TEMPORAL MODELS FOR HISTORY-AWARE EXPLAINABILITY IN SELF-ADAPTIVE SYSTEMS

JUAN MARCELO PARRA ULLAURI

Doctor of Philosophy

ASTON UNIVERSITY

September 2022

Abstract

The complexity of real-world problems requires modern software systems to be able to autonomously adapt and modify their behaviour at runtime to deal with unforeseen internal and external fluctuations and contexts. Consequently, these self-adaptive systems (SAS) can show unexpected and surprising behaviours which stakeholders may not understand or agree with. This may be exacerbated due to the ubiquity and complexity of Artificial Intelligence (AI) techniques which are often considered "black boxes" and are increasingly used by SAS.

This thesis explores how synergies between model-driven engineering and runtime monitoring help to enable explanations based on SAS' historical behaviour with the objective of promoting transparency and understandability in these types of systems. Specifically, this PhD work has studied how the use of runtime models extended with long-term memory can provide the abstraction, analysis and reasoning capabilities needed to support explanations when using AI-based SAS. For this purpose, this work argues that a system should i) offer access and retrieval of historical data about past behaviour, ii) track over time the reasons for its decision making, and iii) be able to convey this knowledge to different stakeholders as part of explanations for justifying its behaviour.

Runtime models stored in Temporal Graph Databases, which result in Temporal Models (TMs), are proposed for tracking the decision-making history of SAS to support explanations. The approach enables explainability for interactive diagnosis (i.e. during execution) and forensic analysis (i.e. after the fact) based on the trajectory of the SAS execution. Furthermore, in cases where the resources are limited (e.g., storage capacity or time to response), the proposed architecture also integrates the runtime monitoring technique, complex event processing (CEP). CEP allows detecting matches to event patterns that need to be stored instead of keeping the entire history. The proposed architecture helps developers in gaining insights into SAS while they work on validating and improving their systems.

To my family and friends:

I could not have done it without you.

Thank you for always believing in me and for your constant support.

Simplemente gracias!

Declaration.

I declare that this thesis was written by myself and presents my own work developed in the context of my PhD in the Computer Science Department at Aston University. The work reported in this thesis has not been previously submitted for a degree, in this or any form. The approach based on temporal models, its extension including complex event processing, and its implementation were developed by myself for supporting the ideas and design challenges reported in the thesis. Nevertheless, this work does build on the ideas proposed in the paper "Reflecting on the past and the present with temporal graph-based models" [57], the Eclipse Hawk project and its query language, and has benefited considerably from the Software Engineering at Aston (SEA) research group. Hopefully, this thesis contributed to the progress and expansion of the vision of the group and the department. This research was fully funded by a Faculty Postgraduate Studentship at Aston University.

Acknowledgements

I would like to express my deepest gratitude to my supervisors, Dr Antonio Garcia-Dominguez and Dr Nelly Bencomo, who have been great inspirations to me during my studies. Thank you for your invaluable patience and feedback. I have learned from you in every meeting and I am thankful for both the professional and life advice you have given to me during this time. Thank you for supporting as well as criticising my ideas and work, which has helped me to grow as a researcher. I hope this thesis reflects everything you have taught me and makes you proud.

I also want to thank the staff in the Computer Science department. Special thanks to the Software Engineering in Aston (SEA) research group for the valuable discussions we have had during these years. Thank you to SEA members Luis, Huma, and Owen with whom I collaborated during my PhD. Thanks should also go to Dr Paul Grace who acted as a reviewer in my first and second-year reports and who is now my main supervisor. Additionally, this work would not have been possible without the generous support from Aston University, who financed my research.

I would like to extend my sincere thanks to Dr Juan Boubeta-Puig and Dr Guadalupe Ortiz from the University of Cadiz for their support during my research stay in Cadiz. I am also thankful to Dr Shufan Yang from Napier University, Changgang Zheng from the University of Oxford and Chen Zhen from the University of Science and Technology of China for your collaboration and contribution to my PhD. I am truly grateful to have had the opportunity to work with you.

To my friends in the UK, Thomas, Monse, Salma, and others, thank you for making me feel at home in Birmingham. To my friends back home and in South America, Wilson, Daniel, David, Jose, Ignacio, Rodrigo, Franco, and others, thank you for all the support and good energy from a distance. To my godmother, Madre Amadita, and our family friend

Padre Ramiro, thank you for all the blessings.

To my girlfriend, Dr Lily Hawkins, thank you for the patience, support and love during these years. It has not been easy, and you helped me in every stage. Te Quiero. We did it!

Lastly, to my family, thank you for encouraging me to dream big. To my sisters Andrea, Adriana, and Mayra, I am truly thankful to have you in my life. We are always together even though we are in different parts of the world. To my father Marcelo, thank you for always believing in me and for supporting me. To my grandmother, Rosario, thank you for being a role model for all of us. We miss you. Finally, to my mother, Marlene, words cannot express how grateful I am to have you. Thank you for every Monday message and encouraging words. Thank you for always being supportive and for teaching us that with hard work, self-belief, passion, and love, everything is possible. Le Amo Ma. I hope this makes you proud.

List of Publications

1) A. García-Domínguez, N. Bencomo, J. M. Parra-Ullauri, and L. H. García-Paucar. Towards history-aware self-adaptation with explanation capabilities. In *2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS* W)*, pages 18–23. IEEE, 2019 [59].

2) A. García-Domínguez, N. Bencomo, J. M. Parra-Ullauri, and L. H. García-Paucar. Querying and annotating model histories with time-aware patterns. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 194–204. IEEE, 2019 [60].

3) J. M. Parra-Ullauri, A. García-Domínguez, L. H. García-Paucar, and N. Bencomo. Temporal models for history-aware explainability. In *Proceedings of the 12th System Analysis and Modelling Conference*, pages 155–164, 2020 [119].

4) J. M. Parra-Ullauri, A. García-Domínguez, J. Boubeta-Puig, N. Bencomo, and G. Ortiz. Towards an architecture integrating complex event processing and temporal graphs for service monitoring. In *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, pages 427–435, 2021 [121].

5) C. Zheng, S. Yang, J. M. Parra-Ullauri, A. Garcia-Dominguez, and N. Bencomo. Reward-reinforced generative adversarial networks for multi-agent systems. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2021 [165].

6) J. M. Parra-Ullauri, A. García-Domínguez, and N. Bencomo. From a series of (un) fortunate events to global explainability of runtime model-based self-adaptive systems. In *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 807–816, 2021. doi: 10.1109/MODELS-C53483.2021.00127 [122].

7) J. M. Parra-Ullauri, A. Garcia-Dominguez, N. Bencomo, S. Yang, C. Zheng, C. Zhen, J. Boubeta-Puig, and G. Ortiz. Event-driven temporal models for explanations - etemox: explaining reinforcement learning. *Software and Systems Modeling*, 2021. doi: 10.1007/s10270-021-00952-4 [120].

8) J. M. Parra-Ullauri, A. García-Domínguez, and L. G.-P. Bencomo, Nelly. History-aware explanations: Towards enabling human-in-the-loop in self-adaptive systems. In *Proceedings of the 14th System Analysis and Modelling Conference*, 2022. To be published [123].

Contents

List of Figures

List of Tables

Glossary

- **ABS:** Airborne Base Station.
- **AI:** Artificial Intelligence.
- **CEP:** Complex Event Processing.
- **DQN:** Deep Q-Network.
- **EMF:** Eclipse Modeling Framework
- **EOL:** Epsilon Object Language.
- **EDM:** Event-Driven runtime Monitoring.
- **ETeMoX:** Event-driven Temporal Models for Explanations.
- **EPL:** Event Processing Language.
- **GDPR:** General Data Protection Regulation.
- **H-A:** History-Aware.
- **MDE:** Model Driven Engineering.
- **MDP:** Markov Decision Process.
- **ML:** Machine Learning.
- **MQTT:** MQ Telemetry Transport.
- **QL:** Q-Learning.
- **RL:** Reinforcement Learning.
- **RSRP:** Reference Signal Received Power.
- **RTM:** Runtime Monitoring.
- **SARSA:** State - Action - Reward - State - Action.
- **SAS:** Self-Adaptive System.
- **SINR:** Signal-to-Interference-plus-Noise Ratio.
- **TGDB:** Temporal Graph Databases.
- **TMs:** Temporal Models.
- **XAI:** Explainable Artificial Intelligence.
- **XRL:** Explainable Reinforcement Learning.

Chapter 1

Introduction

1.1 Research Problem

Humankind increasingly entrusts computerised systems with complex and critical tasks [30]. Such systems are usually required to autonomously adapt and modify their behaviour at runtime based on their observations in order to cope with uncertain and dynamic environments [111, 35]. These, so-called Self-Adaptive Systems (SAS), collect data during operation to reason about themselves and based on their goals, to reconfigure or adjust their behaviour to satisfy evolving conditions and complexities [159]. Early solutions to self-adaptation tended to adapt according to monitored changes based on the knowledge that was known at design time (e.g., rule-based systems), providing limited reasoning and reflection capabilities [35].

Modern solutions can learn new information during execution to provide estimations about the future that support better-informed and proactive decision-making [108, 31, 124]. However, self-adaptive actions may run into problems or present unexpected behaviour due to uncertainty in the environment, which is exacerbated by the ubiquity of Artificial Intelligence (AI)-based SAS [133]. Sudden actions performed by the SAS can create surprise and consternation in users that may not understand the behaviour of the system even if correct. Further, these users may cease to use the system due to the lack of trust and understanding [141]. Providing understandable explanations for surprising behaviour can be key to tackling the challenges described and has become a topic of relevant interest for the SAS research community [141, 31, 29]. Explainability of decision-making processes can be key to enabling users' understanding and increment confidence, in order to promote the

widespread adoption of SAS [119, 96].

Explanations shown by a running system can help someone observing the system's behaviour to analyse and trace actions to help fix potential faults, convey knowledge about the decision-making, and foster the trust of end users [59]. This is ratified by the General Data Protection Regulation (GDPR) law, which enshrines the right to explanation [32], and the IEEE P7001 - Transparency of Autonomous Systems [162]. In SAS, explainability can be described as the capability of answering questions about the system's past, present and future behaviours [57]. The answers to these questions can explain why a decision was made or why a particular state was reached [119]. It is argued that history-awareness can help to explain coherently to various stakeholders (e.g. end-users or other SAS) which situations have caused the system's current behaviour [121].

In order to enable these history-aware explanatory capabilities, this PhD work discusses that a SAS should be equipped with traceability management facilities and offer temporal links to provide (i) the history of the decisions of the system and the evidence that supports the decisions made under the observed environmental conditions, and (ii) the impacts of the adaptation actions over the quality properties of the system over time. These history-awareness requirements present several challenges such as how the history is structured and stored considering the availability of resources, how this stored data can be distilled to derive meaning from the history, and how to minimise the impact of adding these capabilities on the SAS performance, and how to present explanations for a specific purpose and target using historical information. A more in-depth discussion about the challenges for history-awareness in SAS will be presented in Section 1.1.1.

Given the above, this thesis explores how Runtime Monitoring (RTM) and the abstraction capabilities provided by Model-Driven Engineering (MDE) can be combined to support the tackling of the previously mentioned challenges to enable explanations based on SAS historical behaviour with the main objective of promoting transparency and understandability in this type of systems.

On one hand, RTM enables engineers and maintenance personnel to keep track of the system's behaviour during operation and to check the interactions that occur between its components, as well as between the system and its environment [128]. Event-driven monitoring is a common RTM approach to gain insights about a system based on particular situations of interest (i.e. events) [81]. A goal of event-driven RTM is to determine whether

a system behaves as intended, allowing the potential correction of the behaviour when required [44]. However, it can come at a cost of the impact on the performance of the monitored system [128]. Therefore, it is important to deploy the right type of monitoring service to achieve its objectives whilst minimising negative impact as well as complexity [71].

On the other hand, MDE has evolved substantially and its adoption in industry is increasing [25]. MDE aims to describe (i.e. specify and build) software systems supported by abstractions called *models* [69]. MDE has made substantial contributions to leverage abstraction and automation in many areas of software and systems development and analysis [25]. Specifically, runtime models [15] seek to extend the applicability of models in MDE approaches to the execution of a system. They have the potential to be used to monitor and verify particular aspects of runtime behaviour, and to implement self-* capabilities (e.g., adaptation technologies used in self-healing, self-managing, self-optimising systems) [13]. Following a models@run.time (i.e. runtime models) approach, where the system operates from a formal set of rules described in a modelling language, is a technique used in the development of SAS [14] and it has been recognised as the *third wave* of engineering SAS [159]. Crucially, runtime models can be also used to provide the abstraction, analysis and reasoning capabilities needed to support explanations when using AI-based SAS.

An extended review of these two areas, RTM and MDE for SAS, is presented in Chapters 3 and 4.

1.1.1 Challenges when enabling history-aware explainability in SAS

Different challenges when enabling explanations based on the historical behaviour of SAS are listed next:

CH1: Collection, organisation and storage of the historical data produced by the system

Storing the historical data of a running system can be costly for its performance. For example, scalability issues may arise when large amounts of logs are produced as a result of processes in the systems as they need to be stored on disk. Keeping each version of a time-evolving system would further increase the storage capacity needed exponentially, and the performance of the system may be affected due to dealing with such an amount

of data. Therefore, providing the capability of reasoning based on the history of a system would need to mine a large amount of data, extract a relevant view, and finally analyse it. This capability would require considerable computational power while being highly time-consuming, conflicting with the strict real-time response time requirements of modern software systems [67]. Consequently, it is necessary to find scalable and efficient alternatives for storing large amounts of data without compromising the SAS performance.

Logs are prevalent in all kinds of computer software. However, principally, they are text-based and are usually addressed to humans, while showing limited support for automated processing such as basic filtering and tagging [58]. The increasing level of automation in cloud deployments has motivated some IaaS (Infrastructure as a Service) platforms to explicitly collect historical data intended to be used by software systems as well [18]. For instance, the Google Cloud platform is known to track memory usage and recommend VM changes[1].

There has been considerable work on time-series data mining which attempts to extract knowledge by looking at the shape of the data as described in the survey presented by Esling and Agon in [50]. This survey lists a wide variety of approaches for querying by content, clustering, classification, segmentation, and prediction, among other tasks. However, the history of a system can be more complex than a sequence of numbers. In its most general form, the configuration of a system is a complex entity that changes over time. In order to make a system more explainable, tracking the history requires a fitting data structure. Therefore, this PhD dissertation defines three crucial aspects to be considered for collecting a SAS history: (i) formatting the logs to make them machine-parseable, (ii) structuring the data on a reusable manner to make it suitable for explanations for different SAS, and (iii) minimising negative impacts on the performance of the monitored SAS.

CH2: Query and extraction of information to provide explanations

If the system shows surprising or unexpected behaviour, users or stakeholders may want to know why. Answering emerging questions related to such situations would require execution-tracing features to allow the analysis back and forth in the system's history for monitoring the decision-making performance against the available evidence at one or more points in time. Working with historical information that is accessed through a data structure involves

[1] `https://archive.ph/lAMvF`

being able to perform queries on a given data storage [23]. Accordingly, beyond compact storage (related to Challenge 1), history-awareness requires an efficient manner to represent and query the history. To cover these demands, it is argued that a flexible and scalable approach for querying the history of the SAS is needed to study the evolution for the validation of the system. Having a suitable way to write queries over the history of the system is an important aspect of reusable explanatory capabilities [57].

CH3: Provision of explanations for different purposes and consumers

Transparency and trust play an important role in the acceptance of information systems. Systems able to explain their decisions, concepts, and information sources to users can demonstrate their trustworthiness and support users' understanding [40]. Further, systems able to generate explanations for their own internal use may be able to increase their robustness in dealing with unexpected situations, as well as to improve their future performance, by refining their internal models and reasoning processes [133]. All of these potential benefits depend on the ability of the systems to generate high-quality explanations. Accordingly, it is important to understand which is the main purpose of the explanations, who or what is going to be the consumers of these [1]. If the explanations are human-oriented, it is important to define first what is the technical expertise of the user involved: for example, a developer may need a different level of detail than an end-user. In this PhD work, explanations are mainly targeted to SAS developers. Another approach could be machine-oriented. In this case, the explanations may be used for either internal reasoning of the SAS or for the case of a system explaining its behaviour to other systems in a system-of-systems deployment. All these scenarios are valid and need research towards history-aware explainability.

1.1.2 Research questions

Explanations in philosophy have been the focus of vast research over the years [105]. The philosopher David K. Lewis stated:

> "*To explain an event is to provide some information about its causal history. In an act of explaining, someone who is in possession of some information about the causal history of some event –the explanatory information– tries to convey it to someone else.*" — Lewis, D.[90]

Based on this premise, this PhD work investigates the importance of history for transferring knowledge and providing explanations. Consequently, and based on the problem statement and the key issues and challenges identified above, the research gap and motivation for this thesis were defined.

To explore the challenges of history-aware explanations discussed in this chapter this far and considering the selected approaches of RTM and MDE to undertake these challenges, there are three primary questions and concrete measurable sub-questions that this thesis aims to address:

- **RQ1:** How can model-driven engineering and runtime monitoring enable scalable and structured historical data storage?
  - RQ1.1: What is the effectiveness of the proposed solution which combines Temporal Models from model-driven engineering and Complex Event Processing from runtime monitoring for history-aware explainability?
  - RQ1.2: How does the proposed approach compare to existing approaches?
- **RQ2:** How can the exploration of the stored SAS history support developers wishing to improve or validate their systems?
  - RQ2.1: How useful is the proposed approach, which uses post-hoc explanations extracted during runtime monitoring, in enabling developers or external entities to monitor and refine SAS systems?
  - RQ2.2: What insights can be gained to improve the system's behaviour and decision-making using the proposed approach?
- **RQ3:** How can external entities using history-aware explanations influence the SAS decision-making in an informed way?
  - RQ3.1: How can a history-aware architecture enable a bi-directional communication approach between stakeholders and a SAS?
  - RQ3.2: How can feedback from external entities through explanations be integrated into the SAS decision-making process to enhance its performance?

Each research question is related to the previously defined challenges. RQ1 will focus on the feasibility of keeping track of the history of SAS (CH1). RQ2 explores temporal

assertions that will allow the elicitation of historic information that will conform explanations (CH2). Finally, RQ3 analyses the importance of history-aware explanations for the purpose of enhancing collaboration between the SAS and external entities as the consumers or recipients of the explanations (CH3). To study and tackle these research questions, a series of experiments were designed, conducted and evaluated.

1.2 Contributions of this Thesis

The intention behind the efforts presented within this PhD work was to gain an understanding into how runtime models –from MDE– combined with runtime monitoring of SAS can be used to keep track of the system's reasoning over time to extract history-aware explanations on demand. Concretely, the work proposes ***Temporal Models*** that seek to add short and long-term memory to runtime models through the use of temporal databases [119, 120] (a more in-depth description is provided in Chapter 6). A generic post-hoc (i.e., after the event) framework based on temporal models towards history-aware SAS is proposed with two main objectives: i) to allow users to gain trust on a SAS through explanations based on the system's historical behaviour, ii) to empower SAS decision-making processes with explicit reflective capabilities about past performance. Rather than an all-or-nothing situation, it is argued that it is more cost-effective to develop these capabilities in stages or *levels*. Colleagues from the Software Engineering at Aston (SEA)[2] research group proposed in [59] a 4-level spectrum of reflective capabilities in SAS. They go from forensic explanations to autonomous history-aware decision-making. This research roadmap that was further refined and presented in [119], acts as the research roadmap for this PhD work. Considering the above, the main contributions of this thesis are as follows:

i The study, design and implementation of the first three levels of the mentioned gradual approach for a spectrum of reflective capabilities for history-aware SASs.

ii A novel approach for combining runtime models and temporal databases –Temporal Models– which record on an structured fashion the goals of a SAS, its decisions, its observations and its reasoning over time.

iii An evaluation of a set of time-aware extensions to an existing model querying language

[2] `https://cs.aston.ac.uk/sea/`

that enable access to the stored system history to extract information that will conform explanations.

iv A novel generalizable architecture based on Temporal Models and event-driven runtime monitoring, called *ETeMoX* (presented in Chapter 7), for the extraction of history-aware explanations from data-intensive SAS on a post-hoc manner.

v Two comprehensive experimental studies and accompanying analyses, designed to investigate the proposed research questions about effects of history-aware explanations, that establish:

- Temporal models are a convenient solution for storing the history of SAS. The provision of temporal assertions based on the temporal query language allows access and exploration of the SAS history.
- Temporal models combined with event-driven runtime monitoring can tackle the challenges in volume and throughput posed by data-intensive systems in a resource-aware manner.
- Explanations based on the historical behaviour of SAS help developers and other stakeholders validate and verify the system's runtime decision-making processes.
- The consumers of explanations, whether humans or machines, can use the provided information for internal reasoning and for providing feedback to the system, online or offline, to the system based on the information acquired.

1.3 Research strategy

In order to meet the established goals, the following approach has been applied:

- To undertake a survey of the state of the art on approaches to support explanations in SAS, to identify the research gaps.
- To examine and evaluate the state of the art in MDE and RTM for SAS in the context of knowledge discovery.
- To study the proposed research roadmap for history-awareness in SAS.

- To investigate, following the research roadmap, how temporal models, model-driven and event-driven RTM can be complemented between them to support explanations in SAS.
- To design, implement and evaluate the proposed architecture in different case studies to validate the feasibility of the approach.
- To report results and limitations after the validation is performed.

The research done aims to demonstrate the viability and the benefits of the generalizable approach using the proposed techniques to support explanations based on the historical behaviour of SAS.

1.4 Structure of this Document

The remainder of this thesis is organised into eight chapters with the following structure. Chapters 2, 3 and 4 describe the baseline and preliminary literature review needed to comprehend this work. On the other hand, Chapters 5, 6 and 7 presents the contributions of this thesis. Figure 1.1 depicts the flow of chapters. Chapter 2 presents the foundations regarding SAS and motivates the need for explanations on such systems that underlies this research. An introduction to MDE and its use in the context of SAS is depicted in Chapter 3. Chapter 4 describes different approaches of runtime monitoring. The proposed gradual approach for reflective capabilities in SAS is described in Chapter 5. The following 2 chapters are self-contained: Chapter 6 presents a first experiment of the proposed model-driven approach applied to the case study of a network management SAS, and Chapter 7 describes how model-driven and event-driven monitoring can be combined to support explanations. The approach is demonstrated on a case study for a telecommunications SAS. Finally, Section 8 presents the conclusions and future work.

