}, \mathit{eval}(\mathit{riskModel}))$ is a function that updates the base model to a more detailed model.
- armOn is a Boolean variable that indicates if the robot arm is on.
- $\mathit{riskThreshold}$ is the risk threshold.

Funded by the European Union

5 Conclusion

Section 1 presented the findings of the carried out survey of the state of the art of MAPE-K loops. These results expand the findings reported in D3.1 [GIK+24] and together form the basis for the extension and development of the RoboSAPIENS MAPLE-K loop, as well as the most challenging aspects of the thurstworthiness checker. As as a result, section 3 introduces the ongoing work in dynamic property monitoring and distributed monitoring have been introduced as monitoring implementations for the RoboSAPIENS trustworthiness checkers.

Section 4 shows, how this monitoring for Trustworthiness checkers can be integrated in the four different RoboSAPIENS case studies. Specific properties were therefore developed together with the case studies.

Next steps in work package 3 will focus on the synthesis of verified trustworthiness checkers (task T3.3) and further preparation of integrating trustworthiness checkers into the case studies, through the RoboSAPIENS adaptative platform (task T3.4 and WP5).

Funded by
the European Union

References

[ADS+22] Giorgio Audrito, Ferruccio Damiani, Volker Stolz, Gianluca Torta, and Mirko Viroli. Distributed runtime verification by past-CTL and the field calculus. Journal of Systems and Software, 187:111251, May 2022.

[ANM24] Bert Van Acker, Sahar Nasimi Nezhad, and Paul De Meulenaere. Initial architecture for the RoboSAPIENS platform. Technical report, RoboSAPIENS Deliverable, D5.1, September 2024.

[BBLN] Ezio Bartocci, Luca Bortolussi, Michele Loreti, and Laura Nenzi. Monitoring Mobile and Spatially Distributed Cyber-Physical Systems. In Proceedings of the 15th ACM-IEEE International Conference on Formal Methods and Models for System Design, pages 146–155. RV specifically for spatially distributed systems.

[BF16] Andreas Bauer and Yliès Falcone. Decentralised LTL monitoring. Formal Methods in System Design, 48(1):46–93, April 2016.

[BFFR18] Ezio Bartocci, Yliès Falcone, Adrian Francalanza, and Giles Reger. Introduction to runtime verification. Lectures on Runtime Verification: Introductory and Advanced Topics, pages 1–33, 2018.

[BLGJ91] Albert Benveniste, Paul Le Guernic, and Christian Jacquemot. Synchronous programming with events and relations: The SIGNAL language and its semantics. Science of Computer Programming, 16(2):103–149, September 1991.

[BMNS25] Borzoo Bonakdarpour, Anik Momtaz, Dejan Nickovic, and N. Ege Saraç. Approximate Distributed Monitoring Under Partial Synchrony: Balancing Speed & Accuracy. In Erika Ábrahám and Houssam Abbas, editors, Runtime Verification, pages 282–301, Cham, 2025. Springer Nature Switzerland.

[CF16] Christian Colombo and Yliès Falcone. Organising LTL monitors over distributed systems with a global clock. Formal Methods in System Design, 49(1):109–158, October 2016.

[DS19] Luis Miguel Danielsson and César Sánchez. Decentralized Stream Runtime Verification. In Bernd Finkbeiner and Leonardo Mariani, editors, Runtime Verification, pages 185–201, Cham, 2019. Springer International Publishing.

[DSS+05] B. D'Angelo, S. Sankaranarayanan, C. Sanchez, W. Robinson, B. Finkbeiner, H.B. Sipma, S. Mehrotra, and Z. Manna. LOLA: Runtime monitoring of synchronous systems. In 12th International Symposium on Temporal Representation and Reasoning (TIME'05), pages 166–174, June 2005.

[GIK+24] Claudio Gomes, Carlos Isasa, Morten Haahr Kristensen, Thomas M. Roehr, and MORE. Requirements for safe and trustworthy MAPE-K loops. Technical report, RoboSAPIENS Deliverable, D3.1, September 2024.

Funded by
the European Union

[GXJ+22] Ritam Ganguly, Yingjie Xue, Aaron Jonckheere, Parker Ljung, Benjamin Schornstein, Borzoo Bonakdarpour, and Maurice Herlihy. Distributed Runtime Verification of Metric Temporal Properties for Cross-Chain Protocols. In 2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS), pages 23–33, July 2022.

[HHHL+24] Cathrine Hasse, Stephan Holmberg-Hansen, Ane Nykjær Lemvik, Houxiang Zhang, Guoyuan Li, Mikkel Labori Olsen, Robert Scharping, and Thomas Peyrucains. Case study compendium. Technical report, RoboSAPIENS Deliverable, D4.1, June 2024.

[HJMR25] Léo Henry, Thierry Jéron, Nicolas Markey, and Victor Roussanaly. Distributed Monitoring of Timed Properties. In Erika Ábrahám and Houssam Abbas, editors, Runtime Verification, pages 243–261, Cham, 2025. Springer Nature Switzerland.

[KWG+25] Morten Haahr Kristensen, Thomas Wright, Cláudio Gomes, Lukas Esterle, and Peter Gorm Larsen. Dynsrv: Dynamically updated properties for stream runtime verification. In Runtime Verification. Springer Nature, 2025.

[MAB23] Anik Momtaz, Houssam Abbas, and Borzoo Bonakdarpour. Monitoring Signal Temporal Logic in Distributed Cyber-physical Systems. In Proceedings of the ACM/IEEE 14th International Conference on Cyber-Physical Systems (with CPS-IoT Week 2023), ICCPS '23, pages 154–165, New York, NY, USA, May 2023. Association for Computing Machinery.

[MB15] Menna Mostafa and Borzoo Bonakdarpour. Decentralized Runtime Verification of LTL Specifications in Distributed Systems. In 2015 IEEE International Parallel and Distributed Processing Symposium, pages 494–503, May 2015.

[MLG94] Olivier Maffeïs and Paul Le Guernic. Distributed implementation of SIGNAL: Scheduling & graph clustering. In Hans Langmaack, Willem-Paul de Roever, and Jan Vytopil, editors, Formal Techniques in Real-Time and Fault-Tolerant Systems, pages 547–566, Berlin, Heidelberg, 1994. Springer.

[PFJM14] Srinivas Pinisetty, Yliès Falcone, Thierry Jéron, and Hervé Marchand. Runtime enforcement of regular timed properties. In Proceedings of the 29th Annual ACM Symposium on Applied Computing, SAC '14, page 1279–1286, New York, NY, USA, 2014. Association for Computing Machinery.

[Pnu77] Amir Pnueli. The temporal logic of programs. In 18th Annual Symposium on Foundations of Computer Science (SFCS 1977), pages 46–57, 1977.

[SS14] Torben Scheffel and Malte Schmitz. Three-valued asynchronous distributed runtime verification. In 2014 Twelfth ACM/IEEE Conference on Formal Methods and Models for Codesign (MEMOCODE), pages 52–61, October 2014.

Funded by the European Union

[SVAR04] K. Sen, A. Vardhan, G. Agha, and G. Rosu. Efficient decentralized monitoring of safety in distributed systems. In Proceedings. 26th International Conference on Software Engineering, pages 418–427, May 2004.

[YJ22] Ruixuan Yan and Agung Julius. Distributed Consensus-Based Online Monitoring of Robot Swarms With Temporal Logic Specifications. IEEE Robotics and Automation Letters, 7(4):9413–9420, October 2022.

[ZC06] Ji Zhang and Betty H. C. Cheng. Using temporal logic to specify adaptive program semantics. Journal of Systems and Software, 79(10):1361–1369, October 2006.

Funded by
the European Union

A Appendix

A.1 State of the Art of the MAPE-K Loop: Architecture, Implementation and Verification

The appended paper follows.

Funded by
the European Union

State of the Art of the MAPE-K Loop: Architecture, Implementation and Verification

CARLOS ISASA, Aarhus University, Denmark
ZIGGY ATTALA, University of York, United Kingdom
MORTEN H. KRISTENSEN, Aarhus University, Denmark
SAHAR NASIMI NEZHAD, University of Antwerp, Belgium
THOMAS M. ROEHR, Simula Research Laboratory, Norway
ROBERT SCHARPING, Fraunhofer Institute for Factory Operation and Automation IFF, Germany
PAVLOS TOSIDIS, Aristotle University of Thessaloniki, Greece
THOMAS WRIGHT, Aarhus University, Denmark
CLAUDIO GOMES, Aarhus University, Denmark
VASILEIOS MOUSTAKIDIS, Aristotle University of Thessaloniki, Greece
THEODOROS MANOUSIS, Aristotle University of Thessaloniki, Greece
LUKAS ESTERLE, Aarhus University, Denmark
ANA CAVALCANTI, University of York, United Kingdom
PETER GORM LARSEN, Aarhus University, Denmark

Adding autonomy to systems is at the core of the transformation that industry is currently undertaking, with human interaction being taken out of the loop whenever possible. It is important that this revolution is carried out safely for humans and all elements of the environment in which the autonomous systems operate. So, for such systems to adapt to unexpected circumstances, a systematic approach to safety is needed. The most well-known architectural pattern for adaptation is the MAPE-K loop. This survey reports on a systematic review of more than 300 publications about MAPE-K. Our main contributions are (1) an analysis of the alternative architectural patterns and extensions of MAPE-K; (2) a categorisation of the alternative design approaches for each of the MAPE-K phases; and (3) an analysis of the different approaches taken to ensure that the MAPE-K autonomy is safe and thus can be trusted. We further identify research gaps and provide a roadmap for future research to enable safe self-adaptation in autonomous systems of the future.

Authors' addresses: Carlos Isasa, Aarhus University, Aarhus, Denmark, cisasa@ece.au.dk; Ziggy Attala, University of York, York, United Kingdom, ziggy.attala@york.ac.uk; Morten H. Kristensen, Aarhus University, Aarhus, Denmark, mhk@ece.au.dk; Sahar Nasimi Nezhad, University of Antwerp, Antwerp, Belgium, sahar.nasiminezhad@uantwerpen.be; Thomas M. Roehr, Simula Research Laboratory, Oslo, Norway, roehr@simula.no; Robert Scharping, Fraunhofer Institute for Factory Operation and Automation IFF, Magdeburg, Germany, robert.scharping@iff.fraunhofer.de; Pavlos Tosidis, Aristotle University of Thessaloniki, Thessaloniki, Greece, ptosidis@csd.auth.gr; Thomas Wright, Aarhus University, Aarhus, Denmark, thomas.wright@ece.au.dk; Claudio Gomes, Aarhus University, Aarhus, Denmark, claudio.gomes@ece.au.dk; Vasileios Moustakidis, Aristotle University of Thessaloniki, Thessaloniki, Greece, vmousta@csd.auth.gr; Theodoros Manousis, Aristotle University of Thessaloniki, Thessaloniki, Greece, thodorisman@gmail.com; Lukas Esterle, Aarhus University, Aarhus, Denmark, lukas.esterle@ece.au.dk; Ana Cavalcanti, University of York, York, United Kingdom, ana.cavalcanti@york.ac.uk; Peter Gorm Larsen, Aarhus University, Aarhus, Denmark, pgl@ece.au.dk.

ACM XXXX-XXXX/2025/6-ART
https://doi.org/10.1145/nnnnnnn.nnnnnnn

ACM Reference Format:
Carlos Isasa, Ziggy Attala, Morten H. Kristensen, Sahar Nasimi Nezhad, Thomas M. Roehr, Robert Scharping, Pavlos Tosidis, Thomas Wright, Claudio Gomes, Vasileios Moustakidis, Theodoros Manousis, Lukas Esterle, Ana Cavalcanti, and Peter Gorm Larsen. 2025. State of the Art of the MAPE-K Loop: Architecture, Implementation and Verification. 1, 1 (June 2025), 35 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Autonomous systems can adapt to changes in their environment without human involvement. Governed by autonomous software agents, these systems can be employed in a wide range of applications, even in collaboration with humans, with the potential of transforming society at large. Over the past 40 years, various architectures have been proposed to enable engineered systems to adapt to their dynamically changing environments. Some examples are the subsumption architecture [35], BDI agents [143], or the LRA-M loop [104].

Our focus in this paper is a leading architectural pattern called the MAPE-K loop [94]. It is characterised by components to Monitor, **A**nalyse, **P**lan and **E**xecute over shared **K**nowledge, to govern all kinds of computational systems and adapt its actions, resources and goals at runtime. With a history of over 20 years, MAPE-K has been applied in a wide range of applications.

A MAPE-K loop is typically associated with a managed system. Yet, the type of managed system varies greatly, ranging from distributed systems, such as cloud-based solutions, down to a very narrowly scoped subsystem. Deud Guimarães and De Almeida Neris [56] show that even a human or rather the human-state can be a target of control. In their work, the emotional state of the user, which is influenced by the changing presentation of a user interface, is controlled. Table 1 summarises the main application domains in the literature. In some areas of application, such as robotics, authors do not focus on describing MAPE-K aspects or compatibility of their software design. Yet, feedback-control loops, widely used in robotics and other areas, can be seen as an application of a semantically equivalent concept (cf. Rutten et al. [146]).

Due to the vast research on self-adaptive and autonomous systems, and on MAPE-K specifically, there are numerous surveys reviewing specific areas covering this topic. We can cite, for example, architecture-based self-adaptation [164], machine learning in self-adaptive systems [79], and decentralised adaptation using the MAPE-K loop [141]. However, with the ability of a system to adapt to a changing environment, protection of humans and other elements of the environment becomes a concern. A comprehensive survey of the MAPE-K loop and its individual components with a dedicated focus on ensuring safety in adaptation processes is currently missing.

This article fills this gap by offering the following contributions.

(1) We present a comprehensive overview of research and work on the MAPE-K loop by systematically surveying more than 300 publications.
(2) We provide insights on different implementations of each of the MAPE-K components, and highlight the requeriments of each implementation technique. This provides practitioners with a valuable overview of existing implementations and approaches, and the required and desired boundaries they impose on a system that adopts them.
(3) We cover the approaches for verification of adaptions of an autonomous system governed by a software agent implemented using a MAPE-K framework. We present an overview of available techniques and application-specific implementations.
(4) We outline a roadmap for future research to bridge current research gaps, combining adaptation and safety for trustworthy self-adaptation in future autonomous systems.

The structure of the paper is as follows. Section 2 explains the methodology followed in this survey together with the research questions we present. Section 3 introduces the MAPE-K architecture and covers different alternative approaches of MAPE-K realisations in regard to design, integration,

Table 1. Application domains of MAPE-K control loops

Domain	References
cloud / provisioning, networks	[1, 43, 50, 114, 119, 121, 123, 169]
workflow, compute optimization IoT, industry 4.0, production systems	[26, 28, 52, 106],
CPS control loop, robotics, greenhouse control	[44, 58, 72]
smart city: traffic management, health management	[7, 27]
architecture (adaption, e.g., Service Oriented Architecture)	[11, 133]
security, monitoring: security source, application-layer attacks, network	[17, 20, 63]
business process management	[83, 147, 150]
monitor service level agreements	[43]

and further properties. Afterwards, Section 4 presents the main findings regarding the alternative patterns of the different stages of MAPE-K. This is followed by Section 5 examining the surveyed publications for verifying the trustworthiness of the MAPE-K loops. In Section 6, we return to all the research questions, and we discuss gaps in the literature that could be addressed in the future. We conclude in Section 7, where we summarise and give pointers for future work.

2 METHODOLOGY

Our study follows the three macro-phases of systematic mapping studies defined in the literature [99], namely, planning, conducting, and reporting. Figure 1 describes the design of our study, which has five phases numbered 1 to 5. Phase 1 is planning; Phases 2–4 correspond to the conducting macro-phase, and finally Phase 5 corresponds to reporting.

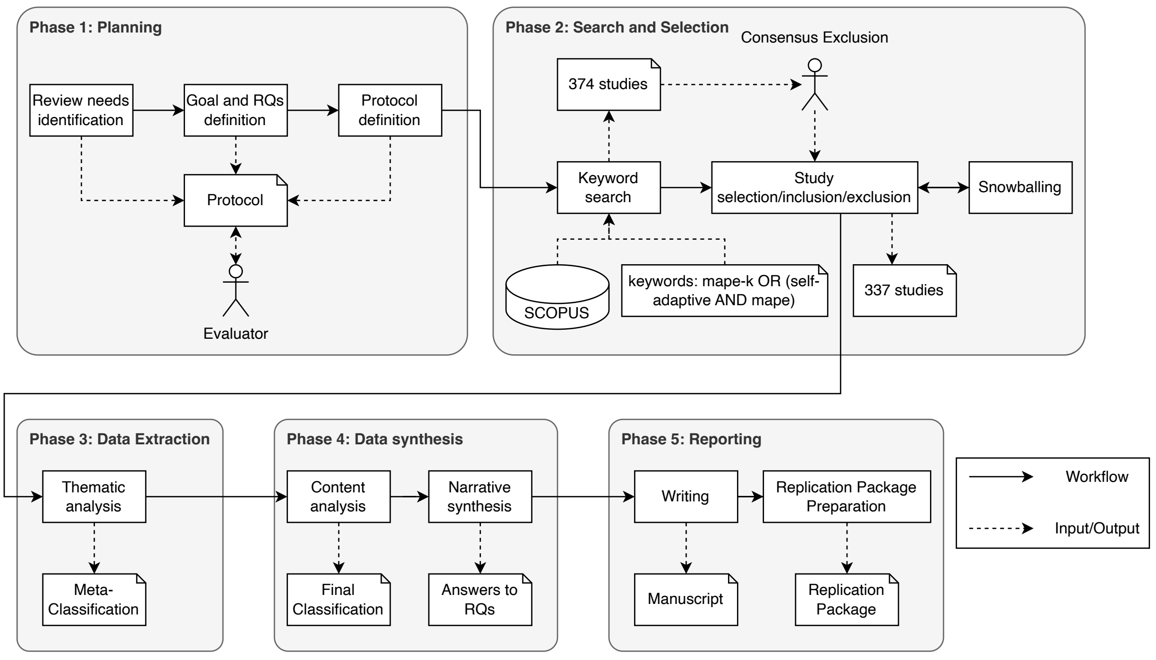


Fig. 1. Study Design. Adapted from [12].

Table 2. Research Questions

RQ1	What possible ways are there to implement the M/A/P/E/K component of the MAPE-K loop and how can they be classified?
RQ2	Under what conditions are each of the main implementation techniques for MAPE-K phases suitable?
RQ3	What are the existing verification approaches in the literature?
RQ4	How are the different implementations of the MAPE-K phases related?
RQ5	What are the most common architectural patterns and properties for MAPE-K loops?

Phase 1 started by identifying gaps within the existing surveys, namely, topics that are not the target of a systematic and exhaustive review of papers related to the MAPE-K loop and need to be covered. Phase 1 then proceeded with the definition of our goals and Research Questions (RQs). These tasks were all supported by an evaluator not in the core group, that is the group of authors in charge of reading and classifying the papers. The outcome of Phase 1 was a research protocol, defining the study's context, goals, and RQs listed in Table 2. The protocol underwent internal evaluation by co-authors experienced in systematic studies. Feedback from this evaluation was used to refine the protocol, addressing identified threats to validity.

Phase 2 aimed to identify primary studies relevant to, or concerned with, MAPE-K loops in self-adaptive systems. First, a keyword-based search was conducted. The database of choice was Scopus and the search string was `TITLE-ABS-KEY (mape-k) OR (self-adaptation AND mape)`. Other inclusion criteria were studies written in English and published in journals and conference proceedings in the fields of computer science or engineering. In this way, 374 studies were found.

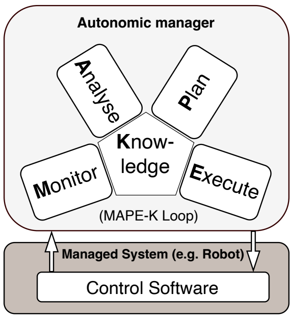
Exclusion decisions were then carried out by consensus among the authors. This process removed studies not explicitly addressing MAPE-K loops, such as those that only make high-level references to MAPE-K (for instance, [34]) or position papers about the use of MAPE-K loops in unrelated disciplines (for instance, [160]). The paper count after this step was 293.

Finally, snowballing was conducted to supplement the initial search, resulting in 54 more studies cited by and citing the primary studies identified. This process was done by checking both the references and the citations of papers with more than 20 citations. Finally, an additional criteria excluded papers published before 2010 to maintain a dataset that was both manageable and still relevant. In total 337 studies were included in the final dataset used for data extraction and synthesis.

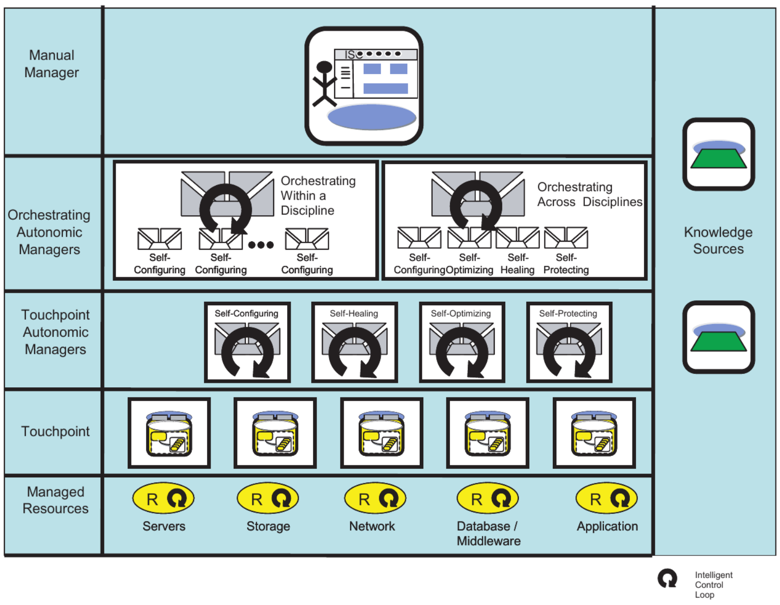
Phase 3 aimed at collecting information from each paper relevant to the research questions, such as implementation details, extensions, and verification approaches. Based on the topics presented, a thematic analysis led to the meta-classification of verification, implementation and architectural extension papers. This early meta-classification is the output of this phase.

Phase 4 consisted of content analysis and narrative synthesis. First, in the content-analysis step, the papers were read and taxonomies for the different meta-classes were developed. Next, in the narrative-synthesis step, quantitative and qualitative techniques were used to identify trends, patterns, and gaps in the literature. Findings were mapped to the research questions, providing answers with actionable insights for practitioners and researchers.

Finally, in Phase 5 the results were then compiled into the current manuscript, detailing the methodology, findings, and implications. A replication package, including the extracted data, has been made publicly available to ensure transparency and reproducibility.



(a) MAPE-K loop in an autonomic element (adapted from [107])



(b) IBM Reference Architecture - Source: [85]

Fig. 2. The MAPE-K reference architecture and its phases.

Threats to Validity. Potential biases, such as language and keyword-selection biases, were mitigated by iterative refinement of the search strategy and snowballing. For instance, snowballing revealed 54 papers that were about MAPE-K but did not include the keywords of the search formulated in the title, abstract, or keywords. Regular meetings of the authors for sharing and reviewing results minimized subjective biases during all phases. The dynamic nature of the field, with an increasing number of publications, poses a threat to capturing all relevant studies. Snowballing and periodic reviews of classification criteria addressed this issue. All records were maintained for transparency.

3 ARCHITECTURE FOR MAPE-K SYSTEMS

Traditional control systems (along with their more sophisticated supervisory (layered) [117] and model-based [84] extensions) can be considered as already providing support for self-adaptive behaviour. However, their space of adaptations is carefully controlled and restricted at design time. In contrast, the MAPE-K loop, depicted in Fig. 2a, is a well-recognized engineering approach [22] for structuring software in self-adaptive systems. Fig. 2a shows an autonomic element, that is, an individual system containing resources and delivering services to humans and other autonomic elements. Each autonomic element includes a managed component, described as a Managed System (Robot) in Fig. 2a, and an Autonomic manager. There are various approaches to implement a MAPE-K control loop. In this section, we therefore give an overview of the structure of existing system architectures that take advantage of one or more MAPE-K control loops.

In Section 3.1, we describe the default MAPE-K control loop, and analyse the standard approach to its implementation with respect to the original proposal by IBM [85]. (Design of the individual phases is discussed in Section 4.) Due to the lack of a strict specification, researchers and practitioners have come up with several interpretations of MAPE-K-based control loops. Hence, Section 3.2 presents a variety of ways in which MAPE-K loops are designed and built into larger architectures. This includes, for instance, approaches where MAPE-K loops define an overall system architecture, and where multiple control loops interact and take advantage of additional functional layers. In doing so, we describe typical implementation patterns for MAPE-K based architectures. Subsequently, we discuss architectures that extend the phases of the MAPE-K loop in Section 3.3. Section 3.4 gives an overview of architectural properties, and Section 3.5 concludes with implementation frameworks.

3.1 Control-loop design

The basic layout of the MAPE-K loop has originally been proposed by Kephart and David M. Chess [94], and detailed in a whitepaper [85]. Those works list the four phases and the knowledge component, but give only loose descriptions thereof, quoted below.

- Monitor: "collect, aggregate, filter and report details (such as metrics and topologies) collected from a managed resource" - it performs these activities "until it determines a symptom that needs to be analysed".
- Analyse: "correlate and model complex situations [...] to learn about the IT environment and help predict future situations" - the main purpose of this component is "to determine if some change needs to be made".
- Plan: "construct actions needed to achieve goals and objectives [..., using] policy information to guide its work".
- Execute: "control the execution of a plan with consideration for dynamic updates".

The authors indicate that all phases take advantage of a knowledge component, that allows storage and access to data. The term data is used by the authors in a very abstract sense, so that the detailed methods of storing and accessing it remain unspecific. Likewise, other phases of the MAPE-K loop are also not strictly specified, and moreover, the authors of [85] do not imply a strict monitor-analyse-plan-execute control flow, and consider also the partial implementation of these phases. So, implementing only the monitor phase for one subsystem is possible, for example.

The predominant, and thus standard implementation structure of MAPE-K is a single control loop using the monitor-analyse-plan-execute phases, with examples given by [49, 51, 57, 158, 162, 177]. For these typical MAPE-K implementations, the component structure goes hand-in-hand with the control flow, where the execution order is: monitor, analyse, plan, and lastly execute.

In the next sections, we cover variations developed over the years.

3.2 Architectural Patterns

Individual MAPE-K phases can be implemented by one or multiple components. The possible cardinalities are listed below as mappings to the real components.

n:1 all phases are implemented in a single component. This pattern is called monolith (single program) [51].

n:n each phase has one component as counterpart in the implementation [9, 30].

n:m each phase has one or many (coordinated) components implementing it [11].

Calinescu et al. [39] consider also the redundant implementation of a single phase, which we note as **n:m[+]**, meaning n:m in 1 or more parallel implementations. Meanwhile, some authors likewise merge the plan and execute phases [11], or leave out a phase and call the MAPE-K based approach "lightweight" [31]. Thus, we use **n:m[-]** to denote merged or absent phases.

As soon as multiple components are used to implement a MAPE-K loop, they can be realised in a distributed way, with components running on multiple devices. Albassam et al. [11], for instance, present a distributed MAPE-K architecture, where the monitor is represented as an "Architecture Discovery Layer", which communicates detected node failures to a "Configuration Maintenance Layer". An "Application Recovery Layer" combines the plan and execute components of a MAPE-K loop, and is responsible for recovery of the overall component-based architecture. Overall, the literature points to multiple patterns: (i) **single/monolith** with no distribution, since there is only one component [51]; (ii) **single/component-based** with a single system with a MAPE-K loop, but implemented by multiple components [82, 119, 176, 178]; and (iii) **distributed/component-based** where MAPE-K runs across multiple systems, implemented by multiple components [7, 11, 27, 122].