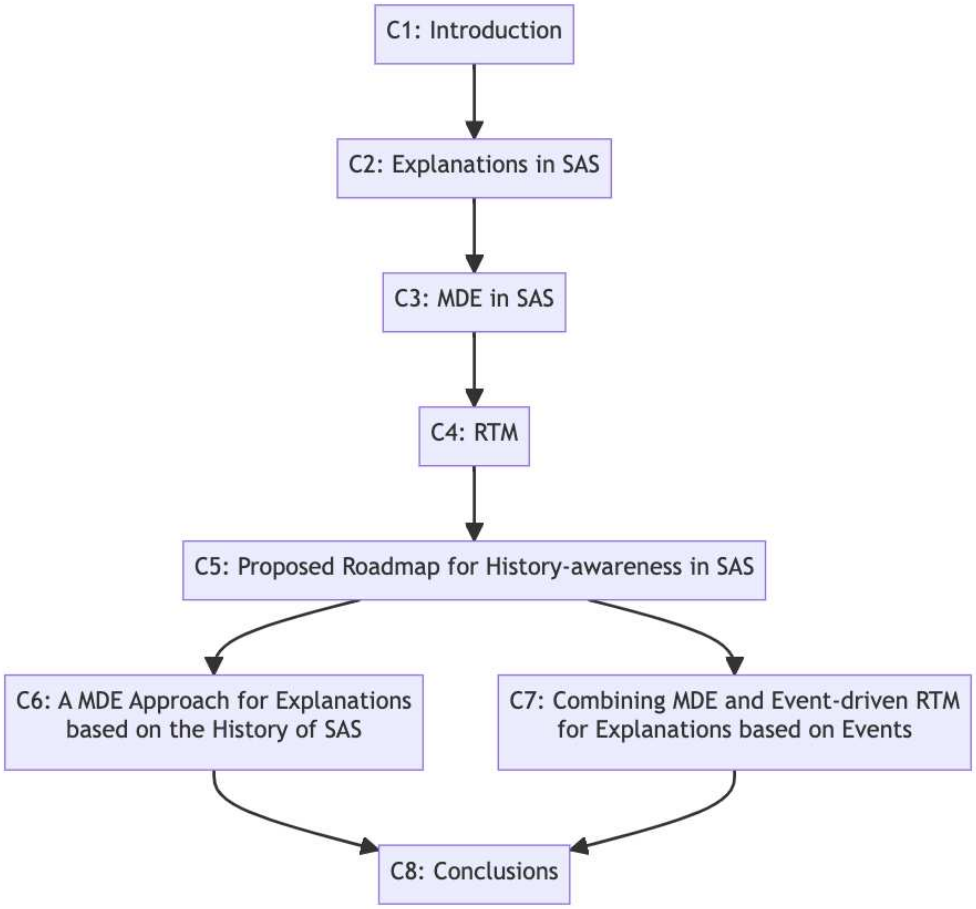


Figure 1.1: Thesis structure

Chapter 2

Explanations in Self-Adaptive Systems

This chapter provides a brief introduction about SAS and important concepts relevant for the present document. It also motivates the need for explanations in SAS, especially in AI-enhanced SAS. Current approaches to support explanations and its classification are also presented.

2.1 Self-Adaptive Systems (SAS)

As computer systems become more pervasive in our every day lives with the continual advancement of technology, software systems are expected to dynamically adapt to changes in the highly volatile and heterogeneous environments that characterise them. SAS address these challenges by being able to automatically modify their behaviour in response to changes in their operating environment [88]. Scientists and engineers have made significant efforts to design and develop SAS. These systems address adaptivity in various concerns including performance, security, and fault management [137].

The applications of SAS are broad and diverse, they go from adaptable user interfaces to autonomous robots, from embedded systems to mobile ad-hoc networks, and many other areas of application and research [35]. The common element that enables adaptability is usually software [43]. Self-adaptive software is expected to fulfil its requirements and to adjust its behaviour in response to its perception of the environment and the system itself [35, 137]. For achieving this adaptive behaviour, basic system properties are self-

awareness and context-awareness [88]. Self-awareness can be seen as the capability of a system to acquire and access knowledge about its own state and resources [31]. Such knowledge allows for better understanding and reasoning about its adaptive behaviour. Self-awareness of a computing system can be related to different specific capabilities such as goal-awareness [35], requirements-awareness [141] or time-awareness [91]. Time-awareness is the use of knowledge of historical and perhaps future phenomena [91]. Therefore, history-awareness is implied in time-awareness [57]. On the other hand, context-awareness means that the system is aware of its operational environment, the so called context [88].

2.1.1 A conceptual model of SAS and the MAPE-K loop

The workflow of sensing and applying adaptations is typically characterised by a feedback loop involving four key activities: i) collect data from the system and its context, ii) analyse this data, iii) subsequently, make a decision regarding the adaptive behaviour, and iv) and then perform an action [24, 35]. Within this frame of reference, Weyns, D. in [159], defined a conceptual model of a self-adaptive system (Figure 2.1). It is conformed by four main elements:

- Adaptation goals: They are the drivers of the system's reconfiguration and are usually related to the software quality of the system [159]. Four main adaptation goals can be recognised: self-configuration (systems that configure themselves automatically), self-optimisation (systems that continually seek ways to improve their performance or efficiency), self-healing (systems that detect, diagnose, and repair problems resulting from defects or failures), and self-protection (systems that defend themselves from malicious attacks or cascading failures) [159].

- Environment: The environment refers to the part of the external world with which the self-adaptive system interacts: its context [88]. The environment is where the effects of the system will be observed and evaluated [159].

- Managed system: It is the system that is controllable and subject to adaptation [63]. In order to realise its functionality, the managed system senses and affects the environment. To support adaptations, the managed system has to be equipped with sensors to enable monitoring and actuators to execute adaptations [159].

- Managing system: It is the system that, based on the adaptation goals, controls the managed system through an adaptation process [63]. In order to realise the adaptation goals, the managing system monitors the environment and the managed system and adapts the latter when necessary [159].

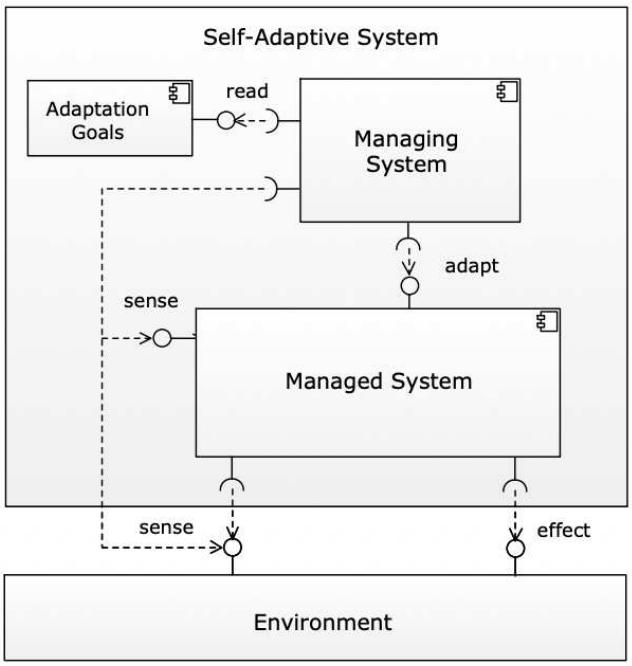


Figure 2.1: Conceptual model of a SAS [159].

The managing system operates the managed system based on the adaptation logic that deals with internal and external concerns [160]. A common approach to describe this adaptation logic is the MAPE-K feedback loop proposed by IBM [111] which consists of four key stages; Monitor, Analyse, Plan, and Execute around a Knowledge base (See Figure 2.2). The monitor stage senses the managed system and the environment in which the system operates (context), filters the accumulated sensor data, and updates the knowledge. The analyser stage uses the up-to-date knowledge to evaluate the need for adaptation. The planner stage then selects the best option based on the adaptation goals and generates a plan to adapt the system from its current configuration to the new configuration. The executor stage performs the adaptation actions of the plan [24, 111]. The MAPE elements map to the basic functions of a feedback loop, while the K component maps to runtime models (causally connected representations) maintained by the managing system to support the MAPE functions [160]. It is worth noticing that SASs are usually systems-of-systems [88],

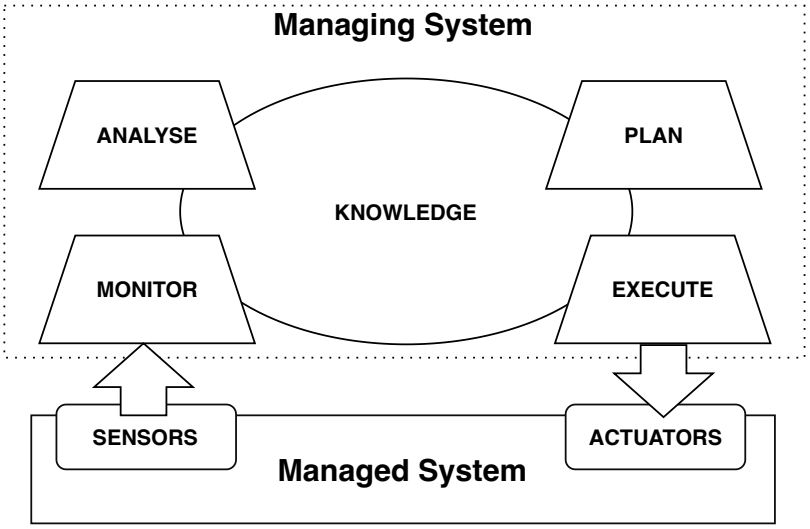


Figure 2.2: MAPE-K loop for SAS [111]

and the managed and managing subsystems can be deployed in a centralised or distributed fashion [160].

2.1.2 Research Challenges in SAS

Various researchers have dedicated their efforts to identify and define the research challenges for SAS [35, 43, 30, 159, 88, 141]. They are related to the fundamentals to engineer SAS, key elements for the concrete realisation of SAS regarding to its goals and assurances, uncertainty management, and the exploration and exploitation of established and new research trends [35, 159]. In this context, the challenges relevant to this thesis are listed and described next:

- *Modeling dimensions* [35]: The challenge is to define models that can represent a wide range of systems properties to support run-time analyses and decision processes [35]. Runtime models have been widely used for SAS management [15]. They underpin self-awareness, by allowing systems to abstract their own state and behaviour in a way that is amenable for automated adaptation strategies [43]. Apart from the challenges in the use of runtime models regarding domain specific modelling languages [159], it is discussed that full potential use of runtime models in SAS is still not accomplished. Runtime models for SAS will be further discussed in Sections 3.1.2 and 3.2

- *SAS requirements* [141, 43]: Once uncertainty is considered at the requirements definition stage, requirements monitoring is necessary for the reason that deviations

between the system's run-time behaviour and the requirements model may trigger the need for a system modification [43]. As such, a SAS is likely to exhibit emergent behaviour. Developers need to be able to trace the origin of this behaviour and users need to gain confidence in the system. These both require an ability for the system to account for its behaviour in some appropriate form [141].

- *SAS assurances* [30, 43]: The provision of perpetual assurances during the entire life cycle of a SAS poses three key challenges: how to obtain a better understanding of the nature of uncertainty in software systems and how it should be balanced, how to monitor and quantify uncertainty, and how to derive and integrate new evidence. Run-time validation and verification (V&V) is crucial in SAS to guarantee the system's desired properties and goals [30]. However, performing V&V tasks can come at a cost on the performance of the monitored system. Therefore, it is important to perform the right V&V tasks at an appropriate time and location to achieve its objectives whilst minimising negative impact as well as their complexity [71].

The present work tries to tackle some features of the previous stated challenges in the following manner:

- *Runtime Models to convey knowledge:* This thesis further argues that runtime models can be exploited not only for internal reasoning but also external. Runtime models can be used to provide abstraction, analysis and reasoning capabilities needed to explain to external entities (i.e. humans or other systems) why the system shows a given emerging behaviour. Section 3.1.2 will describe runtime models more in detail.

- *Explaining a SAS surprising behaviour:* SAS's behaviour is best explained in terms of the satisfaction of its requirements [12]. The emerging behaviour caused by a SAS trying to fulfil its requirements under the uncertainty in the environment, needs to be explained [141]. This thesis' main goal is to provide the tools to extract explanations about a SAS' historical behaviour to promote understandability and trustworthiness in this type of systems.

- *Explanations for Validations in runtime V&V:* This thesis argues that post-hoc explanations extracted during run-time monitoring can be used for Validation in V&V processes by developers and/or external entities. By combining MDE and effective

run-time monitoring techniques, this work tries to minimise the impact of the Validation processes in the SAS performance.

In addition to the previous challenges, this thesis targets AI-enhanced SAS, one recognised challenge in [159]. Rather than focusing on how to introduce AI into SAS, this work tackles the "black-box" challenges [1, 141] that the use of AI brings to SAS. The following section describes some applications of AI in SAS.

2.1.3 Artificial Intelligence and Machine Learning in SAS

Artificial Intelligence (AI) aims to mimic cognitive functions for real-world problem solving, building systems that learn and think like people do [125]. AI represents an effective way of emulating adaptivity, making organised and efficient systems easier to reconfigure and more highly adaptive [100]. Machine Learning is a sub-discipline of AI that focuses on building AI models of human learning and understanding how machines can be empowered with the ability of learning [106]. In ML, agents [1] learn either from training data or from policies to create AI models with minimal or no human intervention [102]. ML can broadly be categorised into supervised, unsupervised and reinforcement learning (RL) [106].

In recent years, we witness a rapid increase in the use of machine learning in SAS [141]. ML has been used for a variety of reasons, ranging from learning a model of the environment of a system during operation to filtering large sets of possible configurations before analysing them [140]. Gheibi et al. in [63] differentiates the adaptation problem and the learning problem (Figure 2.3). The adaptation problem is related to the managing system's problem-solving based on the adaptation goals (commonly driven by the MAPE-K loop). Meanwhile, the learning problem is related to the AI approach used by a machine learner[2] to support the MAPE-K loop in solving the adaptation problem.

The machine learner as part of the managing system can assist in different stages of the MAPE-K loop described in Section 2.1.1.

- *At the Monitor stage*, ML can be used for pre-processing the incoming data from the sensors and to update the knowledge models [63]. For example, in [11] the authors use Bayesian Learning to update and infer the models (i.e., descriptions) that the managing system has about the managed system.

[1] agent: autonomous or semi-autonomous AI-driven system, in other words, the learner [147]
[2] machine learner or agent: autonomous or semi-autonomous AI-driven system. [148]

- *At the Analyse stage*, ML can be used to classify large sets of adaptation options reducing the adaptation space such that only the relevant options need to be analysed, improving the efficiency of the system [127].

- *At the Plan stage*, ML can be applied for updating rules/policies [63]. For instance, in [73] the authors propose model-based Reinforcement Learning for policy evolution.

- *At the Execution stage*, ML can support the realisation of an adaptive action. For example, in [117], the authors proposed a classifier to choose the actuator that the system should use to adapt.

Besides the previous mentioned applications of AI in SAS, ML has been also applied to SAS without the concept of the an architecture or framework as the MAPE-K loop. These approaches use traditional control loops which refer to collect, analyse, decide, and act towards an adaptation without a formal architecture [140].

The adoption of AI has become ubiquitous in software-based systems when needing to provide better levels of autonomy and self-management in modern SAS. AI and ML has been successfully applied in vast domains such as transportation, recommendation systems or natural language processing among others [1]. Despite its broad applicability, the nature of ML is still considered as a "black-box" where system decisions can become opaque to stakeholders [162]. In this context, explanation-aware computing has received growing interest due to the ubiquity and complexity of AI-based systems, creating the notion of explainable AI (XAI) [138] to gain insight into the "black boxes" associated with AI. The next section will describe some approaches that have been used for providing explanations in AI-based SAS.

2.2 Explanations in SAS

Calinescu et al., emphasise uncertainty in the environment or domain in which the software is deployed as a prevalent aspect of SAS [29]. This uncertainty may cause surprising or unexpected behaviour [136]. The emergent behaviour caused by uncertainty may result on users not understanding or trusting a SAS. Such a lack of intelligibility can mean that users may cease to use a SAS [141]. Systems able to explain their decisions, concepts, and information sources to users can demonstrate their trustworthiness and support users'

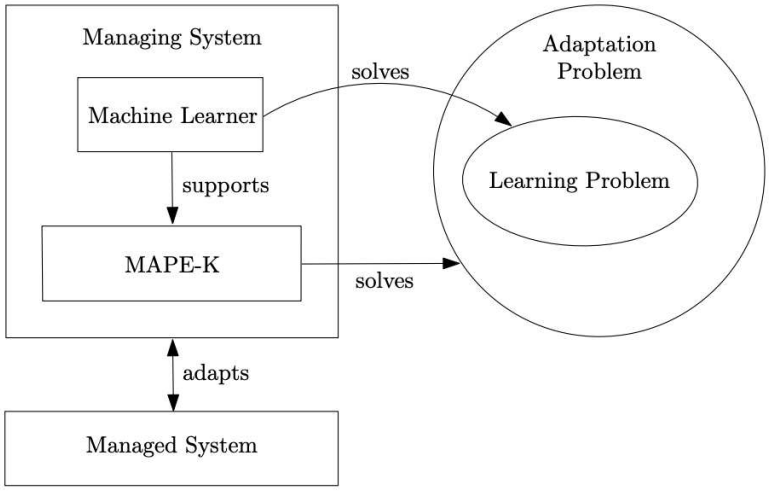


Figure 2.3: The learning problem and the adaptation problem in ML for SAS [63]

understanding [133].

Explanations provide a key capability to shape the human understanding of the environment, especially when their perceptions diverge from their expectations [40]. Explanations can prove or refute user hypotheses or mental models about system behaviour, and help fill the gaps in those incomplete mental models for causal accountability [92]. SAS increasingly use AI-based approaches for their flexible decision-making, which often appear to users as "black boxes" that are not inherently interpretable [159, 162]. There are different arguments in favor of explanations in AI. Adadi et al., stated four arguments in [1]:

- *Explain to justify:* AI is involved in more and more areas of our everyday lives. People affected by AI-influenced decisions (e.g. when refused a loan) may demand a justification for the particular outcome. This transparency is needed to ensure fair and ethical decisions [150] are being made.

- *Explain to control:* Explanations can often be used to keep agent actions inside an envelope of good behaviour. The explanations allow to discover the origin of a problem or to clarify misunderstandings between the system and the user [3]. Indeed, explanations can contribute to prompt identification of errors in non-critical situations [1].

- *Explain to discover:* Modern AI systems can process large amounts of data that otherwise would be difficult for humans to process. Asking for explanations is a helpful tool to extract insights about the knowledge acquired by this processing [1].

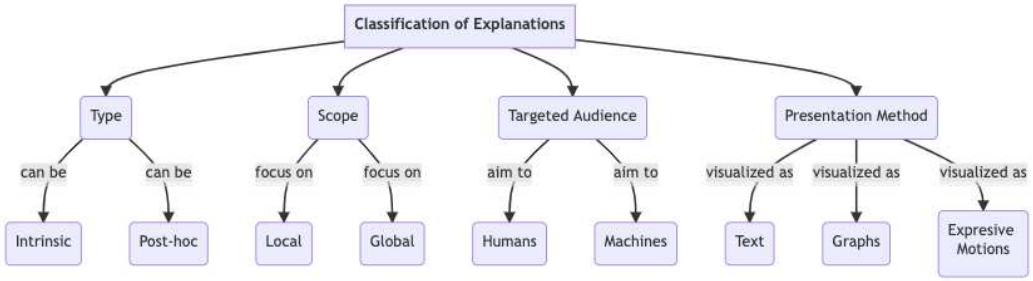


Figure 2.4: Classification of Explanations. Dimensions selected from [1] and [138]

- *Explain to improve:* In order to improve an AI system, it is key to discover its flaws. An AI system that can be explained and understood can be easier to enhance and use to the best advantage [138].

The XAI community has differentiated three main phases for building and conveying explanations [3]: i) the *explanation generation*, this phase focuses on the methods that allow the construction of the explanation (i.e, how to enable access to information that will build explanations) ii) the *explanation communication*, which deals with what information will be provided and how will it be presented and iii) *explanation reception* which studies how well the human understands the explanation. In order to assess this, typically, research relies on user studies and subjective evaluation [70].

This thesis focuses on the first two stages: explanation generation and explanation communication for explanations to control and improve AI-based SAS.

2.2.1 Classification of Explanations

From the literature, there are different techniques and approaches that are proposed to confer explainability [1, 138, 126, 70, 130, 41]. This section will focus on four dimensions selected from [1] and [138]: the type of the explanations, the scope of the explanations, the method for presenting the explanations and the profile of the audience targeted by the explanations. Figure 2.4 depicts the selected criteria.

By type

Explanations differ on whether they arise from the system's decision-making processes, or whether their generation requires post-processing [42]. In that sense, the extraction and/or

generation of explanations can be *intrinsic* or *post-hoc* [1]. The most straightforward way to get an interpretable AI system is to make it intrinsically explainable, thus self-explainable by design. One example are decision trees: they have a defined structure and can provide convincing capabilities to gain the trust of domain experts [42]. This type of explanations are model-specific by definition [1, 126]. Post-hoc explainability, on the other hand, aims to mimic the original system to provide the needed explanations without altering or even knowing the inner works of the original AI-model [42]. Rule extraction is an example of this type. By analysing the input and output of an artificial neural network, it provides a description of the knowledge learned by the network during its training by extracting rules that approximate the decision-making processes [130]. This type of explanations are generally AI-model -agnostic [1, 126]. This thesis focuses on post-hoc explainability of AI-enhanced SAS.

By scope

The explanations can either be *local* or *global* [42]. Local explanations focus on data and provide individual explanations, helping provide trust on AI-model outcomes. Local explanations focus on why the AI-model made a certain decision for one or a group of instances [126], whereas global explanations focus on the AI-model and provide an understanding of the overall decision process. A global explanation aims to provide a general understanding of how the AI-model works [1]. For example, a local explanation would be explaining the denial of a loan by a financial system to an end user (a single decision/prediction), while a global explanation would be the case of developers trying to understand the whole logic and reasoning that leads to all possible outcomes for the acceptance or denial of a loan for any user.

By targeted audience

Besides the motivation for explaining a system, it is also important to understand who or what is going to receive the explanations, in other words, the public to whom the explanations will be addressed [162]. It is essential to take into account the concept of audience, as the intelligibility and comprehensibility of a model is dependent on the goals and the cognitive skills of its users [70]. The target audience can consist both of *humans* or *machines*. In the case of humans, its crucial to identify the level of expertise of the explanation

consumer. They can be developers, domain experts or lay-users [70]. On the other hand, the explanations could be directed at another computer system. The SAS can explain its behaviour to other systems for example, on a systems of systems community. Furthermore, the consumer of the explanations can be the system itself, where explanations could enable reflective capabilities in SAS.

By presentation method

Different ways to communicate the explanations have been recognised [3]. *Text-based* explanations are the most common type presented in the form of natural language or logs [42]. This type of explanations *Visual representations* such as graphs, plots, or heatmaps among others are also used to depict explanatory information [70]. Other approaches include *expressive motions* and indicators, as well as text-to-speech for explaining robots [3]. The selection of the presentation method should be coherent with the targeted audience. For example an end-user may find easier to understand a high-level well defined graph than a set of logs. On the other hand, detailed information, may be more useful for developers trying to comprehend system's inner works.

2.2.2 Current approaches to support explanations in SAS

In recent years, explainability has been in the agenda of the SAS community [29, 141, 136]. In SAS, explainability can be described as the capability of answering questions about the system's past, present and future behaviours. The answers to these questions can explain why a decision was made or a particular state was reached [59]. Users may require explanations about why, for example, the system reached a certain state from the current state, which goals and requirements caused the system current behaviour, or how an external entity can help the system to achieve a certain goal [65].

This thesis discusses that explaining a SAS behaviour is related on explaining the adaptation problem, different from XAI approaches that focus on explaining the learning problem from Section 2.1.3. The previous can be mapped to the idea of Machine Reasoning Explainability proposed by Ericsson Research in [41], where explanations of a system are conformed not only by explaining AI black-boxes but, also explaining all the inner workings and system observations, and its interactions with the environment.

Although the field can be considered relatively new for SAS, there has been some re-

search interest in providing support for explanations as a self-* capability. For example, Bencomo et al., in [12] describes how goal-based requirements models can be adopted to offer explanation of how a system is meeting its requirements. In [48] the authors design methodology for self-explainable systems, and argue that beyond user acceptance, self-explanation also has other applications such as self-verification and reconfiguration. In [19], the authors propose an architecture for building self-explainable systems. They propose the MAB-EX loop as a framework that can extend the SAS MAPE-K loop in order to support explanations. The basic idea of MAB-EX is to first Monitor and Analyse a certain behaviour of a system, then Build an explanation from explanation models and convey this explanation in a suitable way to a stakeholder. Li et al. [92] propose explanations for human-in-the-loop. Their target is to define when an explanation should be provided as a tactic of the SAS to support human interaction. They analyse the cost or the latency of the explanation receptions (the third stage in [3]). Reynolds et al. proposed in [131] automated provenance graphs to explain the behaviour of SAS based on runtime models. Provenance graphs relate the entities, actors and activities in the system over time, recording the reasons why the system reached its current state. In [87], the authors of propose a framework based on self-descriptions and the Smart Object Description Language (SODL) for the realisation of self-explainability in SAS. By using this framework, smart objects and applications can be connected dynamically to control each other.

All the previous approaches are focused on the intrinsic type of explanations. Therefore, they focus on explanations that are generated and planned as a self-* capability of a SAS (i.e., self-explanations). While this is a valid approach, it is model-specific by definition and requires changes in the SAS, which is difficult to generalise [1]. Moreover, the existing work has focused on justifying specific decisions (i.e., local explanations). However, it is argued that an approach able to justify the whole logic of a model and follow the entire system reasoning is also required (i.e., global explanations) [126]. Apart from the previous mentioned, the approaches found in the literature don't define formally a targeted audience. This thesis focuses on AI-enhanced SAS developers and AI-enhanced SAS domain experts. These two groups of users are familiar with developing and/or using SAS and are hence interested in understanding, diagnosing, as well as refining such systems in a given application context [97]. In summary, there are some approaches towards enabling explanatory capabilities in SAS. However, to the best of the author's knowledge, none focus on explain-

ing of self-adaptive actions in a decoupled post-hoc manner nor allow both the extraction of local and global explanations targeting SAS developers and experts.

Chapter 3

Model-Driven Engineering in SAS

The development of self-adaptive software requires the engineering of an adaptation engine (i.e., managing system) that controls the underlying adaptable software (i.e., managed system) in feedback loops [159]. This separation decouples the engine from the adaptable software but it makes the feedback loop a crucial element of the overall software architecture [156]. Models can be helpful abstract representations of feedback loops and their interactions, among other wide range of aspects in SAS [35, 156]. In this context, MDE in SAS refers to the systematic use of models as primary artifacts for engineering, refining and managing such systems both at design time and runtime [43].

This chapter provides a brief introduction to MDE, and key features of the approach that will be used to enable post-hoc explanations in SAS. Important concepts relevant for the present work as run-time models and the notion of time in MDE are presented. Additionally, how MDE has been used in SAS is also discussed.

3.1 MDE in a nutshell

The human mind inadvertently and continuously re-works reality by applying cognitive processes that alter the subjective perception of it. Among the various cognitive processes that are applied, *abstraction* is one of the most prominent ones. Abstraction is also widely applied in science and technology, where it is often referred to as *modeling* [22]. In software engineering, applying advantages of modeling approaches for the development of software artifacts can be defined as Model-Driven Engineering (MDE) [22]. The process of analysing a problem, conceiving a solution, and expressing a solution in a high-level programming

language can be viewed as an implicit form of modeling and thus one can argue that software development is essentially a model-based problem solving activity [56].

Abstraction, automation, and analysis are the peculiar aspects of Model Driven Engineering that refer to the systematic use of models as first class entities throughout the software development life-cycle [142]. Some of the benefits of MDE are:

- *Systematic reuse of development knowledge and experience:* models can be used to capture knowledge about a specific domain and describe processes and principles. MDE supports how explicitly and systematically reusing these knowledge and experience representations [22].

- *Higher levels of abstraction:* MDE allows the level of abstraction at which developers operate to be raised, with the goal of both simplifying and formalising the activities and tasks of the software life-cycle [79].

- *The ability to synthesise artifacts through generators and transformation engines:* which allows access and analysis of certain aspects of models to automatically synthesise various types of artifacts, such as source code, deployment descriptions, or even other kinds of models representations [142].

MDE brings and adapts well-understood and long-established principles and practices of trustworthy systems engineering to software engineering; it is unthinkable to start constructing e.g. a bridge or an aircraft without designing and analysing several models of it first [83]. It's used extensively in organisations that produce business or safety-critical software, including in the aerospace, automotive and robotics industries, where defects can have catastrophic effects (e.g. loss of life), or can be very expensive to remedy, for example requiring large-scale product recalls. MDE is also increasingly used for non-critical systems due to the productivity and consistency benefits it delivers, largely through automated code generation [83]. The high re-usability and reliability of the code generated by this software paradigm, as well as its increased productivity and less costly maintenance, have led to its application in various fields including railway systems, automotive, business process engineering among many others [25].

In order for models to be amenable to automated processing, they need to be defined in terms of rigorously specified languages [114]. In this scope, a metamodel describes the structure of models in an abstract way. Particularly, a metamodel is defined using a metamodel

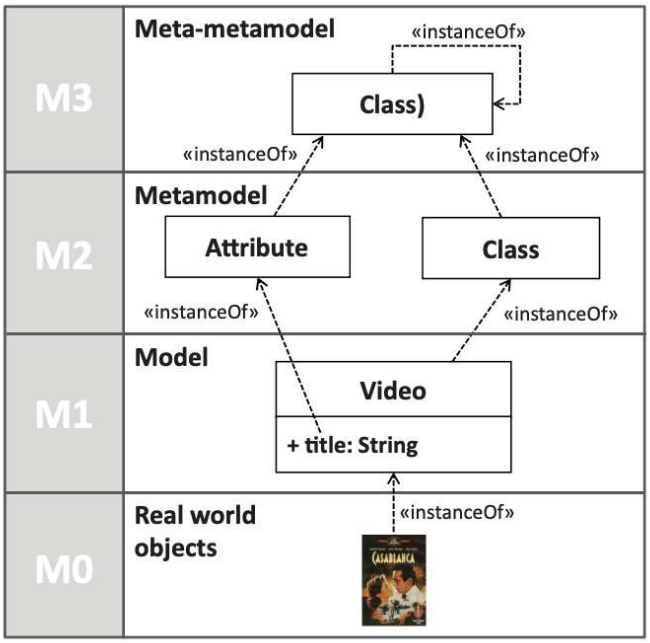


Figure 3.1: Relationships between models, metamodels and meta-metamodels [145]

language joined to a set of rules which specify the constraints so that the metamodel is well-formed [21]. Consequently, models represent abstractions of real systems that are analysed and engineered by identifying a coherent set of interrelated concepts precisely captured in metamodels. A model therefore is said to conform to a metamodel and model transformations generate target artefacts from source models. Models are analysed by means of queries specifically conceived with respect to the properties to be checked [36] that is a key element of the present document.

The relationship between models and metamodels can be explained with the four-level architecture [22] depicted in Figure 3.1, as detailed below:

- *Data level (M0)*: it represents real-world data that conforms to a given model. For example a movie, in this case the 1942 film Casablanca directed by Michael Curtiz.
- *Model level (M1)*: it characterises models describing M0-level data. Every model is an instance of a metamodel. In the given example, the model depicts the concept of Video with an attribute of the type string that describes the title of the movie.
- *Metamodel level (M2)*: it represents metamodels describing M1-level models. Every metamodel is an instance of a meta–metamodel. The metamodel describes the concepts used at M1 for defining the model, such us: Class, Attribute and Instance.

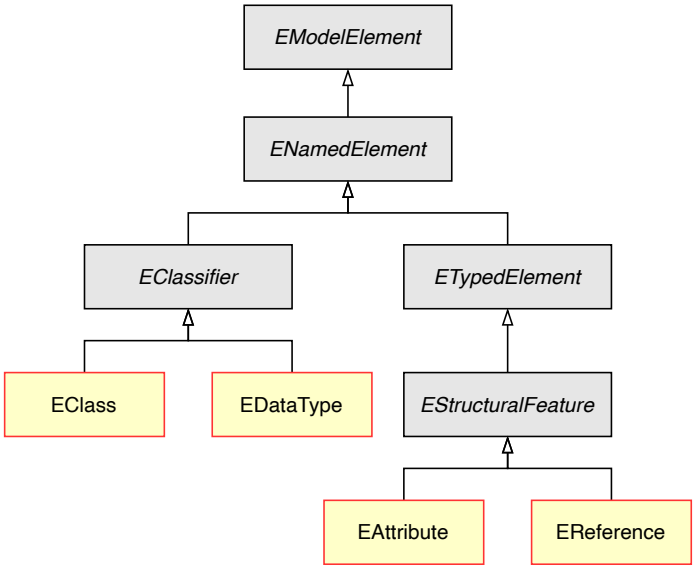


Figure 3.2: Simplified Diagram of the Ecore metamodeling Language [145].

- *Meta-metamodel level (M3)*: it characterises meta-metamodels describing M2-level metamodels. A meta–metamodel is an instance of itself.

One of the most popular and widely-used frameworks that facilitate the creation of metamodels is The Eclipse Modeling Framework (EMF) [1] which is used in this thesis. In EMF, metamodels are defined using the Ecore metamodeling language. A simplified overview of Ecore concepts is illustrated in the class diagram of Figure 3.2 with four main Ecore classes [145, 7].

- *EClass* is used to represent a modeled object including name, attributes and references.
- *EAttribute* is used to represent a modeled attribute. They have a name and a type.
- *EReference* is used to depict one end of an association between classes. It has a name, a Boolean flag to indicate if it represents containment (i.e. ownership relation), and a target type, which is another class.
- *EDataType* is used to represent the type of an attribute. It can be primitive (e.g., int) or an object type (e.g., java.util.Date)

EMF offers a robust framework for MDE that streamlines the modeling process and improves the quality of software development [145]. By providing a standardised way of

[1] `https://www.eclipse.org/modeling/emf/`

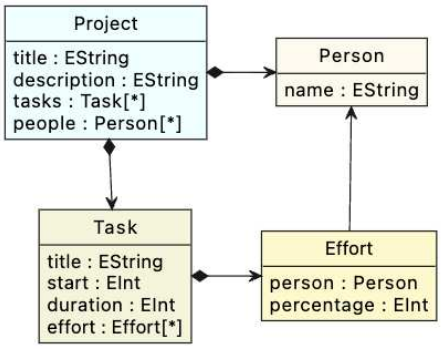


Figure 3.3: Metamodel describing a Project and its features.

defining models using a metamodel, EMF enhances reusability and maintainability of models across different projects. Furthermore, EMF facilities simplify code generation and ease interoperability with other frameworks and tools, which promotes collaboration and sharing of models across teams. Additionally, EMF allows for automated testing of models using automated tools, which improves the quality and consistency of models. Lastly, EMF and its variants support large-scale modeling, allowing complex models to be easily created and managed. Overall, EMF provides a robust framework for MDE that streamlines the modeling process and enhances the quality of software development [145].

The Epsilon Object Language (EOL) [85] is a programming language that plays an important role in MDE [85]. EOL facilities the navigation and modification of models. EOL can be use to manage models from diverse technologies such as XML, and the previous mentioned EMF. This metamodel-independent language is inspired by the Object Constraint Language (OCL) first developed by IBM in 1995. OCL enables expressing all kinds of (meta)-model querying, manipulation and specification requirements [28]. However, OCL has its limitations as a general-purpose language for different model management tasks [85]. To overcome these constraints, EOL proposes a syntax and set of constructs that allow the creation and manipulation of (meta)models in a technology-agnostic fashion. Similar to EMF, in EOL, developers establish the design of their models by means of a metamodel. This metamodel outlines the types of components that can be present in a model and the connections between them. Subsequently, models are constructed using instances of these components specified in the metamodel and can be altered with the help of operations supplied by the EOL libraries and frameworks [85].

EOL programs are organised in *modules*: each one contains a *body* and a number of

operations. When the module is executed, the body block is evaluated. Operations have a kind of objects on which they are applicable (i.e. a context), a name, a set of parameters and optionally a return type. Based on the Epsilon Playground tool proposed by Kolovos and Garcia-Dominguez in [84], an example for the use of EOL is presented next. Consider the metamodel of Figure 3.3 describing a `Project`, its `Task`, and the `Person` objects associated to the `Task`. Then, the EOL program depicted in Listing 3.1 can be used for model navigation. When the body is evaluated, the operation `getTotalEffort()` of the type `Task` can be accessed. This EOL query will print, for every `Task` in the model, its title and the total person-months of the `Task`. Additionally, it will print the count of tasks that are undertaken by a single `Person`.

EOL enables developers to create and modify EMF models, query model contents, and perform model transformations. EOL provides numerous features that facilitate working with EMF models, including a rich syntax for expressing complex queries and transformations [84]. Additionally, EOL has a range of built-in functions and operations that can be utilised to manipulate models, such as selecting, creating, and modifying model elements. In essence, EOL is a programming language purpose-built for EMF models that offers powerful features for manipulating and transforming models [85]. In this PhD work, EOL has been extended with temporal assertions that allow the exploration of model transversion over time. More details will be presented in Section 5.

Listing 3.1: EOL program example

```
for (t in Task.all) {
        (t.title + ": " + t.getTotalEffort()).println();
}
Task.all.select(t|t.effort.size() = 1).size().
        println("One-person tasks: ");
operation Task getTotalEffort() {
        return self.effort.
                collect(e|self.duration*e.percentage/100.0).sum();
}
```

Overall, in Software Engineering, the uses of models and their associated tools have been extensive. They vary from models as description of the system's domain, to models for doc-

umentation, to models as specification for testing, among many others [89]. MDE combines process and analysis with architecture bringing in model abstractions at the various phases of the software life cycle. In this context, models can be broadly classified in *development models* and *runtime models* [56] which will be described further in the following sections.

3.1.1 Development Models

Model-driven approaches are improving the way we build software. They increase developer productivity, decrease the cost (in time and money) of software construction, improve software reusability, and make software more maintainable [94]. In the context of software development, models provide an abstract representation of a software system or a part of it. In the software development process, models are used for documentation and communication purposes in analysis, design, and implementation activities. MDE further increases the importance of models, as in MDE models are not only used for documentation and communication, but as central artefacts of the software development process [22].

MDE researchers have largely applied models to selected elements of the development process, particularly structural and compositional aspects in the design phase and model checking and verification in the testing phase [142]. Development models are models of software at levels of abstraction above the code level. MDE researchers and practitioners focus on how modeling techniques can be used to tackle the complexity of bridging the gap between the problem domain and the software development [56]. Examples of development models are requirements, architectural, implementation and deployment models [135].

Eliciting, defining and agreeing on the requirements for a system demand considerable effort, involving different stakeholders and a large amount of information [141]. At the requirements level, MDE techniques can be used to provide a common framework to integrate requirements capturing activities (i.e., eliciting, defining, modelling, agreeing, communicating and validating) from all necessary perspectives [10]. For example, model transformations may ensure consistency between different kinds of requirements models such as goal, scenario, and domain models. Furthermore, MDE approaches may help to automatically construct architectural models from requirements (e.g., by deriving a more detailed UML model from a goal or scenario model), allowing for a tighter integration of requirements models and architectural/design models [69].

In the architectural space, MDE can empower the specification of the parts and connec-

tors of the system and the rules for the interactions of the parts using the connectors [22]. Architectural models include interface, interconnections, and specifications for components within a system but not their implementation. They describe the interactions between components and their arrangement in the system, but the components themselves are black boxes. The component implementations can be described separately by implementation and or deployment models [38]. The implementation issue deals with the mapping of the models to some existing or future running systems. It consists of defining three core aspects; i) models, ii) transformations or code and, iii) artifacts [22, 145].

All these aspects presented form part of Model-Driven Development (MDD), that is a development paradigm that uses models as the primary artifact of the development process [22]. In the present document, the focus of interest of models goes beyond the development paradigm and targets the analysis of the runtime behaviour of software systems. Further details about models in the context of runtime software behaviour, will be described in the following section.

3.1.2 Runtime Models

As previously discussed, in MDE, models are used as central artefacts of software development. Recent studies take the idea of using models as central artefacts one step further by using models during execution, known as *runtime models*, to cope with the dynamic aspects of ever-changing software and its environment [56]. The basic underlying motivation for runtime models is the need for managing the complexity that arises from the large amounts of information that can be associated with runtime phenomena [159]. Bencomo et al., described a runtime model as an abstract causally-connected representation of a running system. It allows monitoring and controlling the system as well as reasoning about it [14]. It can also serve as a knowledge base for SAS. The approach followed in the present thesis depends on runtime architectural models that allow access to monitoring results. This is a typical use case of runtime models as mentioned in [35].

Given information about the running system, the runtime models allow the system to reason about its state and environment, and take corrective actions at a higher level of abstraction [14]. Furthermore, performing changes at the model level of those models at runtime improves the synchronisation between design artefacts and the implementation of the software system [149]. Different problems can be addressed by runtime models [15,

149]. For example, inaccurate predictions result from unknown requirements evolution (e.g., requirement change produced by the user)and associated impacts. In a changing environment, it is no easy task to monitor and visualise the impact of adaptation rules which are applied when a system should meet new requirements. Runtime models help to visualise such impacts by analysing the affected software parts at the model level and checking whether specified application constraints have been violated [157]. Another example of a problem addressed by runtime models is checking rules or constraints. The previous mentioned means that the model contains or implies rules or constraints, such as consistency requirements with the running system or other artefacts that should not get violated [149].

The purposes and objectives pursued when using runtime models are varied. From the literature [15, 149], they can be classified as follows:

- *Abstraction*: refers to the use of models to provide a higher level of abstraction, regardless of whether used at design time or runtime [14]. The aim is to tackle the problems of level abstraction, maintenance and reusability resulting from hand-written artefacts and manual human interventions [149].

- *Adaptation*: denotes the application of runtime models to build, operate or manage SAS [15]. One main objective of using runtime models is to ease adaptation in environments with continuously changing requirements [149]. Common scenarios addressed by adaptation through runtime models are: user interface adaptation [62], requirements changes [11], contextual changes [54], QoS enforcement [124], and change impact analysis [157]. Further details about runtime models, and in general MDE, for SAS will be described in section 3.3.

- *Error handling*: refers to the application of runtime models to increase the fault tolerance of systems [15]. Runtime models provide system operators with enhanced capabilities to localise faults in behavioural models like workflows [112]. Beyond that, with runtime models faults can not only be localised, but also eliminated by architecture-based self-repair features [2].

- *Interoperability*: denotes the application of runtime models to bridge architectural mismatches between individual systems [15]. In this context, runtime models can be used to capture meta-information about networked systems that need to cooperate, including their interfaces and additional knowledge about their associated behaviour [13].

- *Monitoring, simulation and prediction*: refers to monitoring the system by using models which help to trace application behaviour, to simulate changes at the model level to analyse their consequences, and to predict system properties like performance by analysis at the model level [149, 34].

- *Consistency and conformance checking*: denotes the application of runtime models to assure that there are no contradictions between the different parts of a software system and/or software artefacts related to the system [149]. Conformance ensures that a software system meets a specified standard [149].

- *Policy checking and enforcement*: encompasses models used at runtime to cope with policies, such as real-time constraints, access control and security regulations, or other compliance rules [149].

Similar to the purposes for using runtime models, the types of such models are also diverse [149]. Bencomo et al., in [15], differentiated the following:

- *Structural runtime models*: refer to models capturing the system elements and their state at runtime.

- *Behavioural runtime models*: refer to models describing the runtime dynamics of the systems, i.e., what the system can or will do based on its current state.

- *Quality runtime models*: stand for models capturing the current values of quality properties (i.e., non-functional properties) of a system or its elements.

- *Goal runtime models*: denote models describing the current state of the system's goals (e.g., if they are currently fulfilled or violated).

- *Variability runtime models*: refer to models representing possible variants of the system or it's elements and which variant is currently in use.

- *Design runtime models*: refer to design-time decisions, which are continuously synchronised with an evolving running system (used, e.g, for eternal system approaches).

- *Requirements runtime models*: denote models representing the current set of requirements a system has to meet.

- *Runtime models for feedback loops*: refer to models representing one or more feedback loops, their connections and current state (i.e., are a special type of behavioural model).

- *Physical runtime models*: refer to models capturing the dynamics and current state of physical (i.e., continuous) phenomena (e.g., in Simulink.)

- *Runtime metamodels*: describe runtime models that are used to specify how the system or its environment are modelled.

The present PhD work focuses on runtime models with the purpose of monitoring SAS. Furthermore, the proposed approach aims to exploit runtime models' capabilities to reason and convey knowledge about the behaviour of a SAS in the form of explanations. The types of runtime models used in the proposed approach are behavioural, goals and requirement runtime models to analyse how well does the system fulfil its requirements over time towards a targeted goal. The proposed approach will be further described in Chapters 5, 6 and 7.

3.2 The Concept of Time in MDE

As mentioned in the first chapter of the present document (Chapter 1), the proposed approach exploits the importance of a SAS history to transfer knowledge and to provide explanations. By using MDE, specifically runtime models, the history can potentially be represented on a causally connected abstraction of the system's behaviour. However, before further describing the approach, it is important to understand the concept of time and, therefore, history in MDE.

There are several ways to consider time in MDE. One way is to include it in the concepts of the modelling language itself. For instance, the OMG MARTE (Modeling and Analysis of Real-Time and Embedded Systems) profile [113] includes a framework for representing time, supporting several abstractions: a causal one (modelling event precedence), a clock-based one (dividing time into "instants" in which several processes may run simultaneously), or a physical one (with real-world duration values). Different from [113], the concern of this work is more about the evolution of the models themselves over time, along with the modifications made by either human modellers, by an automated process (e.g. the operation or monitoring process of a system) or by the running system itself. As an example of early

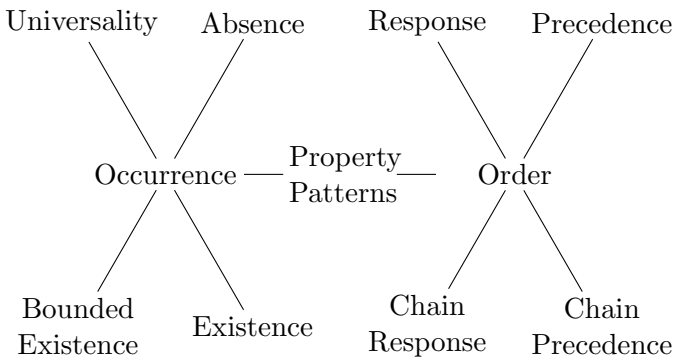


Figure 3.4: Hierarchy of property patterns by Dwyer [49].

work, Ziemann and Gogolla proposed in 2002 TOCL (Temporal OCL), an extension of the OCL for describing temporal constraints on an object-oriented model [166]. Their proposal extended OCL with a number of *past expressions* (e.g. "prev e" is true if e is true in the "previous state") and *future expressions* (e.g. "always e_1 until e_2" is true if e_1 is true "from then on" until e_2 is true "for the first time in the future"). While the proposal included formal semantics and a type system, it did not include any suggestions on how to implement this language.

In 2014, later work by Kanso and Taha [78] stated that existing research on temporal OCL extensions at the time only brought syntactic proposals without any concrete implementation, and required knowledge of specific temporal logics. The work proposed a temporal extension of OCL based on the work of Dwyer, Avrunin and Corbett on specification patterns for finite-state verification [49] to make temporal logics more accessible. Dwyer et al. organised the patterns into a hierarchy (as shown in Figure 3.4). They also added a mapping from each pattern to various formalisms (including linear temporal logic). Dwyer et al. collected 555 specifications from at least 35 different sources, and identified the described patterns to cover 92% of them; with Response, Universality and Absence being the most common out of all three. In terms of scopes, Dwyer et al. defined five, which are shown in Figure 3.5: "global" (the entire execution of the program), "before" a certain event, "after" an event, "between" two events, and "after" a certain event "until" another event.