The monolith pattern is the most restrictive as it allows no distribution, but a related decomposition is still possible, e.g., by running phases in separate threads [98]. For the remaining cases of component-based implementations, complexity increases as soon as a phase requires multiple components (n:m), and when redundancy is introduced by parallel structures [+].

For the control flow it is important to know how the coordination between phases is done. Most descriptions of implementations do not detail this aspect, so that the default has to be assumed, that is, a sequential control flow in which the monitor identifies an anomaly and triggers the analyse phase, which triggers the plan phase, which finally triggers the execute phase.

Riccio et al. [144] explicitly adopt asynchronous execution of the monitor phase, which then triggers an adaptation loop, without detailing if a conflicting overlapping execution of an adaptation loop could be triggered or how it is handled. An important safety-relevant question arising from this is how anomalies that occur during the handling of another anomaly are handled. Hoffmann et al. [83] mention at least the unique identification of anomalies, so that redundant triggering of an analysis from the monitor is prevented. Faraji-Mehmandar et al. [70], on the other hand, describe the periodic execution of their implemented adaptation loop.

Several works, including the original description of MAPE-K [85], consider architectures that combine several instances of a MAPE-K loop. The original approach already uses several MAPE-K loops in the autonomic manager as shown in Fig. 2b. This architecture has five layers. The bottom layer is defined by resources (servers, databases, and so on) managed through interfaces exposed via an interface layer named Touchpoint. These interfaces can be used by autonomic managers in an additional layer named Touchpoint Autonomic Managers.

An autonomic manager can, for instance, perform central coordination of control loops responsible for self-configuration. Another option is for an autonomic manager to coordinate loops with diverse responsibilities: self-configuration, self-protection, self-healing, and self-protecting (also referred to as self-x properties). The authors of [85] refer to the former strategy as "within a discipline" and the latter as "across discipline", where discipline refers to one of the self-x properties.

The second layer from the top, called Orchestrating Autonomic Managers, relies on a hierarchical setup of MAPE-K loops for orchestration. This is guided by a policy defined by goals and objectives that a human operator can set. Hence, orchestration does not need to be fully autonomic. Instead, the top layer of this architecture, Manual Manager, allows a human in the loop.

All layers use common Knowledge Sources for exchange of information.

The use of MAPE-K loops in a hierarchical composition has been suggested by Portocarrero et al. [137]. This pattern is referred to as "tower of adaptation" [27, 36]: MAPE-K loops can be stacked in such a way that a MAPE-K loop can act as managing element for a child MAPE-K loop, while being managed by another parent MAPE-K loop itself.

In many later works, a MAPE-K loop does not define the full architecture of a complex system, but is itself embedded into an enclosing architecture. Weyns and Iftikhar [165] provide one such example: the three-layered architecture ActivFORMS. It comprises a "goal management" layer as interface to external operators, while a "change management" layer comprises all MAPE-K loops. In this approach, the goal and change management layers form the managing system, and the "managed system" layer completes the architecture as a third layer.

A combination of multi-agent systems with MAPE-K is illustrated by Müller et al. [122], and Aguilar et al. [7], while Doran et al. [58] merge MAPE-K with IMD (Intelligent Machine Design). De Benedictis et al. [55] and Feng et al. [72] embed a digital twin in the control loop.

The distribution of a MAPE-K loop can increase the complexity of the control mechanism by itself, due to the need for additional communication and coordination between the phases. For a set-up of multiple MAPE-K loops, an additional coordination and synchronization challenge arises, which Gerostathopoulos et al. [76] address by introducing explicit mechanisms to perform conflict resolution and multi-layer control. This complexity is handled in parts by hierarchical MAPE-K loop setups. where one or more MAPE-K loops are controlled by another MAPE-K loop. Such setup as illustrated by Gerostathopoulos et al. [76] allows, for instance, to activate, deactivate or even add control strategies - all again encoded by MAPE-K control loops.

The coordination of multiple MAPE-K loops requires communication, and for the communication flow between multiple MAPE-K loops, we can identify the following connection and responsibility patterns: (i) **sequential, unidirectional**, with multiple MAPE-K loops connected, with control of communication unidirectional; (ii) **parallel, bidirectional**, with multiple MAPE-K loops running in parallel and communicating with each other, e.g., to adjust parameters; (iii) **parallel, independent**, with multiple MAPE-K loops running, but not influencing each other directly; and (iv) **hierarchical, bidirectional**, with multiple MAPE-K loops forming a hierarchical communication structure, where the parent MAPE-K loops exert more responsibility than any successor loops.

The need for multiple MAPE-K loops and component distribution for federated systems can arise from performance constraints. In addition, it is useful to maintain modularity of the implementation.

3.3 Architectural Extensions

Some authors have extended the MAPE-K architecture, instead of adapting it to their domain via different implementation strategies. We can classify the adaptations into two types, each for different purposes. First, some authors have introduced sub-phases to provide more details regarding the existing five phases to the MAPE-K loop. For example, the techniques presented by Quin et al. [140] and Camelo et al. [40] introduce new phases to enable machine learning to be utilised inside the knowledge components; the intent of these phases is to provide more details regarding the knowledge component, so they are sub-phases. Second, some authors have introduced phases with a purpose separate to any of the four existing phases; we refer to these phases as top-level phases. In this section, we consider only top-level phases, taking the view that sub-phases are used for supporting the implementation of a MAPE-K loop, not for extending it.

Informative Phases. The first type of top-level phase is one whose results inform, or assist, the operation of an existing phase by performing other functionality. For example, Karaduman et al. [93] introduce a Fuzzy Logic Controller phase to help the operation of the execute phase. In this work, the modified MAPE-K loop is used to specify a BDI (Belief, Desire, Intent) agent [75]. The extra Fuzzy Logic Controller phase, together with the planning phase, help provide consistent reasoning about the BDI agent's beliefs when the environment exhibits uncertainty.

Ali [13] similarly adds a phase that assists with the analysis: the Analytical Framework phase that obtains data from the analysis component, and then develops models using key performance indicators, visualization, and machine learning (ML) algorithms to add to the knowledge component. These models also help the analysis component to determine if there is an anomaly. The Analytical Framework phase is distinct from a sub-phase of analysis because the Analytical Framework updates the knowledge with new models, which the analysis phase does not in their technique. Gheibi and Weyns [78] take a similar approach, they introduce a Verifier phase that updates learning models in the knowledge component based on the results of the analysis phase.

Finally, Abdennadher et al. [2] add a Decision phase, connected to the output to each of the analysis, planning, and execute phases. If any such phase reaches a point where they have a decision to make, the Decision phase guides the choice based on information about the managed system obtained at runtime, such as how the tolerance of the hardware to physical changes. Jamshidi et al. [89] similarly introduce an auto-adjuster phase that receives data from the monitor phase, and uses this to dynamically update the parameters of all other phases depending on the context.

Recording Phases. There is another type of top-level phase mainly for recording, analysing, and communicating information about the MAPE-K loop itself, similar to a reflective operation. Yaghoobirafi and Farahani [170] specify the DisMAPE-K loop, a distributed version of the MAPE-K loop that enables multiple MAPE-K controllers to interact and work together. For that, the DisMAPE-K loop has an Entity phase in each MAPE-K loop, whose purpose is to report on its managed system of this MAPE-K loop to the wider system.

Another type of reflective phase is presented by Annighöfer et al. [21], who add a Virtual Certification phase. It follows the plan phase, but acts before the execute phase. The Virtual Certification phase ensures that the software still conforms to pre-defined (aviation) standards, and ensures the system configuration still conforms to them following adaptation.

Reflexive Action Phase. Liess et al. [111] specify a MAPE-K loop with a single top-level phase added: the Reflexive Action phase. It receives data from the managed system via the monitor, and if predefined trigger conditions are met, then the Reflexive Action communicates a course of action directly to the execute phase, skipping all other phases entirely. These trigger conditions represent states where immediate adaptation is required, for example when a physical system may imminently collide with an object. This architecture is designed for real-time and low-latency environments, where reaction time and liveness properties are critical considerations.

Rule Augmentation Phases. In [100, 101], Verena Klös presents approaches to include new rules in the plan component of a MAPE-K loop using additional phases. The rules are defined based on the distance of the system, represented by K_{Sys} in the knowledge component, from a goal, captured by K_{Goal}, a model of that goal in the knowledge component. Another K_{Adapt} component represents the current adaptation rules, and further aids in the definition of the new rules.

Klös et al. [100] add two top-level phases to the MAPE-K loop: the Learner and Evaluation phases. Here, the analyse phase decides if the system has strayed too far from its goal. If so, then the Evaluate phase can either remove rules from the system, or signal the need for further rules via a flag. In the plan phase, if such a flag has been raised, then, in the Learner phase, a learning component attempts to learn another rule; if successful, this rule is recorded in the K_{Adapt} component.

In contrast, in [101], instead of the analyse, the monitor phase communicates with the Evaluation phase, which itself determines if the distance to the goal is too large, and can disable or adjust rules, as needed. Further, this approach contains a Verification phase, which establishes system properties considering the new rule. The rule is added only when this verification passes.

3.4 Architectural Properties

Software architectures can come with different properties to ensure safety of the systems. State-of-the-art of architectures using MAPE-K-loops in particular focus on the following properties.

- Conformance and compatibility: alignment with existing reference architectures, e.g., RAMI 4.0 [167].
- Security: the system is not compromised by using secure communication and source verification.

- Fault tolerance: support for generic fault handling strategies, e.g., to allow for handling of unforeseen faults, mostly domain-specific.
- Verifiability: embedded means for validation and verification (cf. Section 5).

Security is addressed in a small number of implementations and with the main focus on ensuring that the MAPE-K loop is not compromised, e.g., by estimating the trustworthiness of sources [124] and secure message exchange [153]. All MAPE-K based implementations come inherently with a well-specified tolerance to faults in the managed system, since this is one of the design goals of MAPE-K. Verification and security functions can be considered preventive measures against builtin or intentionally triggered faults. Security breaches might still be encountered, and – when detected – handled as faults by following, for instance, a dynamic policy update to prevent future incidents.

Safe adaptation is, however, discussed in very few works. Prenzel and Steinhorst [138] use MAPE-K in the context of safe software updates of industrial control systems and an implementation of IEC61499. Their aim is to reduce the time between identification of an anomaly and implementing the adaption. Thus, they group the first three MAPE-K phases into a decision-making phase and allow to detail the execution phase using the IEC61499 standard (cf. Section 4.4).

The type of trigger, or rather anomaly, for a change varies with the domain and is typically related to direct sensor data. Namal et al. [124] present an architecture in which a derived attribute is monitored; they implement a MAPE-K loop that takes decisions based on trust levels of data sources. They introduce a “Trust Management Service Layer” that receives input for possibly distributed “Trust Agents”. Namal et al. suggest to compute the trustworthiness of sources by measuring availability, reliability, irregularities and capacities. Typically, the monitor triggers a response to an external event. Seiger et al. [150] illustrate how to use a MAPE-K loop to monitor the triggered adaptation itself. In their architecture, the monitor measures “cyber-physical consistency”, defined as the delta between the assumed physical state and the actual physical state of a cyber-physical system (CPS) during adaptation. To fix an encountered issue they escalate to a human.

3.5 Implementation Frameworks

Several frameworks have been developed to facilitate the development of MAPE-K based systems. Pfannemüller et al. [134] offer a framework to implement adaptive systems, targeting mainly IoT systems. They base their work on Clafer [90] to model structure and behaviour.

Similarly Krupitzer et al. [105] use a context manager that provides an abstraction layer between low-level data formats and the control logic. Communication between phases uses a publish-subscribe mechanism and introduces information categories. When mapped to robotics, it resembles the infrastructure established by the Robot Operating System (ROS) [159].

Souza et al. [156] present a framework to adjust the difficulty of a game via a MAPE-K loop. They outline a generic architectural concept for developing a MAPE-K-based system operating with rule-based reactions. Souza et al. [156] consider parametrization of the number of sensors used, the monitoring frequency, and provide interfaces to define anomalies and reaction rules.

Next, we consider the works from the point of view of particular phases of a MAPE-K loop.

4 MAPE-K PHASES

Here, Section 4.1 to 4.5 cover each of the phases of the MAPE-K loop.

4.1 Monitor

The monitoring phase gathers data from the managed system, either updating the knowledge base or initiating the analysis phase. The collected data can range from sensor readings, such as temperature sensors, ADC outputs, and GPS coordinates, to network-related information, including network topology, the number of active hosts, and running services. Monitoring can focus on

Table 3. Summary of Monitoring Approaches in MAPE-K

Category	**Type**	**Examples / Techniques**
Feed Type	Direct Feed	Raw or minimally processed data sent to knowledge base
	Filtered Feed	Rule-based filtering, Kalman filter, CEP, noise filtering and normalization, context-aware monitoring, network mapping, pre-analysis thresholds, symptom aggregation, adaptive ROS monitoring , runtime models for adaptive monitoring
Deployment Type	Centralised	Aggregation at a single node; suitable for cloud or enterprise systems
	Decentralised	Distributed processing at edge or across nodes; suitable for IoT/CPS
Triggering Method	Real-time Monitoring	Continuous data flow for immediate response; useful in dynamic environments
	Batch / On-Demand Monitoring	Periodic or event-triggered data collection; resource efficient

internal or local data, such as sensor-based tracking in IoT and industrial systems or network performance metrics in cloud environments. Additionally, it may involve external monitoring, such as environmental data collection, or a combination of both internal and external data sources.

The critical task of the monitoring system is gathering the data from the managed system, and providing the necessary data for further analysis and decision making. Most surveyed papers use the monitor component only to read the data and update the knowledge base. Here, we classify the papers, based on the strategies, the deployment options, and the triggering method of the monitoring component. The categories that we have identified are summarised in Table 3, and detailed in Sections 4.1.1, 4.1.2, and 4.1.3, where we describe surveyed papers.

4.1.1 Direct feed and Filtered Feed Monitoring. Most of the surveyed papers describe use of the direct feed of monitored data to the knowledge base, but report doing some changes to the data before passing it on. Rachidi and Karmouch [142] employ predefined rules for detecting anomalies and filtering data. Feng et al. [71] deploy a Kalman filter to estimate the states of an incubator, characterised by the air temperature inside an insulated container and the heatbed temperature.

Malburg et al. [115] use Complex Event Processing (CEP) methods to monitor a smart environment. CEP enables detection of matching patterns in IoT sensor streams and derive specified events for higher-level systems. Thus, it is possible to react to sudden misleading situations.

Azimi et al. [26] mention preprocessing methods such as noise filtering and normalization, packetizing, and periodically transmitting data to the system management. Elgendi et al. [60] emphasizes context-aware monitoring to determine the necessity of adaptation dynamically. Papamartzivanos et al. [131] employ network mappers and sniffers to assess network topology, active hosts, services, and vulnerabilities, enhancing the adaptivity of Intrusion Detection Systems (IDS). Pre-analysis methods are used by da Silva et al. [48] to evaluate significance thresholds before forwarding data to the analyze component. Namal et al. [124] apply aggregation and correlation methods to detect symptoms requiring further analysis. Cheng et al. [44] support adaptive ROS-based monitoring by dynamically subscribing to and unsubscribing from relevant data sources at runtime.

Monitoring lower-level phenomena like exceptions and diagnosis data requires identifying the detailed information necessary and feasible to monitor. Adaptive monitoring permits observing a greater variety of details with less overhead if, most of the time, the MAPE-K loop can operate using only a tiny subset of all those details. Brand and Giese [33] propose an approach that overcomes the outlined problems by providing generic adaptive monitoring via runtime models. It reduces the effort to introduce and apply adaptive monitoring by avoiding additional development efforts to control monitoring adaptation. Although the generic approach is independent of the monitoring purpose, it still allows for substantial savings regarding monitoring resource consumption.

4.1.2 Centralized and Decentralized Monitoring. In centralized systems, all monitoring data is aggregated and processed at a single node or server. This approach is typically used in resource-rich environments, such as cloud systems or enterprise architectures, where latency and data centralization are manageable. Most of the surveyed papers employing this strategy focus on the efficiency and reliability of data aggregation. Alternatively, decentralized monitoring distributes the workload across multiple nodes or edge devices. This is common in CPS and IoT applications, where scalability, low latency, and resilience are critical. For instance, Ouareth et al. [130] collect data using subcomponents through an internal interface.

4.1.3 Real time and Batch Monitoring. In most of the papers the monitoring happens in real-time. This is particularly useful in dynamic and real-time environments, such as smart factories, where detecting anomalies or process deviations promptly is essential [115]. On the other hand, Rachidi and Karmouch [142] mention periodic and on-demand data collection of device parameter values.

4.2 Analyse

The objective of the analyse phase in the MAPE-K loop is to thoroughly investigate and understand the current state of the system and to identify any deviations from the expected state. This phase is essential for diagnosing issues, evaluating performance, and gathering insights that will guide the plan phase. Advanced techniques, such as statistical analysis, ML, and data mining, are often employed to enhance the accuracy and depth of the analysis. The outcomes from this phase inform the subsequent plan and decision-making processes in later phases.

There are also works where the analyse phase selects an adaptation plan or implements a filtering procedure. Additionally, in some cases, the analyse phase generates useful features from the monitored data, which are then utilized in the plan phase. Based on these observations, we classify the papers we surveyed into four categories (see Table 4) described in Sections 4.2.1 to 4.2.4. We conclude in Section 4.2.5, discussing novel views of the role of the analyse and plan phases.

4.2.1 Anomaly Detection. Anomalies are rare items and events, or, more generally, observations that significantly differ from the majority of the others according to gathered data. The process of anomaly detection in MAPE-K loop in the analyse phase employs various techniques such as rule-based methods, statistical methods, and ML or Deep Learning (DL) approaches.

Rule-Based Methods. The most commonly used methods for anomaly detection among the surveyed papers were rule-based methods [52, 142, 148]. They use a set of explicit, predefined rules to identify deviations from normal behavior. These rules are often derived from expert knowledge and historical data. However, due to adaptation, those rules can change over time to fit with the new input data. Rule-based methods are commonly used because, in many cases, it is easier to detect anomalies using thresholds, rules, or statistical metrics.

Machine Learning & Deep Learning Methods. ML and DL methods provide more powerful tools for anomaly detection capable of handling complex and high-dimensional data, detecting intricate

Table 4. Summary of Analysis Approaches in MAPE-K

Category	Type	Examples / Techniques
Detection Techniques	Rule-Based Detection	Threshold rules, expert heuristics for anomaly flags [52, 148].
	ML / DL Detection	Supervised or unsupervised models (e.g., classifiers, autoencoders, forecasting networks) used for anomaly detection [60, 96].
Data Engineering	Feature Extraction	PCA, encoder-based models; ontology-driven reasoning with OWL + SPARQL/SWRL for semantically rich features [5, 26, 131]
	Pre-processing	Noise/outlier filtering, semantic filtering, aggregation, and normalization performed prior to planning [106].
Decision Integration	Adaptation Plan Selection	Utility- or prediction-driven tactic choice; Kalman-based recalibration, tactic ranking [44, 72]
	Analyse–Plan	RL, proactive workload forecasting, real-time analytics that directly yield adaptations [97, 172]

patterns that rule-based methods might miss. ML tools are commonly used due to their ability to detect patterns from larger datasets, enabling even prediction of anomalies [60, 96].

4.2.2 Feature Extraction. Sometimes, the monitored data may not directly facilitate anomaly detection or offer insights into adaptation [5, 26, 131]. In such cases, the focus of the analyse phase shifts to generating features used in the plan phase. This approach ensures that even if direct anomaly detection or system adaptation insights are challenging from raw data, the derived features can still inform effective planning for adaptation and performance optimization. Feature extraction is mainly implemented by applying ML and DL methodologies such as Principal Component Analysis, Encoder-based models, etc. Semantic ontologies have been introduced to enhance the generation of semantically meaningful features. For example, ontologies modeled in OWL and reasoning techniques such as SPARQL queries and SWRL rules enable advanced interpretation and runtime reasoning, directly supporting adaptation plan generation and semantic interoperability [106].

4.2.3 Data Preprocessing. Given that the analyse phase bridges the gap between the monitor and plan phases, several studies have utilized it for preprocessing monitored data before passing it to the plan phase [30]. Authors recommend this approach particularly when raw input data contains noise or outliers. Filtering methods and semantic reasoning are frequently employed to refine and prepare data for the subsequent plan phase. This ensures that the plan phase can extract new execution plans based on cleaner and more reliable data, enhancing the overall effectiveness of the system's adaptation and decision-making capabilities.

4.2.4 Adaptation Plan selection. Although adaptation plans are typically formulated during the plan phase, there are scenarios where adaptation plans can also be utilized in the analyse phase [44, 72]. This is because the plan phase may transition from creating adaptation strategies to executing actions aimed at achieving specified goals. As a result, the analyse phase can incorporate insights derived from these actions to refine its processes and optimize feature extraction or data processing techniques. Therefore, while the primary role of the plan phase is to execute actions to achieve goals, its decisions can also shape and enhance the subsequent analyses.

In some works, the analyse phase plays a direct role in selecting adaptation strategies based on utility evaluations, predictive models, or observed system behavior [44, 72]. Rather than merely detecting anomalies, the analyse step may proactively determine suitable adaptations, as seen in utility-driven or prediction-based approaches [8, 120]. However, when no anomalies are detected or predicted, these systems often lack fallback mechanisms, making the availability and selection of adaptation plans in the analyse phase critical for robust self-adaptation.

4.2.5 Analysis and Plan. A key observation in the surveyed papers is that the distinction between the analyse and plan phases is becoming blurred. In particular, studies employing Reinforcement Learning (RL), proactive workload prediction, and runtime analytics demonstrate that adaptation decisions are often derived directly from the analysis. For example, RL has been applied for dynamic fog-service provisioning in IoT systems, while proactive workload prediction has been used for autonomic resource allocation in cloud computing [97]. Similarly, adaptive virtual network allocations leverage real-time analytics to enhance network optimization [172].

This evolving perspective suggests a tighter coupling between the analyse and plan phases, where advanced analytics not only assess system behavior, but also actively inform and refine adaptation strategies. As a result, the analyse phase is no longer a passive evaluation stage but a key driver of intelligent, autonomous adaptation within the MAPE-K loop.

4.3 Plan

The plan phase determines an adaptation strategy to transition the system from its current state to a desired state, leveraging insights from the analyse phase. The approach to finding a (near-)optimal plan varies depending on system requirements and domain and can encompass any self-x goals [163]. However, a well-designed adaptation plan is crucial in ensuring the system's functionality and performance in dynamic environments. Consequently, much of existing contributions within the literature highlight this phase as a key focus of their MAPE-K research. Planning approaches have been divided into six categories summarised in Table 5 and described below (Sections 4.3.1 to 4.3.6).

4.3.1 Rule- and Policy-based. These approaches constrain the adaptation space by using a combination of logical rules and system policies to guide decision making. This ranges from rules that select specific actions under specific conditions to high-level directives that act as guidelines for achieving the desired system state. For instance, one of the rules defined by Al-Dhuraibi et al. [9] specifies that if the CPU usage exceeds 90% then an additional vCPU core must be used. In contrast, the Track-based Traffic Control System developed by Bagheri et al. [27] exemplifies the use of a policy instead of a rule, specifying that "adaptation should be designed in a way that avoids a collision and considers the fuel level required for the new schedule/flight plan."

Many strategies fall between these two examples, employing a combination of rules and policies. The set of rules- and policies can either be fixed, as demonstrated by Al-Dhuraibi et al. [9], or dynamically updated based on the knowledge base, as shown by Affonso et al. [3].

This rule- and policy-based planning approach has the benefit of allowing for simple, interpretable, and efficient planning algorithms. A limitation of this is the inability to dynamically evolve rules in response to dynamic environments. Additionally, there is no guarantee that predefined rules will yield (near-)optimal adaptation outcomes. To address these shortcomings, there is research on dynamically evolving rules to reflect real-time information about the environment [174]. Several studies utilize variations of planning approaches discussed below to dynamically generate the rules. For instance, Zhao et al. [174] use reinforcement learning, Liu et al. [112] use a genetic algorithm, and Ghahremani et al. [77] compare various supervised ML methods.

Table 5. Summary of Planning Approaches in MAPE-K

Category	Type	Examples / Techniques
Declarative planning	Rule- and Policy	Logical rules and system policies determines plan [27]. Either fixed [9] or dynamically updated based on knowledge [3].
	Semantic Reasoning	Planning is driven by an ontological representation defining concepts, properties, and relationships. Plan is often found using a semantic reasoning engine [150, 158].
Optimization		Planning is defined as an optimization problem where different, mainly heuristic, techniques are used [14, 29, 91].
Learning	Offline Learning	Data-driven planning using ML techniques that are trained before deployment. Used at runtime to generate plans [61, 83, 108].
	Online Learning	Data-driven planning using ML techniques that continually learn during system execution. Typically done with RL [114, 121].
Other		Planning that combines multiple approaches or did not fit within any category.

4.3.2 Semantic Reasoning-based. Ontological knowledge representations define concepts, properties, and relationships within a domain. In this section, we cover works in which the planning approaches are driven by an ontological representation. While this section emphasizes the planning aspects of the approaches, Section 4.5 describes the details of the knowledge representation.

Similar to rule- and policy-based approaches, ontology-based planners employ logical rules to generate plans. However, they are particularly suited for complex domains where domain-specific notations are required to accurately represent the environment's state. In such domains, the intricate relationships between entities and the need for semantic reasoning make ontology-based planners essential for ensuring accurate and context-aware decision-making.

In these studies, the plan component leverages semantic-reasoning engines to analyse the current system state and subsequently generate an adaptation plan. This plan is typically devised based on how multiple domain-specific components within the environment relate to each other. For example, Teimourikia and Fugini [158] utilize an OSHA-based ontological model for safety expertise, where risk analysis is initiated by hazardous events, and the planner devises risk-preventive strategies using predefined hazard-analysis documents or an adaptive risk-treatment model.

An alternative approach is presented by Seiger et al. [150] who address inconsistencies between the sensed physical world and the assumed cyber world in CPSs. In this framework, the ontology defines observable changes within components, and when discrepancies are detected, the planner formulates a self-healing adaptation based on the results of a "Compensation Query".

4.3.3 Optimization-based. When mathematical optimization techniques are used to generate a new plan formulated as an optimization problem, we have an optimization-based planner. An optimization problem consists of three key elements: the objective function, decision variables, and constraints. The objective function captures the desired adaptation outcome, while the decision variables represent the controllable parameters that can be adjusted to achieve it. Constraints impose system limitations that must be respected during adaptation. Given these elements, optimization algorithms identify the optimal set of values for the decision variables that maximize or minimize the objective function while satisfying the given constraints.

A common characteristic of optimization-based planners is the prevalent use of heuristic-based optimization methods, perhaps due to their computational efficiency compared to exact methods [14, 72]. Beyond this, a diverse range of optimization algorithms were employed, with a slight preference for particle swarm optimization algorithms [6, 14]. Other approaches include ant-colony optimization algorithms [91, 126] and planning as a constraint satisfaction problem [29, 127].

4.3.4 Offline machine learning-based. This category includes data-driven planning using ML models trained prior to system deployment and utilized at runtime to generate new plans. These models leverage a diverse range of ML techniques, spanning both supervised and unsupervised learning, such as generative adversarial networks [83], K-means clustering [61], DL [108], federated learning [59], and K-nearest neighbors [128]. Their strength lies in their ability to recognize complex patterns and handle high-dimensional data, enabling effective analysis and decision-making in dynamic environments [83]. However, practical limitations may arise due to their data-driven nature, as they rely on high-quality datasets for effective model training.

4.3.5 Online machine learning-based. This category includes data-driven planning models that, like their offline counterparts, have the option to be trained offline using pre-collected data. However, unlike purely offline models, they continually learn and adapt during system execution, refining their predictions and improving decision-making in real-time [114, 121].