In the work by Kanso and Taha cited above, events are of two types: either a certain operation *op* has been invoked in a context where *pre* is a pre-condition and *post* is a post-condition ("isCalled(*op*, *pre*, *post*)"), or a certain predicate has become true after some

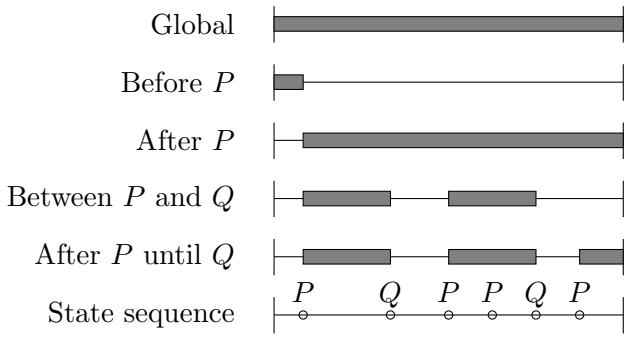


Figure 3.5: Pattern scopes by Dwyer [49].

invocation ("becomesTrue(P)", a shorthand for "isCalled(any,$\neg P$,P)"). The notation was implemented as an extension of the Eclipse MDT OCL plugin [2], with a new grammar, new editors, and new pivot metamodels. A transformation from OCL temporal properties to test scenarios for object-oriented software was implemented. The work by Kanso and Taha focused on generating test cases, and did not consider the use of these extensions for querying the histories of the models themselves [78].

In 2016, Hilken and Gogolla presented a mapping of several Linear Temporal Logic primitives to standard OCL, using so-called *filmstrip models* [72]. These models conform to an enriched version of their original metamodels, which consists of a trace of model states that are linked by operation calls. This enriched structure allows for implementation of LTL operators such as "finally" or "globally" with regular OCL closures and logic quantifiers. Unfortunately, in their work the authors did not address how the filmstrip-based temporal constraints would scale to longer traces, neither in terms of both memory requirements nor processing time.

Later in 2017, Dou, Bianculli and Briand presented TemPsy [47], a pattern-based specification language inspired as well by the primtives of Dwyer et al. The TemPsy patterns were translated to OCL constraints on the fly, as requested by their industrial pattern. The constraints ran on models which conformed to dedicated trace metamodels. The authors compared their TemPsy-Check tool against MonPoly, an existing tool based on MFOTL (metric first-order temporal logic), using a case study from the eGovernment domain. TemPsy-Check had similar or better performance, while having a notation that was easier to use than MFOTL.

Beyond temporal extensions to constraint languages, Benellalam et al. argued for the

[2]Eclipse Model Development Tools (MDT) `https://www.eclipse.org/modeling/mdt/?project=ocl`

need to raise time-awareness in model-driven engineering in 2017 [16], identifying gaps in the capabilities of conceptual modelling approaches, query languages and storage solutions. Their work used an example based on the smart grids domain, where a *SmartMeter* meta-class is annotated with time sensitivity, periodicity and precision concerns. The authors suggest that time should be a first-class property orthogonal to all model elements, allowing each element to evolve independently over time (e.g. more time-sensitive concepts would store more versions).

Mouline et al. presented a metamodel for interactive diagnosis of adaptive systems which combines design-time and run-time concerns [110]. The design-time parts cover the available strategies and actions, whereas the run-time parts cover the observations made and the decisions that were taken. The authors propose allowing users to use temporal queries to find out why a specific action was taken, helping with the self-explainability of the system.

Meyers et al. described ProMoBox in [104], a framework for building behavioural domain-specific modeling languages with the ability to define and verify temporal properties. A DSML description is turned into five DSMLs: design-time concerns, run-time concerns, event inputs, traces, and a Dwyer-inspired temporal property language. Properties are translated into Promela models for the Spin model checker.

This thesis examines the trade-offs imposed by adding time-awareness to a model-based approach. Specifically, the present work focus on the use of runtime models and the timeline of the running system for the explanation generation and explanation communication about a SAS's behaviour over time. Further details regarding the proposed approach will be discussed in Chapters 5, 6 and 7.

3.3 The application of MDE in SAS

Model definition and transformations between models and from models to code are key factors in developing automatic systems. The use of MDE techniques in self-adaptive autonomous software systems has mainly focused on the use of runtime models to support adaptation management, run-time analyses and decision processes [35]. The goals of runtime models in SAS are to: depict abstractions of runtime phenomena, automate runtime adaptation, and analyse the running system and its domain [156]. The predominant application of runtime models in SAS is the use of structural models to support self-adaptation [15].

A structural runtime model upholds self-adaptation with a high-level, holistic view of the running system, in such a way that the self-adaptation engines can use the model to analyse the runtime phenomenon and enact the system directly [15]. For example, Fouquet et al. in [55] proposed μ-Kevoree, a dynamic component model which relies on runtime models to support dynamic adaptation of distributed resource-constrained devices such as microcontrollers. The authors use the concepts of *Component, Node, Channel* and *Group* to model the infrastructure and communications semantics of a running SAS. This modeling layer enables efficient and safe reasoning for adapting microcontroller-based nodes [55].

On the other hand, goal runtime models describe the relationships between a system and its environmental context. With contextual information, goal models allow the system to assess candidate solutions against high-level criteria towards an objective [149]. For example, the work proposed in [158] where the authors propose the use of goal runtime models for the realisation of requirements-aware systems by checking assumptions made at design time. When the assumptions are not longer valid, system wide adaptations are triggered to enable alternative goal realisation strategies [158]. Silva Souza et al. in [144] proposed requirements runtime models which are characterised syntactically as requirements that refer to other requirements or domain assumptions and their success or failure at runtime. Awareness Requirements (AwReqs) are represented in a formal language and can be directly monitored by a requirements monitoring framework. Moreover, they propose a graphical representation that allows to define the AwReqs in goal models to enact the communication among system developers and users [144].

A prominent work that exploits the benefits of runtime models for SAS is the EUREMA (ExecUtable RuntimE MegAmodels) project by Vogel and Giese [156]. The authors propose the use of different types of runtime models, their integration and synchronisation to support the execution of adaptation engines (i.e., managing systems) and feedback loops in SAS. In order to systematically describe and automate the coaction between runtime models and adaptation activities, runtime megamodels (i.e., models constituted of other models) are defined [156, 15]. EUREMA includes a domain-specific modeling language and a runtime interpreter for managing systems using feedback loop concepts. Figure 3.6 depicts the runtime models considered in the approach based on the MAPE-K feedback loop described in Section 2.1. *Reflection models* correspond to the knowledge component in the MAPE-K loop and reflect the managed system and its environment. *Monitoring models* denote the

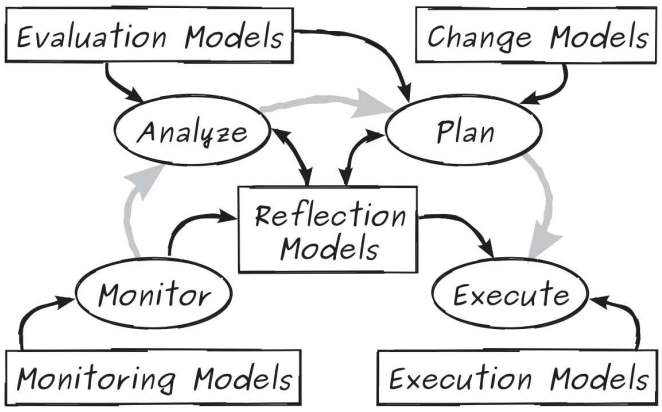


Figure 3.6: EUREMA - Runtime models for feedback loops [156].

mapping of system-level observations to the reflection models. *Evaluation models* support the analysis of the reflection models to diagnose adaptation needs. *Change models* are related to the planning stage depicting the SAS variability space (i.e. reconfiguration or adaptation space). Finally, *execution models* refine model-level adaptation to system-level adaptation [156].

Apart from the use of runtime models, MDE have been also used to develop SAS [134]. For example, Burmester et al. in [26] presented Mechatronic UML a model-driven development approach for reconfigurable mechatronic systems. The authors extended UML to specify and generate a hierarchical scheme that addresses control, reconfiguration, and planning by distinct feedback loops at different layers. However, the adaptation is defined before deployment and cannot be dynamically changed [26, 156]. Sansores and Pavon proposed an adaptive agent model for self-organising multi-agent systems (MAS) using an extended version of the INGENIAS methodology [139]. INGENIAS is a methodology for the development of MAS. Its development tools rely on its MAS modeling language, which is specified with a meta-modeling language, MOF (Meta-Object Facility), a standard by OMG [113, 139]. A similar work was developed by Rougmaille et al., in [134]. The authors presented an model-driven engineering approach to improve the development process and the quality of the software. The goal is to reduce the duration and the complexity of the designing of adaptive MAS [134].

The use of MDE techniques in SAS have increased with the evolution and adoption of this area of software engineering [25, 20]. As described in this section, both development and runtime models have been utilised for the engineering and management of SAS includ-

ing applications at design time and runtime. However, only little research has been done regarding to the exploitation of the benefits of MDE for explainability and transparency in SAS, important in the context of trustworthiness and the understandability of these systems. The approach presented in this thesis aims to address these aspects using runtime models for post-hoc explainability in SAS. For doing so, runtime models are used to trace and monitor the system's, thus to perform run-time monitoring (RTM). RTM will be further described in the following section.

Chapter 4

Runtime Monitoring

The full behaviour of complex software systems often only emerges during operation. They thus need to be monitored at runtime to check that they adhere to their requirements [128]. In order to explain AI "black boxes", their behaviour should be observed [1]. Run-time monitoring (RTM) is a common approach to gain insights about a running system [81]. This chapter discusses the role of RTM in SAS and describes the RTM approaches relevant for the comprehension of this thesis (i.e., Event-driven Monitoring).

4.1 What is Runtime Monitoring (RTM)?

RTM is a lightweight and dynamic verification technique that involves observing the internal operations of a software system and/or its interactions with other external entities, with the aim of determining whether the system satisfies or violates a correctness specification [33]. RTM has been extensively studied in different areas of computer science, such as distributed systems, requirements engineering, programming languages, and aspect oriented development among others [5]. In SAS, RTM can be related to: i) how the managing system correctly monitors the managed system and the environment to guarantee that their assurances and assumptions are accurate [30] or, ii) run-time V&V tasks over the entire system to ensure that desired properties, goals and requirements are fulfilled [43]. This thesis focuses on the latter, on RTM of SAS to provide explanations about the system's behaviour regarding its goals and requirements.

Existing RTM approaches are very diverse. Some present monitoring tools to support end-users, while others demand expert domain knowledge by their users (e.g., [132] and

[155]) ; a number focus on specific architectural components, while others focus on general purposes (e.g., [6] and [132]); some automatically generate monitors based on models, while others require that probes to be manually developed (e.g., [132] and [153]) [128]. Despite its diversity, RTM approaches can be categorised into two main classes, *online* and *offline* monitoring [33]. In offline RTM, the system is not directly monitored at runtime. Instead, events of interest are recorded as execution traces inside a data store for further analysis after the system completed its executions. On the other hand, in online RTM, the executing system is dynamically monitored for violations or satisfactions, for example on the Quality of service (QoS) attributes, during the course of its execution [33]. The present work proposes a flexible approach that is for either class of RTM, where explanations from a monitored SAS can be obtained during its execution or after-the-fact.

Most RTM frameworks employ compilation techniques that synthesise monitors from high-level specifications (i.e. properties), expressed in terms of a formal logic, which are executed in tandem with the monitored system [33]. The approaches for performing RTM can be either time-driven or event-driven [81, 44, 128]. Time-driven RTM, also known as sampling, only allows statistical statements about the program behaviour [81]. Event-driven RTM checks specific sequences of events that have to occur, the presence and absence of a specific event, or data/performance properties [128]. In the present document, an event-driven approach is used and is further described in the following section.

4.2 Event-driven Monitoring

During the past two decades, event-driven programming (EDP) has emerged as a central and almost ubiquitous concept in modern software development [99]. In RTM, event-driven monitoring is a common approach to gain insights about a system based on particular situations of interest [81]. An event is defined as a detectable condition that can trigger a notification. In this context, a notification is an event-triggered signal sent to a runtime-defined recipient [51]. Event-driven monitoring focuses on detecting the occurrences of predefined events on one or multiple incoming data streams, in order to notify interested stakeholders and/or run some palliative processes [45]. Event-driven monitoring approaches are commonly designed to listen for system events and handle them in the background without interfering in any way with the system's execution [109].

4.2.1 Current event-driven monitoring approaches

There are several proposals for event-driven monitoring in the literature [45]. These include monitoring QoS attributes of the services, resource utilisation, as well as business-related information relevant for Key Performance Indicator (KPI) progression or Service Level Agreement (SLA) fulfilment [109]. For instance, Konno et al.'s work [86] uses a rule inference method that integrates dynamic case-based reasoning and root cause analysis. It allows for autonomous recovery and failure prevention to guarantee long-term QoS of cloud systems. Another event-driven monitoring approach integrates Wireless Sensor Networks (WSNs) with sentinel nodes [52] to detect heavy road vehicles as well as raising alarms in monitoring nodes. In [37], Cicotti et al. presented a cloud-based platform-as-a-service based on event-driven monitoring and cloud computing. SLAs can be analysed by collecting KPIs and defining event patterns. When KPIs exceed certain thresholds, the violation condition is prevented.

However, some challenges arise when using an event-driven approach. Klar et al., defined the essential questions to be asked when using such approaches [81]:

- *What* is the aim of the measurement?
- *Which* events are necessary for describing the functional behaviour?
- *How* should the events be defined?

Answering these questions requires; i) knowledge about the aim of the measurement and ii) precise formal knowledge about the functional behaviour of the object program at an adequate level of abstraction [81]. Moreover, Moser et al., in [109] stated four key requirements for event-driven monitoring:

- *Platform agnostic and unobtrusive*: the monitoring system should be independent of any concrete monitored system while being unnoticeable for the monitored system.
- *Integration with other systems*: the monitoring system should be capable of integrating monitoring data from other subsystems. This enables a holistic view of all monitoring data in a system.
- *Multi-process monitoring*: the monitoring system should enable monitoring across multiple services, and instances.

- *Detecting anomalies*: The monitoring system should be capable of unveiling potential anomalies in the monitored system.

Complex Event Processing (CEP) is an event-driven monitoring approach that has been widely used to address these requirements [109, 37, 21]. CEP is a core technology used in the present thesis and is further discussed in the following subsection.

4.2.2 Complex Event Processing (CEP)

CEP [98] is a technology that can capture, analyse and correlate large amounts of data in real time in a domain-agnostic way. CEP is used to identify complex meaningful circumstances and to respond to them as quickly as possible [21]. The main objective is to detect situations of interest in a specific domain or scenario [163]. These situations of interest are detected through a set of *event patterns* that specify the conditions that incoming events to the system must fulfil. An incoming event can be *simple* (something that happens in the system at a point in time) or *complex* (patterns of two or more events that happen over a period of time). Any detected complex events can be fed back to the CEP system for further matching, which creates a hierarchy of complex event types [74]. The defined patterns must be deployed to a CEP engine, i.e. the software responsible for analysing and correlating the data streams received from different sources, as well as for raising alerts to users or systems interested in complex events generated by the detected event patterns [21]. Each CEP engine provides its own Event Processing Language (EPL) for defining the patterns to be deployed.

As depicted in Figure 4.1, CEP is conformed by three main components [98]:

- *Event Sources*: which send events in the form of data streams (e.g. sensor updates) or message queues (e.g. business events).

- *CEP Engine*: the event processor, which performs various actions such as adding timestamps, adding information or metadata and processing multiple information at once. It is conformed by a rule engine where event patterns are defined, and an event handler where a match of the defined patterns generates a complex event. Some examples of CEP engines are Apache Flink[1], IBM InfoSphere Streams[2], and Esper[3].

[1] https://flink.apache.org/
[2] hhttps://www.ibm.com/docs/en/streams/4.1.0?topic=welcome-introduction-infosphere-streams
[3] https://www.espertech.com/

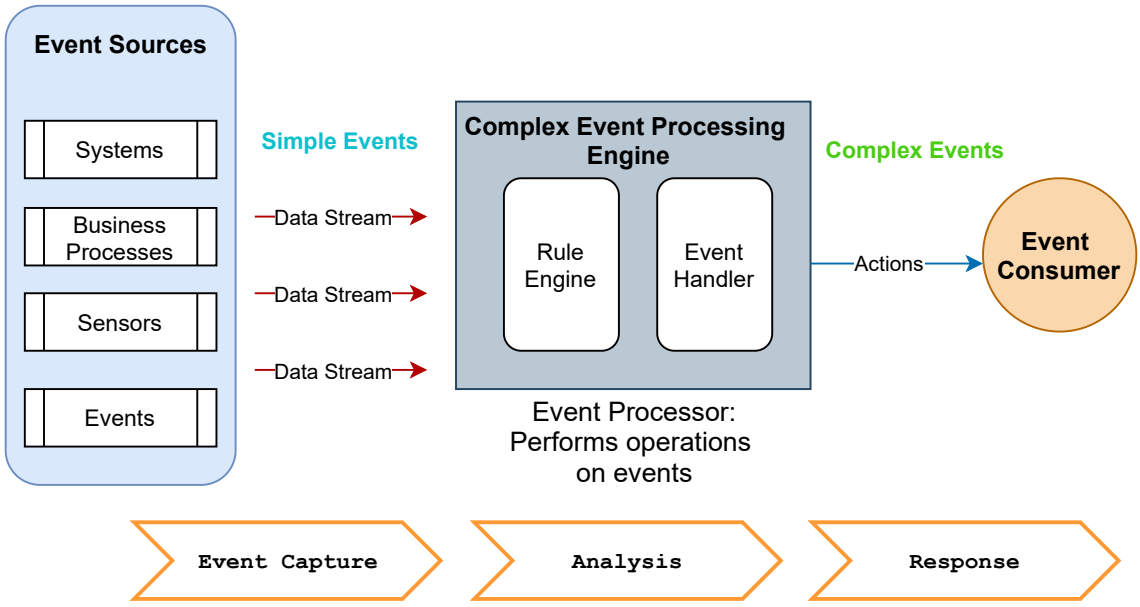


Figure 4.1: Components and stages of Complex Event Processing [21, 98]

Apache Flink is an open-source stream processing framework that supports both batch and stream processing and provides a high-performance, distributed processing engine for analysing data in real-time. IBM InfoSphere Streams is a CEP engine that is specifically designed to handle real-time analytics for processing high-speed data streams at rates of up to millions of events per second. Esper, another open-source CEP engine, is capable of processing and analysing high-volume event streams in real-time and offers a range of features for complex event processing, pattern matching, aggregation, and correlation. Each engine has its unique features and capabilities, and choosing the right one for a specific use case depends on several factors, including the complexity of the data, the volume of events, and the performance requirements.

- *Event Consumers*: which perform a certain action upon being notified of a complex event. For instance; updating a database, interacting with external services, starting/closing new applications or services.

Also from Figure 4.1, it can be noted that CEP is performed in three stages [21]:

- *Event capture*: this stage focuses on the definition of the *simple events* to be analysed. Simple events are occurrences of particular low-level patterns of something that happened at a point in time [98]. These low-level occurrences can then be processed and analysed to produce higher-level occurrences/events [21]. For example, the occurrence

of a water drop (low-level) plus the occurrence of the sound of a thunder (low-level) can suggest that a storm (high-level) is approaching.

- *Analysis*: this is stage where operations over events are performed. The CEP engine will use the defined event patterns to correlate the arriving events and detect situations of interest in real time.

- *Response*: after the detection of a concrete situation of interest in the analysis stage, there will be a notification to an event consumer that can trigger an action in response.

The present thesis proposes the use of CEP as technology to detect temporal and causal dependencies between events, in order to gain insights from events as they occur during execution (Chapter 7).

4.3 MDE and Event-driven Monitoring

Now that event-driven monitoring and CEP have been introduced, the combination of these approaches with MDE will be explored. Putting in place a working event-driven monitoring process involves several activities, including the collection of raw observation data, the interpretation of such raw information to recognise higher-level events that are relevant at the business level, and the effective presentation of the monitoring results [17]. Consequently, in order to answer the *what, which and how* questions mentioned in Section 4.2.1, event-driven monitoring approaches require domain experts to define the activities indicating such situations of interest and the appropriate actions to be executed in their information systems [21]. Unfortunately, the outcome of such effort is time-consuming, hard to generalise and to reuse, and, as a matter of fact, the resulting monitoring configuration typically is only relevant in the specific situation at hand [17]. In order to address some of these challenges, MDE has been used in the context of event-driven monitoring [81, 21, 17, 82, 4].

The key MDE features exploited in event-driven monitoring are; i) the use of models of different levels of abstraction for graphical abstract representation of a system and ii) model transformation and model to code features [21]. For example, Klar et al. in [81] presented a concept for model-driven execution of event-driven monitoring. The authors propose the use of MDE tools to represent the instrumentation needed to perform RTM on a system. The monitoring model can be used as a guideline for dynamic visualisation of the

system behaviour (execution animation). In a more recent work, Bertolino et al., proposed a property-driven approach to event-driven monitoring that is based on a comprehensive Property Meta-Model (PMM) [17]. PMM allows the definition of prescriptive/descriptive and qualitative/quantitative properties, and the metrics needed to quantify them. PMM is combined with GLIMPSE, a CEP based engine for model-driven monitoring based on the occurrence of events. In [82], the authors propose aPro, a modular architecture for business process optimisation that uses CEP as a monitoring tool. This work's main contribution is to provide a model-driven approach for monitoring business processes. They propose a modeling language (ProGoalML) for process metrics, KPIs and goals, which automates the creation and setup of the monitoring infrastructure including the CEP engine setup. Similarly, Boubeta-Puig et al., in [21], propose the use of MDE to facilitate user-friendly design of complex event patterns for the CEP engine setup. The authors introduce MEdit4CEP, a model-driven approach for CEP together with graphical model editors for defining the CEP domain, event patterns and actions, and code generators. Figure 4.2 shows an screenshot of the editor that allows the definition of the situation of interest to be detected as well as rules, logic and actions to be performed by an event pattern. This event pattern will be automatically validated, transformed into code and deployed into a CEP engine.

In the present thesis, MDE is combined with event-driven monitoring for gaining insights about SAS behaviour. The approach is used to present explanations at runtime using Temporal Models and CEP. Further details will be described in Chapter 7.

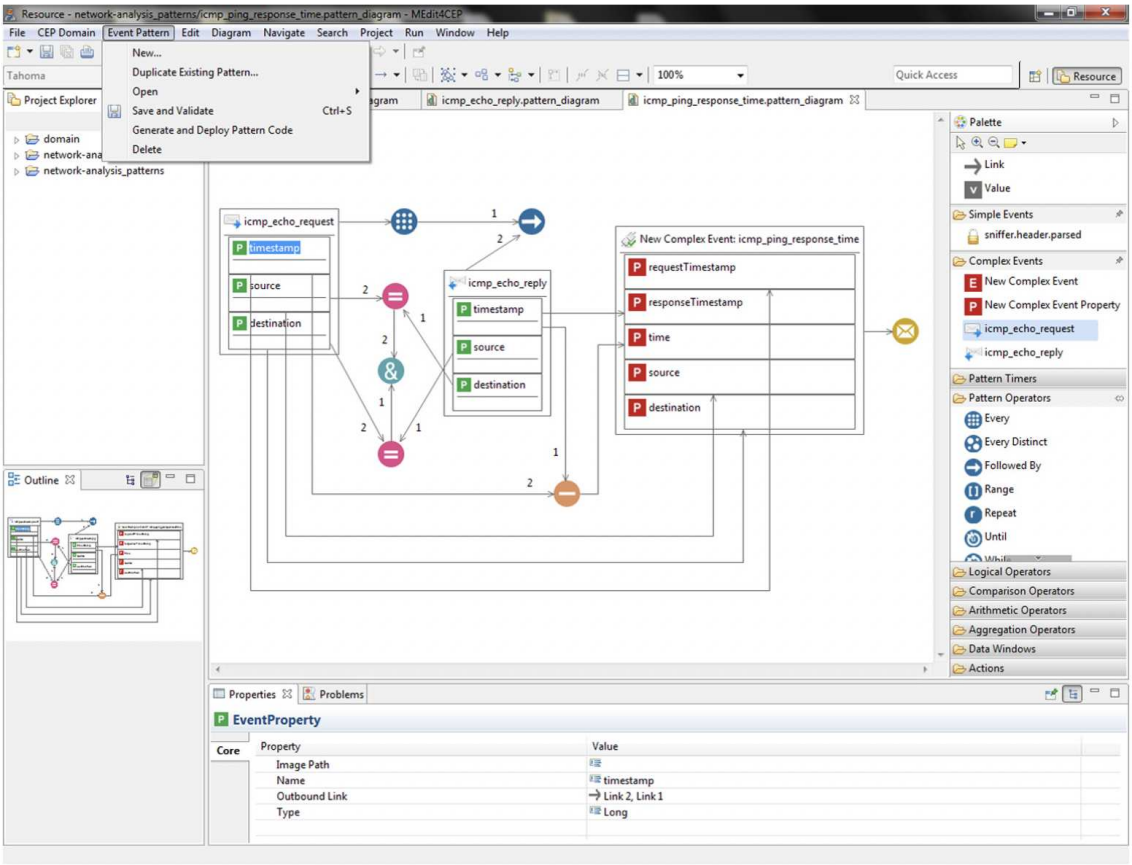


Figure 4.2: Screenshot of the model-driven GUI for CEP, MEdit4CEP [21].

Chapter 5

Temporal Models for History-aware Explainability in SAS

The work presented in this chapter has been adapted from the following publications:

[119] J. M. Parra-Ullauri, A. García-Domínguez, L. H. García-Paucar, and N. Bencomo. Temporal models for history-aware explainability. In *Proceedings of the 12th System Analysis and Modelling Conference*, pages 155–164, 2020.

[59] A. García-Domínguez, N. Bencomo, J. M. Parra-Ullauri, and L. H. García-Paucar. Towards history-aware self-adaptation with explanation capabilities. In *2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS* W)*, pages 18–23. IEEE, 2019.

[60] A. García-Domínguez, N. Bencomo, J. M. Parra-Ullauri, and L. H. García-Paucar. Querying and annotating model histories with time-aware patterns. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 194–204. IEEE, 2019.

This thesis proposes that ideally, a SAS should be able to access its decision-making history and adjust future adaptations taking into account past results of previous adaptations. These so-called reflective and history-aware capabilities can be exploited for the system itself to perform better-informed decisions based on past behaviours, and this information

can be conveyed to different stakeholders to analyse SAS historical behaviour. However, adding these capabilities into a SAS can be costly and hard to evaluate. In the context of this document and from the challenges identified in Section 1.1, these trade-offs are related to how to store, access and extract historical information, and how to communicate the extracted information to different stakeholders in the form of explanations to promote understandability and trustworthiness in SAS. This chapter presents the proposed approach using Temporal Models to tackle these challenges and the research roadmap that guided the research in this thesis.

5.1 Temporal Models

This PhD study delves into the concept of history-awareness, which refers to a SAS's ability to use past behaviour to inform decisions or predictions about future behaviour. The study also discusses how enabling a SAS's reflective capabilities can lead to a better understanding of its past behaviour. By examining what the system did in a specific context, one can more coherently explain the SAS's decision-making process. To achieve this history-awareness for explainability, a SAS needs to be equipped with appropriate tools to track its decision-making, store this information, extract knowledge from the stored data, and communicate this knowledge to various stakeholders. Various methods, such as data mining, knowledge distillation, and MDE, can be employed to enable these capabilities. The thesis proposes an MDE approach for achieving history-awareness.

As reviewed in previous chapters, in MDE, models are used as central artefacts of software development [56]. Further, runtime models are used to cope with the dynamic aspects of ever-changing software and its environment [14]. The basic underlying motivation for runtime models is the need of managing the complexity that arises from the large amounts of information that can be associated with runtime phenomena [56]. In this context, runtime models are described as an abstract causally-connected representation of a running system. It allows monitoring and controlling the system as well as reasoning about it [14].

Temporal Models (TMs) go beyond representing and processing the current state of systems [64]. They seek to add short and long-term memory to runtime models through the use of temporal databases [103]. Examples of temporal databases used for TMs, include Time Series Databases (TSDB) and Temporal Graph Databases (TGDB) [103, 120].

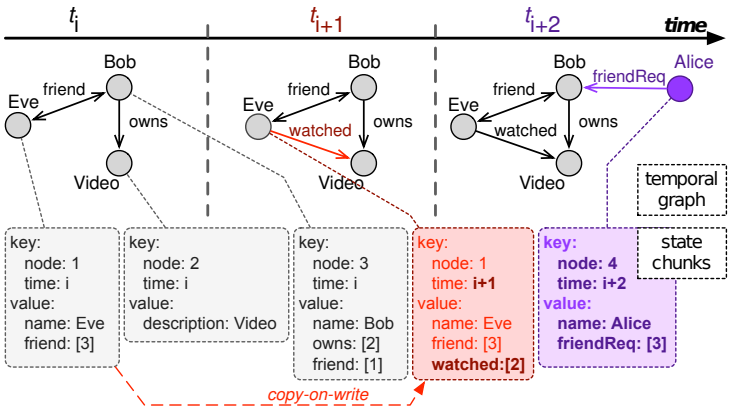


Figure 5.1: Example of a time-evolving temporal graph, by Hartmann et al. [68]

Conceptually, each attribute to be monitored in a running system can be considered as a time series: a sequence of values along an axis [50]. TGDB go beyond traditional event logs and time series in their ability to track the appearance and disappearance of entities and connections, and the capability to relate the changes in several properties of an entity over time [121]. The approach proposed in this thesis uses TGDBs.

TGDBs record how nodes and edges appear, disappear and change their key/value pairs over time. Several TGDB implementations have been proposed. They include Greycat [68] from Hartmann et al. and Chronograph [66] from Haeusler et al. In particular, Greycat is an open-source solution which reuses several existing database engines (e.g. the LevelDB key/value store) to implement a TG data model. Nodes and edges in Greycat have a lifespan: they are created at a certain time-point, they may change in state over the various time-points, and they may be "ended" at another time-point. Greycat considers edges to be part of the state of their source and target nodes. It also uses a copy-on-write mechanism to store only the parts of a graph that changed at a certain time-point, to save disk space.

In this data model (shown in Figure 5.1), the graph is stored as a collection of nodes, which are *conceptual identifiers* that are mapped to specific *state chunks* depending on the *world* and *timepoint* chosen to visit it. Nodes have a lifespan between two specific timepoints, and within that lifespan they may take on a sequence of state chunks. Each state chunk appears at a specific timepoint and overrides any previous state chunk.

In the example in Figure 5.1, during timepoint $i+1$ a "watched" edge is created from "Eve" to "Video", and in $i+2$ "Alice" enters the graph and posts a "friendReq" to "Bob". Instead of storing the three full graphs outright, only new state chunks are created for "Eve"

and "Alice" as needed, using a *copy-on-write* style. State chunks are keyed by node, time and (in Greycat) by *world*. This third coordinate makes it possible to "fork" the graph into multiple branching paths, which enables what-if analyses.

The approach presented in this document adopts the data model by Hartmann et al. of an evolving labelled attributed graph. If models the system operates upon can be turned into this type of temporal graphs, it could allow the system to reflect on what it has been doing in the long and the short term, and provide clear explanations about its history to the user. In the present work, TMs build on top of Greycat TGDB, allow accessing and retrieving causally connected historical information about runtime behaviour of SAS. The end goal of this thesis is to develop a generic and reusable framework to allow SAS to reflect upon their past execution and to improve the explanations provided to the users about their behaviour. In this context, a generic tracing facility and a method to access and retrieve the historical information are required. They will be described next.

5.1.1 Problem-independent execution trace metamodel

SAS are generally built as feedback loops (e.g. those following the MAPE-K architecture). At each timepoint or *time slice*, observations are made and analysed, then future behaviour is planned, and those plans are executed. Since the goal is to make the queries on the execution history reusable, the history must be expressed in a language that can be reused across multiple problems (e.g. network management).

The proposed execution trace metamodel for SAS that need to switch between multiple configurations is shown in Figure 5.2. It has been implemented on top of the previously described EMF for linking the system goals and decisions to its observations and reasoning. In the trace metamodel (Fig. 5.2), the *Log* records the *Decision*s to be made by a the *Agent*, the available *Action*s and their probabilities for satisfying a requirement and the *Observation*s of the environment. Each *Decision* is based on an *Observation* of the environment, which produces a set of *Measurement*s of the *Metric*s. Different types of measurements are allowed (e.g., *DoubleMeasurement* and *StringMeasurement*).

The proposed infrastructure built upon the trace metamodel allows the behaviour of measures/observations and decisions to change from time-step to time-step. However, in a SAS, at each timepoint or time-step, observations are made and analysed, then future behaviour is planned, and those plans are executed. In practice, it is expected to see a stable

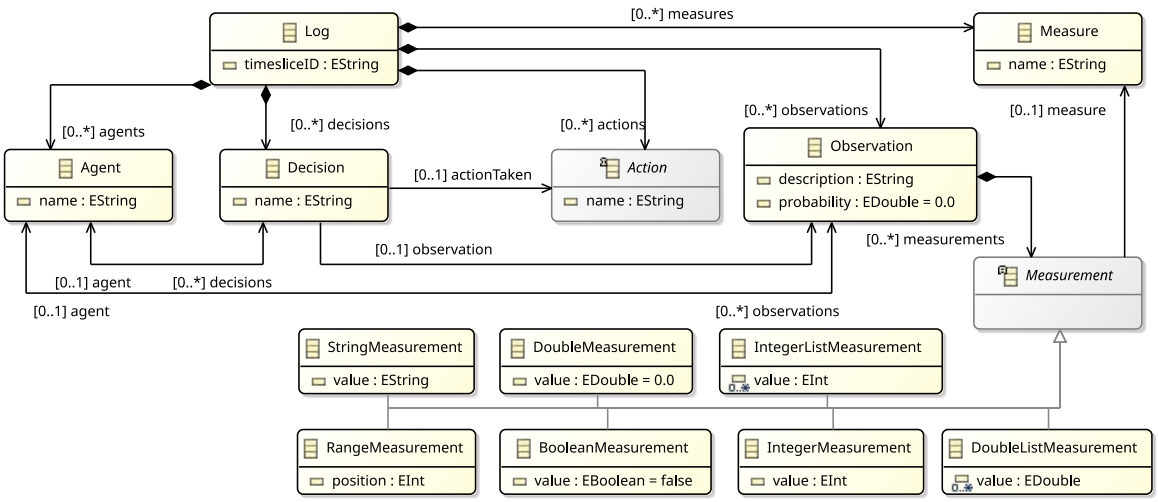


Figure 5.2: Class diagram for the core metamodel used to record system history.

set of SAS agents with their own decisions over time. In the metamodel of Figure 5.2, a Log for a give time slice records the Observations, Measures, and the different Actions that can be taken. These are used in the various Decisions that the system must take. Depending on the sensors, probes, and data collection mechanisms (e.g., observations collected with a certain data rate) implemented, different observations and measures can be recorded in the trace log.

The proposed approach intends to be generalisable, problem-independent and can be feasible for different log recording approaches. However, some rules or conditions must be satisfied in order for a new model instance to be considered well-formed. The instance should have a timestamp tag and a log ID. Once these conditions are met, a log trace can be reshaped into the form of the core metamodel used to record system history by the use of a Log Parser. To ensure that models are consistent, complete, and syntactically correct they should follow some well-formedness constraints. These constraints specify the rules that a model must adhere to in order to be considered valid. They are classified into *structural*, *type*, and *multiplicity* constraints. These well-formedness constrains (WFC) are:

- Structural constraints:
  - WFC1: Every *Log* must have a `timesliceID`.
  - WFC2: Every *Agent* must have a `name`.
  - WFC3: Every *Decision* must have a `name`.
  - WFC4: Every *Action* must have a `name`.

  - WFC5: Every *Measure* must have a `name`.
  - WFC6: Every *Observation* must have a `description`.
  - WFC7: Every *Observation* must have a `probability`.
  - WFC8: Every *RangeMeasurement* must have a `position`.
  - WFC9: Every *IntegerMeasurement* must have a `value`.
  - WFC10: Every *IntegerListMeasurement* must have a list of `values`.
  - WFC11: Every *BooleanMeasurement* must have a `value`.
  - WFC12: Every *DoubleMeasurement* must have a `value`.
  - WFC13: Every *StringMeasurement* must have a `value`.
  - WFC14: Every *DoubleListMeasurement* must have a list of `values`.

- Type constraints:
  - WFC1.1: Every `timesliceID` must be of type `string`.
  - WFC2.1: Every `name` must be of type `string`.
  - WFC3.1: Every `name` must be of type `string`.
  - WFC4.1: Every `name` must be of type `string`.
  - WFC5.1: Every `name` must be of type `string`.
  - WFC6.1: Every `description` must be of type `string`.
  - WFC7.1: Every `probability` must be of type `double`.
  - WFC8.1: Every `position` must be of type `integer`.
  - WFC9.1: Every `value` must be of type `integer`.
  - WFC10.1: Every `value` in a list must be of type `integer`.
  - WFC11.1: Every `value` must be of type `boolean`.
  - WFC12.1: Every `value` must be of type `double`.
  - WFC13.1: Every `value` must be of type `string`.
  - WFC14.1: Every `value` in a list must be of type `double`.
  - WFC15: *DoubleListMeasurement* must be derived from the *Measurement*.
  - WFC16: *IntegerListMeasurement* must be derived from the *Measurement*.

  - WFC17: *IntegerMeasurement* must be derived from the *Measurement*.
  - WFC18: *BooleanMeasurement* must be derived from the *Measurement*.
  - WFC19: *DoubleMeasurement* must be derived from the *Measurement*.
  - WFC20: *RangeMeasurement* must be derived from the *Measurement*.
  - WFC21: *StringMeasurement* must be derived from the *Measurement*.
- Multiplicity constraints:
  - WFC22: *Log* must have a one-to-many relationship with *Decision*.
  - WFC23: *Log* must have a one-to-many relationship with *Action*.
  - WFC24: *Log* must have a one-to-many relationship with *Measure*.
  - WFC25: *Log* must have a one-to-many relationship with *Observation*.
  - WFC26: *Log* must have a one-to-many relationship with *Agent*.
  - WFC27: *Decision* must have a many-to-one relationship with *Agent*.
  - WFC28: *Decision* must have a one-to-one relationship with *Observation*
  - WFC29: *Decision* must have a one-to-one relationship with *Action*.
  - WFC30: *Agent* must have a one-to-many relationship with *Decision*.
  - WFC31: *Agent* must have a one-to-many relationship with *Observation*.
  - WFC32: *Observation* must have a one-to-one relationship with *Agent*.
  - WFC33: *Observation* must have a one-to-many relationship with *Measurement*.
  - WFC34: *Measurement* must have a one-to-one relationship with *Measure*.

The resulted runtime model will then be used to update the TGDB, creating a new snapshot at the current point in time: all relevant versions are kept. A model indexer is used to automatically compare the runtime model as an object graph against the current version of the temporal graph model. A model indexer is a tool that enables indexing and querying of models based on various criteria. An indexer can help to improve the efficiency of model queries and support model discovery and reuse. Some examples of models indexers are EMF Index[1], NeoEMF[2], and Eclipse Hawk[3]. The proposed approach uses Eclipse Hawk given its

[1]https://www.eclipse.org/proposals/emf-index/
[2]https://neoemf.atlanmod.org/
[3]https://www.eclipse.org/hawk/

maturity, ease to use, and community support. Hawk operates on Greycat temporal graphs as firstly described in [57]. By using TGDBs, it is possible to track the evolution of certain metrics at each node, as well as the changes in the relationships of the various entities in the system, or their appearance and disappearance. The backend used by Hawk (i.e. Greycat) uses a copy-on-write approach that only creates a new version which updates the temporal graph model where needed, for efficient storage.

In order to implement the proposed solution, some storage specifications are required. These storage requirements refer to the specific features and capabilities that the storage layer needs to provide in order for the proposed approach to work. The current implementation bases its functionality on a temporal graph model construction with elicitation time-aware capabilities. The time-aware indexing in Hawk relies on the implementation of the `ITimeAwareGraphDatabase` interface[4], which specifies the requirements for a temporal graph with lifetimes for nodes and edges, per-node and per-edge versioning, and time-aware lookup indices. Hawk uses Greycat for the implementation of the basic temporal graph and combines it with Apache Lucene for time-aware lookup. Using Hawk, a evolving temporal graph database conforming the defined metamodel is created. This can be reused for different goal-oriented SAS. Moreover, it can be easily extended to meet the need of specific applications as will be shown in the Chapters 6 and 7.

5.1.2 Reusable time-aware query language

Eclipse Hawk already had its own query language, the previously described EOL [85] which conceptually is a mix of OCL and JavaScript. EOL was extended with time-aware primitives, and with the concept of "history" for any type and its instances. The definitions for the history of a model element and a type are as follows:

- The history of a model element starts from the moment it is created, and ends when it is destroyed. Model elements are assumed to have a unique identity, which could be a natural or artificial identifier or its location within the model. There will be a new version of a model element every time its state changes, whether by changing the value of an attribute or the target of one of its references to other model elements.

- Model element types are considered "immortal", in the sense that they are created at

[4] https://www.eclipse.org/hawk/advanced-use/temporal-queries/

the first timepoint in the graph and last to the virtual "end of time" of the graph. We will have a new version of a model element type every time an instance of the type is created or destroyed.

For model elements and model element types, the basic time-aware operations that must be supported by the temporal graph backend (e.g. Greycat) are: i) retrieving all versions, ii) all versions within a range, iii) versions from/up to a certain timepoint (included), iv) earliest/previous/next/latest version, and v) retrieving the timepoint for that version. The semantics for these basic operations represented using EOL are shown in the Table 5.1 as *Version Transversal*.

In an outcome of this thesis [60], the time-aware EOL dialect was further extended with primitives inspired on Dwyer's work on temporal specification patterns mentioned in Section 3.2, with the ability to use *temporal assertions* (e.g. *always*, *never*) and *version scopes* (e.g. *when*, *until*). The same work presented a first version of *timeline annotation*, a mechanism for automatically annotating specific moments in history where an event of interest happened, speeding up its retrieval in a later query. These extensions to EOL are found in Table 5.1 and are described next.

In order to find model elements in the history according to these time-aware patterns, users need a conceptual model about what "the versions of `x`" means. In these queries, `x` may be a model element (in the EMF an *EObject*), or it may be a type of model element (in EMF, an *EClass*). `x` has a certain identity and a lifespan:

- Model elements may receive an identity through a natural identifier (a "business key"), an artificial identifier (e.g. XMI identifiers), or their location within the model. Their lifespans are limited by the moment they are created, and when they are destroyed.
- Types are identified by a combination of the metamodel identifier (in EMF, the metamodel URL) and its name. Types are "immortal", as their lifespans start at the "beginning of time" and end at the "end of time" (the latest timepoint that the system is able to represent).

With the lifespans and the versions of both model elements and their types, time-awareness can be added as a new set of operations. These operations are collected in Table 5.1. The operations can be divided into several types, which will be discussed in separate subsections next.

Operation	Syntax
Version traversal	
All versions, from newest to oldest	`x.versions`
Versions within a range	`x.getVersionsBetween(from, to)`
Versions from a time point (included)	`x.getVersionsFrom(from)`
Versions up to a time point (included)	`x.getVersionsUpTo(to)`
Earliest / latest version	`x.earliest`, `x.latest`
Next / previous version	`x.next`, `x.prev` / `x.previous`
Version timepoint	`x.time`
Temporal assertions	
True for all versions?	`x.always(v | p)`
False for all versions?	`x.never(v | p)`
Any matching version?	`x.eventually(v | p)`
At least n matching?	`x.eventuallyAtLeast(v | p, n)`
At most n matching?	`x.eventuallyAtMost(v | p, n)`
Predicate-based version scoping	
View with versions since match (inclusive, exclusive)	`x.since(v | p)`, `x.after(v | p)`
... until match (i., e.)	`x.until(v | p)`, `x.before(v | p)`
... with matching	`x.when(v | p)`
Context-based version scoping	
View with versions since current (i., e.)	`x.sinceThen`, `x.afterThen`
... until current (i., e.)	`x.untilThen`, `x.beforeThen`
Version unscoping	
Original without version scoping	`x.unscoped`

Table 5.1: Proposed new operations for time-awareness for the Epsilon Object Language, divided by type. `p` stands for a Boolean predicate.

Basic version traversal

These primitives provide access to all the versions of a type or a model element. To retrieve the instances of the latest version of a type, a user would write `Type.latest.all` instead of `Type.all`. The new semantics of `Type.all` would need to default to either the earliest or the latest version. `x.versions` would return the collection of the versions of `x`: therefore, finding the versions of `x` that meet a predicate could be done with `x.versions.select(version | predicate)`. `Type.latest.all.first` would produce the first instance of that type in its latest version: the instance would be at the timepoint of the latest version of that type.

Temporal assertions

These operations provide concise ways to check if certain temporal properties hold for a type or model element. For instance, `x.always(version | predicate)` would answer if a particular predicate has always held for `x`. `eventuallyAtLeast` and `eventuallyAtMost` would not be as easy to implement in one line, given their ability to stop as soon as enough or too many elements are found, respectively.

Predicate-based version scoping

In some cases, it may be desired to examine specific version ranges, e.g. from the first moment P was true onward. Rather than extending the assertions with arguments for version ranges, a better design is to have dedicated primitives which will return a view of the original type or model element, which only exposes some of the versions. This can produce richer patterns by composition: for example, with `x.since(v | Q).always(v | R)`, it would be able to check whether since an arbitrary predicate `Q` first held from the current timepoint of `x`, `R` was always true. The scoping can also be composed: for instance, `x.since(v|Q).until(v|R)` would produce a view over `x` that would only report the versions between `Q` and `R`. The returned views are still model element or types, as the originals, and the view will always switch to the oldest matching version within the scope. For instance, if `x` was a *Person*, `x.since(v|Q)` will still be the same *Person*, but at the first timepoint when `Q` held.

Context-based version scoping

As mentioned above, every type and model element visited at each step of a query will be at a specific timepoint. This timepoint may be used as the reference for the version scoping: there are `xThen` variants of `since`, `after`, `before` and `then` that use the timepoint of the current model element or type rather than a predicate.

Version unscoping

Crucially, one final operation which was not part of the works by Dwyer and Kanso is the `unscoped` operation. This operations returns the same model element or type, but without any of the version scoping: all versions are visible once more. This is useful in more advanced queries which need to switch between different scopes in the same query.

Time-aware EOL example

As an example using these primitives, suppose that there is a *Semaphore* model element `s` with two features: the shown `color` (red, yellow or green), and the `count` with the number of vehicles that passed in the last 5 seconds. Assume that it is desired to check, for example, if across all intervals where the light was yellow, no more than 5 vehicles passed.
To determine the querying primitives for a particular scenario, it is necessary to locate the versions that define the intervals in question. In this case, the intervals refer to the moments when the light changes to yellow. To do this, one must determine if there was no previous version or if the previous version had a different colour.
Once these delimiters have been found, the test function `.always(v | predicate)` can be applied to each of them. The predicate would be evaluated using a variable `v` that only includes versions when the semaphore changed to yellow. In order to access the appropriate interval within the `always` predicate, it is necessary to first undo the scoping with the `unscoped` function. The final query would involve undoing the scoping within `always` and adding together the vehicle counts for all versions from that point until the semaphore changes colour. This total would then be compared with the upper bound to complete the query, as shown in Listing 5.1.

```
s.when(v | v.color = 'yellow'
  and (v.prev.isUndefined()
    or v.prev.color <> 'yellow'))
 .always(v | v.unscoped.sinceThen
  .before(v | v.color <> 'yellow')
    .versions.collect(v | v.count).sum() <= 5)
```

Listing 5.1: Example Epsilon Object Language query for the `unscoped` primitive: *Semaphore* `s` does not let more than 5 vehicles through in any of its yellow intervals.