This characteristic allows these models to handle truly dynamic environments, where the rules governing interactions, states, and outcomes may evolve unpredictably over time. As such, they present a promising approach for managing uncertainty and addressing unprecedented scenarios, a significant challenge for self-adaptive systems as discussed by Weyns [163]. The majority of studies in this category employ reinforcement learning techniques, particularly Q-learning, while others explore learning automata or leverage a combination of various ML approaches.

A key challenge, particularly when considering reinforcement learning techniques, is the existence of the exploration-exploitation dilemma. While exploring new planning policies is crucial for discovering optimal strategies, executing them can lead to suboptimal performance. Conversely, exploiting known policies can provide immediate performance benefits but may hinder the discovery of superior strategies. Balancing this trade-off is complex and significantly impacts the method's effectiveness. The works of Lewis et al. [109] and Esterle [65] explore this dilemma in the context of self-organizing systems with a focus on the individual system and their local decisions as well as the group behaviour in collaborative systems, respectively.

4.3.6 Other methods. Some surveyed papers employed planning approaches that did not align with any of the above categories. This is either because planning utilizes a combination of techniques from different categories or the technique itself does not fit within any category, and the number of contributions in this area did not warrant the creation of a separate category.

4.4 Execute

The execution phase in the MAPE-K loop is responsible for applying the generated plan to the managed system, ensuring the transition from planning to operational implementation. This phase is not merely about forwarding execution plans, but also involves dynamic resource allocation, error recovery, and security enforcement. Execution strategies vary based on automation levels, ranging from fully automated processes in cloud computing to semi-automated systems requiring

Table 6. Summary of Execution Approaches in MAPE-K

Category	Type	Examples / Techniques
Execution Role	Passive Execution	Directly applies plan without further verification
	Active Verification	Consistency checks before execution, Strategy validation and filtering, Security integration

human oversight. Additionally, execution methods incorporate workflow orchestration, control systems, and multi-level execution strategies, integrating safety checks such as authentication, policy enforcement, and data integrity verification. Understanding these aspects is crucial for optimizing execution efficiency, reliability, and security. We identified the categories in Table 6.

Some approaches treat execution as a passive phase that directly applies the changes received from the plan phase without additional verification. Rachidi and Karmouch [142] highlights that execution involves structuring and formatting configuration plans and dispatching them to managed devices without intervention. Singh and Kim [153] reinforces this idea by stating that execution can be predefined based on rules stored in the knowledge component, limiting its role to plan dissemination. In [26], the authors allocate data transmission (i.e., bandwidth) management to the execute component. This can be extended to cover other types of system resources (e.g., energy) by adding a computing component in the fog layer to set system behaviors during monitoring.

Seiger et al. [150], on the other hand, emphasize that execution does not simply forward the plan but actively verifies consistency through the MAPE-K loop, ensuring objective fulfillment before proceeding. Poggi et al. [136] highlight that execution includes model adaptation by filtering and validating strategies before implementation, ensuring alignment with real-time conditions.

Security adaptation is an integral component of the execute phase in some systems. Amoud and Roudies [17] discuss how execution integrates security enforcement, intercepting data before adaptation and applying policy-driven modifications. Eryilmaz et al. [64] describe execution as responsible for reconfiguring security settings dynamically by selecting the best-suited data sources and processing chains to maintain system security and accuracy.

Faraji Mehmandar et al. [69] and Gamal et al. [73] present execute as an active phase where dynamic resource allocation and load balancing occur before executing tasks. Bucchiarone et al. [37] further elaborate on execution as a self-healing mechanism, with not only plans being applied, but also dynamic scaling, rebalancing, and restarting microservices carried out to maintain efficiency.

Some frameworks extend execution beyond task initiation by incorporating real-time adaptation. Bozhinoski et al. [32] detail execution strategies that involve stopping, starting, and reconfiguring system nodes as needed, rather than merely executing static instructions. Nazeri et al. [126] describe an execution phase that leverages intelligent scheduling and optimization algorithms to ensure energy efficiency and balanced workflow deployment in fog computing environments.

4.5 Knowledge

The knowledge component records previous experiences and evolves continuously with adaptations, ensuring that configurations utilized in decision making are closely tied to results from past iterations. The specific structure and implementation of the knowledge component can vary significantly based on application and system context, lacking a universally defined standard. Table 7 provides a summary. A significant portion of the reviewed literature, however, does not provide details of the knowledge component. This makes it difficult to identify commonalities.

Table 7. Summary of Knowledge Approaches in MAPE-K

Category	**Type**	**Examples / Techniques**
Connectivity	Global	Shared and accessible throughout all MAPE-K phases.
	Local	Only used by one or a few MAPE-K phases.
Content	Data	Represents all forms of monitored data.
	Logs	Encompasses all recorded logs.
	Strategies	Includes high-level plans, policies, decision strategies, goal definitions, and adaptation tactics.
	Rules	Refers to predefined conditions that dictate system behaviour.
	Metrics	Comprises performance indicators used for system evaluation.
	Models	Encompasses a wide range of modelling elements.
	Resources	Includes all necessary resources required for MAPE-K operations.
Representation	Ontology- and Rule-Based	Ontology languages to formally represent and reason about knowledge in a semantic format.
	Graph- and Tree-Based	Capture relationships and hierarchical structures in contextual knowledge.
	Structured and Relational	Structured formats for data storage.

We, therefore, concentrate on works where the knowledge was a major focus of the paper, with approximately half of the reviewed works barely addressing the knowledge component at all.

4.5.1 Uses of knowledge. Arcaini et al. [22] explain how the knowledge component stores data shared across components, promoting smooth coordination within a decentralized MAPE-K loop. In some instances, the knowledge component supports security-related decisions, with Amoud and Roudies [18] presenting a knowledge Unit that synchronizes system components by sharing critical data, including a Service States File linking security concerns to environmental factors. Closson et al. [47] assert the knowledge component's critical role in making informed decisions during execution. Ettahiri and Doumi [67] describe a process for monitoring the system, comparing current cases with existing knowledge, analyzing similarities, and proposing new plans based on past experiences. Iglesia and Weyns [87] emphasize that a dynamic knowledge component enables real-time adaptations, ensuring flexibility and responsiveness to changing conditions.

4.5.2 Architecture. The architecture of the MAPE-K loop differs significantly across implementations (see Section 3), impacting how the knowledge is structured and managed. Rachidi and Karmouch [142] propose an architecture where the knowledge component, referred to as Ontology & Policy Management, primarily interacts with the configuration planner and analysis components. Seiger et al. [149] highlight a model-based approach in which data is stored as ontologies, updated with each data change. Abdennadher et al. [1] describe a dual-layer structure, with the knowledge component comprising a knowledge and a decision layer. Elsayed et al. [62] set the knowledge component into a storage layer, while Hakim et al. [82] ensure consistency and reflexivity through a storage and self-management layer that maintains static context knowledge. Mazidi et al. [119] detail how the knowledge is stored in the database layer, while Müller et al. [122] position it within a subdivision of the cyber layer, with data presented in UML/XML for configuration.

4.5.3 Contents of the knowledge component. We have classified the contents of the knowledge component in the various works to gain insights into their purpose and applications. Different works refer to similar concepts using varying terminology, making it challenging to directly compare different knowledge-component implementations. The following classification gives a (potentially incomplete) description of the various contents types and forms of the knowledge component: data, logs, configurations, strategies, rules, metrics, models, and resources.

4.5.4 Data representation and formats. Various works focus on data storage and representation. Almeida et al. [15] use XML for information storage. Silva et al. [152] and Teimourikia and Fugini [158] employ the Ontology Web Language (OWL) for structuring knowledge. Ashraf et al. [25] integrate OWL with rules defined in the Semantic Web Rule Language (SWRL). Amja et al. [16] also use SWRL. Parvizi-Mosaed et al. [133] formalize and store knowledge using OWL.

Seiger et al. [149] utilize a graph-based representation of the context ontology. Khorsand et al. [96] adopt graphs and trees for representing models and ontologies. Examples are given in Mehmandar et al. [121], Khorsand et al. [96], and Koehler et al. [102], where application data, resources, and metrics are stored in a database. Seiger et al. [148] use a graph-based structure for efficient knowledge representation, echoed in Seiger et al. [150], where both graph-based and ontological representations of context data are employed. Senthilvelan et al. [151] opt for a tabular format for data structuring.

4.5.5 Knowledge evolution and adaptation. Aguayo and Sepúlveda [4] specify that, with knowledge and ML, systems can learn from past actions. As highlighted by Calinescu et al. [39], the knowledge can comprise “a priori knowledge”, often initial and imprecise, and “a posteriori knowledge”, generated from runtime experiences. Dautov et al. [52] suggest that knowledge is a common vocabulary for a managed system, adapting over time. Ma and Wang [113] outline a progression from basic rules to increasingly sophisticated decision making. Krupitzer et al. [105] highlight that learning cycles contribute to the knowledge, enabling adaptations based on past experiences. Parsaeefard et al. [132] emphasize the knowledge’s role in learning and anticipating conflicts.

The knowledge component implementation presents several challenges. Ashraf et al. [25] note that the knowledge components’s size can adversely affect performance. Nascimento et al. [125] point out that knowledge representation may be difficult for humans to interpret, suggesting natural language extensions as a potential solution. Arcaini et al. [23] highlight challenges in distributed loops, such as preventing inconsistent updates and avoiding redundant knowledge.

5 VERIFICATION OF MAPE-K LOOPS

Formal verification of MAPE-K loops attempts to ensure that systems meet their requirements and behave safely throughout their operation. This is naturally a challenging task, since self-adaptive systems are designed to respond to unanticipated circumstances. Verification requires us to handle uncertainty in modelling the system’s behaviour in these circumstances.

Researchers have applied a variety of existing formal methods to verify the different components of systems involving MAPE-K loops, whilst developing new approaches and abstractions to model the self-adaptation process. In this section, we cover the variety of modelling techniques that have been used to model the MAPE-K loop (Section 5.1), the specification languages used to express the requirements of self-adaptive systems (Section 5.2), and the verification approaches used to check that a system meets its requirements (Section 5.3). We present final considerations in Section 5.3.2.

5.1 Semantics for MAPE-K verification

A key challenge in verifying MAPE-K loops is establishing a formal semantics that gives mathematical meaning to both the behaviour of the managed system, and the self-adaptation process implemented in the MAPE-K loop. Such semantics provide mathematical representations of the states and behaviors of computational systems providing a formal model of the system. We consider works that use existing behavioural-modelling approaches in Section 5.1.1, and work that introduce specific approaches to model self-adaptation in Section 5.1.2.

5.1.1 Applicable modelling notations. Many works rely on variants of state machines to model the underlying behaviour of the system. These approaches include Abstract State Machines (ASMs) [22, 171] and Petri nets [41]. ASMs form the basis of the ASMETA framework [74],a general-purpose framework for state-based modelling. Yang et al. [171] propose a variation of ASMs named Interactive State Machines (ISMs). These machines distinguish between environmental variables and controller variables; this allows the authors to model environmental constraints and uncertainty by defining intervals of possible values for variables.

An alternative approach uses the Z notation [54], a formal specification language based on set theory, for providing logical contracts of the operations. Applications of this approach to model self-adaptivity and the MAPE-K loop include FORMS (the FOrmal Reference Model for Self-adaptation) [166], which introduces a formal model of self-adaptation (see the next section).

The next category of work captures real-time aspects of the system behaviour. This makes it possible to reason about the timing of events, and to verify properties regarding real-time constraints on operations. Approaches include uses of timed automata, which extend finite state machines with clocks to model the timing of transitions. Other options are the SMARTS framework, which uses Timed Communicating Object Z [139] (a timed, concurrent extension of the Z notation), and the CARSS framework [95], which uses timed-arc Petri nets [88].

Finally, there are works that model stochastic aspects of a system's behavior. This makes it possible to reason about reliability, performance, and Quality of Service (QoS) requirements. The sources and potential impacts of stochastic behavior covered are diverse, and include environmental uncertainty and load and performance characteristics variability.

The most common way to model stochastic behavior is through Markov chains [68, 103, 154]. These are state-based models in which behavior depends only on the current state of the system, not its history. States are advanced through actions that commonly have a statistical component that dictates the next state. These actions also have a reward, which is also statistical in nature. Stochastic behaviour can be combined with timed behaviour, as shown by Li et al. [110], who use UPPAAL-SMC's models: timed automata with probabilistic transition functions.

Queuing networks are another formalism for modelling stochastic behaviour which focus on performance analysis and have applied to modelling self-adaptive systems in [24]. In the next section, we focus on modelling of self-adaptation specifically.

5.1.2 Modelling self-adaptation. In this section, we discuss works that introduce direct modelling support for self-adaptation, and how these have been applied to modelling the MAPE-K loop.

The conceptual framework for adaptation introduced by Bruni et al. [36] models systems as labelled transition systems. It characterizes self-adaptive systems as a parallel composition $FC \parallel CD$ between some fixed component FC and a control data component CD. Here, $\parallel$ denotes the parallel composition of two labelled transition systems, synchronizing on shared labels. FD represents the non-adaptive components of the system (i.e. the managed subsystem), and CD captures changes in response to adaptations (i.e. by a managing subsystem). Whilst conceptually simple, this model can account for very general schemes of adaptation in response to changing external environments or goals. It was applied directly to modelling the MAPE-K architecture, with the loop determining the control data. Within this model the environment also plays a key role and is treated as a labelled transition system E in composition with the system and its control data.

An alternative abstract model captures self-adaptation through the use of reflective subsystems whose role is to monitor and adapt the managed components of the system [19]. This approach requires the reflective subsystem to contain a reflection model that refines aspects of the managed system (e.g. architecture, submodules, data) required for detecting planning adaptations. This reflection-based approach is a key feature of the FORMS framework [166], which explicitly models reflective components and computations corresponding to each component of the managed system. This reflective view is unified with the MAPE-K architecture by identifying reflective computations corresponding to each stage of the MAPE-K loop [166, Section 3.3].

In a similarly approach, the model of Camilli et al. [41] defines an API for modifying the behavior and state of the managed system. In this case, it is realised through an emulator Petri net to handle dynamically updatable data representing the managed system. This uses an extension of Petri nets focused on dynamic evolution, and takes a two-level approach to modelling self-adaptive systems, with the first level capturing the managed system and the second, the MAPE-K loop.

5.2 Requirements formalisms

In addition to modelling the system under study, verification also requires mechanisms to specify the properties that the system should uphold. Capturing these requirements brings us to two related but distinct challenges. First, it is necessary to decide which requirements are relevant and express them with sufficient precision to enable formal verification. Second, we need to relate the meaning of these requirements to the underlying semantics of the system and its self-adaptations, making it possible to evaluate whether a given system model meets its requirements.

A large number of works use temporal logics, which specify timed properties using logics that combine assertions about the events or states of the system, with standard logical and temporal operators. One of the most common temporal logics used in the literature is Linear Temporal Logic (LTL), which uses the modal temporal operators *eventually* ($\Diamond$), which states that a property ϕ must be true at some point in the future, and *always* ($\Box$) which states that a property ϕ must be true at all points in the future. Since its introduction to computer science by Pnueli [135] in 1977, LTL has been one of the most widely used formal languages for specifying requirements of computational systems, and of MAPE-K loops in particular [22, 23, 116, 161].

A number of works [22, 101] have also used the Computation Tree Logic (CTL) [46], which extends LTL with temporal operators that allow assertions about the branching structure of the computation tree. There are also extensions of LTL specifically focused on adaptations.

Zhao et al. [175] propose that properties for self-adaptive systems with distinct modes fall within the following categories: (a) local properties, which are expected to hold within a particular mode, but may not persist after subsequent adaptations; (b) adaptation properties, which characterise how the state of a system changes upon adaptation; and (c) global properties, which must always hold regardless of when and how the system is adapts. Zhao et al. address these classes of properties through the introduction of mode-extended LTL [175], which adds operators that make it possible to reason about the modes of a system and the mode locality of properties.

Adapt operator-extended LTL (ALTL) [173] is another extension of LTL with an adaptation operator that makes it possible to specify the state of a system before and after an adaptation event. This makes it possible to express adaptation properties in addition to local and global properties, and to reason about the impact of adaptations on the system's behavior.

Timed aspects of MAPE-K loops have also be specified using timed temporal logics such as MTL (Metric Temporal Logic) [92] and TCTL [86]. On the other hand, probabilistic temporal logics such as PCTL (Probabilistic Computation Tree Logic) [38, 103] have also been applied to specify probabilistic properties such as reliability requirements.

The hard distinction between the role of modelling and specification languages is not always clear, since many modelling languages support abstraction mechanisms that can be used to omit details of a systems behaviour until the model becomes more of a description of system requirements than a description of the system implementation. Under this approach, modellers start with a high-level model of the system based on the requirements and iteratively refine it to a concrete model that can be implemented, with each refinement step being verified to ensure that the new model conforms to the higher-level model. Such refinement-based approaches are popular in the Z community, and readily applicable to FORMS [166] and SMARTS models [139]. Hachicha et al. [81] apply such a refinement-based approach to verifying that Event-B models of MAPE-K loops follow certain timed patterns, whilst Göthel et al. [80] apply refinement checks on models in the CSP process algebra [145] to check the abstract design patterns based on the FDR model checker.

5.3 Verification methods

We cover offline verification, carried out at design time (Section 5.3.1), and at runtime (Section 5.3.2).

5.3.1 *Offline formal verification.* In the CARSS framework [95], the authors use the TAPAAL model checker [53] to verify timed properties specified in TCTL. For probabilistic properties, Korn et al. [103] propose the use of probabilistic model checking in PRISM to check PCTL properties of different strategies using a Markov Decision Process (MDP). These strategies are then stored in the knowledge database and the plan component decides which one to run at runtime.

Statistical model checkers can give approximate estimates of the satisfaction probabilities of formulae in a manner that scales better to realistic systems. This is used by Li et al. [110], via the UPAAL-SMC model checker, to verify stochastic behaviours of decentralized MAPE-K loops.

Yang et al. [171] use strategies encoded as paths of a state machine and extract logical statements from the guards of the transitions of thestrategies. These statements are enhanced with uncertainty and, in conjunction with a failure condition, sent to a SAT solver to see if the failure condition can hold at any point of the path. Probabilistic approaches can synthesise optimal control loops, as shown by Carwehl et al. [42], who present techniques to synthesise loops that guide exploration to maximize reduction of uncertainty (as defined by a parametric DTMC specified in PRISM). Song et al. [154] propose a cheaper verification process: they use model slicing before model checking.

5.3.2 *Runtime verification.* These techniques tend to be more lightweight, as they are based on a system execution. So, they do not provide guarantees about all possible behaviours. However, online verification can respond to unforeseen environmental conditions and adaptations. There is a significant link between the concept of runtime verification and the MAPE-K loop monitoring.

Runtime verification can also be applied to the MAPE-K loops themselves, as a meta-adaptation approach. Klös et al. [101] represent meta-adaptation information within the knowledge base, allowing a MAPE-K loop to monitor and adapt to the performance of past adaptations.

A related approach is that of correct-by-construction methods that ensure correct behaviour through their use of formal models at runtime, as the basis of their planning phases. In Activ-FORMS [86], this is done through the use of timed automata models of the underlying system. The authors identify runtime monitoring with the adaptation goals of a MAPE-K loop, and use a virtual machine to switch between different models at runtime.

Another such approach is proposed by Cifuentes et al. [45], who produce correct-by-construction plans by solving Constraint Satisfaction Problems (CSPs) at runtime. Kamburjan et al. [92] apply a related X-by-construction approach to a digital twin-based system, where a MAPE-K loop maintains the relationship between the digital twin and the physical system. This approach uses a knowledge graph to represent the physical world and models, and uses Metric Temporal Logic (MTL) to specify requirements relating them. Wright et al. [168] propose an approach for using a continuous time dynamical system as a model for a digital-twin–based MAPE-K loop, and then use the Flow* tool to verify the Signal Temporal Logic (STL) properties of the system at runtime.

Probabilistic models and probabilistic model checking forms the basis of MAPE-K based runtime verification proposed by Calinescu et al. [38]. This approach is refined by Fang et al. [68] who propose the PRESTO framework to improve the scalability of runtime probabilistic verification.

6 DISCUSSION

In this section, we first we provide an analysis of the numbers of papers in each of the categories of the taxonomies used in our survey (Section 6.1). Next, we present an statistical analysis that identifies how those categories are correlated (Section 6.2). With these results, if a designer decides to implement a phase of the MAPE-K loop using a specific technology (for example, use machine learning for Analysis), we provide here insight on how to implement the rest of the phases. We cover verification in Section 6.3. We finally identify research gaps (Section 6.4).

6.1 Results analysis

As already said (in Section 2), as part of the survey, we have classified each paper that describes an implementation of a MAPE-K loop according to a taxonomy for each phase. For convenience, we have plotted that information in Fig. 3, with a graph for each phase. The value n defined in the caption of each graph defines how many papers discuss the corresponding phase.

The Monitor phase is a good representative of the distribution of papers in each classification. A category, usually the simplest, dominates the taxonomy. In this case, the category "Direct-feed" covers the most common technique used to implement the Monitor component. This is a good approach for applications that do not collect complex or abundant data.

The Analyse phase is most commonly used as a way to detect anomalies, operating over raw environment data coming from the Monitor phase. In this case, either through rule-based or ML approaches, the Analysis phase aborts the MAPE-K loop's execution if the data contains no anomalies. Another two uses of the Analyse phase is feature extraction and data preprocessing, which are less focused on the environment; the applications are less common.

For the Plan phase, the distribution of surveyed papers is more balanced. It is natural that the phase with the most complex responsibilities, bearing the load of deciding on the new behaviour of the system, has papers spread through more categories. Still, over half of the papers are in the "Rules and policy" category. The rest of the approaches are mostly uniformly distributed, with solving an optimization problem being the second most common.

The Execution phase, in most surveyed papers, is the simplest one. Very commonly, it is just a simple pass-through execution. However, there is still a significant amount of papers that report the use of a controlled execution, specifying the method by which they deploy the plan.

For the Knowledge component, we surveyed data that is stored. Most commonly, it is the data read by the Monitor. Models of the system or adaptation rules are also commonly stored.

6.2 Inter-phase Analysis

We discuss here the the impact of selecting an approach from a particular category to implement a phase on the category of the approach that should be chosen for the remaining phases. To do that, we built contingency tables for each pair of phases and conducted a Fisher's exact test, running Monte Carlo simulations (as the marginals are not fixed). Our null hypothesis is that the tables are not associated, that is, it is probable enough to generate them from a table following the uniform distribution with the same marginals. The p-values (smaller means more associated) showing the

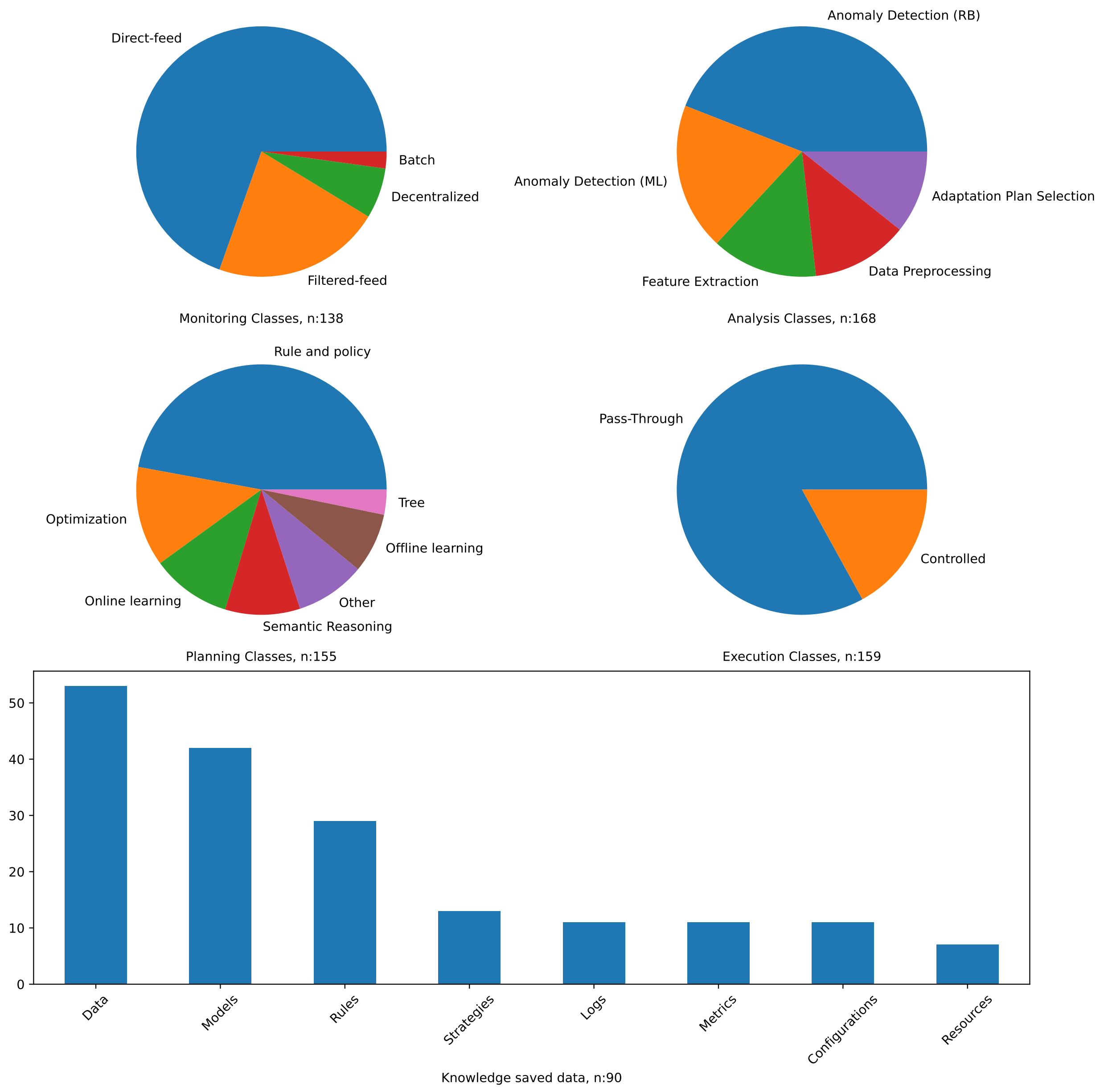


Fig. 3. Distribution of classes of the MAPE phases and objects stored in K.

	M	A	P	E
M	—	0.4859	0.0868	0.0001
A	0.4859	—	0.0289	0.2492
P	0.0868	0.0289	—	0.0113
E	0.0001	0.2492	0.0113	—

Table 8. P-values from Fisher's exact test for pairwise comparisons between classes.

probability to generate a table as bad or worse for each pair of phases are given in Table 1. A graph visualizing the contingency tables (normalized by dividing the value in each cell by the sum of the amount of papers on each corresponding category) for each pair of phases is shown in Fig. 4.

As we can see from Table 8, the design choices for the Plan phase have the biggest association with the choices for the other phases, coinciding with Plan being the phase with the largest responsibility. It dictates the adaptive behaviour of the MAPE-K loop, and as such choosing how to implement it influences an engineer's decision on how to implement the other phases.