Timeline Annotation

Scalability is a challenge when indexing models with potentially very long histories: both in terms of storage size, and in terms of query times. Finding rare events across many versions can take a long time if it is needed to iterate through each version while testing a predicate. For design-time models, this may happen in artefacts that have been worked on for a long time, whereas runtime models may quickly run into this problem if they are updated very frequently. Being able to answer a query as soon as possible could be critical for approaches based on runtime models which need to interact with the outside world.

In [9] Barmpis et al. proposed expensive calculations in advance, storing them into *derived attributes* that extended existing types, and keeping the model elements indexed by these values. Finding which elements matched a predicate could be done with an efficient lookup, rather than expensive iteration. The same approach can be adopted for finding occurrences of an event in the history of a model instance. The predicate describing such an event can be defined as a derived Boolean attribute on its type, which would be pre-computed and indexed incrementally across the versions of each of its instances. Queries will naturally be multidimensional in this case: they will have to look up matches based on the desired value, and the appropriate range of timepoints.

In effect, such an extension is annotating the timeline of each model element with markers for these events of interest. These markers can be used in queries through alternative versions of the version scoping primitives of Table 5.1. `whenAnnotated(a)` will produce a view that exposes the versions for which the derived Boolean attribute named `a` will be true, by index lookup. `sinceAnnotated(a)` / `afterAnnotated(a)` will produce views with left-closed and left-open version ranges by index lookup. `untilAnnotated(a)` / `beforeAnnotated(a)` will produce views with right-closed and right-open version ranges.

Beyond model queries, these annotations could also be used for incrementally computed

model monitoring as well. If a certain model element matched a certain predicate, it could trigger a notification to an external system or a modeller. Users would be able to quickly recall past occurrences of the same event, for the sake of comparison.

These time-aware extensions to EOL and timeline annotations have been used in this research for analysing back and forth the stored SAS history in the TMs. The extracted information conforms the explanations to be presented. Detailed examples will be described in Chapters 6 and 7.

5.2 Research roadmap for History-awareness with Explanatory capabilities in SAS

When enabling reflective and history-aware capabilities in SAS, it is argued that rather than an all-or-nothing situation, it is easier, safer and more rewarding in the short-term, to do it in stages or *levels*. Figure 5.3 shows the envisioned levels of reflective capabilities that a SAS should offer (adapted from [59, 119]). These capabilities are framed in four incremental levels which are: i) *forensic history-aware explanations*, ii) *live history-aware explanations*, iii) *externally-guided history-aware decision-making*, and iv) *autonomous history-aware decision-making*. Incremental capabilities imply that capabilities at *level* n should be available for capabilities at *level* $n + 1$. This spectrum of reflective history-aware capabilities in SAS acts as a research roadmap for this thesis and will be described further in the following sections of this chapter.

5.2.1 Level 1: Forensic history-aware explanations

This level operates very much like a "black box". From the runtime monitoring approaches discussed in Section 4.1, this is part of the offline type of runtime monitoring where the system runs as normal, while capturing logs in a machine-parseable form. After the system has finished its execution (whether gracefully or crashing), the history of the system is stored and processed for after-the-fact analysis. In this document, *Temporal Models* are proposed and used as a key enabler for efficient and reusable storage and analysis of a SAS's history. Users can then study its history with a *temporal query language*. In the present work, a temporal database supports the storage of massive amounts of historical data, while providing fast querying capabilities to support reasoning about runtime properties in the

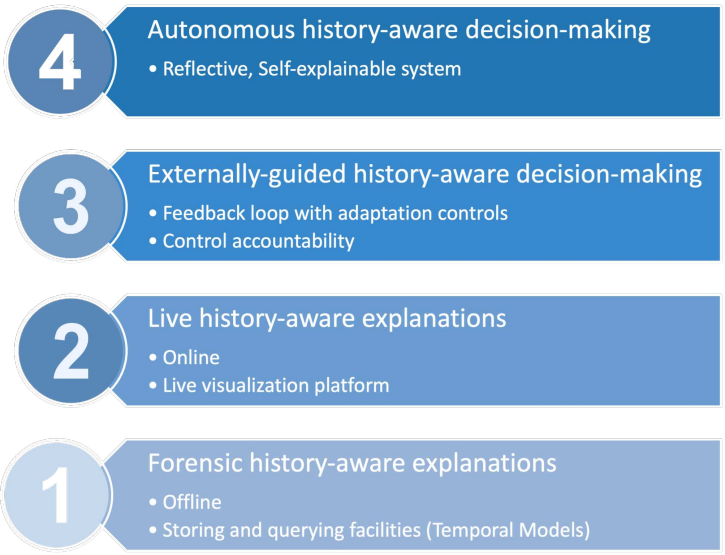


Figure 5.3: Proposed research roadmap for history-awareness in SAS (adapted from [59, 119]).

monitored SAS.

Figure 5.4 depicts the flow diagram of level 1. The SAS starts its execution, the log traces are stored after every point in time in conformance of the metamodel of Figure 5.2. Once the execution has finished, the information is available for extraction. The extracted information will conform the explanation to be presented. Finally, the explanation's recipient can perform and action if required. This could be the case of SAS developers trying to validate and improve their systems. This first level is useful for either post-mortem analysis after an unexpected behaviour, or for internal evaluation during development. The storage and querying facilities present in this level are the base for the subsequent levels.

5.2.2 Level 2: Live history-aware explanations

This level allows users to evaluate past observations, decisions and performance on a running system without having to stop it or waiting until the end of the system's execution. It exploits the storing and querying capabilities of level 1 but requires extra features to allow online monitoring. As such, it requires an *incremental importer mechanism* that loads periodically the latest state of the decision algorithms into the TM, adding one more timepoint to its history. To keep storage and memory costs manageable, the history of the TM may be *bounded* to a specific *time window*. The TM may be structured as a strict linear sequence of system states, or as a graph of states that the system may go to and from: this will depend

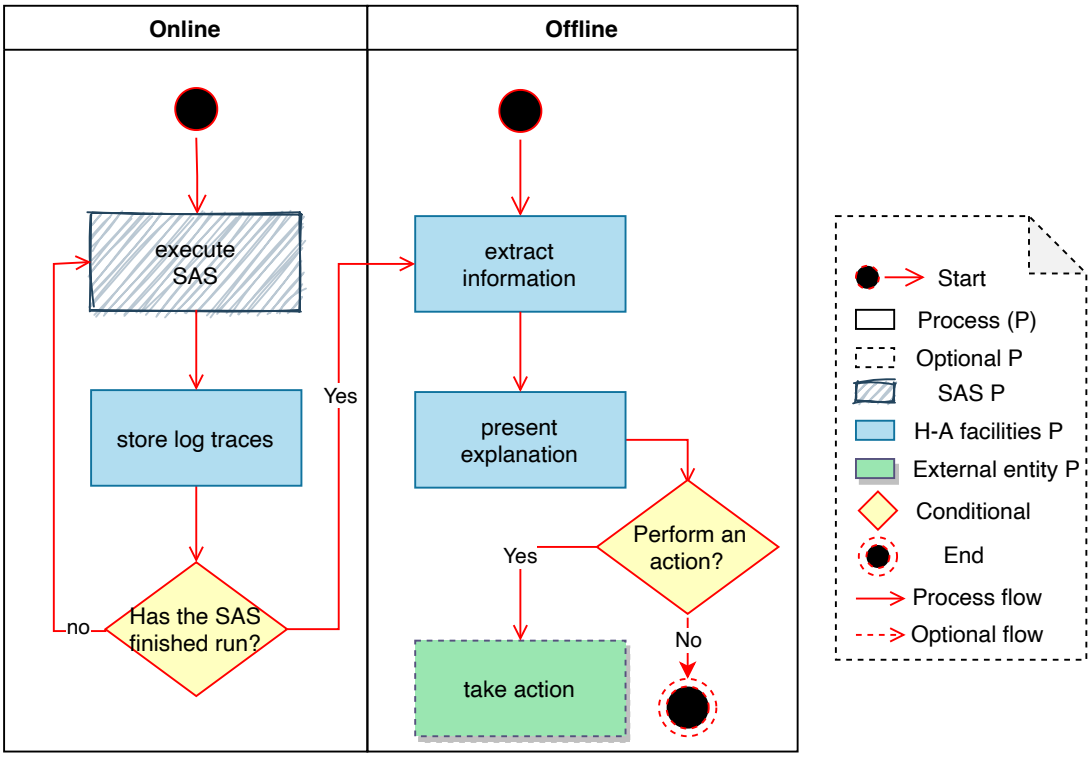


Figure 5.4: Level 1: Forensic history-aware (H-A) explanations

on whether there is a restricted and finite number of possible system states, or not.

A *live visualisation platform* will allow various types of stakeholders (developers, end users) to study the history of the system. This level can help users gain trust in the SAS during its day-to-day operation, and does not require modifying the existing decision making process.

Figure 5.5 depicts the flow diagram of level 2. The SAS starts its execution and after each time step traces are reshaped to conform to the trace metamodel (Figure 5.2) and stored as a new version in the TM. If an explanation is required at any point in time, the information is extracted and presented using a live visualisation platform. The forensic capabilities from level 1 are available for off-line analysis if required.

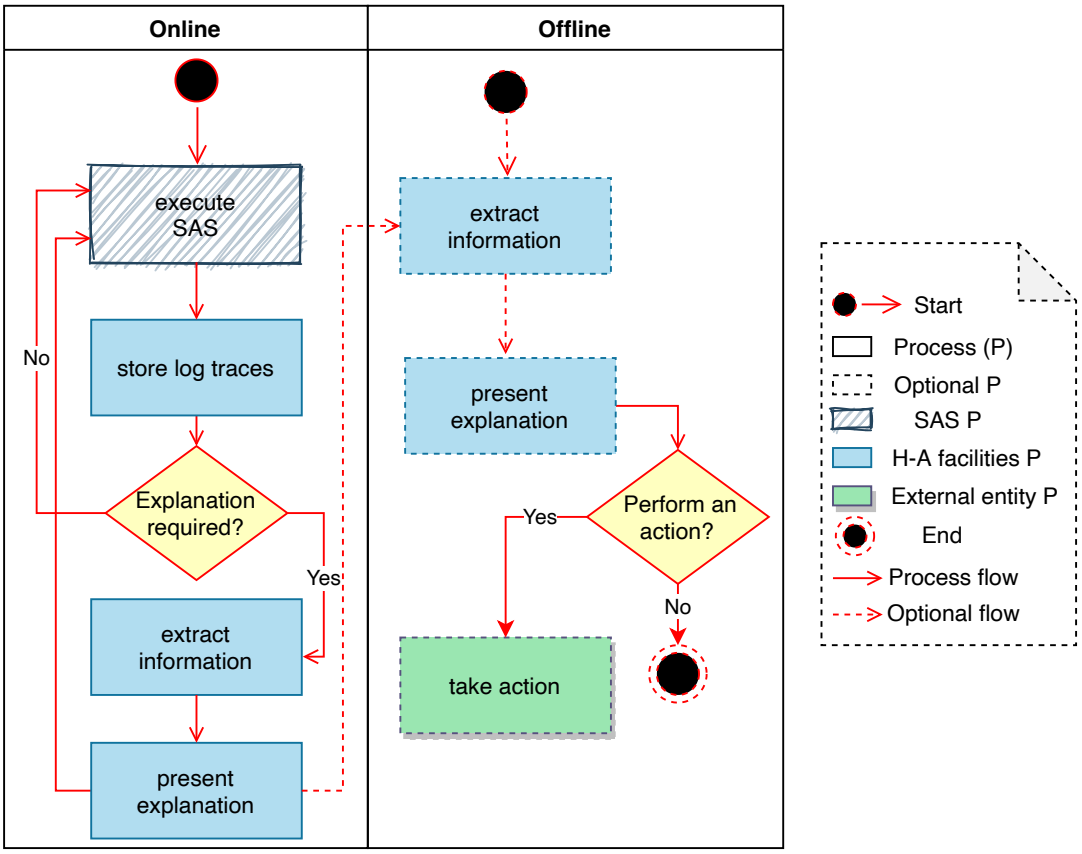


Figure 5.5: Level 2: Live history-aware (H-A) explanations

5.2.3 Level 3: Externally-guided and history-aware decision making with explanation capabilities

This level allows external entities to interact actively with the SAS at runtime. These external entities can be either a human or another software system. The explanation's recipients are able to perform changes in the SAS through a set of effectors (i.e. input parameters or some type of configuration facility). For example, at the *Plan* stage of the MAPE-K architecture, an external entity could influence the high-level goals of the system by guiding the system and their priorities, or the internal parameters governing their algorithm (i.e, hyperparameters): in both cases, the explanation's recipient would need an appropriately abstracted explanation to allow them to make an informed decision about the relevance and impact of the intended changes. At the *Execute* stage, effectors could allow external entities to explicitly select certain actions; for example, on an autonomous car the user could decide to go on a specific direction that goes against the system's reasoning. In this case the system may need to reconfigure its decision-making to meet a new preference

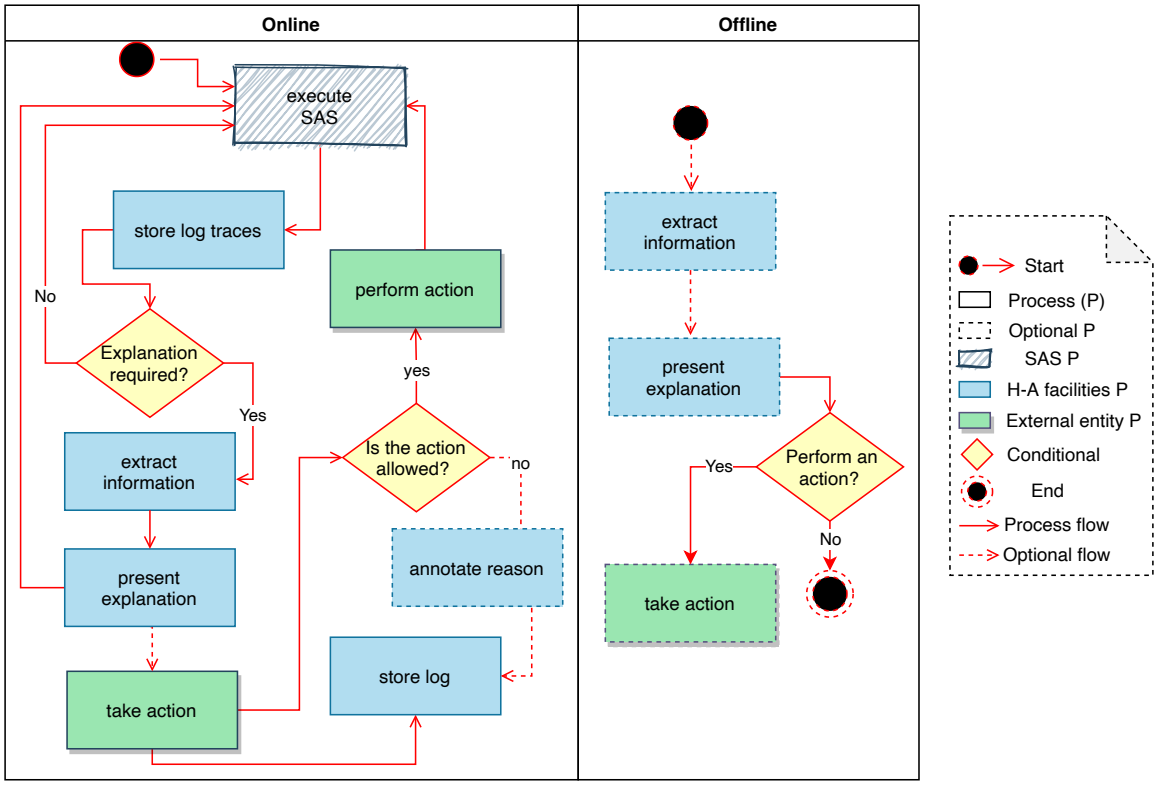


Figure 5.6: Level 3: Externally-guided history-aware (H-A) decision-making with explanation capabilities

introduced by a user.

In the case of humans, collaboration, communication and trustworthiness between the human and SAS is promoted by the provision of history-based explanations extracted from the TM, and a set of effectors allow human users to influence the system based on the received information. Effectors designed for humans should be defined in a notation flexible enough to express the evolving preferences of the user, while also concise enough to not overwhelm the user with low-level details. As control would be partly given to an external entity, it would be important for the trustworthiness and accountability of the level 3 system to record these interventions accordingly.

Figure 5.6 describes the flow diagram for level 3. Similar to the previous level, an explanation is presented if required. If the external entity (i.e., the explanation's recipient) considers that an action should be performed to steer the SAS behaviour, a validation process should take place to avoid inconsistencies in the system decision-making. If the intended action is allowed, the changes are performed in the SAS execution. Proposed changes by external entities should be recorded as part of the TM for accountability. Finally,

the same offline analysis capabilities of level 1 are available if required.

5.2.4 Level 4: Autonomous history-aware decision-making with explanation capabilities

At this last level, a *history-aware decision making process* is introduced to further support autonomous behaviour. It will be an improved version of the existing decision-making process that takes its own control over its history as one more dimension to adapt [154]. For example, one way to use the history would be to recognise major trends that may require reconfiguration, and which may not be evident from a single timeslice: e.g. continued performance degradation over time in a particular indicator. This may trigger its own adaptations aiming at long-term effects. As there may be too few observations to support it, or these observations may be too different to the original ones, the system should evaluate its confidence level on the estimated trajectory.

Any interventions based on the history of the system should also have to be tracked into a dedicated *automated control accountability*. At this level, the explanation capability based on the infrastructure provided by level 1 and level 2, will enable reasoning about the history of the SAS, i.e. the system and the adaptation logic will be history-aware (HA).

Figure 5.7 shows the flow diagram for level 4. This level mimics level 3 but with the difference that all the runtime processes (i.e. online processes) are part of the SAS decision-making. The system is able to look back and reflect on previous experiences. If desired, the SAS history can be recorded for post-mortem analysis (level 1).

From the point of view of the type of explanations recognised in Section 2.2.1, the levels 1 to 3 can be categorised as post-hoc explanations. Thus, the explanation generation and communication are processes that are external to the SAS' inner workings. The H-A facilities identified in every graph are processes running in parallel to the main SAS. In the case of level 4, the H-A facilities and, therefore, the explanation generation and communication should be part of the SAS itself. Thus, they would be categorised as the intrinsic type where the system is self-explainable. The level 4 is out of the scope of this thesis. The following chapters will describe the implementation of levels 1 to 3 for different case studies, analysing different trade-off, of enabling history-aware explainability in SAS through the use of TMs.

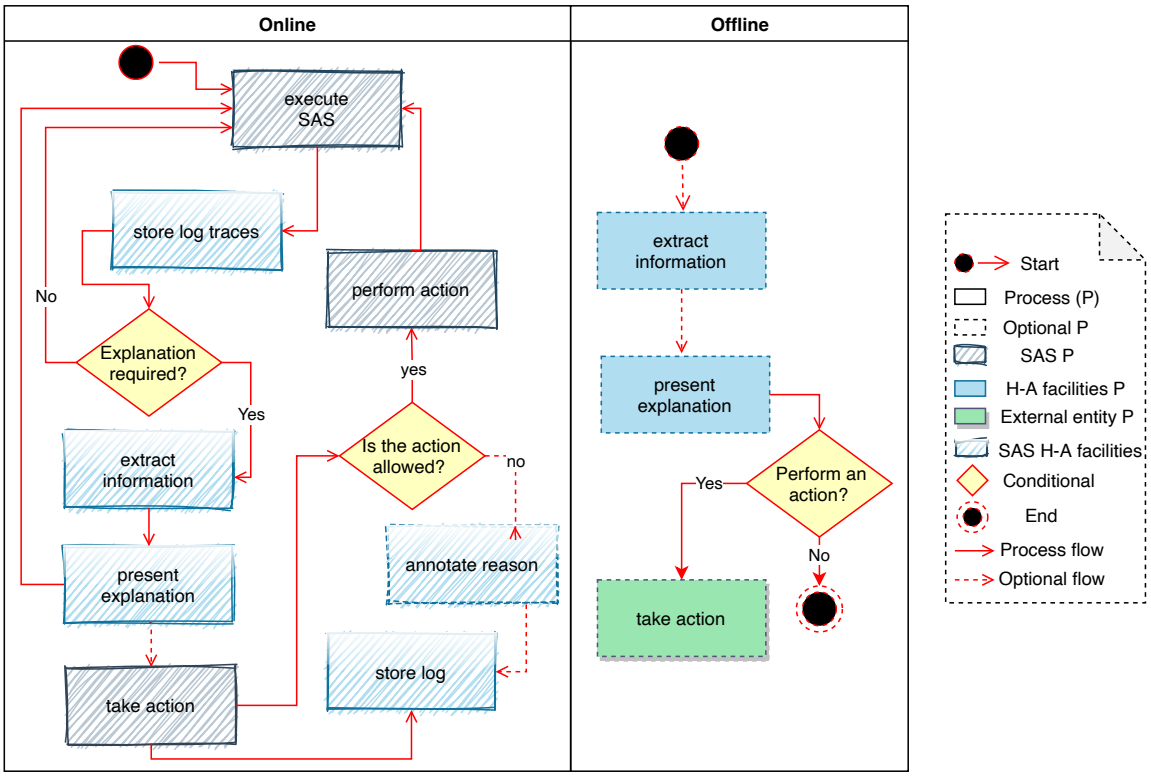


Figure 5.7: Level 4: Autonomous history-aware (H-A) decision-making with explanation capabilities

Chapter 6

Explaining SAS with Temporal Models

The work presented in this chapter has been adapted from the following publications:

[119] J. M. Parra-Ullauri, A. García-Domínguez, L. H. García-Paucar, and N. Bencomo. Temporal models for history-aware explainability. In *Proceedings of the 12th System Analysis and Modelling Conference*, pages 155–164, 2020.

[121] J. M. Parra-Ullauri, A. García-Domínguez, J. Boubeta-Puig, N. Bencomo, and G. Ortiz. Towards an architecture integrating complex event processing and temporal graphs for service monitoring. In *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, pages 427–435, 2021.

[123] J. M. Parra-Ullauri, A. García-Domínguez, and L. G.-P. Bencomo, Nelly. History-aware explanations: Towards enabling human-in-the-loop in self-adaptive systems. In *Proceedings of the 14th System Analysis and Modelling Conference*, 2022. To be published.

[60] A. García-Domínguez, N. Bencomo, J. M. Parra-Ullauri, and L. H. García-Paucar. Querying and annotating model histories with time-aware patterns. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 194–204. IEEE, 2019.

[59] A. García-Domínguez, N. Bencomo, J. M. Parra-Ullauri, and L. H. García-Paucar. Towards history-aware self-adaptation with explanation capabilities. In *2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS* W)*, pages 18–23. IEEE, 2019.

6.1 Motivation

It is argued that explanations shown by a running SAS about its decisions and behaviour helps someone diagnose the behaviour of the system to analyse and trace past actions, helping fix potential faults and fostering the trust of the end users [119]. To enable these capabilities, this work proposes that SAS should be equipped with traceability management facilities and offer temporal links to provide (i) the impacts of the adaptation actions over the quality properties of the system over time and (ii) the history of the decisions of the system, together with the data that informed those decisions.