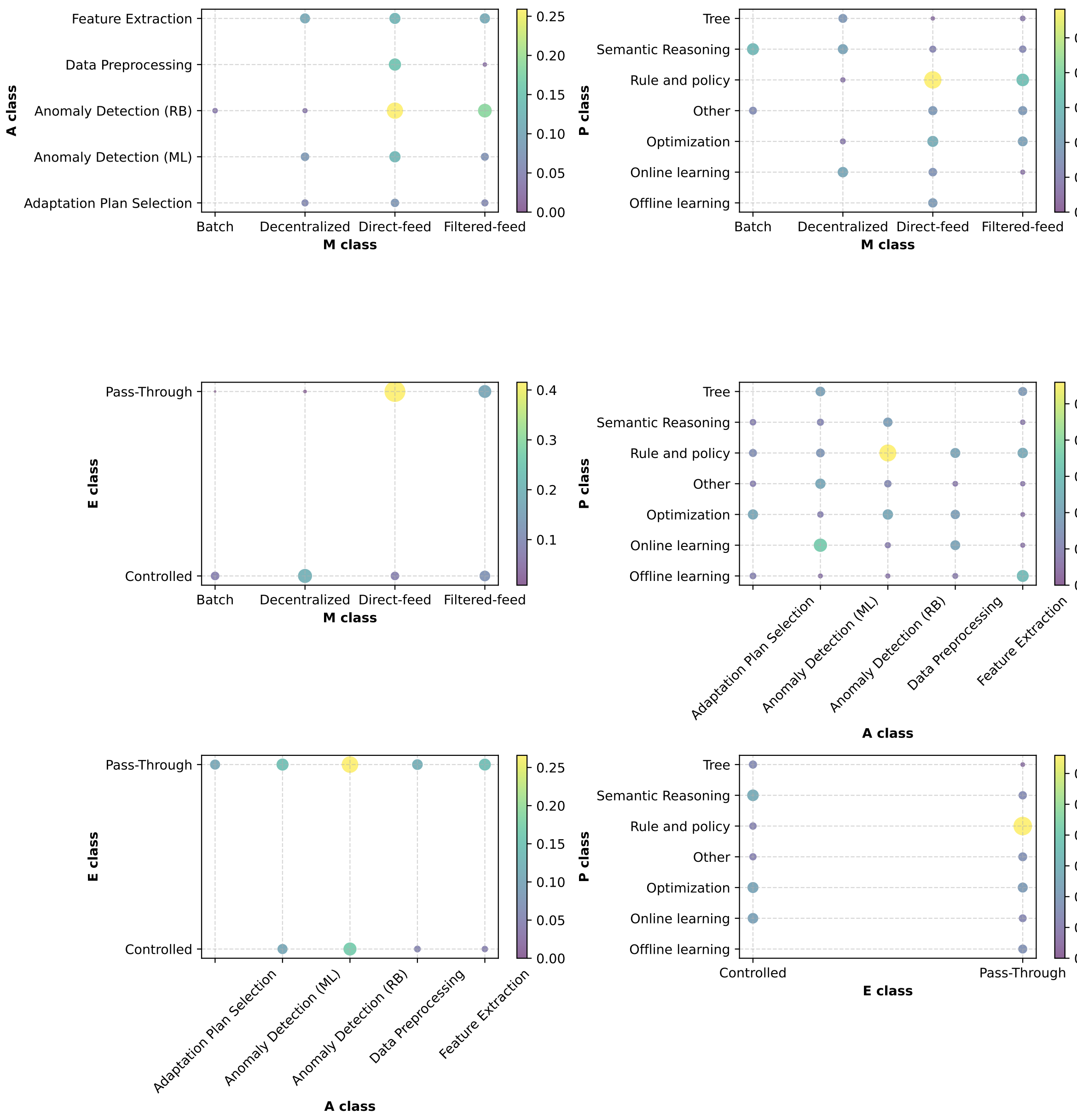


Fig. 4. Visualization of contigency tables between MAPE phases (normalized).

Looking at the M-P contingency table, we can see that the strongest category correlation Rule and Policy planning with Direct Feed monitoring, as papers that focus on applications of the MAPE-K loop, like the work of Al-Dhuraibi et al. [9], or focus on other phases, like the work of Mazidi et al. [118], report on works where the option was simplifying the Monitor and Plan phases. There is also a connection between Batch monitoring with Semantic Reasoning planning. However, the lack of papers that report on work that adopt Batch monitoring led to this unexpected result, as we have found no connection between the two categories.

The A-P contingency table also shows some interesting correlations. First, we can see that Rule Based Anomaly Detection analysis is correlated to Rule and Policy planning. The reason for this correlation is again due to empirical rules being the simplest way to implement these phases, and as such are commonly used on papers exploring new applications like the work of Albassam et al. [10]. Furthermore, the learning-based planning categories are also highly correlated to some of the Analyse categories. On the one hand, ML Anomaly Detection and Online Learning planning are

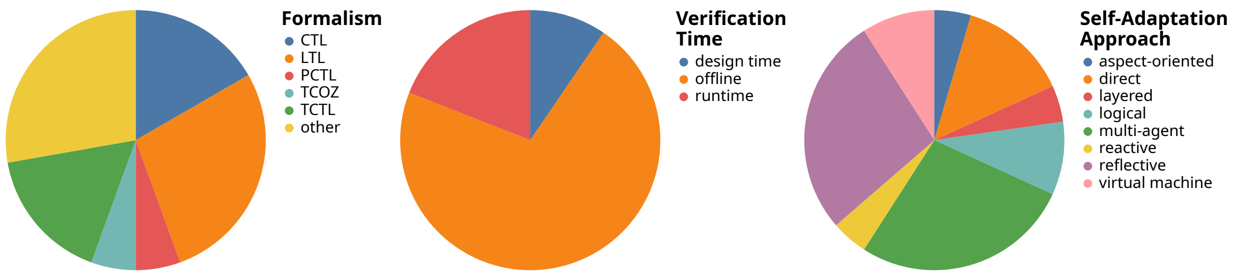


Fig. 5. An overview of the papers considered by requirement specification language, time of verification, and the approach taken to modelling self-adaptation.

commonly used in conjunction, since the same neural network can be used for both phases. An example is the work of Soto et al. [155], where learning is based directly on the data passed from the Monitor phase. On the other hand, Feature Extraction analysis and Offline Learning planning show some correlation as well, as extracting the most relevant dimensions of the data is commonly used alongside ML techniques, as in the work of Ortiz et al. [129].

Finally, there is also some association in the relationships M-A, M-E and P-E. Relationships between the Execute phase and other phases are not worth discussing, as there are only two categories in the Execute taxonomy and the distribution is extremely unbalanced, leading to significant statistical noise. However, there is a noteworthy category relation in the M-A contingency table concerning Data Preprocessing and Direct Feed, since if Filtered Feed monitoring preprocesses the data, it is redundant to do it again on the Analyse phase.

6.3 Verification

The techniques we have identified cover almost all the state of the art in formal methods, from modelling to the verification approaches (see Fig. 5). There is one clear exception: hybrid systems modelling and verification. Hybrid systems are systems that have a continuous timed domain and a discrete computational one. These systems are commonly seen in self-adaptation, as the environment can be modelled with differential equations and the controller is discrete.

The majority of works have focused on using the MAPE-K loop to monitor its own behaviour, or as a verified monitor for the managed system. Only few works have considered runtime verification of the whole self-adaptive system as a means to provide additional safety guarantees.

6.4 Research Gaps

Based on the survey, we can identify a several research gaps and propose a roadmap to close them.

Monitoring. Figure 3 highlights that current work focusses on direct-feed analysis and some limited work on filtered feeds, and on decentralised and distributed monitoring. Distribution of monitoring tasks can gain more information about the environment by allowing the system to adapt monitoring tasks based on the state of the system and of the environment. Another unexplored direction is the introduction of an awareness in monitoring to improve autonomy. With this, the system can adapt its own monitoring regarding the different properties and aspects that should be monitored as well as the frequency and accuracy of such monitoring.

Analysis. The majority of current implementations of the analysis phase focus on anomaly detection and data preprocessing with and without machine learning approaches as highlighted in Section 4.2. While some work has been done incorporating uncertainty in the planning phase, there seems to be little attention to consider uncertainty in the analysis phase. Future research should explore uncertainty identification and uncertainty quantificatioin analysis.

Planning. Planning has received a lot of attention in the surveyed literature and, according to our results (see Figure 3), covers mainly rule- and policy-based approaches, semantic reasoning, optimizations, and machine learning techniques as outlined in Section 4.3. Similar to the Analysis phase, the current implementations of the Planning phase are mostly based on predefined action spaces. This limits the system's ability to adapt to unforeseen situations. Future work should focus on introducing novel approaches that allow the system to explore its own capabilities and potentially adapt to new situations. This can be either by enabling the system to figure out new capabilities when everything is working correctly and it is operating in a safe state, or by allowing the system to explore new operations in situations that cannot worsen the state of the system. Both reinforcement learning and generative artificial intelligence models are expected to play a crucial part to explore the action space and generate new actions. With such approaches, novel planning algorithms that can incorporate such exploration capabilities in its core are expected.

Execution. In Section 4.4, we highlight that Execution usually refers to the different steps of the generated or selected plan from the planning phase. Recent work has expanded this with verification techniques to ensure safety, highlighted by our research on the controlled execution category in Figure 3. This is already a well-investigated topic within the formal methods and verification community. Concerns not covered, however, are security risks, use of predictions and what-if simulations, and distribution of execution tasks.

Knowledge. The discussed work on the Knowledge component in Section 4.5 covers areas such as architectures, data representations, and knowledge evolution. A core element for future work is considering the knowledge component beyond its use as data storage. Making independent inference and generating actual *knowledge*, in contrast to only information [157], is a key step towards a more autonomous system. Such step can be related to introducing computational [66] and goalawareness. With such a capability, the system can reason about its own goals, select from a set of goals, and even adapt or generate goals at runtime.

General. Taking into consideration the the interaction of different phases as highlighted in Figure 4, it becomes apparent that there is little work on batch monitoring and its relation to the other phases. This can help to improve the overall performance of the MAPE-K loop by allowing for more efficient and effective monitoring. Adaptation of plan selection is another area with the Analysis component that can receive more attention.

Finally, there is a lack of papers addressing synchronization between the different phases as they can have different execution paces. For example, a direct-feed monitor may take significantly less execution time than ML based Analyse or Plan. These differences can lead to bottlenecks for other phases that need to sync with a rapidly evolving environment or managed system.

7 CONCLUSIONS

In this paper, we report on an extensive survey of the literature on MAPE-K loops. We have identified five research questions (see Section 2), RQ1-5. The answers to RQ1 and RQ2 are given at Section 4 and elaborated on Section 6.1. RQ3 is answered on Section 5 and Section 6.3. Section 3 covers the answer to RQ4. Finally, RQ5 is answered on Section 6.2.

Future work will need to update the survey and results as we go forward. We observe large activity by the community, with an increasing ammount of papers being published every year. We expect that new lessons and exciting results will be available in a few years.

ACKNOWLEDGMENTS

This work is supported by the RoboSAPIENS Project financed by the European Commission's Horizon Europe programme under Grant 101133807 (https://cordis.europa.eu/project/id/101133807). The work of Ana Cavalcanti is funded by the Royal Academy of Engineering. The authors would also like to thank Dr. Jalil Boudjadar, Prof. Anastasios Tefas and Mikkel Labori Olsen for their thorough reviews.

REFERENCES

[1] Imen Abdennadher, Ismael Bouassida Rodriguez, and Mohamed Jmaiel. 2017. A Design Guideline for Adaptation Decisions in the Autonomic Loop. *Procedia Computer Science* 112 (2017), 270–277.

[2] Imen Abdennadher, Ismael Bouassida Rodriguez, and Mohamed Jmaiel. 2016. An Overview of a Decision Approach for Autonomic Applications Architectural Adaptation. In *Proceedings of the International Conference on Smart City*. 1095–1101.

[3] Frank José Affonso, Gustav Leite, Rafael A.P. Oliveira, and Elisa Yumi Nakagawa. 2015. A framework based on learning techniques for decision-making in self-adaptive software. In *Proceedings of the International Conference on Software Engineering & Knowledge Engineering*. 24–29.

[4] Oscar Aguayo and Samuel Sepúlveda. 2022. Variability Management in Dynamic Software Product Lines for Self-Adaptive Systems—A Systematic Mapping. *Applied Sciences* 12, 20 (2022), 10240.

[5] Oscar Aguayo, Samuel Sepúlveda, and Raúl Mazo. 2024. Variability Management in Self-Adaptive Systems through Deep Learning: A Dynamic Software Product Line Approach. *Electronics* 13, 5 (2024).

[6] Jose Aguilar, Alberto Garcés-Jiménez, Jose Manuel Gómez-Pulido, Maria Dolores Rodríguez Moreno, José Antonio Gutiérrez De Mesa, and Nuria Gallego-Salvador. 2021. Autonomic Management of a Building's Multi-HVAC System Start-Up. *IEEE Access* 9 (2021), 70502–70515.

[7] Jose Aguilar, M. Jerez, M. Mendonça, and M. Sánchez. 2020. Performance analysis of the ubiquitous and emergent properties of an autonomic reflective middleware for smart cities. *Computing* 102, 10 (2020), 2199–2228.

[8] O. Aissaoui, F. Atil, and A. Amirat. 2013. Towards a generic reconfigurable framework for self-adaptation of distributed component-based application, Vol. 488. 399–408.

[9] Yahya Al-Dhuraibi, Fawaz Paraiso, Nabil Djarallah, and Philippe Merle. 2017. Autonomic Vertical Elasticity of Docker Containers with ELASTICDOCKER. In *Proceedigns of the International Conference on Cloud Computing*. 472–479.

[10] E. Albassam, H. Gomaa, and D.A. Menascé. 2017. Model-based recovery and adaptation connectors: Design and experimentation, Vol. 743. 108–131.

[11] Emad Albassam, Jason Porter, Hassan Gomaa, and Daniel A. Menascé. 2017. DARE: A Distributed Adaptation and Failure Recovery Framework for Software Systems. In *Proceedings of the International Conference on Autonomic Computing*. 203–208.

[12] Elvin Alberts, Ilias Gerostathopoulos, Ivano Malavolta, Carlos Hernández Corbato, and Patricia Lago. 2025. Software architecture-based self-adaptation in robotics. *Journal of Systems and Software* 219 (1 2025), 112258.

[13] Mubashir Ali. 2020. Big Data and Machine Intelligence in Software Platforms for Smart Cities. In *Software Architecture*, Henry Muccini, Paris Avgeriou, Barbora Buhnova, Javier Camara, Mauro Caporuscio, Mirco Franzago, Anne Koziolek, Patrizia Scandurra, Catia Trubiani, Danny Weyns, and Uwe Zdun (Eds.). 17–26.

[14] Nour Ali and Carlos Solis. 2015. Self-Adaptation to Mobile Resources in Service Oriented Architecture. In *Proceedings of the International Conference on Mobile Services*.

[15] Andre Almeida, Everton Cavalcante, Thais Batista, Nelio Cacho, and Frederico Lopes. 2014. A Component-Based Adaptation Approach for Multi-Cloud Applications. In *Proceedings of the Conference on Computer Communications Workshops*. 49–54.

[16] Anne Marie Amja, Abdel Obaid, and Hafedh Mili. 2016. Combining Variability, RCA and Feature Model for Context-Awareness. In *Proceedings of the International Conference on Innovative Computing Technology*. 15–23.

[17] Mohamed Amoud and Ounsa Roudies. 2016. MaPE-K-Based Approach for Security @ Runtime. In *Proceedings of the International Conference on Software Science, Technology and Engineering*. 138–140.

[18] Mohamed Amoud and Ounsa Roudies. 2017. Using Combination of MAPE-K and DSPL to Secure Smart Camera Networks. In *Proceedings of the International Conference on Industrial Engineering and Operations Management*. 2061–2070.

[19] Jesper Andersson, Rogerio De Lemos, Sam Malek, and Danny Weyns. 2009. Reflecting on self-adaptive software systems. In *2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*. IEEE, 38–47.

[20] Roberto O. Andrade and Sang Guun Yoo. 2019. Cognitive security: A comprehensive study of cognitive science in cybersecurity. *Journal of Information Security and Applications* 48 (2019), 102352.

[21] Björn Annighöfer, Johannes Reinhart, Matthias Brunner, and Bernd Schulz. 2021. Requirements and Concept for a Self-organizing Plug&Fly Avionics Platform. In *Proceedings of the Digital Avionics Systems Conference.* 1–10.

[22] Paolo Arcaini, Elvinia Riccobene, and Patrizia Scandurra. 2015. Modeling and Analyzing MAPE-K Feedback Loops for Self-Adaptation. In *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems.* 13–23.

[23] Paolo Arcaini, Elvinia Riccobene, and Patrizia Scandurra. 2017. Formal Design and Verification of Self-Adaptive Systems with Decentralized Control. 11, 4 (2017), 1–35.

[24] Davide Arcelli. 2020. Towards a Generalized Queuing Network Model for Self-adaptive Software Systems. In *Proceedings of the International Conference on Model-Driven Engineering and Software Development.* 457–464.

[25] Qazi Mamoon Ashraf, Mohammad Tahir, Mohamed Hadi Habaebi, and Jouni Isoaho. 2023. Toward Autonomic Internet of Things: Recent Advances, Evaluation Criteria, and Future Research Directions. *IEEE Internet of Things Journal* 10, 16 (2023), 14725–14748.

[26] Iman Azimi, Arman Anzanpour, Amir M. Rahmani, Tapio Pahikkala, Marco Levorato, Pasi Liljeberg, and Nikil Dutt. 2017. HiCH: Hierarchical fog-assisted computing architecture for healthcare IoT. *ACM Transactions on Embedded Computing Systems* 16 (2017). Issue 5s.

[27] Maryam Bagheri, Marjan Sirjani, Ali Movaghar, and Edward A Lee. 2018. Coordinated actor model of self-adaptive track-based traffic control systems. *The Journal of Systems & Software* 143, September 2017 (2018), 116–139.

[28] Ahmed Bali, Mahmud Al-Osta, Soufiene Ben Dahsen, and Abdelouahed Gherbi. 2020. Rule based auto-scalability of IoT services for efficient edge device resource utilization. *Journal of Ambient Intelligence and Humanized Computing* 11, 12 (2020), 5895–5912.

[29] M. Baruwal Chhetri, H. Luong, A.V. Uzunov, Q.B. Vo, R. Kowalczyk, S. Nepal, and I. Rajapakse. 2018. ADSL: An embedded domain-specific language for constraint-based distributed self-management. 101–110.

[30] Nelly Bencomo and Luis Hernan Garcia Paucar. 2019. RaM: Causally-Connected and Requirements-Aware Runtime Models using Bayesian Learning. 216–226.

[31] Sree Ram Boyapati and Claudia Szabo. 2022. Self-adaptation in Microservice Architectures: A Case Study. In *Proceedings of the International Conference on Engineering of Complex Computer Systems.* 42–51.

[32] Darko Bozhinoski, Mario Garzon Oviedo, Nadia Hammoudeh Garcia, Harshavardhan Deshpande, Gijs van der Hoorn, Jon Tjerngren, Aandrzej Wąsowski, and Carlos Hernandez Corbato. 2022. MROS: runtime adaptation for robot control architectures. *Advanced Robotics* 36, 11 (2022), 502–518.

[33] Thomas Brand and Holger Giese. 2018. Towards generic adaptive monitoring. In *Proceedings of the International Conference on Self-Adaptive and Self-Organizing Systems.* 156–161.

[34] Melanie Brinkschulte, Christian Becker, and Christian Krupitzer. 2019. Towards a QoS-aware Cyber Physical Networking Middleware Architecture. In *Proceedings of the 1st International Workshop on Middleware for Lightweight, Spontaneous Environments.* 7–12.

[35] Rodney Brooks. 1986. A robust layered control system for a mobile robot. *IEEE Journal on Robotics and Automation* 2, 1 (1986), 14–23.

[36] Roberto Bruni, Andrea Corradini, Fabio Gadducci, Alberto Lluch Lafuente, and Andrea Vandin. 2012. A conceptual framework for adaptation. *Lecture Notes in Computer Science* 7212 LNCS (2012), 240–254.

[37] Antonio Bucchiarone, Claudio Guidi, Ivan Lanese, Nelly Bencomo, and Josef Spillner. 2022. A MAPE-K Approach to Autonomic Microservices. In *Proceedings of the International Conference on Software Architecture Companion.* 100–103.

[38] Radu Calinescu, Carlo Ghezzi, Marta Kwiatkowska, and Raffaela Mirandola. 2012. Self-adaptive software needs quantitative verification at runtime. *Commun. ACM* 55, 9 (9 2012), 69–77.

[39] Radu Calinescu, Lars Grunske, Marta Kwiatkowska, Raffaela Mirandola, and Giordano Tamburrelli. 2011. Dynamic QoS management and optimization in service-based systems. *IEEE Transactions on Software Engineering* 37, 3 (2011), 387–409.

[40] Miguel Camelo, Luca Cominardi, Marco Gramaglia, Marco Fiore, Andres Garcia-Saavedra, Lidia Fuentes, Danny De Vleeschauwer, Paola Soto-Arenas, Nina Slamnik-Krijestorac, Joaquin Ballesteros, Chia-Yu Chang, Gabriele Baldoni, Johann M. Marquez-Barja, Peter Hellinckx, and Steven Latre. 2022. Requirements and Specifications for the Orchestration of Network Intelligence in 6G. In *Proceedings of the Consumer Communications and Networking Conference.*

[41] Matteo Camilli, Carlo Bellettini, and Lorenzo Capra. 2018. A high-level petri net-based formal model of Distributed Self-adaptive Systems. In *Proceedings of the European Conference on Software Architecture: Companion.*

[42] Marc Carwehl, Calum Imrie, Thomas Vogel, Genaína Rodrigues, Radu Calinescu, and Lars Grunske. 2024. Formal Synthesis of Uncertainty Reduction Controllers. In *Proceedings of the 19th International Symposium on Software Engineering for Adaptive and Self-Managing Systems.* 2–13.

[43] Emiliano Casalicchio and Monica Palmirani. 2015. A Cloud Service Broker with Legal-Rule Compliance Checking and Quality Assurance Capabilities. *Procedia Computer Science* 68 (2015), 136–150.

[44] Betty H C Cheng, East Lansing, Robert Jared Clark, East Lansing, Michael Austin Langford, and Philip K Mckinley. 2020. AC-ROS : Assurance Case Driven Adaptation for the Robot Operating System. In *Proceedings of the International Conference on Model Driven Engineering Languages and Systems*. 102–113.

[45] Julián Cifuentes, Andrés Paz, and Hugo Arboleda. 2017. PISCIS: A constraint-based planner for self-adaptive systems. In *Advances in Computing*, Andrés Solano and Hugo Ordoñez (Eds.), Vol. 735. 282–296.

[46] Edmund M Clarke and E Allen Emerson. 1981. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Workshop on logic of programs*. Springer, 52–71.

[47] Louis Closson, Christophe Cerin, Didier Donsez, and Denis Trystram. 2022. Towards a Methodology for the Characterization of IoT Data Sets of the Smart Building Sector. In *Proceedings of the International Smart Cities Conference*. 1–7.

[48] Jorge Luiz da Silva, Márcio Miranda Assis, Alexandre Braga, and Regina Moraes. 2019. Deploying privacy as a service within a cloud-based framework. In *2019 9th Latin-American Symposium on Dependable Computing (LADC)*.

[49] Rafael Ferreira da Silva, Gideon Juve, Mats Rynge, Ewa Deelman, and Miron Livny. 2015. Online Task Resource Consumption Prediction for Scientific Workflows. In *Parallel Processing Letters*, Vol. 25. 1541003.

[50] Thiago Pereira Da Silva, Aluizio F.Rocha Neto, Thais Vasconcelos Batista, Frederico A.S. Lopes, Flavia C. Delicato, and Paulo F. Pires. 2021. Horizontal Auto-Scaling in Edge Computing Environment using Online Machine Learning. In *Proceedings of the International Conference on Dependable, Autonomic and Secure Computing*. 161–168.

[51] Gabriella D'Andrea, Tania Di Mascio, and Giacomo Valente. 2019. Self-adaptive loop for CPSs: Is the Dynamic Partial Reconfiguration profitable?. In *Proceedings of the Mediterranean Conference on Embedded Computing*. 1–5.

[52] Rustem Dautov, Dimitrios Kourtesis, Iraklis Paraskakis, and Mike Stannett. 2013. Addressing self-management in cloud platforms: a semantic sensor web approach. 11–18.

[53] Alexandre David, Lasse Jacobsen, Morten Jacobsen, Kenneth Yrke Jørgensen, Mikael H. Møller, and Jiří Srba. 2012. TAPAAL 2.0: Integrated Development Environment for Timed-Arc Petri Nets. In *Tools and Algorithms for the Construction and Analysis of Systems*, Cormac Flanagan and Barbara König (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 492–497.

[54] Jim Davies and Jim Woodcock. 1996. *Using Z: Specification Refinement and Proof.*

[55] Alessandra De Benedictis, Francesco Flammini, Nicola Mazzocca, Alessandra Somma, and Francesco Vitale. 2023. Digital Twins for Anomaly Detection in the Industrial Internet of Things: Conceptual Architecture and Proof-of-Concept. *IEEE Transactions on Industrial Informatics* 19, 12 (2023), 11553–11563.

[56] Patrícia Deud Guimarães and Vânia Paula De Almeida Neris. 2024. A Tool-Supported Approach to Adapt Web User Interfaces Considering the Emotional State of the User. In *Proceedings of the Brazilian Symposium on Human Factors in Computing Systems*. Association for Computing Machinery, Article 30, 11 pages.

[57] Alhassan Boner Diallo, Hiroyuki Nakagawa, and Tatsuhiro Tsuchiya. 2021. Adaptation space reduction using an explainable framework. In *Proceedings of the Annual Computers, Software, and Applications Conference*. 1653–1660.

[58] Martin Doran, Roy Sterritt, George Wilkie, and George Wilkie. 2020. Autonomic architecture for fault handling in mobile robots. *Innovations in Systems and Software Engineering* 16, 3 (2020), 263–288.

[59] Manuel Dworzak, Marcel Großmann, and Duy Thanh Le. 2024. Federated Autonomous Orchestration in Fog Computing Systems. In *Proceedings of Eighth International Congress on Information and Communication Technology*, Xin-She Yang, R. Simon Sherratt, Nilanjan Dey, and Amit Joshi (Eds.). Singapore, 639–649.

[60] Ibrahim Elgendi, Md. Farhad Hossain, Abbas Jamalipour, and Kumudu S. Munasinghe. 2019. Protecting Cyber Physical Systems Using a Learned MAPE-K Model. *IEEE Access* 7 (2019), 90954–90963.

[61] D. Elsayed, E. Nasr, A. El Ghazali, and M. Gheith. 2020. A self-healing model for qos-aware web service composition. *International Arab Journal of Information Technology* 17, 6 (2020), 839–846.

[62] Doaa Elsayed, Eman Nasr, Alaa El Ghazali, and Mervat Gheith. 2020. A Self-Healing Model for QoS-aware Web Service Composition. *The International Arab Journal of Information Technology* 17, 6 (11 2020), 839–846.

[63] Mahsa Emami-Taba, Mehdi Amoui, and Ladan Tahvildari. 2015. Strategy-Aware Mitigation Using Markov Games for Dynamic Application-Layer Attacks. *Proceedings of IEEE International Symposium on High Assurance Systems Engineering* 2015-Janua, January (2015), 134–141.

[64] Elif Eryilmaz, Frank Trollmann, and Sahin Albayrak. 2019. Quality-Aware Service Selection Approach for Adaptive Context Recognition in IoT. In *Proceedings of the International Conference on the Internet of Things (IoT '19)*. Article 3, 8 pages.

[65] Lukas Esterle. 2018. Goal-Aware Team Affiliation in Collectives of Autonomous Robots. In *Proceedings of the International Conference on Self-Adaptive and Self-Organizing Systems*. 90–99.

[66] Lukas Esterle and John N. A. Brown. 2020. I Think Therefore You Are: Models for Interaction in Collectives of Self-aware Cyber-physical Systems. *ACM Transactions on Cyber-Physical Systems* 4, 4, Article 39 (jun 2020), 25 pages.

[67] Imane Ettahiri and Karim Doumi. [n. d.]. A Novel Dynamic Enterprise Architecture Model: Leveraging MAPE-K Loop and Case-Based Reasoning for Context Awareness. 14, 2 ([n. d.]), 1875.