History-awareness requires an efficient manner to represent and query past history. Logs are prevalent in all kinds of computer software: however, most of them are text-based and are usually intended to be used by humans, with precarious support for automated processing such as simple filtering and tagging. This chapter presents contributions towards allowing the system to support explanations to operators and end users based on the generic metamodel defined in Section 5.1.1 (Figure 5.2). This generic metamodel structures execution traces of SAS, using a parser, it shows how a sequence of traces can be turned into a TM. The temporal query language of Section 5.1.2 is used to extract information from TMs that will construct explanations. Explanations should be human-friendly and machine-friendly. Specifically, explanations should be available to different stakeholders such as end-users, developers, external systems, or the SAS itself.

This chapter describes an implementation of the proposed approach to a case study from the domain of SAS using runtime models: a Remote Data Mirroring (RDM) system [61, 75, 129]. A SAS frequently updates its knowledge about the environment and its decision processes. Being able to query this evolution would allow users to know the reasons why particular actions were taken. The section starts with a description of the RDM system. This is followed by an outline of the experimental setup for implementing levels 1, 2 and 3 of the proposed research roadmap (Section 5.2). Finally, the results are discussed.

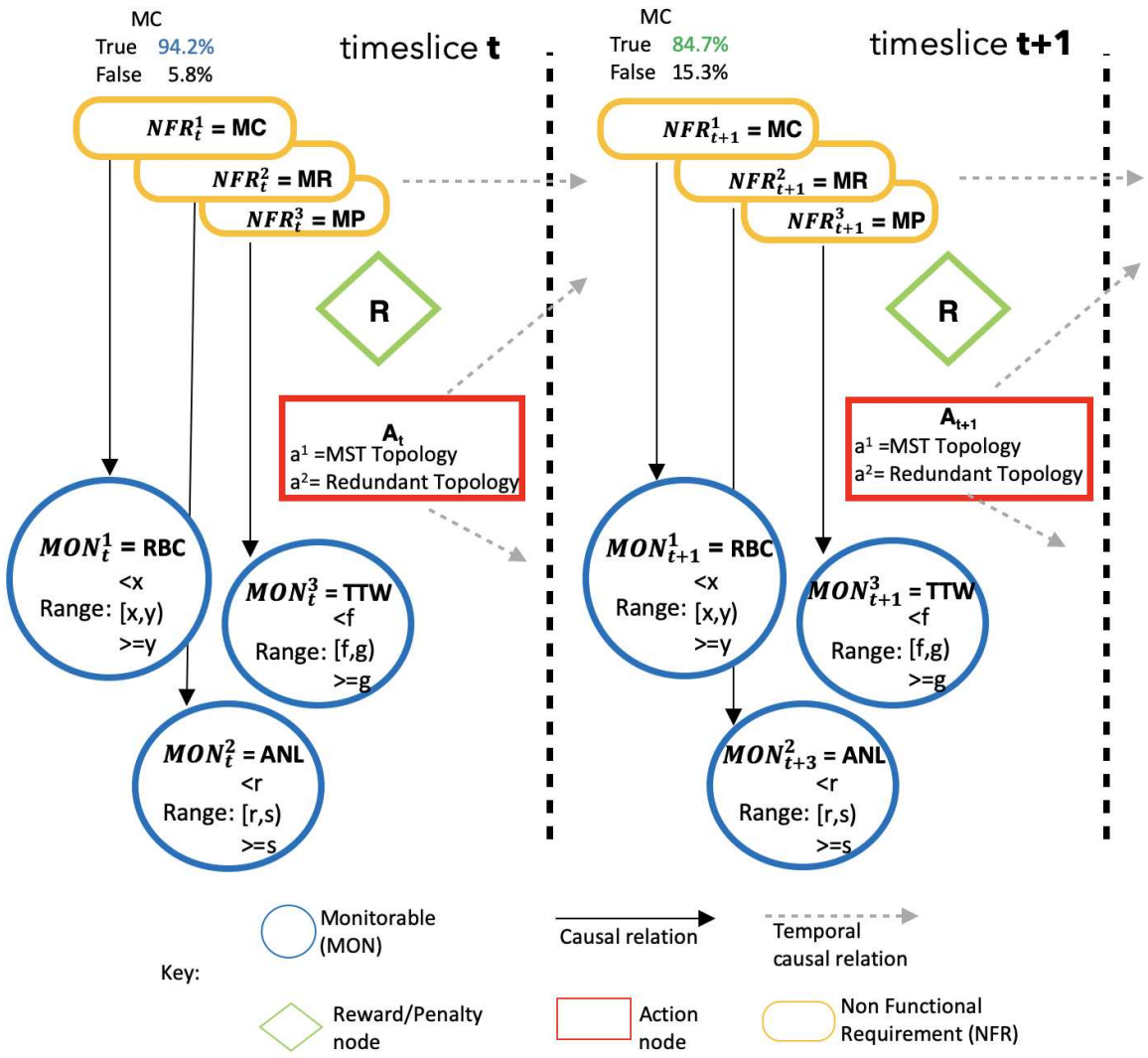


Figure 6.1: RDM Case Study

6.2 Experimental study: RDM SAS

The first case study in this dissertation is the RDM SAS, which uses Bayesian Learning (an AI approach) [61]. RDM is a technique to protect data against inaccessibility to therefore provide further resistance to data loss [75, 129]. RDM maintains data availability and prevents data loss by storing copies (i.e., replicas) on servers (i.e., data mirrors) in physically remote locations [11].

Its overall structure is shown in Figure 6.1. Uncertainty exists due to different unexpected situations such as delayed or lost messages, noise in sensors, or network link failures. RDM self-adapts to these situations by reconfiguring itself. Specifically, RDM can use two topologies: minimum spanning tree (MST) and redundant topology (RT). These two

possible configurations allow the RDM selectively activate and deactivate network links to change its overall topology at runtime [11].

The RDM SAS self-adapts at runtime according to the changes in its environment. Each network link in the RDM brings upon an operational cost and has a measurable throughput, latency, and loss rate. The performance and reliability of the RDM are determined by these metrics according to the following trade-off: while RT is more reliable than MST, RT can be prohibitively more computationally expensive than MST in some contexts. Each configuration provides its own levels of reliability and energy costs which are taken into account while estimating the levels of satisfaction of the Non-Functional Requirements (NFRs) observed, the Maximization of Reliability (MR), the Minimization of Energy Consumption (MC), and the Maximization of Performance (MP). `MST` is more efficient in terms of energy consumption and performance, whereas `RT` is more reliable.

It is not possible to directly observe whether these NFRs are being met or not. In that sense, observations about their states are obtained by using monitoring variables (called MON variables in the Figure 6.1). The three MON variables are Ranges of Bandwidth Consumption (RBC <x, RBC in [x,y) and RBC>=y), Active Network Links (ANL<r, ANL in [r,s) and ANL >=s) and Total Time for Writing (i.e., TTW<f, TTW in [f,g) and TTW >=g). In [124], the authors show the requirements specification based on Partial Observable Markov Decision Processes (POMDP) that enables reasoning and decision-making about partial satisfaction of NFRs. Trade-offs are analysed using evidence collected at runtime based on the formalism for decision-making under uncertainty provided by POMDPs (See Figure 6.1). POMDPs provide an approach to making rational decisions in the face of uncertainty within changing environments [11].

A POMDP model can be specified as a 6-tuple (S, A, Z, T, O, R), where: S, A, and Z represent the system's state space, action space and observation space, respectively. T represents the transition function that describes the stochastic nature of actions' effects over the state of the system. O is the observation function that describes the conditional probability of observing Z when an action A is performed and a state S is reached. Finally, R is the reward function [11]. In the case of RDM, S are the possibles states for the NFRs (MR, MC, MP), shown in yellow in Figure 6.1. A corresponds to the action of switching topologies from MST to RT and vice versa, shown as red in Figure 6.1. Z the observation space of the monitored variables (RBC, TTW, ANL), shown as blue in Figure 6.1.

The approach considered by the implementation of RDM is proactive, where R-POMDP considers future evolutions (i.e. projections into the future) of the satisfaction of the NFRs to decide the next action $a \in A$, to reason about long-term effects of immediate actions [61]. These future evolutions are represented by a belief b over possible states. Proactive approaches can be produce not favourable results at first while improving the behaviour in the long term. The operators may find themselves asking the reasons why the RDM SAS has used one topology instead of the other. These kind of situations may require explanations.

6.2.1 RDM and the MAPE-K loop

This section presents the RDM architecture based on the MAPE-K loop (described in Section 2.1.1) and the role played by the Requirements-aware Model (RaM-)POMDPs [11]. A RaM-POMDP serves as a runtime model that supports the decision making by a SAS and which is driven by the trade-off of the levels of satisfaction of the NFRs. It uses Bayesian inference based on evidence collected from the environment [11].

Figure 6.2 describes the different stages of the MAPE-K loop in RDM proposed by Garcia-Paucar et al in [61]. They are described next:

- Monitor: During the SAS execution the variables RBC, ANL and TTW are monitored to offer evidence that is later used to compute the probability distributions in order to check the satisfaction of the NFRs [11].

- Analyse: At this stage, adaptation needs are evaluated. Examples of data processing needed are given by the fact that the RBC can be directly observed by sensors, while ANL is computed using a reachability algorithm which studies which RDM nodes can be reached by traversing active network links [11].

- Plan: The online POMDP planning is used to choose the best action under the current state of the RDM system. Online POMDP planning is a technique that interleaves planning with plan execution, which allows computationally tractable solutions for POMDPs [61]. At each time slice, the system searches for an optimal action $a \in A$ at the current belief b. It then executes the chosen action immediately [11].

- Execute: At this stage, model-level adaptation is refined to system-level adaptation. The action selected at the Plan stage is executed. The system will reach a new state

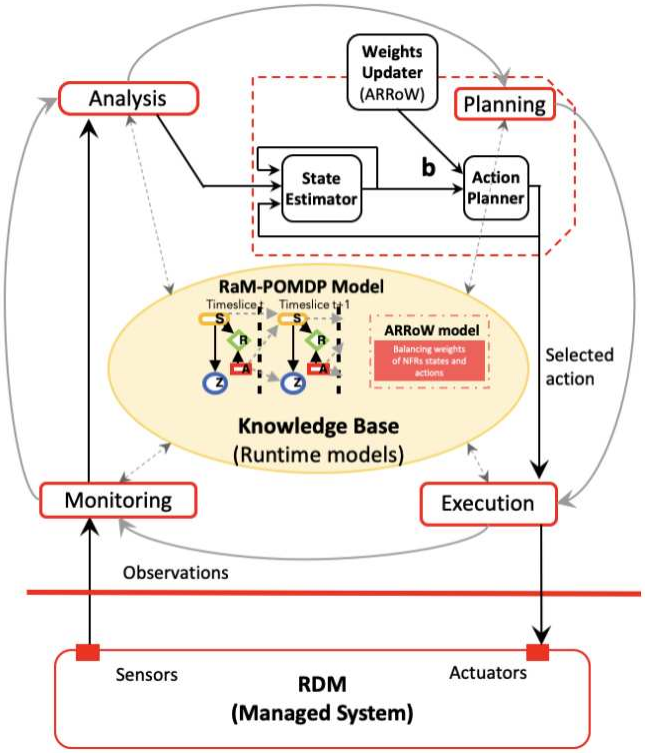


Figure 6.2: Overview of causal relationships between runtime models in the MAPE-K loop of the RDM from [61]

$s' \in S$ with probability T and experience an observation $z \in Z$ with probability O, with the resulting reward value R [11].

- Knowledge: Using the observed monitored values, the action $a \in A$, and the previous belief b, the state estimator infers the current belief b' over the possible states i.e. the current probability distributions about the satisficement of the NFRs for the RDM SAS [11].

The studied implementation of the RDM SAS bases its functionality on runtime models that are causally connected to the MAPE-K architecture. The proposed approach aims to differentiate the SAS functionality to the explanation generation and communication. Therefore, it is desired to provide a post-hoc approach which is agnostic to the SAS architecture. The following section will describe the requirements to be met by the proposed generic and reusable framework based on TMs for the RDM SAS.

6.2.2 Enabling History-Aware explainability in RDM SAS

In order to facilitate the exploration of the history to generate and communicate explanations through TMs, a different steps are required. The process will be described in the following subsections.

Log collection and preprocessing

The RDM SAS produces execution traces for each time slice. The traces (logs) contain information related to the observations made by the agent about its decisions, actions, states, rewards, and environment, and the preferences currently being applied in the decision process. The log is made available to the proposed framework for explanation purposes. Listing 6.1 shows an excerpt of the log for the first time slice. The system collects a large number of logs and transform them into a TM, answering queries away after the system has completed its execution (i.e., in an "post-hoc" fashion).

The steps required for this process are:

1. Collection of monitoring variables: The variables used for decision-making in RDM are collected for every timeslice. They include (from Figure 6.1 and Listing 6.1) timeslice ID, RBC, TTW, ANL, MC satisfaction, MR satisfaction, MP satisfaction,and the action performed, among others. This process does not add extra computing overheads as the variables are monitored by the system as part of the decision-making process.

2. Log construction: The log can follow structured (JSON / XML) or unstructured (plain text) formats: JSON has been selected for this implementation as it is a widely adopted. This JSON log containing unprocessed data is converted into the data format required by the model indexer, Eclipse Hawk.

3. Log exposure: In order to construct the TM that will then enable history exploration, the JSON trace logs have to be exposed to Eclipse Hawk. Hawk offers different approaches for adding the repositories to be indexed (i.e., TM versions). They include Git-based, SVN-based, and file-based configurations. Additionally, it is possible to develop customised configurations. In the present work, both approaches (existing and customised connectors) have been explored and will be described it the following sections.

Listing 6.1: Excerpt of the original JSON trace execution logs from the Remote Data Mirroring self-adaptive system

```
{
    "0": {
        "current_belief_mec_true": 0.5,
        "current_belief_mr_true": 0.25,
        "current_observation_code": −1,
        "current_rewards": [
            [90.0, 45.0, 25.0, 5.0],
            [100.0, 10.0, 20.0, 0.0]
        ],
        "ev.mst": 465.104345236406,
        "ev.rt": 326.710194366562,
        "flagUpdatedRewards": 0,
        "observation_description": "None",
        "observation_probability": 0.0,
        "selected_action": "MST"
    },
    "1": {
        "current_belief_mec_true": 0.94, ...
    },...
}
```

Temporal Model construction

After the JSON trace logs are exposed to the model indexer, the TM construction can take place. The log information about the state of the system is reshaped into the trace metamodel (Figure 5.2) based on the EMF metamodel of figure 6.3 for linking the system goals and decisions to its observations and reasoning. This metamodel is generic for goal-oriented autonomous systems. However, a more specific metamodel for systems that take into account NFRs and their satisfaction to support the system's decision-making can facilitate the history exploration for this domain specific case study. In this regard, the proposed metamodel of Figure 5.2 was extended for systems that uses POMDPs. The proposed extebsuis are shown in Figure 6.3.

The RDM trace metamodel contains the *RewardTable*, which is a lookup table made up of *RewardTableRow*s. The lookup key is the truth value of the *NFRSatisfaction*s for each NFR, and the *Action* under consideration. To produce these Boolean values, the estimated probability of each *NFRBelief* is compared against the matching *RewardTableThreshold*. For instance, the NFR MEC (minimisation of energy consumption) may be considered to

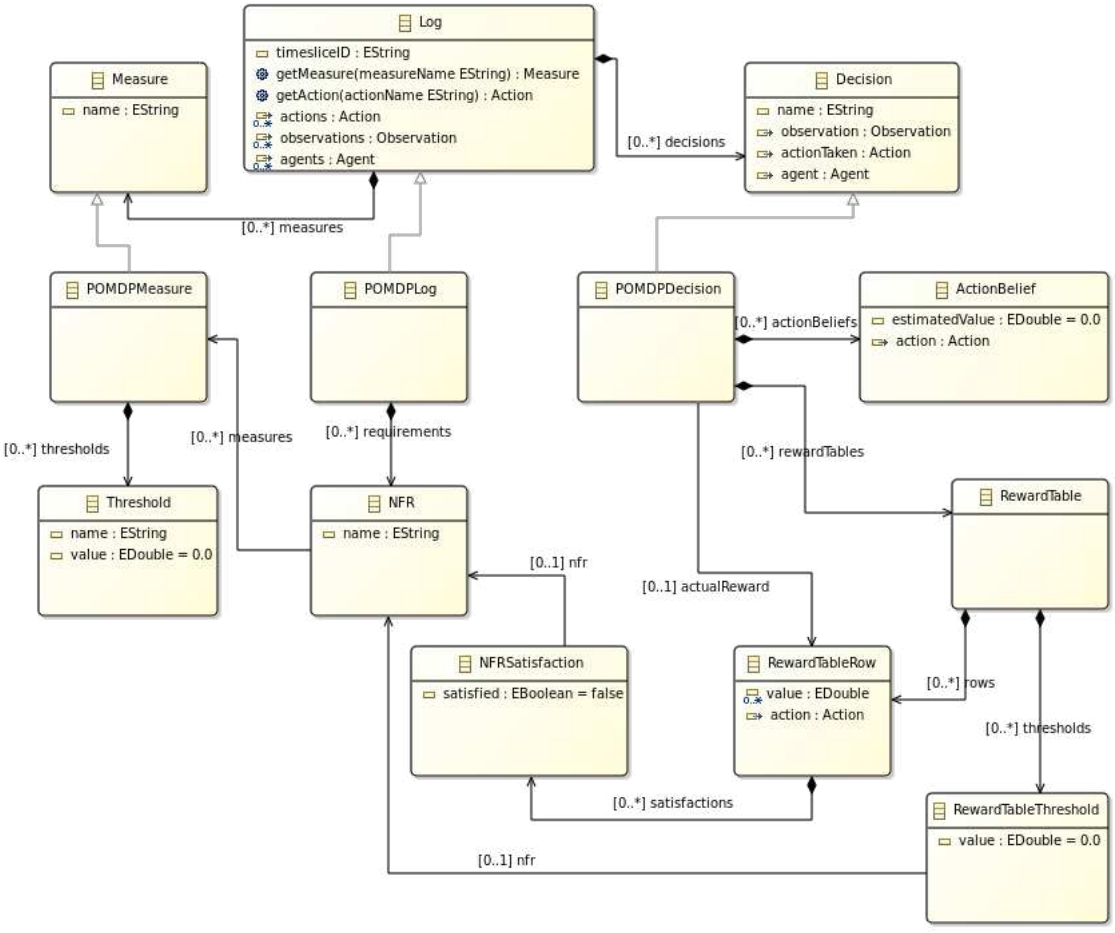


Figure 6.3: Class diagram for the extensions to the core metamodel used to record POMDP-based systems, such as history

be satisfied if the estimated probability is higher than 70% (as stated by the developer's requirements specifications).

Once the trace metamodel is defined, it can be registered in an Eclipse Hawk instance using the guidelines described in its documentation[1]. The main steps are:

1. Create Hawk instance.
2. Define backend: The Greycat TGDB is the backend used throughout this work.
3. Define the parser to transform the SAS log to an EMF model in line with the trace execution metamodels.
4. Select the query language: Temporal EOL which include the time-aware extensions to EOL presented in 5.1.2.
5. Register the metamodel, by providing the .ecore files representing the metamodels in Figure 5.2 and 6.3.
6. Add a connector that will access the exposed JSON logs (e.g., by reading from Git, SVN, customised repositories).
7. (Optional) Add annotations through derived attributes. This step will be further described in 6.2.4.
8. Leave Hawk running in the background as it monitors the JSON logs while querying the TM using the reusable time-aware query language described in Section 5.1.2.

Depending on the approach followed from the proposed research roadmap described in Section 5.2, the monitor will vary, and further steps may be taken. The following sections will describe the experimentation for Levels 1, 2 and 3 of the mentioned roadmap for the RDM SAS.

6.2.3 Level 1: Forensic history-aware explanations in RDM

According to Section 5.2, the first step to achieve automated history-awareness in SAS is to offer *forensic* capabilities. This section shows a description of this level's implementation for RDM. It also describes the scenario that motivates the need for explanations in the RDM SAS. Finally, the experiment results are presented.

[1]Hawk: `https://www.eclipse.org/hawk/basic-use/core-concepts/`

Timeslice	Action	Monitored NFR	Satisfaction	SLA
0	MST	MC	0.91	0.90
1	RT	MC	0.80	0.90
2	RT	MC	0.81	0.90
3	RT	MC	0.84	0.90
4	RT	MC	0.92	0.90

Table 6.1: Example of Long Term Effects (LTEs) in the RDM system

a. Scenario

An interesting aspect to be monitored is the effect of the proactive adaptation exposed by the RDM system [61]. An example is when an apparent bad decision turns out to be a good one in the long run. A decision may be, surprisingly and even unexpectedly, considered by the user, non-ideal at first when the level of satisfaction of an NFR is below the threshold defined in the Service Level Agreements (SLAs) and the action suggested by the system results in a reduction on the level of satisfaction after the decision made. Nonetheless, as the system continues under the same action the level of satisfaction gradually increases until reaching or even exceeding its threshold. These kinds of events are called *Long Term Effects* (LTEs) in the present work. Let's consider the simplified information of timeslice information shown in Table 6.1: the NFRs are monitored over timeslices. The monitored NFR is `Minimization of Cost (MC)`. From ts 0 to 1, RDM decided to change (adapt) the topology from `MST` to `RT`. It ended in a reduction of the level of satisfaction of `MC` that drops below the threshold of the defined SLA, i.e. $0.8 < 0.9$. At first this can be considered a "bad decision". However, for the following ts 2 to 4, the system kept the same topology while the level of satisfaction of the NFR reached and even exceeded its SLA, i.e. $0.92 > 0.9$ shown in the last row.

If a user sees a SAS dip in performance momentarily, they may become anxious about its long-term viability. RDM may have cases like this, as it is proactive and estimates the future trajectory of the system: it may decide to make a decision that is bad in the short-term but good in the long-term. Therefore, these surprising situations required to be explained.

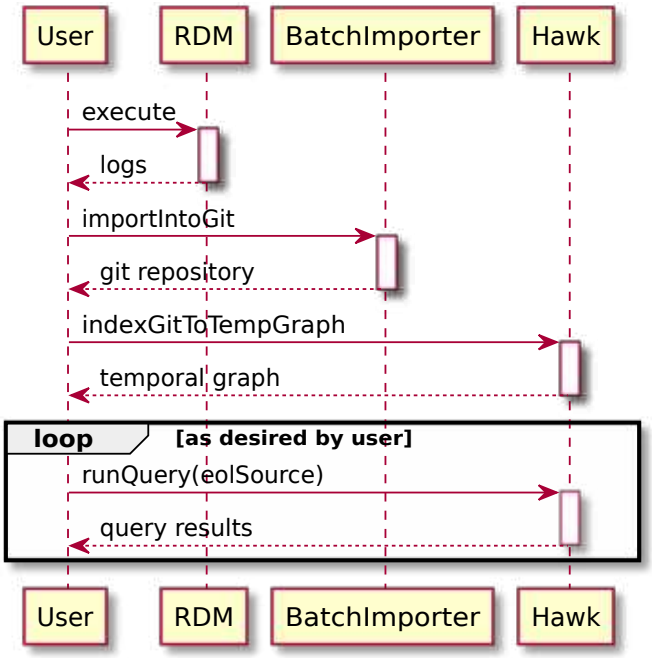


Figure 6.4: UML sequence diagram for interaction between components (RDM case study, level 1)

b. Implementation

Figure 6.4 shows a UML sequence diagram of how the components communicate in level 1. The user asks RDM to run, producing an all-timeslices log. This log is given to a BatchImporter (further described below), which produces a Git repository with one revision per timeslice, containing an XMI file conforming to the metamodel in Figure 6.3 per timeslice. The user then tells Hawk to turn the Git repository into a TM, which can be queried as needed. More details about the implementation are described next:

1. The RDM SAS had been previously run in a different machine through a simulation over 2000 time slices, producing a sequence of entries in JSON format which took 536KB.