[68] Xinwei Fang, Radu Calinescu, Colin Paterson, and Julie Wilson. 2022. PRESTO: Predicting System-level Disruptions through Parametric Model Checking. In *Proceedings of the 17th Symposium on Software Engineering for Adaptive and Self-Managing Systems*. 91–97.

[69] Mohammad Faraji Mehmandar, Sam Jabbehdari, and Hamid Haj Seyyed Javadi. 2020. A dynamic fog service provisioning approach for IoT applications. *International Journal of Communication Systems* 33, 14 (2020).

[70] Mohammad Faraji-Mehmandar, Sam Jabbehdari, and Hamid Haj Seyyed Javadi. 2023. Fuzzy Q-learning approach for autonomic resource provisioning of IoT applications in fog computing environments. *Journal of Ambient Intelligence and Humanized Computing* 14, 4 (2023), 4237–4255.

[71] Dawei Feng, Cécile Germain, and Julien Nauroy. 2015. Sequential fault monitoring. In *Proceedings of the International Conference on Cloud and Autonomic Computing*. 25–34.

[72] Hao Feng, Cláudio Gomes, Santiago Gil, Peter H. Mikkelsen, Daniella Tola, Peter Gorm Larsen, and Michael Sandberg. 2022. Integration Of The Mape-K Loop In Digital Twins. In *Proceedings of the Annual Modeling and Simulation Conference*. 102–113.

[73] Islam Gamal, Hala Abdel-Galil, and Atef Ghalwash. 2022. Osmotic Message-Oriented Middleware for Internet of Things. *Computers* 11, 4 (2022).

[74] Angelo Gargantini, Paolo Arcaini, Patrizia Scandurra, and Silvia Bonfanti. [n. d.]. The ASMETA toolset website. http://asmeta.sourceforge.net/. Accessed: 2024.

[75] Michael Georgeff, Barney Pell, Martha Pollack, Milind Tambe, and Michael Wooldridge. 1999. The Belief-Desire-Intention Model of Agency. In *Intelligent Agents V: Agents Theories, Architectures, and Languages*, Jörg P. Müller, Anand S. Rao, and Munindar P. Singh (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1–10.

[76] Ilias Gerostathopoulos, Dominik Skoda, Frantisek Plasil, Tomas Bures, and Alessia Knauss. 2019. Tuning self-adaptation in cyber-physical systems through architectural homeostasis. *Journal of Systems and Software* 148 (2019), 37–55.

[77] Sona Ghahremani, Christian M. Adriano, and Holger Giese. 2018. Training Prediction Models for Rule-Based Self-Adaptive Systems. In *Proceedings of the International Conference on Autonomic Computing*. 187–192.

[78] Omid Gheibi and Danny Weyns. 2024. Dealing with Drift of Adaptation Spaces in Learning-based Self-Adaptive Systems Using Lifelong Self-Adaptation. *ACM Trans. Auton. Adapt. Syst.* 19, 1, Article 5 (Feb. 2024), 57 pages.

[79] Omid Gheibi, Danny Weyns, and Federico Quin. 2021. Applying Machine Learning in Self-adaptive Systems. *ACM Transactions on Autonomous and Adaptive Systems* 15, 3 (2021).

[80] Thomas Göthel, Nils Jähnig, and Simon Seif. 2017. Refinement-Based Modelling and Verification of Design Patterns for Self-adaptive Systems. In *Formal Methods and Software Engineering*, Zhenhua Duan and Luke Ong (Eds.). 157–173.

[81] Marwa Hachicha, Riadh Ben Halima, and Ahmed Hadj Kacem. 2017. Design and timed verification of self-adaptive systems. In *Proceedings of the International Conference on Computer and Information Science*. 227–232.

[82] Amira Hakim, Abdelkrim Amirat, and Mourad Chabane Oussalah. 2020. Non-intrusive contextual dynamic reconfiguration of ambient intelligent IoT systems. *Journal of Ambient Intelligence and Humanized Computing* 11, 4 (2020), 1365–1376.

[83] Maximilian Hoffmann, Lukas Malburg, and Ralph Bergmann. 2022. ProGAN: Toward a Framework for Process Monitoring and Flexibility by Change via Generative Adversarial Networks. In *Business Process Management Workshops*, Andrea Marella and Barbara Weber (Eds.), Vol. 436 LNBIP. 43–55.

[84] Zhong-Sheng Hou and Zhuo Wang. 2013. From model-based control to data-driven control: Survey, classification and perspective. *Information Sciences* 235 (2013), 3–35. Data-based Control, Decision, Scheduling and Fault Diagnostics.

[85] IBM. 2005. *An architectural blueprint for autonomic computing*. Technical Report.

[86] Usman M. Iftikhar and Danny Weyns. 2014. ActivFORMS: Active formal models for self-adaptation. In *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. 125–134.

[87] Didac Gil De La Iglesia and Danny Weyns. [n. d.]. MAPE-K Formal Templates to Rigorously Design Behaviors for Self-Adaptive Systems. 10, 3 ([n. d.]), 1–31.

[88] Lasse Jacobsen, Morten Jacobsen, Mikael H Møller, and Jiří Srba. 2011. Verification of timed-arc Petri nets. In *SOFSEM 2011: Theory and Practice of Computer Science: 37th Conference on Current Trends in Theory and Practice of Computer Science, Novỳ Smokovec, Slovakia, January 22-28, 2011. Proceedings 37*. Springer, 46–72.

[89] Pooyan Jamshidi, Javier Cámara, Bradley Schmerl, Christian Käestner, and David Garlan. 2019. Machine Learning Meets Quantitative Planning: Enabling Self-Adaptation in Autonomous Robots. In *2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. 39–50.

[90] Paulius Juodisius, Atrisha Sarkar, Raghava Rao Mukkamala, Michał Antkiewicz, Krzysztof Czarnecki, and Andrzej Wąsowski. 2019. Clafer: Lightweight Modeling of Structure, Behaviour, and Variability. *Art, Science, and Engineering of Programming* 3, 1 (2019).

[91] Somayeh Kalantari, Eslam Nazemi, and Behrooz Masoumi. 2021. Emergence-based self-advising in strong self-organizing systems: A case study in NASA ANTS mission. *Expert Systems with Applications* 182 (2021).

[92] Eduard Kamburjan, Crystal Chang Din, Rudolf Schlatte, S. Lizeth Tapia Tarifa, and Einar Broch Johnsen. 2022. Twinning-by-Construction: Ensuring Correctness for Self-adaptive Digital Twins. In *Leveraging Applications of Formal Methods, Verification and Validation. Verification Principles*, Tiziana Margaria and Bernhard Steffen (Eds.). 188–204.

[93] Burak Karaduman, Baris Tekin Tezel, and Moharram Challenger. 2022. Enhancing BDI Agents Using Fuzzy Logic for CPS and IoT Interoperability Using the JaCa Platform. *Symmetry* 14, 7 (2022).

[94] Jeffrey O. Kephart and David M. Chess. 2003. The Vision of autonomic computing. *Computer* 36, 1 (2003), 41–50.

[95] Atif Ishaq Khan, Syed Asad Raza Kazmi, and Awais Qasim. 2023. Formal Modeling of Self-Adaptive Resource Scheduling in Cloud. *Computers, Materials and Continua* 74, 1 (2023), 1183–1197.

[96] Reihaneh Khorsand, Mostafa Ghobaei-Arani, and Mohammadreza Ramezanpour. 2018. FAHP approach for autonomic resource provisioning of multitier applications in cloud computing environments. *Software: Practice and Experience* 48, 12 (2018), 2147–2173. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.2627

[97] R. Khorsand, M. Ghobaei-Arani, and M. Ramezanpour. 2018. FAHP approach for autonomic resource provisioning of multitier applications in cloud computing environments. *Software - Practice and Experience* 48, 12 (2018), 2147–2173.

[98] Mira Kim and Jennifer Jin. 2022. Autonomous Sprinkler System with MAPE-K. In *Proceedigns of the International Conference on Computational Science and Computational Intelligence*. 147–152.

[99] Barbara Kitchenham, Stuart Charters, et al. 2007. Guidelines for performing systematic literature reviews in software engineering.

[100] Verena Klös, Thomas Göthel, and Sabine Glesner. 2015. Adaptive Knowledge Bases in Self-Adaptive System Design. In *Proceedings of the Conference on Software Engineering and Advanced Applications*. 472 – 478.

[101] Verena Klös, Thomas Göthel, and Sabine Glesner. 2018. Comprehensible and dependable self-learning self-adaptive systems. *Journal of Systems Architecture* 85-86 (2018), 28–42.

[102] Martin Koehler, Yuriy Kaniovskyi, and Siegfried Benkner. 2011. An adaptive framework for the execution of data-intensive MapReduce applications in the Cloud. *IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum* (2011), 1122–1131.

[103] Max Korn, Philipp Chrszon, Sascha Klüppelholz, Christel Baier, and Sascha Wunderlich. 2023. Effectiveness of Pre-computed Knowledge in Self-adaptation - A Robustness Study. In *Computer Performance Engineering*, Katja Gilly and Nigel Thomas (Eds.), Vol. 13659 LNCS. 19–34.

[104] Samuel Kounev, Peter Lewis, Kirstie L. Bellman, Nelly Bencomo, Javier Camara, Ada Diaconescu, Lukas Esterle, Kurt Geihs, Holger Giese, Sebastian Götz, Paola Inverardi, Jeffrey O. Kephart, and Andrea Zisman. 2017. *The Notion of Self-aware Computing*. Springer International Publishing, 3–16.

[105] Christian Krupitzer, Felix Maximilian Roth, Sebastian Vansyckel, and Christian Becker. 2015. Towards reusability in autonomic computing. In *Proceedings of the International Conference on Autonomic Computing*. 115–120.

[106] An Ngoc Lam and Oystein Haugen. 2018. Supporting IoT semantic interoperability with autonomic computing. *Proceedings - 2018 IEEE Industrial Cyber-Physical Systems, ICPS 2018* (2018), 761–767.

[107] Peter G. Larsen, Shaukat Ali, Roland Behrens, Ana Cavalcanti, Claudio Gomes, Guoyuan Li, Paul De Meulenaere, Mikkel L. Olsen, Nikolaos Passalis, Thomas Peyrucain, Jesús Tapia, Anastasios Tefas, and Houxiang Zhang. 2024. Robotic safe adaptation in unprecedented situations: the RoboSAPIENS project. *Research Directions: Cyber-Physical Systems* 2 (oct 2024), e4.

[108] Sabah Lecheheb, Soufiane Boulehouache, and Said Brahimi. 2022. Improving Self-Adaptation by Combining MAPE-K, Machine and Deep Learning. In *2022 2nd International Conference on New Technologies of Information and Communication (NTIC)*. 1–6.

[109] Peter R. Lewis, Lukas Esterle, Arjun Chandra, Bernhard Rinner, Jim Torresen, and Xin Yao. 2015. Static, Dynamic, and Adaptive Heterogeneity in Distributed Smart Camera Networks. *ACM Transactions on Autonomous and Adaptive Systems* 10, 2 (June 2015), 1–30.

[110] Nianyu Li, Di Bai, Yiming Peng, Zhuoqun Yang, and Wenpin Jiao. 2018. Verifying stochastic behaviors of decentralized self-adaptive systems: A formal modeling and simulation based approach. In *Proceedings of the International Conference on Software Quality, Reliability and Security*. IEEE, 67–74.

[111] M. Liess, J. Demicoli, T. Tiedje, M. Lohrmann, M. Nickel, M. Luniak, D. Prousalis, T. Wild, R. Tetzlaff, D. Göhringer, C. Mayr, K. Bock, S. Steinhorst, and A. Herkersdorf. 2023. X-MAPE: Extending 6G-Connected Self-Adaptive Systems with Reflexive Actions. In *Proceedings of the Conference on Network Function Virtualization and Software Defined Networks*. 163–167.

[112] Yang Liu, Di Bai, and Wenpin Jiao. 2018. Generating Adaptation Rules of Software Systems: A Method Based on Genetic Algorithm. In *Proceedings of the International Conference on Machine Learning and Computing*. 347–356.

[113] Shunan Ma and Yazhe Wang. [n. d.]. Self-Adaptive Access Control Model Based on Feedback Loop. In *Proceedings of the International Conference on Cloud Computing and Big Data*. 597–602.

[114] Basel Magableh and Muder Almiani. 2020. A deep recurrent Q network towards self-adapting distributed microservice architecture. *Software - Practice and Experience* 50, 2 (2020), 116–135. arXiv:1901.04011

[115] Lukas Malburg, Maximilian Hoffmann, and Ralph Bergmann. 2023. Applying MAPE-K control loops for adaptive workflow management in smart factories. *Journal of Intelligent Information Systems* 61, 1 (2023), 83–111.

[116] Allen Marshall, Sharmin Jahan, and Rose Gamble. 2018. Toward evaluating the impact of self-adaptation on security control certification. In *Proceedings of the International Conference on Software Engineering for Adaptive and Self-Managing Systems*. 149–160.

[117] Nikolai Matni, Aaron D. Ames, and John C. Doyle. 2024. A Quantitative Framework for Layered Multirate Control: Toward a Theory of Control Architecture. *IEEE Control Systems Magazine* 44, 3 (2024), 52–94.

[118] A. Mazidi, M. Golsorkhtabaramiri, and M.Y. Tabari. 2020. Autonomic resource provisioning for multilayer cloud applications with K-nearest neighbor resource scaling and priority-based resource allocation. *Software - Practice and Experience* 50, 8 (2020), 1600–1625.

[119] Arash Mazidi, Mehregan Mahdavi, and Fahimeh Roshanfar. 2021. An autonomic decision tree-based and deadline-constraint resource provisioning in cloud applications. *Concurrency and Computation: Practice and Experience* 33, 10 (2021).

[120] Jeremy Mechouche, Roua Touihri, Mohamed Sellami, and Walid Gaaloul. 2022. Conformance checking for autonomous multi-cloud SLA management and adaptation. *Journal of Supercomputing* 78, 11 (2022), 13004–13039.

[121] Mohammad Faraji Mehmandar, Sam Jabbehdari, and Hamid Haj Seyyed Javadi. 2020. A dynamic fog service provisioning approach for IoT applications. *International Journal of Communication Systems* July (2020).

[122] Timo Müller, Simon Kamm, Andreas Löcklin, Dustin White, Marius Mellinger, Nasser Jazdi, and Michael Weyrich. 2022. Architecture and knowledge modelling for self-organized reconfiguration management of cyber-physical production systems. *International Journal of Computer Integrated Manufacturing* 36, 12 (2022), 1842–1863.

[123] Negin Najafizadegan, Eslam Nazemi, and Vahid Khajehvand. 2021. An autonomous model for self-optimizing virtual machine selection by learning automata in cloud environment. *Software - Practice and Experience* 51, 6 (2021), 1352–1386.

[124] Suneth Namal, Hasindu Gamaarachchi, Gyu MyoungLee, and Tai-Won Um. 2016. Autonomic trust management in cloud-based and highly dynamic IoT applications. In *2015 ITU Kaleidoscope: Trust in the Information Society (K-2015)*.

[125] Nathalia Nascimento, Paulo Alencar, and Donald Cowan. [n. d.]. Self-Adaptive Large Language Model (LLM)-Based Multiagent Systems. In *Proceedings of the International Conference on Autonomic Computing and Self-Organizing Systems Companion*. 104–109.

[126] Mohammadreza Nazeri, Mohammadreza Soltanaghaei, and Reihaneh Khorsand. 2024. A predictive energy-aware scheduling strategy for scientific workflows in fog computing. *Expert Systems with Applications* 247 (2024).

[127] Septimiu Nechifor, Dan Puiu, Bogdan Târnaucă, and Florin Moldoveanu. 2015. Autonomic Aspects of IoT Based Systems: A Logistics Domain Scheduling Example. In *Interoperability and Open-Source Solutions for the Internet of Things*, Ivana Podnar Žarko, Krešimir Pripužić, and Martin Serrano (Eds.). 153–168.

[128] Jesús Ortiz. 2023. Dealing with the evolution of event-based choreographies of BPMN fragments. In *CEUR Workshop Proceedings*, Vol. 3618.

[129] Jesús Ortiz, Victoria Torres, and Pedro Valderas. 2023. Dealing with the Evolution of Event-Based Choreographies of BPMN Fragments: Definition and Proof of Concept. In *Proceedings of the International Conference on Conceptual Modeling*. 296–313.

[130] Selma Ouareth, Soufiane Boulehouache, and Smaine Mazouzi. 2021. An Approach for Composing Multiple Control Loops Hierarchically. In *Proceedings of the International Conference on Theoretical and Applicative Aspects of Computer Science*.

[131] Dimitrios Papamartzivanos, Félix Gómez Mármol, and Georgios Kambourakis. 2019. Introducing Deep Learning Self-Adaptive Misuse Network Intrusion Detection Systems. *IEEE Access* 7 (2019), 13546–13560.

[132] Saeedeh Parsaeefard, Pooyan Habibi, and Alberto Leon Garcia. 2022. Towards Interaction and Conflict Management in AI-assisted Operational Control Loops in 6G. In *2022 IEEE Future Networks World Forum (FNWF)*. 670–675.

[133] Alireza Parvizi-Mosaed, Shahrouz Moaven, Jafar Habibi, Ghazaleh Beigi, and Mahdieh Naser-Shariat. 2015. Towards a self-adaptive service-oriented methodology based on extended SOMA. *Frontiers of Information Technology and Electronic Engineering* 16, 1 (2015), 43–69.

[134] Martin Pfannemüller, Martin Breitbach, Christian Krupitzer, Markus Weckesser, Christian Becker, Bradley Schmerl, and Andy Schurr. 2020. REACT: A model-based runtime environment for adapting communication systems. In *Proceedings of the International Conference on Autonomic Computing and Self-Organizing Systems*. 65–74.

[135] Amir Pnueli. 1977. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*. 46–57.

[136] Francesco Poggi, Davide Rossi, Paolo Ciancarini, and Luca Bompani. 2016. An application of semantic technologies to self adaptations. In *2016 IEEE 2nd International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI)*.

[137] Jesús M T Portocarrero, Flavia C Delicato, Paulo F Pires, and Thais V Batista. 2014. Reference Architecture for Self-adaptive Management in Wireless Sensor Networks. In *Proceedings of the International Conference on Adaptive and Intelligent Systems*, A. Bouchachia (Ed.). 110–120.

[138] Laurin Prenzel and Sebastian Steinhorst. 2022. Towards Resilience by Self-Adaptation of Industrial Control Systems. In *Proceedings of the International Conference on Emerging Technologies and Factory Automation*.

[139] Awais Qasim and Syed Asad Raza Kazmi. 2019. Formal modelling of real-time self-adaptive multi-agent systems. *Intelligent Automation and Soft Computing* 25, 1 (2019), 49–63.

[140] Federico Quin, Danny Weyns, Thomas Bamelis, Singh Buttar Sarpreet, and Sam Michiels. 2019. Efficient analysis of large adaptation spaces in self-adaptive systems using machine learning. In *Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, Vol. 2019-May. 1 – 12.

[141] Federico Quin, Danny Weyns, and Omid Gheibi. 2021. Decentralized Self-Adaptive Systems: A Mapping Study. In *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. 18–29.

[142] Houda Rachidi and Ahmed Karmouch. 2011. A framework for self-configuring devices using TR-069. In *Proceedings of the International Conference on Multimedia Computing and Systems*. 1–6.

[143] Anand S Rao and Michael P Georgeff. 1997. Modeling rational agents within a BDI-architecture. *Readings in agents* (1997), 317–328.

[144] Vincenzo Riccio, Giancarlo Sorrentino, Matteo Camilli, Raffaela Mirandola, and Patrizia Scandurra. 2023. Engineering Self-adaptive Microservice Applications: An Experience Report. In *Service-Oriented Computing*, Flavia Monti, Stefanie Rinderle-Ma, Antonio Ruiz Cortés, Zibin Zheng, and Massimo Mecella (Eds.), Vol. 14419 LNCS. 227–242.

[145] A. W. Roscoe. 1997. *The Theory and Practice of Concurrency.* Prentice Hall PTR, USA.

[146] Eric Rutten, Nicolas Marchand, and Daniel Simon. 2017. Feedback control as MAPE-K loop in autonomic computing. In *Software Engineering for Self-Adaptive Systems III. Assurances*, Rogério de Lemos, David Garlan, Carlo Ghezzi, and Holger Giese (Eds.), Vol. 9640 LNCS. 349–373.

[147] Manuel Sanchez, Ernesto Exposito, and Jose Aguilar. 2020. Autonomic computing in manufacturing process coordination in industry 4.0 context. *Journal of Industrial Information Integration* 19, July (2020), 100159.

[148] Ronny Seiger, Stefan Herrmann, and Uwe Aßmann. 2017. Self-healing for distributed workflows in the internet of things. In *Proceedigns of the Conference on Software Architecture Workshops*. 72–79.

[149] Ronny Seiger, Stefan Huber, Peter Heisig, and Uwe Aßmann. 2016. Enabling Self-adaptive Workflows for Cyber-physical Systems. In *Enterprise, Business-Process and Information Systems Modeling*, Rainer Schmidt, Wided Guédria, Ilia Bider, and Sérgio Guerreiro (Eds.). 3–17.

[150] Ronny Seiger, Stefan Huber, Peter Heisig, and Uwe Aßmann. 2019. Toward a framework for self-adaptive workflows in cyber-physical systems. *Software and Systems Modeling* 18, 2 (2019), 1117–1134.

[151] Prasanth Senthilvelan, Jialong Li, and Kenji Tei. [n. d.]. Similarity-Based Shield Adaptation under Dynamic Environment. In *Proceedings of the International Conference on Software Engineering and Artificial Intelligence*, Vol. 16. 33–39.

[152] Gustavo Rezende Silva, Juliane Päßler, Jeroen Zwanepol, Elvin Alberts, S. Lizeth Tapia Tarifa, Ilias Gerostathopoulos, Einar Broch Johnsen, and Carlos Hernández Corbato. 2023. SUAVE: An Exemplar for Self-Adaptive Underwater Vehicles. In *Proceedings of the Symposium on Software Engineering for Adaptive and Self-Managing Systems*. 181–187.

[153] Madhusudan Singh and Shiho Kim. 2017. Reconcile security requirements for intelligent vehicles. In *Proceedings of the International Conference on Control, Automation and Systems*. 1646–1651.

[154] Jiyoung Song, Jeehoon Kang, Sangwon Hyun, Eunkyoung Jee, and Doo-Hwan Bae. 2022. Continuous verification of system of systems with collaborative MAPE-K pattern and probability model slicing. *Information and Software Technology* 147 (2022).

[155] P. Soto, M. Camelo, D. De Vleeschauwer, Y. De Bock, C.-Y. Chang, J.F. Botero, and S. Latre. 2023. Network Intelligence for NFV Scaling in Closed-Loop Architectures. *IEEE Communications Magazine* 61, 6 (2023), 66–72.

[156] Carlos H. R. Souza, Saulo S. De Oliveira, Luciana O. Berretta, and Sergio T. de Carvalho. 2024. DDA-MAPEKit: A Framework for Dynamic Difficulty Adjustment Based on MAPE-K Loop. In *Proceedings of the Brazilian Symposium on Games and Digital Entertainment*. 1–10.

[157] Dick Stenmark. 2001. The relationship between information and knowledge. In *Proceedings of IRIS*, Vol. 24. 11–14.

[158] Mahsa Teimourikia and Mariagrazia Fugini. 2016. Ontology development for run-time safety management methodology in Smart Work Environments using ambient knowledge. *Future Generation Computer Systems* (2016).

[159] Dirk Thomas, William Woodall, and Esteve Fernández. 2014. Next-generation ROS: Building on DDS. https://api.semanticscholar.org/CorpusID:218740227

[160] Benoit Tremblay, Karol Kozubal, Wubin Li, and Chakri Padala. 2016. A Workload Aware Storage Platform for large scale computing environments: Key challenges and proposed directions. In *Proceedings of the ACM 7th Workshop on Scientific Cloud Computing*. 27–33.

[161] Geetha Lekshmy V, Amal S Pillai, and Arun Raj. 2022. Modeling & Verification of an Adaptive IoT System using Uppaal. In *Proceedings of the Global Conference for Advancement in Technology.*

[162] Felipe Volpato, Madalena Pereira Da Silva, Alexandre Leopoldo Goncalves, and Mario Antonio Ribeiro Dantas. 2017. An autonomic QoE-aware management architecture for software-defined networking. In *Proceedings of the International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises.* 220–225.

[163] Danny Weyns. 2019. Software engineering of self-adaptive systems. *Handbook of software engineering* (2019), 399–443.

[164] Danny Weyns and Tanvir Ahmad. 2013. Claims and Evidence for Architecture-Based Self-adaptation: A Systematic Literature Review. In *Software Architecture*, Khalil Drira (Ed.). 249–265.

[165] Danny Weyns and Usman M. Iftikhar. 2023. ActivFORMS: A Formally Founded Model-based Approach to Engineer Self-adaptive Systems. *ACM Transactions on Software Engineering and Methodology* 32, 1 (2023).

[166] Danny Weyns, Sam Malek, and Jesper Andersson. 2012. FORMS: Unifying reference model for formal specification of distributed self-adaptive systems. *ACM Trans. Auton. Adapt. Syst.* 7, 1, Article 8 (5 2012), 61 pages.

[167] Michael Weyrich and Christof Ebert. 2016. Reference architectures for the internet of things. *IEEE Software* 33, 1 (2016), 112 – 116.

[168] Thomas Wright, Cláudio Gomes, and Jim Woodcock. 2022. Formally Verified Self-adaptation of an Incubator Digital Twin. In *Leveraging Applications of Formal Methods, Verification and Validation. Practice*, Tiziana Margaria and Bernhard Steffen (Eds.). 89–109.

[169] Mahendra Pratap Yadav, Rohit, and Dharmendra Kumar Yadav. 2021. Maintaining container sustainability through machine learning. *Cluster Computing* 24, 4 (2021), 3725–3750.

[170] Kamaleddin Yaghoobirafi and Ali Farahani. 2022. An approach for semantic interoperability in autonomic distributed intelligent systems. *Journal of Software: Evolution and Process* 34 (2 2022).

[171] Wenhua Yang, Chang Xu, Yepang Liu, Chun Cao, Xiaoxing Ma, and Jian Lu. 2014. Verifying self-adaptive applications suffering uncertainty. *Proceedings of the International Conference on Automated Software Engineering* (2014), 199 – 209.

[172] Zhiyong Ye, Yang Wang, Shuibing He, Chengzhong Xu, and Xian-He Sun. 2021. Sova: A software-defined autonomic framework for virtual network allocations. *IEEE Transactions on Parallel and Distributed Systems* 32, 1 (2021), 116–130.

[173] Ji Zhang and Betty HC Cheng. 2006. Using temporal logic to specify adaptive program semantics. *Journal of Systems and Software* 79, 10 (2006), 1361–1369.

[174] Tianqi Zhao, Wei Zhang, Haiyan Zhao, and Zhi Jin. 2017. A Reinforcement Learning-Based Framework for the Generation and Evolution of Adaptation Rules. In *Proceedings of the International Conference on Autonomic Computing).* 103–112.

[175] Yongwang Zhao, Dianfu Ma, Jing Li, and Zhuqing Li. 2011. Model Checking of Adaptive Programs with Mode-extended Linear Temporal Logic. In *Proceedings of the International Conference and Workshops on Engineering of Autonomic and Autonomous Systems.* 40–48. ISSN: 2168-1872.

[176] Naweiluo Zhou, Gwenaël Delaval, Bogdan Robu, Éric Rutten, and Jean François Méhaut. 2018. An autonomic-computing approach on mapping threads to multi-cores for software transactional memory. *Concurrency and Computation: Practice and Experience* 30, 18 (2018), 1–21.

[177] Parisa Zoghi, Mark Shtern, Marin Litoiu, and Hamoun Ghanbari. 2016. Designing Adaptive Applications Deployed on Cloud Environments. *ACM Transactions on Autonomous and Adaptive Systems* 10, 4 (2016), 1–16.

[178] Abdellah Zyane, Mohamed Nabil Bahiri, and Abdelilah Ghammaz. 2020. IoTScal-H : Hybrid monitoring solution based on cloud computing for autonomic middleware-level scalability management within IoT systems and different SLA traffic requirements. *International Journal of Communication Systems* 33, 14 (2020), 1–23.

A.2 Runtime Verification of Autonomous Systems utilizing Digital Twins as a Service

The appended paper follows.

Funded by
the European Union

Runtime Verification of Autonomous Systems utilizing Digital Twins as a Service

Morten Haahr Kristensen*, Alberto Bonizzi†, Cláudio Gomes*, Simon Thrane Hansen‡, Carlos Isasa*, Hannes Iven§, Eduard Kamburjan¶, Peter Gorm Larsen*, Martin Leucker§, Prasad Talasila*, Valdemar Trøjgård Tang*, Stefano Tonetta†, Lars B. Vosteen§, Thomas Wright*

**Department of Electrical and Computer Engineering*
Aarhus University, Denmark
{mhk, claudio.gomes, cisasa, pgl, prasad.talasila, valdemar.tang, thomas.wright}@ece.au.dk

†*Fondazione Bruno Kessler, Italy*
{bonizzi,tonettas}@fbk.eu

‡*Interdisciplinary Centre for Security, Reliability and Trust*
University of Luxembourg
simon.hansen@uni.lu

§*Institute for Software Engineering and Programming Languages*
Universität zu Lübeck, Germany
hannes.iven@student.uni-luebeck.de, {leucker,vosteen}@isp.uni-luebeck.de

¶*Department of Informatics*
University of Oslo, Norway
eduard@ifi.uio.no

***Abstract*—Autonomous Systems (AS) enable systems to adapt to drastic and unprecedented environmental changes, a capability that can be enhanced through the utilization of Digital Twins (DTs). However, the additional capabilities of AS come at the cost of explainability, as the expanding adaptation space complicates the reasoning about the system's behavior. For certain types of systems, it is crucial to ensure that specific properties are upheld despite the system's autonomous behavior. To facilitate the monitoring of these properties, we propose the use of Runtime Verification (RV). This tutorial demonstrates the integration of RV tools into the Digital Twins as a Service (DTaaS) platform to monitor and verify the behavior of AS in real-time. By exploring various methods to incorporate RV tools within a DT context, the tutorial aims to advance the application of RV technologies in autonomic computing and self-adaptive system design. Specifically, we demonstrate how the behavior of a self-configuring DT can be verified utilizing RV. This is accomplished through the DTaaS platform, which supports seamless deployment of DT-based AS.**

***Index Terms*—Self-adaptivity, runtime verification, digital twins, monitor, TeSSLa, NuRV**

I. Introduction

The vision of autonomic computing inspired new approaches to designing flexible Autonomous Systems (AS) capable of adapting to dynamic environments [14]. Initially, research on AS primarily focused on reducing the complexity of large-scale software systems, which often comprise tens of millions of lines of code, particularly within purely software-based environments such as cloud computing. Since then, the field has advanced significantly, extending its concepts to new domains including dynamic software architectures [1], robotics [4], business process management (BPM) [16], and cyber-security [17]. However, the increasing level of adaptability makes the verification of such systems significantly more challenging. For instance, within the domain of robotics, there is a critical need for flexible self-adaptive robots that can operate reliably despite dynamic environmental changes. Nevertheless, the safety of human life must never be compromised, and ensuring that these robots adhere to safety standards despite their self-adaptivity remains an ongoing challenge. Research addressing these challenges, such as [4] and [13], involves formulating and validating the adherence to requirements at runtime. Similar requirements for continuous monitoring of system properties are present in other domains utilizing AS. Within this context, self-adaptivity can be considered as a component that increases the possible behaviors of the system, while Runtime Verification (RV) serves as the component that excludes unwanted behaviors.

The purpose of the tutorial is to demonstrate how researchers within the AS community can utilize RV tools within their work to ensure the correctness of their systems. In doing so, emphasis is placed on the different ways that RV tools can be integrated within a deployment platform and utilized by the existing system. Through this process, we aim that attendees will become familiar with how monitoring services are deployed, as well as gain practical insight into how to build them. The tutorial adopts a hands-on approach, utilizing the

The work presented here is partially supported by the RoboSAPIENS project funded by the European Commission's Horizon Europe programme under grant agreement number 101133807, the O5G-N-IoT project, funded by the German Federal Ministry for Economic Affairs and Climate Action, due to a resolution of the German Bundestag, and the SM4RTENANCE project under grant no. 101123423.

Incubator case study [9], [10], which features a Digital Twin (DT) capable of self-configuring during anomalous situations. Although the tutorial is presented within the context of a DT, the majority of the concepts discussed extend beyond this specific application. Through the Incubator, we explore five different scenarios for integrating an RV tool within an existing AS, all of which are deployed on the Digital Twin as a Service (DTaaS) platform (detailed in Section II).

The tutorial is structured as follows: Section II introduces the main background concepts for following the tutorial. Then Section III presents five examples showcasing the implementation of monitoring using two different RV frameworks. Finally, Section IV concludes the tutorial.

II. Background

A. Runtime Verification

RV is a lightweight method of improving the integrity of deployed systems by extending a system with additional monitoring functionality to avoid unintended behavior at runtime. This is accomplished through a variety of monitoring techniques, which check whether a system conforms to a specification based on traces or streams of data from the running system.

A wide range of RV methods have been developed over the years, offering a variety of different specification languages for expressing the desired behavior of the system including temporal logics such as Linear Temporal Logic (LTL) [18] and Signal Temporal Logic (STL) [7] as well as domain-specific languages such as TeSSLa [6].

RV encompasses both passive monitoring techniques which focus on detecting errors without changing the behavior of the system, as well as more active techniques (also known as *runtime enforcement* [8]) which aim to block or correct bad behaviors.

B. NuRV

NuRV [5][1] is an extension of the nuXmv model checker for assumption-based LTL RV with partial observability and resets. Monitoring formulas are specified in LTL while assumptions are specified in SMV. Thanks to the assumption, the output of the monitor can be conclusive even in cases where the formula contains future operators or if not all variables are observable.

The tool provides commands for online/offline monitoring and code generation into standalone monitor code. Using the online/offline monitor, LTL properties can be verified incrementally on finite traces from the system under scrutiny. The code generation currently supports C, C++, Common Lisp, and Java, and is extensible. Furthermore, from the same internal monitor automaton, the monitor can be generated into SMV modules, whose characteristics can be verified by Model Checking using nuXmv.

[1]https://nurv.fbk.eu

C. TeSSLa

The Temporal Stream-based Specification Language (TeSSLa) [6] framework[2] combines a language and a suite of tools designed for real-time verification of systems through data stream analysis. TeSSLa allows the declaration of input data types and the transformation of this data into new, derived streams by applying a series of defined operations. This approach enables effective monitoring of complex systems, ensuring accurate tracking and analysis without overly complex processes.

TeSSLa provides extensive libraries and supports the creation of macros. These macros allow users to define custom operations, simplifying the specification of complex behaviors and increasing the accessibility of the language. TeSSLa also supports the generation of detailed output streams, including statistical data with precise event timestamps, and allows integration with monitoring tools developed in modern programming languages such as Rust and Scala. Its integration with the metrics collection agent Telegraf [21] contributes to its effectiveness in real-world applications. At its core, TeSSLa's strength lies in its ability to map input data to meaningful outputs, which is essential for real-time system monitoring and informed decision-making in areas such as DT technologies.

D. Digital Twins as a Service

The DTaaS[3] platform is a collaborative platform to build, use, and share DTs. It is based-off a microservices architecture with dedicated software containers[4] for DT assets, user workspaces, platform services, a front-end website, and service router.

One of the architectural principles used in the development of DTaaS is to conceive DTs as composed of reusable assets, which separate the functionality into their constituent parts. Within DTaaS, data, models [23], tools [19], services [20] and ready to use DTs [2] have been identified as reusable assets. The DT Assets software container provides an interface to perform create, reuse, update, and delete operations on the reusable stored within the DTaaS.

Users utilizing DTaaS have private workspaces in which they can build and use systems, from where they can access assets as a regular part of the filesystem. All workspaces have internet access thereby enabling the integration of DTs running inside workspaces with external software systems.

Out-of-the-box, DTaaS supports multiple commonly used services across DTs and users. The most commonly used are RabbitMQ and MQTT (communication), InfluxDB and MongoDB (data storage), and Grafana (data visualization). Additionally, it is possible to host private services accessible to a selected number of users. These services include the runtime services provided by TeSSLa and NuRV.

[2]https://tessla.io
[3]https://github.com/INTO-CPS-Association/DTaaS
[4]Container is a software component at level-2 of the C4 model.

E. *FMI-based Co-simulation*

Integrating verification methods early in development ensures system correctness from the start [22]. One approach is *co-simulation*, which combines multiple simulation tools into a single simulation [12], [15]. Co-simulation is crucial for modeling complex systems co-developed by multiple organizations and systems whose complexity transcends the capabilities of any single simulation tool.

Interoperability between heterogeneous simulation tools is achieved using Functional Mock-up Units (FMUs) defined by the Functional Mock-up Interface (FMI) standard [3]. An FMU encapsulates the behavior of a *dynamic system*, whose state evolves according to *evolution rules* and *external stimuli*, into a discrete trajectory. This allows complex behaviors to be represented modularly while protecting intellectual property.

Multiple FMUs are composed into a *scenario* by coupling their input and output ports to represent the behavior of a complex system. A *coupling* signifies that the state of one FMU (the output) directly influences the state of another (the input). A scenario is simulated using a co-simulation framework that interacts with the FMUs through their interface to advance them in lockstep and exchange values between the coupled ports.

III. Example integrations

Five examples showcasing the RV integration into the AS are presented below: three utilizing NuRV and two utilizing TeSSLa. For NuRV, the first example demonstrates a scenario where the components of a self-adaptive DT are validated before the system is deployed. This involves exporting the NuRV specification as an FMU and conducting co-simulations with the other components of the system. In the second example, the reusability of the FMU within a service-oriented architecture is demonstrated, enabling RV on the deployed system. It listens to real-time sensing data sent by the Physical Twin (PT), i.e., the physical counterpart of the system, through RabbitMQ to the DT, evaluating the truth value of LTL formulas. In the third example, the NuRV specification is deployed on a standalone server, with its services exposed to the DT. As a result, it is uncoupled from the DT instance. For TeSSLa, the two examples demonstrate passive and active monitoring. In passive monitoring, an alarm is raised in the event of a violation of the monitored conditions. In contrast, active monitoring entails altering the system's behavior if a monitored condition is falsified.

A. *The Incubator*

The different ways of integrating RV monitors are showcased using the Incubator system described in [11]. The objective of the Incubator is to keep the temperature inside a box close to a target temperature, a task that can be difficult to achieve when more sophisticated scenarios, such as the possibility of someone opening the lid or the object inside the box releasing heat, are considered. These considerations have led to the development of a DT [9], which consists of a dynamical model of the physical components of the PT and software components capturing its controller behavior. Additionally, the DT contains a self-adaptation service that reacts to possible changes in the environment and adjusts the Incubator's objective as necessary. In order to do this, a Kalman filter estimates the state of the system and compares it to the empirical data from the sensors. As soon as a deviation is detected, the DT looks at historical data to identify the anomaly and plan accordingly.

The self-adaptation service is divided into two different services: anomaly detection, which handles detecting the difference between the expected temperatures and the sensed temperatures, and energy saving, which changes the target temperature to a lower one in case the anomaly detection service has detected an opening of the lid. An overview of the interaction of these services and the PT can be seen in Fig. 1. The runtime monitoring property that is used throughout the examples, ensures the correct combined behavior of the anomaly detection and energy saver blocks. The STL property can be seen below:

$$\Box(A \implies \Diamond_{[0,3]} S) \tag{1}$$

where A stands for the anomaly detection service detecting the opening of the lid and S stands for the energy saver service changing the target temperature. It can roughly be translated into: "It must always hold that if an anomaly is detected then energy saver is started within 3 seconds".

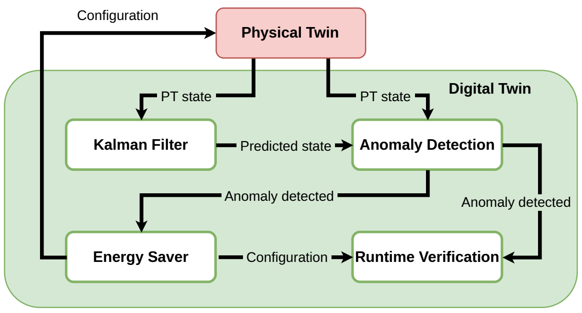


Fig. 1: High-level overview of the DT components relevant to the examples below. Arrows indicate RabbitMQ messages and associated data.

B. *NuRV FMU monitor*

NuRV provides the capability to export runtime monitors as standalone FMU components. This feature enables users to easily interface monitors within custom applications using a variety of libraries that support the FMU standard. This section outlines how to utilize these FMU monitors to perform an early validation of the internal components of the Incubator before its deployment. Specifically, it is shown how to validate the energy saver and anomaly detection components using an FMU monitor.

1) Monitor definition and integration: The monitor is defined by the SMV model seen in Fig. 2. This model specifies a safety LTL property: Whenever an anomaly occurs, the system should reconfigure itself and enter energy-saving mode within a maximum of 3 time steps. The output of this monitor can be a final verdict of either *true* or *false*, or *unknown* if there is insufficient information to reach a definitive conclusion.

```
MODULE main
VAR
   anomaly : boolean;
   energy_saving : boolean;
LTLSPEC -- Safety
   G (anomaly -> F [ 0, 3 ] energy_saving)
```

Fig. 2: NuRV monitor model

2) Simulation environment: For the validation process, a simulation environment was established comprising several components (depicted in Fig. 3): the *energy saver* and *anomaly detection* components, each encapsulated within distinct FMUs, along with the NuRV monitor, exported by NuRV using the specification in Fig. 2. The input data for the simulation is generated by a purpose-built FMU component named *source*, which supplies testing data, simulating an anomaly occurring at time t=60s. A final component, *watcher*, is employed to verify whether the energy saver activates in response to an anomaly reported by the anomaly detector. The FMUs for the energy saver and anomaly detector were constructed packaging their Python code using unifmu[5], while the *source* and *watcher* components were generated using OpenModelica[6]. maestro[7] served as the co-simulation engine.

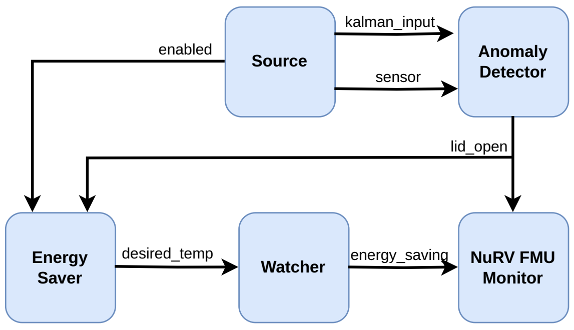


Fig. 3: Simulation architecture of components and their exchanged signals

3) Simulation creation and execution: The setup of the simulation is automatically performed through the create script, which installs the required dependencies and compiles the monitor from its specification model. The simulation is initiated using the DTaaS execute script, which also starts the *maestro* co-simulation engine to simulate the system. Given the characteristics of the FMU monitor exported by NuRV, each invocation of the *doStep* function corresponds to a logical heartbeat of the monitor. Consequently, this allows the monitor to assess the current values of its inputs and determine the appropriate outcome, thus providing validation of the system.

C. NuRV FMU service monitor

It is not possible to directly integrate the NuRV FMU monitor with the deployed Incubator. However, by implementing straightforward wrapper logic, it becomes viable to expose the FMU as an internal service, thereby enabling its utilization within the system. Theoretically, automating this process could be achieved through the development of a dedicated tool; however, no such tool currently exists to the authors' knowledge. For the purposes of this tutorial, a Python-based prototype has been developed to demonstrate the potential functionality of such a tool. It is important to underscore that this solution serves as a prototype only, and certain challenges, such as fault tolerance, remain unaddressed.

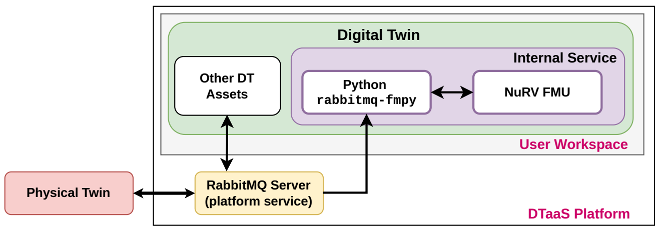


Fig. 4: Overview of components involved with the NuRV FMU service monitor. Notice that rabbitmq and fmpy are libraries.

1) Overview: As depicted in Fig. 4, the tool leverages the Python libraries rabbitmq[8] and fmpy[9], to realize its functionality. rabbitmq facilitates subscription to the RabbitMQ topics that are relevant for the monitor. fmpy enables the simulation of an FMU within Python, allowing the introduction of custom logic between each simulation step. In conjunction, this tool orchestrates its operations such that upon the occurrence of a new message on a RabbitMQ topic, the internal state of the Python program is updated, and the signals are subsequently forwarded to the FMU monitor, resulting in the generation of a new verdict.
Given the reuse of the FMU, the NuRV specification remains identical to the one outlined previously.

2) Creation, execution, and termination: As an extension of the configuration provided in Section III-B, the prerequisites are a superset of those previously outlined. Additionally, the Python libraries fmpy and rabbitmq must be installed. As a consequence, the create script fulfills the same function as

[5]https://github.com/INTO-CPS-Association/unifmu
[6]https://openmodelica.org/
[7]https://github.com/INTO-CPS-Association/maestro

[8]https://pypi.org/project/rabbitmq/
[9]https://pypi.org/project/FMPy/

described above, while also installing the requisite additional Python libraries.
In this configuration of the Incubator, an additional service in the form of the RV monitor is initiated concurrently with the DT. Given that the monitor is deployed as an internal service, it becomes the responsibility of the DT to manage the monitor, thereby intertwining their lifecycles. Consequently, the `execute` script commences the DT as usual but with the inclusion of starting the monitor. This enables the continuous monitoring of the anomaly detection and energy saving blocks, facilitating their verification at runtime.

D. NuRV ORBit2 monitor

Alternatively, NuRV can also be deployed as a standalone monitoring server service accessible to the DT. Consequently, the NuRV monitor and DT operate independently with their lifecycles entirely decoupled. This section delineates the steps to achieve this with the incubator.

1) NuRV monitor server: NuRV supports a network-based *monitoring server* mode: from the interactive shell mode, NuRV can enter with a command into a network listening state. This enables user code to remotely execute the heartbeat command for online monitoring. In server mode, NuRV can accommodate multiple clients connecting to multiple servers. In this context, each *monitor server* refers to a running NuRV process where numerous LTL properties are incorporated alongside their respective runtime monitors, established through the `build_monitor` command. It should be emphasized that a single NuRV process has the capacity to administer multiple monitors, each tailored to different LTL properties.

2) Monitor integration: The process of connecting the monitor server to the DT of the incubator is automated by the `execute` script. This script, in turn, employs a Python script file, that initially launches the `omniNames` CORBA Name Service utility from the *omniORB* toolset, followed by starting the `NuRV_orbit` version of NuRV. Subsequently, a connection is established with the monitor server using the *omniORB* Python library. Once this connection is established, the Python script starts the incubator DT and subscribes to relevant RabbitMQ topics such as energy saver status and lid open status. The lid open status is mapped to the anomaly for the NuRV monitor.
Figure 5 shows the architecture of the system comprising the incubator DT and the NuRV monitoring server. Upon receiving

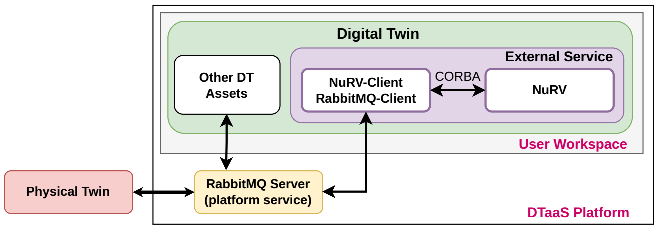


Fig. 5: Overview of components involved with the NuRV ORBit2 monitor.

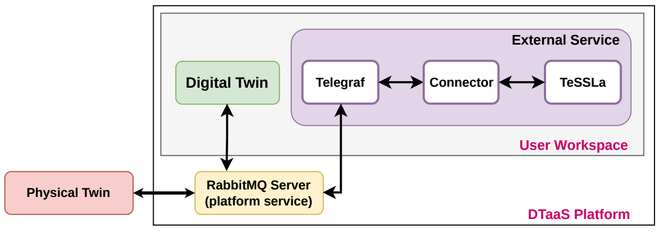


Fig. 6: Overview of components involved with the TeSSLA passive and active monitors.

a message, the DT's status is relayed to NuRV via a heartbeat operation call through the CORBA interface. NuRV responds to this heartbeat by providing the monitor's output. If the monitor's output represents a final verdict, the monitor is reset to prepare for its utilisation for the subsequent execution of the DT.

E. TeSSLa passive monitor

As an alternative to NuRV, the RV tool TeSSLa can be utilized for monitoring the AS properties. Similar to the example presented in [21], the monitor consists of three parts (see Fig. 6). At its core, the TeSSLa monitor processes input streams and produces output streams, but is not itself capable of integrating them into a larger system context. A helper function (Connector) is compiled to handle the streams by connecting TeSSLa streams to sockets with which external tools can interact. Telegraf provides an additional layer of flexibility by adding:

- reconfigurability at runtime – the service can be configured to automatically adapt to a changing configuration file using the `--watch-config` flag, or simply restarted without losing the internal state of the monitor,
- data aggregation with basic statistical operations (such as count, mean, min or histograms),
- stream processing for filtering or transforming data streams, and
- by providing more than 200 integrations with different services and protocols to send or receive streams[10].

The `create` script prepares the system by ensuring that the necessary tools and software are installed and configured. It installs Java, Rust and Telegraf on the system and downloads the necessary files for the TeSSLa-Telegraf Connector[11]. Two files, a TeSSLa specification and a Telegraf configuration, must be provided by the user.

A TeSSLa specification suitable for this scenario (shown in Fig. 7) monitors two key states of the Incubator: whether the lid is open and whether the energy saving mode is used, which are passed to the TeSSLa monitor via different event streams. The helper function `raisingDelay` delays any change from `false` to `true` by three time steps without affecting changes from `true` to `false`. This function

[10]https://docs.influxdata.com/telegraf/v1/plugins/
[11]https://git.tessla.io/telegraf/tessla-telegraf-connector/-/blob/master/Release/tessla-telegraf-connector.zip

is used to define an internal data stream `critical` that represents when the energy saving mode is expected to be active. If it is not, an `alert` stream is set to `true`. This stream is sent back to the system. The `@TelegrafIn` and `@TelegrafOut` annotations allow the compiler to automatically create the Connector function and add to the Telegraf configuration. The Telegraf configuration consists of

```
include "./Telegraf.tessla"

@TelegrafIn("amqp_consumer","host=<hostname>",
 ↪  "lid_open")
in lid_open: Events[Bool]

@TelegrafIn("amqp_consumer","host=<hostname>",
 ↪  "energy_saver_on")
in energy_saver: Events[Bool]

def delayedOpen = raisingDelay(lid_open, 3)
def critical = lid_open && delayedOpen
def alert = critical && !energy_saver

@TelegrafOut("alert")
out alert

def raisingDelay(e: Events[Bool], d: Int):
Events[Bool] = merge3(false, const(true,
 ↪  delay(const(d, boolFilter(e)), e)),
 ↪  const(false, falling(e)))
```

Fig. 7: a TeSSLa specification for the passive monitor

two parts – where to connect to external data sources and sinks (RabbitMQ in this case), and how to connect to the TeSSLa monitor. The first has to be specified manually, as it depends on the specific case. Here it configures the AMQP plugin to connect to the RabbitMQ server, subscribe to the topics `incubator.diagnosis.plant.lidopen` as well as `incubator.energysaver.status` and publish the monitor verdict to the topic `incubator.energysaver.alert`.

The `execute` script uses the following command to add the configuration of how to communicate with the TeSSLa monitor to the supplied Telegraf configuration, create and run the Connector helper function, and compile as well as run the monitor.

```
./TesslaTelegrafConnector -i
 ↪  ./incubator.tessla -c ./telegraf.conf -r
```

Fig. 8: Command used within execute lifecycle script.

The script then starts the Telegraf service with `systemctl start telegraf`. This procedure allows data flow to and from the RabbitMQ broker, facilitating the collection, processing and monitoring of sensor data.

The `terminate` script stops the Telegraf service as well as the TeSSLa monitor and removes all temporary files.

F. TeSSLa active monitor

To use TeSSLa as a monitor for runtime enforcement, the TeSSLa specification (Fig. 7) and the Telegraf configuration must be changed.

To adapt the TeSSLa specification to control the energy saver mode instead of monitoring, the energy saver status is no longer needed as an input and the delayed signal can be provided as an output stream to switch on the energy saver if the lid is still open. Because only the rising edge is delayed, energy saving mode is switched off as soon as the lid closes.

The notable change in the Telegraf configuration is the line shown in Fig. 9 in the AMQP output plugin, which translates the boolean value for controlling the energy saving mode into the JSON format required by the Incubator. By adding this post-processing step to Telegraf[12], the user is able to change the formatting or desired temperature setting in the running system by changing the configuration without recompiling the monitor or losing its internal state.

```
transform = '{"temperature_desired":
 ↪  fields.value ? 21 : 35}'
```

Fig. 9: Telegraf JSON transformation

IV. Concluding remarks

This tutorial paper delineates the process of integrating RV within existing AS, by demonstrating different integration patterns through five use cases. Although the tool integrations has been demonstrated using NuRV and TeSSLa within a DT context utilizing DTaaS, the underlying concepts extend beyond these implementation details. Consequently, the learning outcomes can be generalized, enabling tutorial participants to apply RV tools in their research to provide stronger guarantees of the correctness of their AS. In the physical tutorial conducted at ACSOS 2024, the examples will be demonstrated sequentially, with a discussion of the deployment advantages and disadvantages of each approach. Participants will also have the opportunity to run the examples directly in the DTaaS platform.

References

[1] Albassam, E., Porter, J., Gomaa, H., Menasc, D.A.: DARE : A Distributed Adaptation and Failure Recovery Framework for Software Systems. In: IEEE International Conference on Autonomic Computing (2017), https://doi.org/10.1109/ICAC.2017.12

[2] Aziz, A., Chouhan, S.S., Schelén, O., Bodin, U.: Distributed Digital Twins as Proxies-Unlocking Composability and Flexibility for Purpose-Oriented Digital Twins. IEEE Access **11**, 137577–137593 (2023), https://doi.org/10.1109/ACCESS.2023.3340132

[3] Blockwitz, T., Otter, M., Åkesson, J., Arnold, M., Clauss, C., Elmqvist, H., Friedrich, M., Junghanns, A., Mauss, J., Neumerkel, D., Olsson, H., Viel, A.: Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models. In: Proc. 9th International Modelica Conference. Linköping University Electronic Press (2012)

[12]Telegraf first introduced the JSON transformation feature in version 1.24, which has not yet been widely distributed to package repositories.

[4] Cheng, B.H.C., Lansing, E., Clark, R.J., Lansing, E., Langford, M.A., Mckinley, P.K.: AC-ROS : Assurance Case Driven Adaptation for the Robot Operating System. In: 23rf ACM/IEEE International Conference on Model Driven Engineering Languages and Systems. pp. 102–113. No. 1, ACM (2020), https://doi.org/10.1145/3365438.3410952

[5] Cimatti, A., Tian, C., Tonetta, S.: NuRV: A nuXmv Extension for Runtime Verification. In: Finkbeiner, B., Mariani, L. (eds.) Runtime Verification - 19th International Conference, RV 2019, Porto, Portugal, October 8-11, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11757, pp. 382–392. Springer (2019), https://doi.org/10.1007/978-3-030-32079-9_23

[6] Convent, L., Hungerecker, S., Leucker, M., Scheffel, T., Schmitz, M., Thoma, D.: TeSSLa: temporal stream-based specification language. In: Formal Methods: Foundations and Applications: 21st Brazilian Symposium, SBMF 2018, Salvador, Brazil, November 26–30, 2018, Proceedings 21. pp. 144–162. Springer (2018)

[7] Donzé, A.: On signal temporal logic. In: Runtime Verification: 4th International Conference, RV 2013, Rennes, France, September 24-27, 2013. Proceedings 4. pp. 382–383. Springer (2013)

[8] Falcone, Y.: You should better enforce than verify. In: International Conference on Runtime Verification. pp. 89–105. Springer (2010)

[9] Feng, H., Gomes, C., Gil, S., Mikkelsen, P.H., Tola, D., Larsen, P.G., Sandberg, M.: Integration Of The Mape-K Loop In Digital Twins. In: 2022 Annual Modeling and Simulation Conference (ANNSIM). IEEE (Jul 2022), https://doi.org/10.23919/annsim55834.2022.9859489

[10] Feng, H., Gomes, C., Thule, C., Lausdahl, K., Iosifidis, A., Larsen, P.G.: Introduction to Digital Twin Engineering. In: 2021 Annual Modeling and Simulation Conference (ANNSIM). IEEE (Jul 2021), https://doi.org/10.23919/annsim52504.2021.9552135

[11] Feng, H., Gomes, C., Thule, C., Lausdahl, K., Sandberg, M., Larsen, P.G.: The Incubator Case Study for Digital Twin Engineering. arXiv:2102.10390 (2021)

[12] Gomes, C., Broman, D., Vangheluwe, H., Thule, C., Larsen, P.G.: Co-Simulation: A Survey. ACM Computing Surveys **51**(3) (2018)

[13] Jahan, S., Riley, I., Walter, C., Gamble, R.F., Pasco, M., McKinley, P.K., Cheng, B.H.C.: MAPE-K/MAPE-SAC: An interaction framework for adaptive systems with security assurance cases. Future Generation Computer Systems **109**, 197–209 (2020), https://doi.org/10.1016/j.future.2020.03.031

[14] Kephart, J., Chess, D.: The vision of autonomic computing. Computer **36**(1), 41–50 (2003), https://doi.org/10.1109/MC.2003.1160055

[15] Kübler, R., Schiehlen, W.: Two Methods of Simulator Coupling. Mathematical and Computer Modelling of Dynamical Systems **6**(2) (2000)

[16] Malburg, L., Hoffmann, M., Bergmann, R.: Applying MAPE-K control loops for adaptive workflow management in smart factories. Journal of Intelligent Information Systems **61**(1), 83–111 (2023), https://doi.org/10.1007/s10844-022-00766-w

[17] Papamartzivanos, D., Gómez Mármol, F., Kambourakis, G.: Introducing deep learning self-adaptive misuse network intrusion detection systems. IEEE Access **7**, 13546–13560 (2019), https://doi.org/10.1109/ACCESS.2019.2893871

[18] Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science. pp. 46–57. IEEE (1977)

[19] Qi, Q., Tao, F., Hu, T., Anwer, N., Liu, A., Wei, Y., Wang, L., Nee, A.: Enabling technologies and tools for digital twin. Journal of Manufacturing Systems **58**, 3–21 (2021)

[20] Robles, J., Martín, C., Díaz, M.: OpenTwins: An open-source framework for the development of next-gen compositional digital twins. Computers in Industry **152**, 104007 (2023)

[21] Streichhahn Hendrick and Duodaki, Abdul Rahman and Hyttrek, Christian and Wolf, Jakob and Kreth, Yannick: TeSSLa Telegraf Connector. https://tessla.io/blog/telegrafConnector/ (2024)

[22] Talasila, P., Craciunean, D.C., Bogdan-Constantin, P., Larsen, P.G., Zamfirescu, C., Scovill, A.: Comparison between the HUBCAP and DIGITBrain Platforms for Model-Based Design and Evaluation of Digital Twins. In: Proceedings of the 5th Workshop on Formal Co-Simulation of Cyber-Physical Systems. CoSim CPS (2021)

[23] Zambrano, V., Mueller-Roemer, J., Sandberg, M., Talasila, P., Zanin, D., Larsen, P.G., Loeschner, E., Thronicke, W., Pietraroia, D., Landolfi, G., Fontana, A., Laspalas, M., Antony, J., Poser, V., Kiss, T., Bergweiler, S., Serna, S.P., Izquierdo, S., Viejo, I., Juan, A., Serrano, F., Stork, A.: Industrial Digitalization in the Industry 4.0 Era: Classification, Reuse and Authoring of Digital Models on Digital Twin platforms. Array p. 100176 (2022). https://doi.org/https://doi.org/10.1016/j.array.2022.100176, https://www.sciencedirect.com/science/article/pii/S2590005622000352

A.3 DynSRV: Dynamically Updated Properties for Stream Runtime Verification

The appended paper follows.

Funded by
the European Union

DynSRV: Dynamically Updated Properties for Stream Runtime Verification*

Morten Haahr Kristensen[1] [0009−0008−8467−7567], Thomas Wright[1] [0000−0001−8035−0884], Cláudio Gomes[1] [0000−0003−2692−9742], Lukas Esterle[1] [0000−0002−0248−1552], and Peter Gorm Larsen[1] [0000−0002−4589−1500]

Department of Electrical and Computer Engineering, Aarhus University, Denmark
`{mhk,thomas.wright,claudio.gomes,lukas.esterle,pgl}@ece.au.dk`

Abstract. Systems that adapt to their environment or change based on new requirements pose challenges for runtime verification. Complexity is increased when the system needs to retain its internal state and continue monitoring while also updating properties or adding new ones during runtime. In this work, we propose DynSRV, a Stream Runtime Verification language that allows for dynamic updates of properties. A core benefit of this language is its capability to update properties at runtime without requiring a restart of the monitor, maintaining the internal state of the remaining properties. We formalise the semantics of our core primitives and demonstrate design patterns for allowing adaptations under certain constraints. Finally, we present an implementation of DynSRV and describe three memory strategies that balance memory usage and the ability to resolve dynamically added properties depending on historical data.

Keywords: Runtime Verification, Dynamic Properties, Stream Runtime Verification, Autonomous Systems, Dynamic Software Updating, Self-Adaptive Systems

1 Introduction

Motivation. How do we ensure continuous and accurate runtime monitoring when the system evolves during execution? If the system evolves in simple ways that can be captured in static Runtime Verification (RV) specifications then system evolution is not an issue. However, if significant behavioural changes are introduced by a human through Dynamic Software Updating (DSU) (see,

* The work presented here is supported by the RoboSAPIENS project funded by the European Commission's Horizon Europe programme under grant agreement number 101133807. In addition, the authors would like to thank Amalie Kaastrup-Hansen, Tobias Frejo Rasmussen, and Mikkel Kirkegaard for the fruitful discussions leading up to the paper.

e.g. [13]) or autonomously, then the RV specification must also be updated to ensure that the system is still being monitored correctly. Moreover, changes often entail that the system requirements have evolved [13], and if so, then the RV specification must also evolve to reflect these new requirements. An example of this is shown in [18], where a self-adaptive cloud-edge-end power distribution system requires the deployment of a state-machine based monitor that changes to reflect requirement changes in real-time, as the system reacts to evolving load demands, sensors failure, or maintenance events. Naturally, one possibility is restarting the monitor with an updated specification, but this is not always possible, as it involves loss of internal state potentially leading to incorrect verdicts [20].

Contribution 1. We propose and formally define DynSRV, a Stream Runtime Verification (SRV) language that allows monitors to be updated at runtime without requiring restarts or manual rewriting. Specifically, we introduce two primitives to DynSRV which enable expressing Dynamically Updated Properties (DUPs).

- `defer`(p) allows a RV property p to be specified at a later point in time, enabling exactly one dynamic update.
- `dynamic`(p) extends the concept of `defer`(p) by permitting continual updates, allowing the dynamic property to be modified multiple times throughout execution.

DUPs extend the concept of DSU to SRV by allowing specifications to evolve alongside the system without restarting. Unlike traditional DSU, which modifies the functional aspects of a running system, DUPs focus on changing the RV properties that the system is expected to satisfy.

Contribution 2. While DynSRV enables flexible adaptation, allowing arbitrary dynamic expressions within a monitor introduces challenges in reasoning about specification correctness. Thus, we propose *design patterns* for the specification of DUPs, enabling controlled adaptations, refinements, and demonstrating common adaptation patterns within DSU.

Contribution 3. We highlight the unique challenge presented by allowing adaptive SRV with DynSRV, such as ensuring consistency in monitoring results despite evolving specifications, managing historical data for updated properties, and developing performant interpreters that allow evaluating unforeseen properties.

2 Background & Related Work

We begin by linking the fields of DSU and Self-Adaptive Systems (SASs), which provided the motivation for this work, and we discuss how they relate to DUPs. We then recap the basic concepts of SRV before introducing existing works that express special cases of DUPs in the context of RV.

2.1 Dynamic Software Updating and Self-Adaptive Systems

DSU enables modifying running systems without stopping them, which is critical for applications like financial systems or web servers where downtime is costly. Key challenges include maintaining safety, supporting flexible updates, minimising overhead, and easing the developer's burden. Hicks, Moore, and Nettles [13] address these with a DSU system for C-like languages using type-safe dynamic patches and tools to aid patch creation and application.

SASs autonomously manage and adjust themselves to meet high-level goals [15]. Inspired by biological autonomic systems, they reduce manual intervention through capabilities like self-configuration, optimisation, healing, and modular architectural updates [25]. SASs use feedback loops and distributed components to monitor, analyze, plan, and execute changes in dynamic environments. Key challenges include goal specification, ensuring safety, and handling emergent behaviour.

DSU and SASs are interrelated: DSU enables runtime adaptation for SASs, while SASs frameworks can manage DSU to maintain stability during updates. Virtual machine-based DSU approaches, such as those presented in [23, 24, 14], inspired our monitor architecture, offering runtime control and transformation of system structures.

At design time, formal verification ensures DSU maintains safety and liveness properties. A foundational model by Bierman *et al.* [2] uses a λ-calculus with an update primitive to enable formal reasoning about dynamic updates. Despite extensive research (see surveys [21, 19, 26]), runtime updating of verification properties remains an underexplored area. The following sections address existing work on this topic.

2.2 Dynamically Updated Properties

SRV is a lightweight RV approach that monitors systems producing continuous data streams. It processes input streams, i.e., sequences of event values, into verdict streams following a given specification.

LOLA [8] is a SRV language, inspired by LUSTRE and ESTEREL, supporting basic operations, conditionals, and time-offsets to enable temporal monitoring. LOLA uses a dependency graph to determine if a specification is "Efficiently Monitorable", ensuring bounded memory usage. LOLA pioneered SRV and has influenced many subsequent languages, including TeSSLa [17].

Lola 2.0 improves dynamic RV through dynamic parametrization, enabling quantification over objects and monitor spawning independent of observed instances, and retroactive parametrization, allowing monitors to revisit past events during execution [20]. While the new monitors support parameterizing existing expressions, DynSRV allows dynamically providing any syntactically valid expression that references valid input streams.

Barringer et al. [1] propose Eagle, a general logic framework supporting recursive monitoring rules with fixpoint semantics. Eagle supports dynamic monitor generation and logics like Linear Temporal Logic (LTL), Metric Temporal Logic (MTL), and Statistical Contracts.

First order logic quantification in dynamically created objects in RV was explored by Havelund and Peled [12] and Sokolsky *et al.* [22] with LC_v. LC_v uses first-order and attribute quantifiers to track dynamic entities (e.g., tasks, sensors). This relates to Allocational Temporal Logic (ATL) using history-dependent automata [9].

Actor-based runtime verification [7, 6, 4, 5] has previously been applied to self-adaptive systems, using independent monitor actors that observe and react to behaviour asynchronously.

The most relevant related work in terms of goals (but not methods) is by Carwehl *et al.* [3], who propose dynamically adapting monitors to changing requirements without restarting the monitor. Their monitors are synthesized as automata with error states based on structured English specifications translated into MTL, whereas we use stream-based properties. During execution, a Runtime Verifier checks for violations, and when requirements change, a Requirements Manager applies predefined Property Adaptation Patterns (e.g., updating a time guard or updating events). In contrast, we support arbitrary property expressions as long as they are syntactically valid and use existing input streams. While they argue that adhering to fixed patterns leads to safer adaptations and view fully dynamic RV as undesirable, we take a different stance, and demonstrate through Contribution 2 that we can address these valid concerns while prioritising expressiveness.

3 Specification Language

3.1 Motivational example

To provide a motivational example (Fig. 1), we consider future production lines where different products are manufactured by autonomously moving robots. The robots move around in the production hall and utilise the different tools available in order to produce the desired items. The robot has an understanding of the production process and which tools to utilise for each product. However, while the robot and the production line are developed in parallel, the robot will only get knowledge of the final layout and the respective locations of the different tools upon completion of the production hall. Upon deployment, the robot will be given a product to manufacture, and it will start to move around the production hall. When the product is completed, the robot will receive a new product – potentially with a different requirement for the tools to be used. The robot will then have to adapt its plans, movement and overall behaviour to the new product. Finally, the robot is battery-operated and will need to recharge at certain intervals as well as undergo regular maintenance.

In this scenario, the robot uses the stream $l = \texttt{defer}(l_{in})$ for the layout of the production lines, respective locations of the different tools, and restricted areas, using `defer` since this configuration becomes immutable once it is made. At deployment time, $p = \texttt{dynamic}(p_{in})$ is used to change the rules determining which products the robot is allowed to manufacture as the production line evolves.

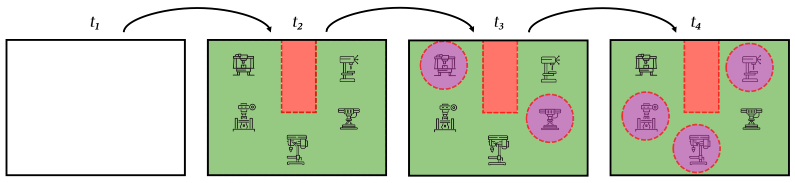


Fig. 1: Example of a production line at different time steps. First is the empty production hall, then the layout with the machines is added – a specific area is for robot maintenance (red square). Last two show which tools are allowed to be used during phases t_3 and t_4 (highlighted in purple).

Here, the new rules can use the information gathered from the layout stream. Finally, the verdict is available with $v = \texttt{update}(p_{init}, p)$ where an initial value of production rules is provided with `update`, which is detailed in Section 3.3.

This example highlights the need for DUPs in scenarios where the monitored system is subject to dynamic changes, and the monitoring properties must adapt accordingly. In addition, the specification can include static properties. For instance, the robot is also subject to regular maintenance and recharging within *at least* certain intervals (e.g., at least every 12 hours), requiring stateful properties to be monitored. As the stateful maintenance information would be lost if the monitor is restarted, a simple restart for each product update is not feasible. While this is only a simple example, the reader can imagine more complex scenarios with multiple robots and multiple products operating in parallel and potentially creating conflicts around resources and tools during execution.

3.2 Syntax

DynSRV defines monitors that transform a set of input streams $I = \{p_1, \ldots, p_n\}$ into a set of output streams $O = \{o_1, \ldots, o_m\}$. Each stream $s = (s_1, s_2, \ldots)$ is a sequence of typed values s_i in some domain $\mathbb{D}$. These domains $\mathbb{D}$ include booleans $\mathbb{B}$, integers $\mathbb{Z}$, floating point numbers $\mathbb{F}$, and, recursively, stream expressions in the DynSRV *expression domain* $\mathbb{E}[\mathbb{D}]$ which we will shortly define with values in some domain $\mathbb{D}$. Streams may also take on a special value $\perp$ (pronounced *deferred*), denoting that no value was sent at the current time step.

A *specification* Φ over input streams and output streams $S = I \uplus O$ (where $\uplus$ denotes the *disjoint union*) consists of a set of equations

$$o_1 = \phi_{o_1} \quad \ldots \quad o_n = \phi_{o_n}$$

where the expressions ϕ_o are defined as stream expressions with output domain $\mathbb{D}$. Stream expressions $\phi \in \mathbb{E}[\mathbb{D}]$ are defined recursively to be a *basic expression*, a *DUP*, or a *DUP helper*. We elaborate on the precise semantics for DUPs in Section 3.3.

A *basic expression* is one of the following:

- a constant $\phi \triangleq c$ for $c \in \mathbb{D}$
- a stream variable $\phi \triangleq v$ for any $v \in S$
- a function application $\phi \triangleq f(\psi_1, \ldots, \psi_n)$
  which lifts an arbitrary data-domain function $f : \mathbb{D}_1 \times \ldots \times \mathbb{D}_n \to \mathbb{D}$
- a temporal index $\phi \triangleq \psi[-j]$ for $\psi \in \mathbb{E}[\mathbb{D}], j \in \mathbb{N}$
  referring to the value of ψ at j time units in the past
- a conditional $\phi \triangleq \texttt{if}\ \sigma\ \texttt{then}\ \psi_1\ \texttt{else}\ \psi_2$ for $\sigma \in \mathbb{E}[\mathbb{B}]$
  $\psi_1, \psi_2 \in \mathbb{E}[\mathbb{D}]$

A *DUP* is one of the following:

- a defer $\phi \triangleq \texttt{defer}(\psi)$ for $\psi \in \mathbb{E}[\mathbb{D}]$
  referring to a dynamic property which is $\perp$ until the first point at which ψ becomes available and behaves like ψ subsequently
- a dynamic $\phi \triangleq \texttt{dynamic}(\psi)$ for $\psi \in \mathbb{E}[\mathbb{D}]$
  referring to a dynamic property which behaves like the most recent value of ψ or is $\perp$ if none has been sent

A *DUP helper* is one of the following:

- a default $\phi \triangleq \texttt{default}(\psi, c)$ for $\psi \in \mathbb{E}[\mathbb{D}], c \in \mathbb{D}$
  which uses the default value c if ψ is $\perp$
- a when $\phi \triangleq \texttt{when}(\psi)$ for $\psi \in \mathbb{E}[\mathbb{D}]$
  which is false until the first time ψ is not $\perp$ and true thereafter
- an update $\phi \triangleq \texttt{update}(\psi_1, \psi_2)$ for $\psi_1, \psi_2 \in \mathbb{E}[\mathbb{D}]$
  which is ψ_1 until the first time ψ_2 is not $\perp$ and ψ_2 thereafter

Standard data-domain operators such as addition, multiplication, logical conjunction, disjunction, and comparison operators are supported as functions f lifted to stream expressions. These operators propagate $\perp$ values such that e.g. $42 + \perp = \perp$.

Furthermore, we define a specification to be *well-defined*, if it has no zero-time cycle of dependencies (similarly to [8]), that is, if any dependency cycle is guarded by a time index. This restriction is necessary for specifications to be monitorable.

3.3 Semantics of DUPs

In this section we define a mathematical semantics for DynSRV specifications Φ. We note that this follows a similar approach to the semantics of TeSSLa [17] and LOLA [8], whilst introducing novel definitions to handle DUPs.

First, we need to formalise the notion of streams, used for specification input and output. We need these to handle both deferred data $\perp$ (for dynamic properties) as well as *partiality*, which uses the special value ? to represent stream values which have not yet been computed.

Definition 1 (Stream). *A* partial stream *(or simply,* stream*) is a function $s : \mathbb{T} \to \mathbb{D} \cup \{?, \perp\}$ such that $s(i) = ?$ implies that for all $j > i$ we must have $s(j) = ?$. We denote the set of streams by* $\textsc{Stream} = [\mathbb{V} \to \mathbb{D} \cup \{?, \perp\}]$.

Additionally, we call a partial stream *total* if $\forall i \in \mathbb{T} : s(i) \neq ?$.

We define the input namespace in($\varPhi$) consisting of the set of input variables, the output namespace out($\varPhi$) consists of the set of output variables, and we define $\mathrm{vars}(\varPhi) = \mathrm{in}(\varPhi) \cup \mathrm{out}(\varPhi)$. We also define vars($\phi$), for any stream expression ϕ to be the set of all stream variables appearing in ϕ, and define $\mathbb{V}$ to be the set of all variable names[1]. This allows us to introduce *contexts*, representing an assignment of partial streams to some stream variables v in the set of all stream variables $\mathbb{V}$.

Definition 2 (Context). *A context is a partial function $C : \mathbb{V} \rightharpoonup$ Stream. We denote the set of all such partial functions as*

$$\textsc{Context} \triangleq [\mathbb{V} \rightharpoonup \mathbb{T} \to \mathbb{D} \cup \{?, \bot\}] = [\mathbb{V} \rightharpoonup \textsc{Stream}].$$

That is, within a given context, for a stream variable $v \in \mathbb{V}$ in the domain of stream variables for which it is defined, we have a stream for this stream variable $C(v) : \mathbb{T} \to \mathbb{D} \cup \{?, \bot\}$. In particular, the inputs to a specification $\varPhi$ can be provided via an *input context* C_{in} such that $\mathrm{dom}(C_{\mathrm{in}}) = \mathrm{in}(\varPhi)$.

We also define the *refinement* partial order on data values by setting $u \sqsubseteq v$ iff $v = ?$ implies $u = ?$. This extends elementwise to a partial order $\sqsubseteq$ on streams, and on contexts sharing the same domain.

Using this, we define the semantics of a specification $\varPhi$ as the least fixed-point of a *single-step semantics*, which expands one recursive step of the stream equations, using refinement to gradually build streams covering the whole time domain.

Definition 3. *We define the single-step semantics for a specification $\varPhi$ to be the function $[\![\varPhi]\!]_1 : \textsc{Context} \to \textsc{Context} \to \textsc{Context}$ defined such that*

$$[\![\varPhi]\!]_1(C)(D)(v) = [\![\phi_v]\!]_1(C)(D \uplus C)$$

for each $v \in \mathrm{vars}(\phi)$, whilst the denotation function for a well-defined specification $\varPhi$ given initial context $C = C_{\mathrm{in}}$ to be the function $[\![\varPhi]\!] : \textsc{Context} \to \textsc{Context}$ defined as the least-fixed point:

$$[\![\varPhi]\!]\,(C) = \mu D.\, [\![\varPhi]\!]_1(C)(D).$$

under the refinement order $\sqsubseteq$.

We also define the shorthand $[\![\psi]\!](C) \triangleq [\![\Psi]\!](C)$ for the semantics of ϕ within the specification $\Psi \triangleq v = \psi$ where v is any fresh variable name.

The fixed-point in the above definition exists and is unique for well-defined specifications $\varPhi$ by Kleene's fixed-point theorem since the definitions of the single-step semantics for individual operators – which we will give shortly – are monotone in the refinement order $\sqsubseteq$, and hence so is $[\![\varPhi]\!]_1(C)$.

[1] To be concrete, we can set $\mathbb{V} = \mathbb{N}$ for countably many numerically-indexed variables.

This depends on the single-step semantics for individual operators, which we define as follows for basic operators,

$$
\begin{aligned}
[\![c]\!]_1(C)(D)(i) &\triangleq c \\
[\![v]\!]_1(C)(D)(i) &\triangleq D(v)(i) \\
[\![f(\psi_1,\dots,\psi_k)]\!]_1(C)(D)(i) &\triangleq f([\![\psi_1]\!]_1(C)(D)(i),\dots,[\![\psi_k]\!]_1(C)(D)(i)) \\
[\![\texttt{if } \sigma \texttt{ then } \psi_1 \texttt{ else } \psi_2]\!]_1(C)(D)(i) &\triangleq \begin{cases} [\![\psi_1]\!]_1(C)(D)(i) & \text{if } [\![\sigma]\!]_1(C)(D)(i) = \text{true} \\ [\![\psi_2]\!]_1(C)(D)(i) & \text{if } [\![\sigma]\!]_1(C)(D)(i) = \text{false} \\ \bot & \text{if } [\![\sigma]\!]_1(C)(D)(i) = \bot \\ ? & \text{if } [\![\sigma]\!]_1(C)(D)(i) = \ ? \end{cases} \\
[\![\psi[-j]]\!]_1(C)(D)(i) &\triangleq \begin{cases} [\![\psi]\!]_1(C)(D)(i-j) & \text{if } i \geq j \\ \bot & \text{otherwise} \end{cases}
\end{aligned}
$$

For the other functions, we first define duration restricted subsets of a context, which can be used to evaluate properties using only data available at a given point in time.

Definition 4. *Given a context C we define the duration-d prefix of C as the context $C|_d$ defined by*

$$
C|_d(v)(i) \triangleq \begin{cases} C(v)(i) & \textit{if } i \leq d \\ ? & \textit{otherwise} \end{cases}
$$

which we use to define the following two helper functions,

Definition 5. *For maximum duration i, expression ψ, and context C, we define the functions* $\text{first}, \text{last} : \mathbb{N} \times \mathbb{E}[\mathbb{D}] \times \textsc{Context} \to \mathbb{N} \cup \{\infty\}$ *defined by*

$$
\begin{aligned}
\text{first}(i,\psi,C) &\triangleq \min\{\, j \in \mathbb{N} \mid [\![\psi]\!](C|_j)(j) \notin \{\bot, ?\} \wedge j \leq i \,\} \\
\text{last}(i,\psi,C) &\triangleq \max\{\, j \in \mathbb{N} \mid [\![\psi]\!](C|_j)(j) \notin \{\bot, ?\} \wedge j \leq i \,\}
\end{aligned}
$$

where each of these functions is set to ∞ if $[\![\psi]\!](C|_j)(j) \in \{\bot, ?\}$ for all j.

Then we define the single-step semantics of dynamic properties by,

$$
\begin{aligned}
[\![\texttt{defer}(\psi)]\!]_1(C)(D)(i) &\triangleq \begin{cases} [\![\psi_j]\!]_1(C)(D)(i) & \text{if } i \geq j \\ \bot & \text{if } j = \infty \end{cases} \\
[\![\texttt{dynamic}(\psi)]\!]_1(C)(D)(i) &\triangleq \begin{cases} [\![\psi_k]\!]_1(C)(D)(i) & \text{if } i \geq k \\ \bot & \text{if } k = \infty \end{cases}
\end{aligned}
$$

where $j = \text{first}(i,\psi,C)$, $k = \text{last}(i,\psi,C)$, $\psi_j \triangleq [\![\psi]\!]_1(C|_j)(D)(j)$, and $\psi_k \triangleq [\![\psi]\!]_1(C|_k)(D)(k)$.

Finally, we define the semantics of each of the DUP helper functions by

$$\llbracket \texttt{update}(\psi_1, \psi_2) \rrbracket_1(C)(D)(i) \triangleq \begin{cases} \llbracket \psi_1 \rrbracket_1(C)(D)(i) & \text{if } \mathrm{first}(i, \psi_2, D) = \infty \\ \llbracket \psi_2 \rrbracket_1(C)(D)(i) & \text{otherwise} \end{cases}$$

$$\llbracket \texttt{when}(\psi) \rrbracket_1(C)(D)(i) \triangleq \begin{cases} \text{false} & \text{if } \mathrm{first}(i, \psi, C) = \infty \\ \text{true} & \text{otherwise} \end{cases}$$

$$\llbracket \texttt{default}(\psi, c) \rrbracket_1(C)(D)(i) \triangleq \begin{cases} \llbracket \psi \rrbracket_1(C)(D)(i) & \text{if } \llbracket \psi \rrbracket_1(C)(D)(i) \neq \bot \\ c & \text{otherwise} \end{cases}$$

4 Design patterns with DUPs

In practical RV scenarios, system requirements change. Supporting such changes with a first-class language construct allows specifications to adapt systematically, and allows expressing which parts are allowed to adapt. With DUPs, not only can the specification itself evolve over time, but it also becomes possible to express meta-properties, i.e., properties about how the specification may change. This section presents design patterns for writing specifications with DUPs in DynSRV.

General design patterns
Open property shows the most permissive use of `dynamic`, where the verdict v, directly reflects the incoming property p. In this case, it is up to the sender to ensure that the provided property is safe and valid.

$$v = \texttt{dynamic}(p)$$

Weaken allows dynamic properties to weaken an existing requirement. In this example, accepting a new goal g normally requires the robot's battery level b to be above 30%. However, in emergencies such as a fire, strictly enforcing this threshold could block critical actions, such as evacuating an area or saving material, thus custom rules p are allowed.

$$v = g \implies b > 30 \vee \texttt{default}(\texttt{dynamic}(p), \text{false})$$

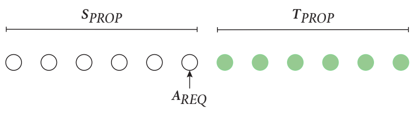
Strengthen allows dynamic properties to strengthen existing requirements. In the example, T is the current time, T_m is the scheduled maintenance time, and p represents dynamic rules that further constrain the maintenance window. For instance, these rules might shorten the service interval if the battery degrades or if the robot moves farther from its charging station.

$$v = T < T_m \wedge \texttt{default}(T < \texttt{dynamic}(p), \text{true})$$

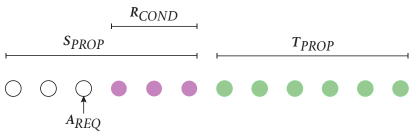
Refinement allows refining an existing property with a new one, where the verdict reflects whether the refinement is valid. In this example, b represents an update of the original condition b_c to a new property p once sent. The refinement expression r evaluates whether the new property remains valid within the context of the original condition. Notably, r is a tautology ($b_c \implies b_c$) when no new property has been provided. The verdict v combines the updated condition b and the refinement r, where true means the requirements are met. If the new property evaluates to $\bot$, the verdict becomes false.

$$b_c = b > 30 \qquad b = \texttt{update}(b_c, \texttt{defer}(p))$$
$$r = b \implies b_c \qquad v = \texttt{default}(b \wedge r, \text{false})$$

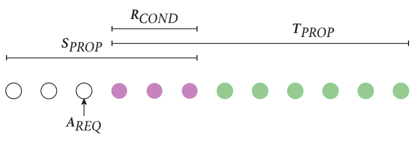
Adaptation patterns
In their works on A-LTL, Zhang and Cheng [27] formalised the semantics of three commonly occurring adaptation semantics: *one-point*, *guided*, and *overlap* adaptation[2]. This is depicted in Fig. 2. To highlight the expressiveness of DynSRV, we demonstrate how the latter two can be expressed with DUPs. One-point adaptation is trivial with DynSRV, as it immediately applies the new property, which is the default behaviour of `defer` and `dynamic`.



(a) One-point adaptation, where T_{PROP} is used immediately upon arrival.



(b) Guided adaptation, where T_{PROP} is used when R_{COND} is satisfied.



(c) Overlap adaptation, where T_{PROP} is used alongside S_{PROP} until the R_{COND} is satisfied, whereafter only T_{PROP} is used.

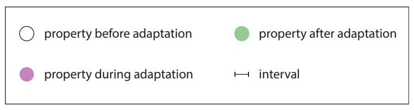
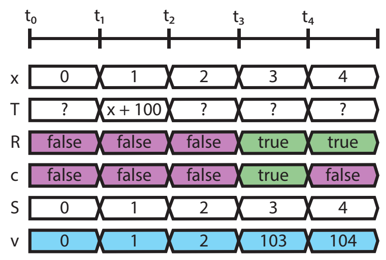

property before adaptation
property after adaptation
property during adaptation
interval

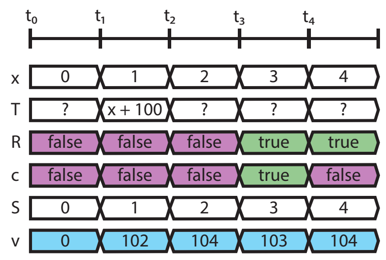
Fig. 2: Adaptation semantics proposed by Zhang and Cheng, figure adapted from [27] with minor modifications for SRV.

Guided adaptation allows a new property to not be used immediately but await for a restriction condition to be satisfied, as depicted in Fig. 2b. This allows the system to delay the application of a new property until appropriate.

[2] In Zhang and Cheng's [27] semantics, adaptation may involve a delay between receiving and applying an adaptation request to ensure the program is in a safe state. This is not required in DynSRV, as it is stateless.



(a) Trace demonstrating guided adaptation. The verdict stream v switches to T after R is satisfied.



(b) Trace demonstrating overlap adaptation. The verdict stream v combines S and T until R is satisfied, after which it uses only T.

Fig. 3: Example traces for guided and overlap adaptation.

The example below demonstrates a specification implementing guided adaptation in DynSRV, with the corresponding trace in Fig. 3a. An integer stream is used for the verdict to more clearly represent its distinct states.

$$S = x \qquad R = \texttt{when}(T) \wedge c \vee \texttt{default}(R[-1], \text{false})$$
$$c = x == 3 \qquad v = \texttt{if}\ R\ \texttt{then}\ \texttt{defer}(T)\ \texttt{else}\ S$$

Here, x and T are input streams, with T representing a property received at time step 1. This property is not applied to the verdict v immediately but is gated by R, which becomes true when c holds after the property is received – at time step 3. The disjunction with $\texttt{default}(R[-1], \text{false})$ ensures that once R holds, it remains true thereafter. The verdict v switches to the deferred property $\texttt{defer}(T)$ only after R holds; otherwise, it yields from S.

Overlap adaptation allows a new property to be used alongside the original property until a condition is satisfied, as depicted in Fig. 2c. Once this condition holds, the new property is used exclusively.

The example below shows a DynSRV specification implementing this behaviour, with its trace depicted in Fig. 3b:

$$S = x \qquad R = \texttt{when}(T) \wedge c \vee \texttt{default}(R[-1], \text{false})$$
$$c = x == 3 \qquad v = \texttt{if}\ \neg\texttt{when}(T)\ \texttt{then}\ S\ \texttt{else if}\ \neg R\ \texttt{then}\ S + T\ \texttt{else}\ T$$

Unlike guided adaptation, overlap adaptation introduces a transition phase where the original stream and the new one are combined $(S + T)$[3] until the condition R becomes true. Initially, the verdict v yields from S. When a new property is received at time step 1 via T, the verdict combines S and T until R is satisfied at time step 3. After that, the verdict uses only T.

[3] Here, addition is used as an example; other operators may apply depending on context.

5 Memory management and History with DUPs

Efficient online monitoring with SRV has long been a focus of the community. Newer languages such as TeSSLa [17] guarantee Bounded Memory (BM) by disallowing future stream indexing – a restriction also adopted by DynSRV. Here, we define BM to mean that at each monitoring step, the monitor's memory usage does not grow with the length of the trace. While memory usage may change dynamically (e.g., when new properties are added), it must remain independent of the trace length. Allowing DUPs in DynSRV introduces a trade-off for ensuring BM, as dynamic expressions may require access to historical data that would otherwise be discarded as a part of the memory management strategy. In static SRV languages, the Dependency Graph (DG) can be used to safely discard unused history once it is no longer needed to resolve equations. In contrast, DUPs can introduce new data dependencies at runtime, potentially referencing historical values that have already been discarded to free memory. This dynamic behaviour makes it impossible to guarantee both BM usage and optimal resolution of expressions introduced by DUPs. Even if a dynamic expression at time t could be resolved to a non-$\perp$ value given the full trace, the monitor may still return $\perp$ if the necessary data has been discarded. This section defines *solvable* stream expressions in relation to memory management and presents three strategies that demonstrate the trade-off.

Solvable stream expressions

For this section, we use the notation that $\phi[-j]$ refer to a stream expression ϕ annotated with an optional temporal index $-j$, where $j = 0$ when no explicit index is given.

Definition 6 (Effective Index). *Given a stream expression $\phi[-j]$ and stream variable $s \in \texttt{vars}(\phi)$, the* effective index $\texttt{index}(\phi, s, j)$ *is defined as:*

$$\texttt{index}(\phi, s, j) = \begin{cases} j + j' & \textit{if } \phi = s[-j'], s \in \texttt{vars}(\phi) \\ j & \textit{if } \phi = \texttt{when}(\psi) \vee \\ & \quad \phi = \texttt{default}(\psi, c) \\ \texttt{index}(\psi, s, j) & \textit{if } \phi = \texttt{defer}(\psi) \vee \\ & \quad \phi = \texttt{dynamic}(\psi) \\ \texttt{index}(\psi, s, j + j') & \textit{if } \phi = \psi[-j'] \\ \max(\{\texttt{index}(\psi, s, j) \mid & \textit{if } \phi = f(\psi_1, \ldots, \psi_n) \\ \quad \psi \in \{\psi_1, \ldots, \psi_n\}\}) & \\ \max(\{\texttt{index}(\sigma, s, j), & \textit{if } \phi = \texttt{if } \sigma \texttt{ then } \psi_1 \\ \quad \texttt{index}(\psi_1, s, j), \texttt{index}(\psi_2, s, j)\} & \quad\quad \texttt{else } \psi_2 \\ \max(\{\texttt{index}(\psi_1, s, j), & \textit{if } \phi = \texttt{update}(\psi_1, \psi_2) \\ \quad \texttt{index}(\psi_2, s, j)\} & \\ j & \textit{otherwise} \end{cases}$$

Effective Index is useful for reasoning about expressions with nested temporal indices. We highlight the case with $\texttt{when}(\psi)$ and $\texttt{default}(\psi, c)$ expressions, which do not introduce new temporal indices and can therefore be used in practice to relax the requirements for *solvable* introduced below.

Definition 7 (Dependency Graph). *Let $s_x \in O, s_y \in S$ be streams and $\phi[-j]$ the expression assigned to s_x. A* Dependency Graph (DG) *is a weighted and directed multigraph $G = (S, E)$, with edges $(s_x, s_y, k, T) \in E$ iff the equations for s_x contains s_y as a subexpression with effective index $k = \texttt{index}(\phi, s_y, j)$, and the edge was introduced at monitor step T.*

The need for T in the DG definition becomes apparent when considering Dynamic Dependency Graphs (DDGs) in strategy 3 below. Until then, it can be assumed that $T = 0$.

Note that if the specification contains DUPs, e.g., if $x = \texttt{dynamic}(p)$ then $(x, p, 0, 0) \in E$, but the properties sent at runtime to p are not. As a result, the DG must be extended to track new dependencies introduced by DUPs, as demonstrated with the strategies below, such that the dynamically received expressions can become *solvable*.

Definition 8 (Solvable). *Let $k = \texttt{index}(\phi, s, j)$ be the effective index of ϕ for stream variable $s \in \texttt{vars}(\phi)$. A stream expression $\phi[-j]$ is said to be* solvable *at monitor step t if there exists an edge $(s', s, j', T) \in E$ such that:*

$$t \geq T + k \wedge j' \geq k,$$

and s evaluated at monitor step $(t - k)$ is not equal to $\perp$.

Intuitively, the first term $t \geq T + k$ ensures that the monitor has progressed sufficiently in steps since the dependency was introduced to solve the expression. The second term $j' \geq k$ ensures that there exists an edge in the DG with a sufficiently large effective index such that the kth last value of s is not discarded.

Theorem 1 claims that a solvable stream expression evaluates to a non-$\perp$ value at monitor step t. The proof follows by structural induction on ϕ and is written out in full in the appendix.

Theorem 1. *Let ϕ be a stream expression that is solvable at monitor step t and may contain DUPs instantiated with solvable stream expressions $(\psi_1, \ldots, \psi_N)$. Then, ϕ evaluated at monitor step t is not equal to $\perp$.*

When writing DynSRV specifications, considering when a stream expression is not solvable is crucial, as an $\perp$ verdict at runtime may not be desirable. We now present three memory management strategies that have different trade-offs between memory efficiency and trace availability (keeping expressions solvable as often as possible).

Strategy 1 – Discard BM: Retain the entire history

Favoring trace availability, this strategy introduces unbounded time dependencies to every other stream in the specification when a DUP is present:

$$E_{\text{DUP}} = \bigcup_{s \in O} \{(s, s', j, 0) \mid \text{hasDUP}(s), s' \in S \setminus \{s\}, j \in \mathbb{N}\}$$
$$G = (S, E \cup E_{\text{DUP}})$$

where hasDUP(s) indicates that stream s's expression contains a DUP.

Using this DG to retain data ensures that any stream expression received dynamically through a DUP is solvable at any monitor step t, if the monitor has progressed sufficiently and none of the referenced stream variables are $\perp$ at the specific monitor steps. That is, the condition $j' \geq k$ from Definition 8 is always met. However, this strategy sacrifices memory efficiency, as any specification involving DUPs will no longer have BM.

Strategy 2 – Preserve BM: Statically specifying dependencies of DUPs
To retain memory efficiency, an alternative approach is to restrict DUPs by requiring users to explicitly annotate their potential temporal dependencies in advance. For example, the expression $\texttt{dynamic}(p, \{(x, -4), (y, -2)\})$ declares that the dynamically introduced property p may depend on stream x up to 4 steps back in time, and on stream y up to 2 steps. A similar change could be made for `defer`. The DG is then defined as:

$$E_{\mathrm{DUP}} = \bigcup_{s \in O} \mathrm{declaredDUP}(s)$$
$$G = (S, E \cup E_{\mathrm{DUP}})$$

where declaredDUP(s) returns the set of edges that are explicitly declared as dependencies of s. This strategy allows each property in the specification to maintain BM, even when using DUPs. However, this comes at the cost of expressiveness as it becomes possible to introduce expressions that never become solvable, specifically, expressions where the condition $j' \geq \texttt{index}(\phi, s, j)$ does not hold, causing them to always evaluate to $\perp$.

Strategy 3 – Preserve BM: Dynamically update dependencies
To balance memory efficiency and trace availability, we propose a DDGs, which extends static DGs by adding dependencies introduced by DUPs at runtime. The DG becomes a time-dependent structure, where the edges are updated based on the declared dependencies of received expressions.

We define the DDG as a stream of DGs that is updated according to the received expressions:

$$E_{\mathrm{DUP}}(t) = \bigcup_{\substack{s \in O \\ \text{where } s \text{ is assigned } e}} \mathrm{dep}(s, t, [\![s]\!]\,(\mathrm{last}(t, e, [\![\Phi]\!]\, C_{in})))$$
$$G(t) = (S, E \cup E_{\mathrm{DUP}}(t))$$

where $[\![s]\!]\,(\mathrm{last}(t, e, [\![\Phi]\!]\, C_{in}))$ denotes the last received expression for stream s at time t in the context $[\![\Phi]\!]\, C_{in}$, and $\mathrm{dep}(s, T, e)$ returns the set of dependencies introduced by the expression e assigned to stream s at monitor step T.

The DDG is used in our DynSRV implementation, detailed in Section 6, to determine how much history to retain at each step. By updating the DDG accordingly, the condition $j' \geq \texttt{index}(\phi, s, j)$ from Definition 8 is guaranteed for new properties as there exists a j' equal to j. However, the monitor step T from

Definition 7 is crucial here: If a new expression ψ referencing stream variable s at effective index k arrives at step T, and at step $T-1$, s had no incoming edges in the DDG, then ψ is not solvable before time $T+k$, and may evaluate to $\perp$ until then. While this approach offers weaker trace availability than the previous two strategies, it preserves bounded memory and supports the full expressiveness of DUPs.

6 Implementation and performance

DynSRV is implemented in Rust as part of the RoboSAPIENS trustworthiness language framework [16], which is a general framework for implementing stream-based languages. The framework is implemented as a modular runtime, with a parser layer that parses specifications into an Abstract Syntax Tree, an extensible execution layer that allows for multiple runtime engine and language semantics to be implemented, and a flexible IO layer allowing input and output streams to be transmitted several sources including files, MQTT, and ROS topics.

As discussed in Section 5, DUPs impose some additional requirements on the implementation of the language compared to existing SRV languages. We meet these requirements via two runtime engine implementations: a constraint-based similar to LOLA [8], and a novel stream-based which translates the specification into a collection of asynchronous Rust actors that communicate over channels. The latter engine features some key design decisions:

- Dependencies between stream variables are handled dynamically, with a publisher/subscriber model used to propagate input values to dependent streams.
- The lifetimes of stream values are handled automatically via Rust's ownership model and the use of channels to communicate between actors. This means that there is no central constraint store or garbage collection step.

This dynamic model does impose some performance challenges since specifications cannot easily be compiled to specifically optimized Rust code for a single specification (as in [11]) and has to keep track of dependencies at runtime. However, as shown in Fig. 4, we are still able to introduce and monitor deferred properties over $100,000$ events in under 500 milliseconds, whilst properties involving no dynamics updates can be monitored with little overhead.

7 Conclusion

In this paper, we have demonstrated how DUPs can be supported with our SRV language DynSRV. Section 2 highlights how DynSRV differs from most related work by enabling truly dynamic expressions to the system's existing properties, whereas prior approaches primarily focus on adapting specific system behaviours.

The semantics presented in Section 3.3 define the denotational meaning of our DUPs primitives, while Section 5 compliments this by describing different memory management strategies based on the trade-off between memory usage and trace availability. Design patterns in Section 4 demonstrate how to weaken,

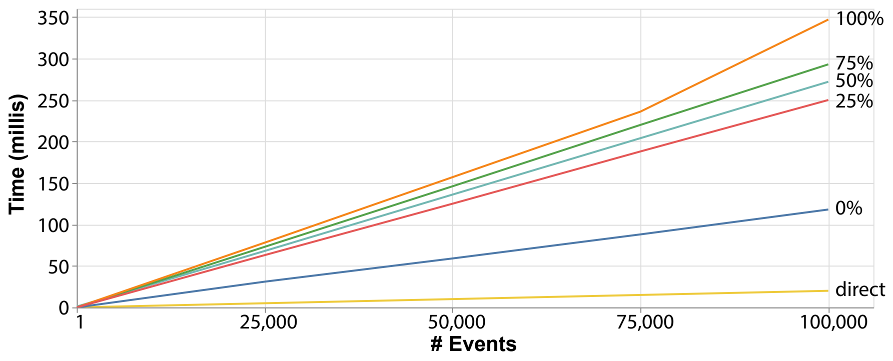


Fig. 4: Time taken to monitor a deferred property $z = \texttt{default}(\texttt{defer}(e), \text{true})$ with the property $e = x \wedge y$ introduced at a certain percentage of the run, compared to direct monitoring of $z = x \wedge y$.

strengthen, and refine properties, as well as describe how to express well-known DSU adaptation patterns.

We consider DynSRV to be especially relevant given the increasing number of systems that require DSU and SASs in our environment and the corresponding need for correctness and safety assurances. While the adaptation for some of these systems is simple enough to be specified statically, others may change in ways where we do not necessarily know what the behaviour in certain situations precisely looks like as shown by Esterle and Brown in [10]. Even if we could specify all possible adaptations in advance, state-space explosion makes it practically infeasible. Our language allows deferring certain parts of the verification to runtime, allowing users to specify new properties as the system evolves, similar to DSU, or allowing a SASs to autonomously specify updated properties as the system evolves.

In the future, we intend to further evaluate the language for industrial use cases, including the case studies provided by the RoboSAPIENS project [16]. We also hope to extend the core language to add full support for asynchronous, timed properties and distributed monitoring of properties.

References

1. Barringer, H., Goldberg, A., Havelund, K., Sen, K.: Rule-Based Runtime Verification. In: Verification, Model Checking, and Abstract Interpretation, pp. 44–57 (2004). `https://doi.org/10.1007/978-3-540-24622-0_5`
2. Bierman, G., Hicks, M., Sewell, P., Stoyle, G.: Formalizing Dynamic Software Updating
3. Carwehl, M., Vogel, T., Rodrigues, G.N., Grunske, L.: Runtime Verification of Self-Adaptive Systems with Changing Requirements. In: IEEE/ACM Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), pp. 104–114 (2023). `https://doi.org/10.1109/SEAMS59076.2023.00024`

4. Cassar, I., Francalanza, A.: On Implementing a Monitor-Oriented Programming Framework for Actor Systems. In: Integrated Formal Methods, pp. 176–192 (2016). https://doi.org/10.1007/978-3-319-33693-0_12
5. Cassar, I., Francalanza, A.: Runtime Adaptation for Actor Systems. In: Runtime Verification, pp. 38–54 (2015). https://doi.org/10.1007/978-3-319-23820-3_3
6. Clark, T., Kulkarni, V., Barat, S., Barn, B.: A Homogeneous Actor-Based Monitor Language for Adaptive Behaviour. In: Programming with Actors: State-of-the-Art and Research Perspectives, pp. 216–244 (2018). https://doi.org/10.1007/978-3-030-00302-9_8
7. Clark, T., Kulkarni, V., Barat, S., Barn, B.: Actor Monitors for Adaptive Behaviour. In: Proceedings of the Innovations in Software Engineering Conference, pp. 85–95 (2017). https://doi.org/10.1145/3021460.3021469
8. D'Angelo, B., Sankaranarayanan, S., Sanchez, C., Robinson, W., Finkbeiner, B., Sipma, H., Mehrotra, S., Manna, Z.: LOLA: Runtime Monitoring of Synchronous Systems. In: Proceedings of the International Symposium on Temporal Representation and Reasoning, pp. 166–174 (2005). https://doi.org/10.1109/TIME.2005.26
9. Distefano, D., Rensink, A., Katoen, J.-P.: Model Checking Birth and Death. In: Proceeings of IFIP International Conference on Theoretical Computer Science (TCS), pp. 435–447 (2002). https://doi.org/10.1007/978-0-387-35608-2_36
10. Esterle, L., Brown, J.N.: The Competence Awareness Window: Knowing what I can and cannot do. In: 2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C), pp. 62–63 (2020). https://doi.org/10.1109/ACSOS-C51401.2020.00031
11. Finkbeiner, B., Oswald, S., Passing, N., Schwenger, M.: Verified Rust Monitors for Lola Specifications. In: Runtime Verification, pp. 431–450 (2020). https://doi.org/10.1007/978-3-030-60508-7_24
12. Havelund, K., Peled, D.: Runtime Verification: From Propositional to First-Order Temporal Logic. In: Runtime Verification, pp. 90–112 (2018). https://doi.org/10.1007/978-3-030-03769-7_7
13. Hicks, M., Moore, J.T., Nettles, S.: Dynamic software updating. ACM SIGPLAN Notices **36**(5), 13–23 (2001). https://doi.org/10.1145/381694.378798
14. Iftikhar, M.U., Weyns, D.: ActivFORMS: Active Formal Models for Self-Adaptation. In: Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), pp. 125–134 (2014). https://doi.org/10.1145/2593929.2593944
15. Kephart, J., Chess, D.: The vision of autonomic computing. Computer **36**(1), 41–50 (2003). https://doi.org/10.1109/mc.2003.1160055
16. Larsen, P.G., Ali, S., Behrens, R., Cavalcanti, A., Gomes, C., Li, G., De Meulenaere, P., Olsen, M.L., Passalis, N., Peyrucain, T., Tapia, J., Tefas, A., Zhang, H.: Robotic safe adaptation in unprecedented situations: the RoboSAPIENS project. Research Directions: Cyber-Physical Systems **2** (2024). https://doi.org/10.1017/cbp.2024.4
17. Leucker, M., Sánchez, C., Scheffel, T., Schmitz, M., Schramm, A.: TeSSLa: Runtime Verification of Non-Synchronized Real-Time Streams. In: Proceedings of the Annual ACM Symposium on Applied Computing, pp. 1925–1933 (2018). https://doi.org/10.1145/3167132.3167338
18. Li, Y., Duan, X., Xu, Y., Zhao, C.: Dynamic Assessment Approach for Intelligent Power Distribution Systems Based on Runtime Verification with Requirements Updates. High-Confidence Computing (2024). https://doi.org/10.1016/j.hcc.2024.100255

19. Lounas, R., Mezghiche, M., Lanet, J.-L.: Formal Methods in Dynamic Software Updating: A Survey. International Journal of Critical Computer-Based Systems **9**(1–2), 76–114 (2019). https://doi.org/10.1504/IJCCBS.2019.098794
20. Pedregal, P., Gorostiaga, F., Sánchez, C.: A Stream Runtime Verification Tool with Nested and Retroactive Parametrization. In: Runtime Verification, pp. 351–362 (2023). https://doi.org/10.1007/978-3-031-44267-4_19
21. Seifzadeh, H., Abolhassani, H., Moshkenani, M.S.: A survey of dynamic software updating. Journal of Software: Evolution and Process **25**(5), 535–568 (2012). https://doi.org/10.1002/smr.1556
22. Sokolsky, O., Sammapun, U., Lee, I., Kim, J.: Run-Time Checking of Dynamic Properties. Electronic Notes in Theoretical Computer Science **144**(4), 91–108 (2006). https://doi.org/10.1016/j.entcs.2006.02.006
23. Subramanian, S., Hicks, M., McKinley, K.S.: Dynamic Software Updates: A VM-centric Approach. In: Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 1–12 (2009). https://doi.org/10.1145/1542476.1542478
24. Walton, C., Krl, D., Gilmore, S.: An Abstract Machine for Module Replacement. In: Proceedings of the Workshop on Principles of Abstract Machines (1998)
25. Weyns, D.: Engineering Self-Adaptive Software Systems – An Organized Tour. In: Proceedings of the IEEE International Workshops on Foundations and Applications of Self* Systems (FAS*W), pp. 1–2 (2018). https://doi.org/10.1109/FAS-W.2018.00012
26. Wong, T., Wagner, M., Treude, C.: Self-adaptive systems: A systematic literature review across categories and domains. Information and Software Technology **148**, 106934 (2022). https://doi.org/10.1016/j.infsof.2022.106934
27. Zhang, J., Cheng, B.H.C.: Using Temporal Logic to Specify Adaptive Program Semantics. Journal of Systems and Software **79**(10), 1361–1369 (2006). https://doi.org/10.1016/j.jss.2006.02.062

Appendix

Proof of Theorem 1

Proof. Assume ϕ is a stream expression that is solvable at monitor step t, potentially containing DUPs receiving solvable subexpressions $(\psi_1, \ldots, \psi_N)$. We proceed by structural induction on the stream expression ϕ:

- Basic expressions: By definition of solvable, all stream variables referenced by ϕ at their respective time indices are not equal to $\perp$ at monitor step t. Therefore, if ϕ is a basic expression it cannot evaluate to $\perp$ at monitor step t, since basic expressions only evaluate to $\perp$ when a referenced stream variable is $\perp$ at that time step.
- $\texttt{default}(\psi, c)$ never evaluates to $\perp$, because it yields ψ if $\psi \neq \perp$, and defaults to the constant $c \in \mathbb{D}$ otherwise.
- $\texttt{when}(\psi)$ is guaranteed to evaluate to a value in $\mathbb{B}$.
- $\texttt{update}(\psi_1, \psi_2)$ evaluates to ψ_2 if $\psi_2 \neq \perp$, which is guaranteed by the definition of solvable.
- DUPs: If ψ_i is a DUP, it will either be $\psi_i = \texttt{defer}(\psi'_i)$ or $\psi_i = \texttt{dynamic}(\psi'_i)$. For ψ_i to be solvable, ψ'_i must also be solvable. By induction, this means that ψ'_i is either a non-DUP expression that evaluates to a non-$\perp$ value at step t or a DUP that is solvable, meaning it will evaluate to a non-$\perp$ value at step t.

Therefore, ϕ evaluated at monitor step t is not equal to $\perp$.