2. BatchImporter: A small Java program (381 lines of code) was created to transform the JSON logs into EMF models conforming to the trace execution metamodel Figure 6.3. The batch log importer works by taking the all-timeslices JSON file and reshaping it into a Git repository with a sequence of XMI files conforming to the metamodel in Figure 6.3. The Git repository can then be indexed by Hawk into a Greycat TDB. The Git repository was produced after 48 seconds, taking up 7.3MB of disk space,

and resulted in 1888 commits. Git naturally ignores cases when the model has not changed at all from one time slice to the next.

3. Hawk was instructed to index the full history of the Git repository into a Greycat temporal graph, using its new time-aware updater component. From the second revision onwards, Hawk used its incremental updating capabilities to propagate any differences observed since the previous revision. The Greycat temporal graph over the 1888 commits was produced after 21 seconds, taking up 31MB of disk space.

Git was chosen over SVN or manually time-stamped files (for example, slice1.xmi,slice2.xmi, and so on) because the version of the local folder indexing component in Hawk at the time would index all time-stamped files separately, rather than as a single evolving model. The Git component in Hawk is designed to provide the full history of each file, which produced the intended results. In the future, a version of the local folder indexing component can be created which understands that files time-stamped according to a certain convention are versions of the same model.

Regardless, ignoring those 112 timepoints when the model did not change did not result in loss of information. Indeed, if only Git revisions were found for timepoints 1 and 10, that means that the model did not change at all between timepoints 2 and 9. Therefore, if the state of the model is asked at timepoint 5, the version at timepoint 1 will be showcased. Omitting timepoints which did not introduce any changes can result in significant space savings when changes are infrequent, i.e. in a stable system configuration. This also reduces the number of results to go through in the queries. It can be thought of as a form of compression.

Once the TM is constructed, the history can be queried to extract explanations based on the scenario described in the previous Section 6.2.3. Algorithm 3 (in the appendix Section) was implemented in the Hawk time-aware dialect of EOL. The algorithm can find the examples to present to the user on demand as a simple plot of estimated requirement satisfaction levels over time. This takes the monitoring beyond a passive set of listings and figures, to allowing users to ask questions or request examples of particular relevant nuances of the SAS. The implementation of this and additional EOL queries can be found in the project repository[2].

[2] `https://gitlab.com/sea-aston/hawk-rdm/-/tree/dronesLocal/bundles/uk.ac.aston.mrt2018.queries/LTL_QueriesRDM_3NFRs`

Listing 6.2: Excerpt of output from Algorithm 3 about long term effect of immediate actions.

```
[..., [719, Minimum Spanning Tree Topology, Maximization of Reliability,
    0.852294921875, [[720, Minimum Spanning Tree Topology, Maximization of
    Reliability, 0.840590259674336], [721, Minimum Spanning Tree Topology,
    Maximization of Reliability, 0.935284515844998], [722, Minimum Spanning Tree
    Topology, Maximization of Reliability, 0.94331412096612]]],
[1597, Minimum Spanning Tree Topology, Maximization of Reliability,
    0.835166769728076, [[1598, Minimum Spanning Tree Topology, Maximization of
    Reliability, 0.0.824842465933626], [1599, Minimum Spanning Tree Topology,
    Maximization of Reliability, 0.934522400804845], [1600, Minimum Spanning Tree
    Topology, Maximization of Reliability, 0.935870208967967]]], ...]
```

c. Results

Listing 6.2 shows an excerpt of the examples found by the query using the time-aware query language, which are shown to the user in a human-readable way. One of the detected sequences started at timeslice 719, when RDM decided to use the Minimum Spanning Tree (MST) topology. As an immediate consequence, a reduction on the satisficement level of the NFR Maximization of Reliability (MR) is observed: from 0.85229 (timeslice 719) to 0.84059 (timeslice 720). However, the satisficement grew during the subsequent timeslices, until it exceeded its threshold in timeslice 722. Similar situations were observed in different timeslices such as timeslice 1597. 29 such situations were found during the 2000-timelice run. This shows us that decisions with apparently immediate negative effects, may produce the required expected increase of the satisficement level of the NFRs in the long term. Developers and end users need to be aware of this kind of behaviour, which otherwise could be found unreasonable at first.

This type of query allows the system to explain why it took a decision and why it is showing the current behaviour. For this specific case, the insight gained through the temporal query would make the user aware of the use of time windows within the decision making process, and would prepare the user to better interact with the SAS if required.

The scalability of this approach is limited by the fact that such a log may grow to be very large: indeed, naively parsing a log which is in the gigabytes may tax the memory capacity of the computer. For the parsing problem, one approach would be to index not a single JSON file, but rather a database (e.g. a collection of Mongo documents) or a stream of events. This would still not prevent the temporal graph from growing too large. For very

long runs, compressing and/or pruning the history may be needed: for instance, it may only keep the last X timeslices (time windows), index only one out of every X timeslices (sampling), and/or keep only versions matching certain situations of interest (filtering). The risk with these strategies is that queries would be limited in scope (with time windows), or would become approximate (sampling and filtering). Studying the impact of these strategies in the query results for long-running systems will be explored in the following sections of this thesis.

The trace metamodel assumes that the system follows a reward-oriented strategy around non-functional requirements. This suggests that queries written against this metamodel may still be reusable beyond R-POMDP, and could work with other types of self-adaptive algorithms (e.g. those based on reinforcement learning). On the other hand, if the SAS follows a different type of strategy, it could still reuse the *Decision* / *Observation* / *Measurement* concepts from the core metamodel described in Figure 5.2 in Section 5.1.1.

6.2.4 Level 2: Live History-aware explanations in RDM

Level 1 has focused on *after-the-fact analysis*, taking a sequence of system models to turn it into a single temporal graph, which can subsequently be queried. It presents the advantage that it does not require any changes in a system that is already producing its own logs in a machine-parsable format. However, users may want to demand questions about the system while it is running, and not just after an event has happened. The next step on the defined research roadmap (Section 5.2), is to enable *online* explanatory capabilities. Extending concepts and components from Level 1, this subsection will describe the implementation of Level 2(Live history-aware explanations) in RDM.

a. Scenario

In order to test the feasibility of the proposed TM approach for both online and offline analysis, the scenario of Level 1 (Subsection 6.2.3) was repeated to find LTEs at runtime. In this scenario, RDM talks directly to Hawk for storing the history of the system. To test the proposed approach, trace logs produced by RDM were monitored while looking for LTEs. The aim of the experiment was to successfully detect these LTEs and keep the stakeholders informed of the otherwise surprising behaviour.

b. Implementation

The proposed approach takes advantage of the fact that Hawk can be run as a network service [58], with the ability to run queries at any time via its Thrift-based API[3]. Therefore, both RDM and Hawk can be running at the same time. The query to detect proactive adaptations (see Algorithm 3), was implemented in the Hawk time-aware dialect of EOL. The EOL query, the incremental importer and the query tool are available in the project's Gitlab repository[4].

Hawk was extended with a component that can read the single-timeslice JSON log produced by RDM, and reshape its contents into an in-memory model while conforming to the trace metamodel of Figure 6.3. This in-memory model can be given directly to Hawk, while significantly reducing overheads. In addition, the RDM SAS has been extended with the ability to notify Hawk when to update its temporal graph, by sending Hawk a message through the same Thrift-based API. This also significantly reduces overheads compared to spawning new Java subprocesses. When told to synchronise, Hawk will compare the trace model represented by the JSON file with the latest version in the temporal graph, to create a new timepoint by applying the differences. The new timepoint then becomes available for querying done by users.

Figure 6.5 shows a UML sequence diagram about the communications between components in level 2. Hawk is assumed to be running and set up, having registered the trace metamodel and the repository with the JSON file to be indexed. The user then starts RDM. At the end of each timeslice, RDM will update the JSON file with the information from the timeslice, and will ask Hawk to update its graph from it. Hawk will acknowledge the update, and then RDM will continue on to the next timeslice. At any point in time, the user can run a query based on the current state of the temporal graph in order to obtain an explanation about how it got there.

c. Timeline annotation

Level 2 can take advantage of the timeline annotations introduced in Section 5.1.2. This allows the system to jump directly to situations of interest without having to scan the full history of the temporal graph. Using this new capability only requires minor preparations.

[3] `https://archive.is/lJwUP`
[4] `https://gitlab.com/a.garcia-dominguez/hawk-rdm`

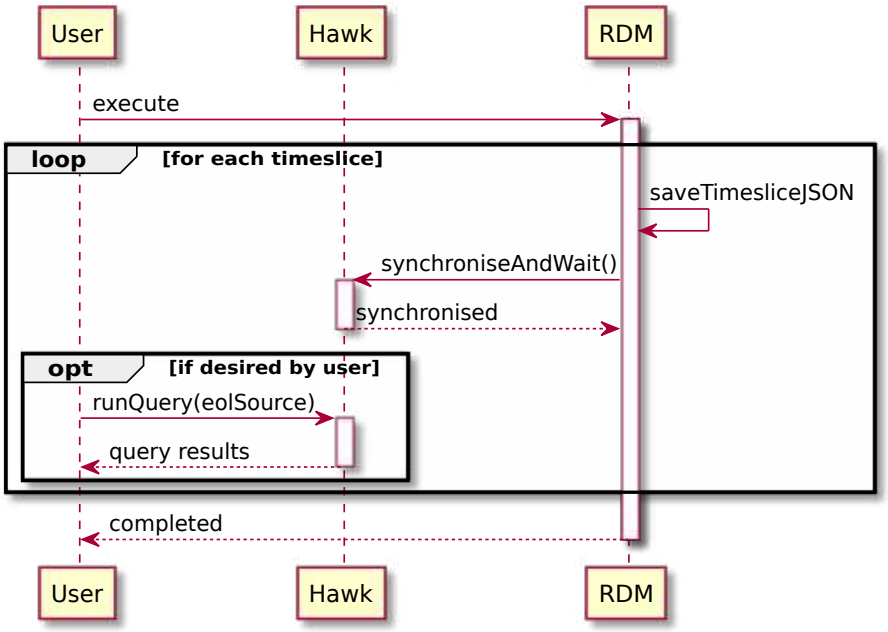


Figure 6.5: UML sequence diagram for interaction between components (RDM case study, level 2)

Before RDM starts, the user will signal Hawk about which situations it should monitor. Once RDM has started, specifically when Hawk notices that a new timepoint matches a situation of interest, it will subsequently record it to therefore provide fast retrieval of the timepoint through the *whenAnnotated* operation.

Listing 1 shows the EOL query implemented using the proposed temporal query language and timeline annotation. The LTE query was written with the full selection of primitives from Table 5.1. The query spans 39 lines of EOL, and it has three sections:

- An operation which detects "apparently bad" decisions in the short-term. This operation does not use any temporal primitives: it only compares pre/post-action satisficement levels and the threshold. This pattern is defined using timeline annotations (line 3).

- An operation which computes information from an version interval where satisficement improved from a certain decision. This version uses the primitives to define an interval of versions of the belief since the timeslice after the change (if it exists), until the timeslice before the action changed or satisficement dropped (lines 9 to 26).

- The main body of the query (Lines 1 to 7). This section finds the belief for the MR NFR and the potentially bad decisions, computes the interval information for each

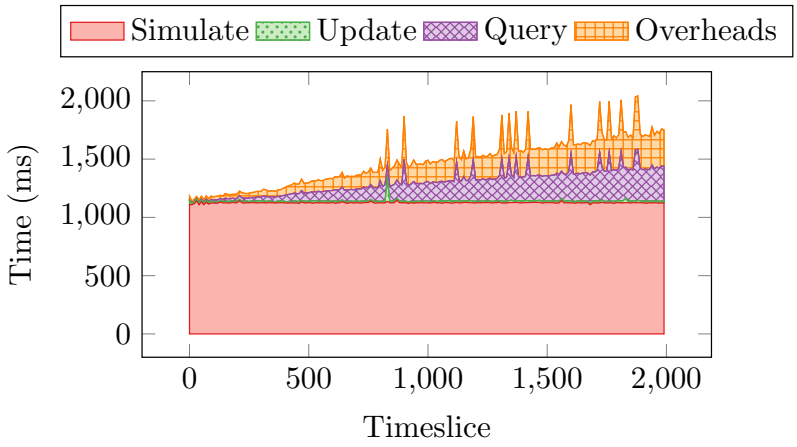


Figure 6.6: Stacked area plot with execution times for RDM SAS simulations in milliseconds, by timeslice and phase, for the queries running without annotations.

decision, and discards decisions with no such intervals. In the first timeslices, there may not be any such decisions: the current definition of `whenAnnotated` will return an undefined value in that situation. To cover against that situation, `ifUndefined` is used to return an empty collection in that case.

In order to study the impact of the temporal assertions and the scoping primitives on conciseness, the query was revised so it would only use the basic version traversal primitives. This required two changes. The first one was simple: `.when().ifUndefined().versions` in the main body was replaced with `.versions.select`. The second change was in the interval computation: the declarative use of `sinceThen` and `before` had to be replaced with a `while` loop, which also accumulated the information to be returned about the interval.

d. Results

The queries for finding LTEs with and without timeline annotation were performed. Figure 6.6 includes the execution times for a simulation of RDM over 2000 timeslices without timeline annotation. This stacked area plot shows the different stages: the simulation of a timeslice, the update of the temporal model within the Hawk indexer, and the full execution of the query. The query times include client-server communication overheads. The simulation times ranged from 1087 and 1148 milliseconds. Temporal graph update times represented the 1.36% in average of the total time and remained stable. Query invocation times grew over time, together with the length of the history of the temporal graph. Query times came to represent up to 14% of the execution time in average. This

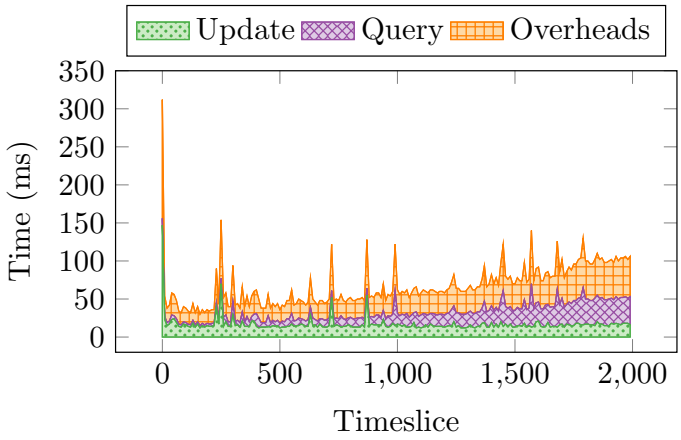


Figure 6.7: Stacked area plot with execution times for RDM SAS simulations in milliseconds, by timeslice and phase, for the queries running with annotations. "Simulate" times are excluded due to small values in the other series, being the same as in Figure 6.6.

is because the query has to go through each time-point in the system history every time the query is executed. For example, at timeslice 500 the query needs to consider all the 500 time points that the simulation has gone through and for the timeslice 2000 the query needs to consider 2000 time-points in history.

One way to attenuate the impact of querying every point in history in the system's performance is to use timeline annotation. This is done in such a way that a situation of interest would be defined in advance, and matching timeslices would be *annotated* during execution. Then, instead of going through the whole system's history, the query would jump to those annotations. For this experiment, the situations to be tagged were the "bad decisions" mentioned in Section 6.2.4. In other words, when the system changed (i.e. adapt) the topology and this action ended in a reduction of the satisficement level of the NFR. Figure 6.7 shows the different stages, except for the simulation time that is the same as in Figure 6.6. Updates represented 1.47% of the total times on average, similar to the first experiment, only presenting a initial peak of 147 ms. This shows the time of setting up the indices for the *annotation*. On the other hand, query times presented a significant improvement in the simulation time, from representing up to 14% of the total time, to only 1.5% of the total time on average.

In total, the 2000-timeslice simulation took 43.54 minutes without timeline annotation and 38.36 with timeline annotation. The simulation time without the *Level 2* capabilities would have been 37.49 minutes. It can be concluded that using timeline annotation, the reduction in the system's performing time can be kept to 2–3% due to overheads. Figure 6.8

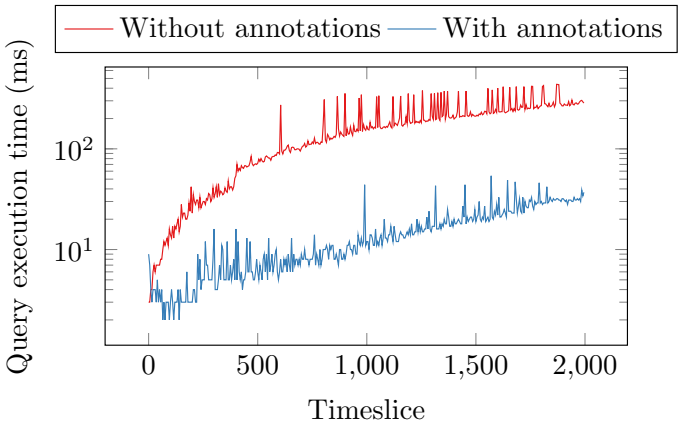


Figure 6.8: Raw server-side execution times of EOL query implementing Algorithm 3 in milliseconds, by timeslice.

shows the raw execution times for the EOL implementation of the query in Algorithm 3 for both approaches, using a logarithmic scale for the times. These execution times exclude the wait for synchronisation with the server and the network overheads, which dominate most of the time in the "Query" series of Figures 6.6 and 6.7. Query times without annotation ranged from 2ms to 486ms, with a median of 162ms. For the timeline annotation approach, times ranged between 2ms to 54ms with a median of 12ms, which shows the advantages of timeline annotation. The peaks can be attributed to the natural variability in inter-process communication times, and overheads.

6.2.5 Level 3: Externally guided history-aware decision making. Introducing the human-in-the-loop in RDM

Users may feel that the decision-making process of SAS is oblivious to the user's own decision making criteria and priorities. Inevitably, users may mistrust or even avoid using the system. Furthermore, SAS could benefit from human involvement for satisfying stakeholders requirements. Integrating humans in this enclosed loop is an ongoing research challenge as these systems are in principle foreseen to be autonomous [35]. The previous levels (Sections 6.2.3 6.2.4 on history-aware explanations have focused on the analysis of decision-making over time with no active role played by the human. Level 1 offered forensic explanations. In contrast, Level 2 provided on-line explanations of the decision making for monitoring purposes. This subsection addresses the third stage proposed in 5.2, using history-aware explanations to enable a human-in-the-loop approach where users take an

active role in the decision-making of the SAS.

a. Scenario

The RDM SAS has been configured with Service Level Agreements (SLAs) for the satisfaction levels of the NFRs MC, MR and MP. These SLAs guide the system on its adaptive decision-making based on values defined by the developer. The defined SLAs were: P(`MC`=True) $\geq$ **0.8** (the observed level of satisfaction of `MC` is greater than or equal to 0.8 out of 1), P(`MR`=True) $\geq$ **0.9**, and P(`MP`=True) $\geq$ **0.75** respectively. Initial stakeholders' preferences about the NFRs and adaptation topologies have also been provided. They are represented by the Reward/Penalty node shown in Fig. 6.1. In RDM, the initial preferences provided by the domain experts favour the MST topology under *stable conditions* [61]. Stable conditions represent a system context where the average satisfaction levels (since the system started the execution) of the NFRs meet their SLAs. This work will showcase how a user can take a more active role in the decision-making based on history-aware explanations provided by the RDM SAS under unexpected contexts (i.e. *unstable conditions*) detected at runtime.

b. Implementation

This PhD work discusses that the human role in the decision-making loop of a SAS can be seen as either passive (i.e. observing and understanding the decision-making process) or active (i.e. steering the decision-making process). Ideally, the decision-making should take into account the execution history [59]. Moreover, the understanding of the decision making by stakeholders should also include the system's reasoning history [119]. Therefore, a SAS should (i) offer access and retrieval to historic data about its behaviour, (ii) track over the time the reasons for its decisions so they can be used to explain and further inform end users and other stakeholders, and (iii) if an active role by the user is required, the SAS should also provide *effectors* to therefore empower human stakeholders to steer the decision-making while being informed by (i) and (ii).

The proposed framework has been extended to support both explanation capabilities and user interaction towards introducing the human-in-the-loop of SAS. A graphical user interface (GUI) including effectors that allow the user to provide feedback to the SAS at runtime was developed. Figure 6.9 shows the designed GUI. Through graphical explana-

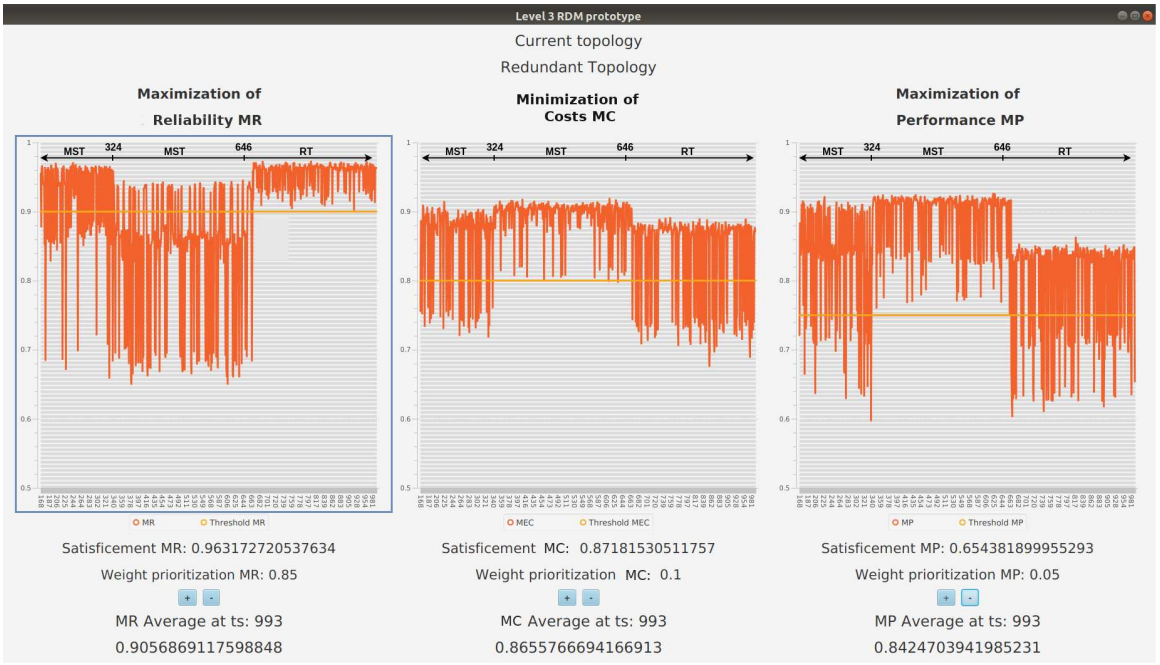


Figure 6.9: GUI showing the system's historical behavior. At time slice 646, the user set a higher priority to the `MR` NFR (left chart).

tions, users are able to improve their mental model about the system's current behaviour. If system behaviour does not agree with the mental model of the user, changes can be requested. Since the decision-making in the RDM is driven by the satisfaction levels of the NFRs, the effectors exposed by the system, the buttons "+" and "-", will allow the user to manipulate the system's preferences (in this case, the relative priorities or weights of the NFRs).

The sequence diagram in Figure 6.10 shows the communication between the various components for level 3 (human-guided history-aware decision-making with explanation capabilities) of the RDM case study. The user initialises the simulation and the GUI assuming that Hawk is running, the trace metamodel is registered and the log file to be indexed is defined. At the end of each timeslice, the system will update the log file with the corresponding information. The GUI is fed by the temporal graph that is being built in Hawk. At any point in time, the user can run a query based on the current state of the temporal graph in order to obtain an explanation about how it got there. Predefined queries can be implemented to run periodically, or the user can develop new queries and implement them using Hawk's interface. If the user considers that the system is not fulfilling the user's preferences or that an external action could improve the system performance, reconfiguration

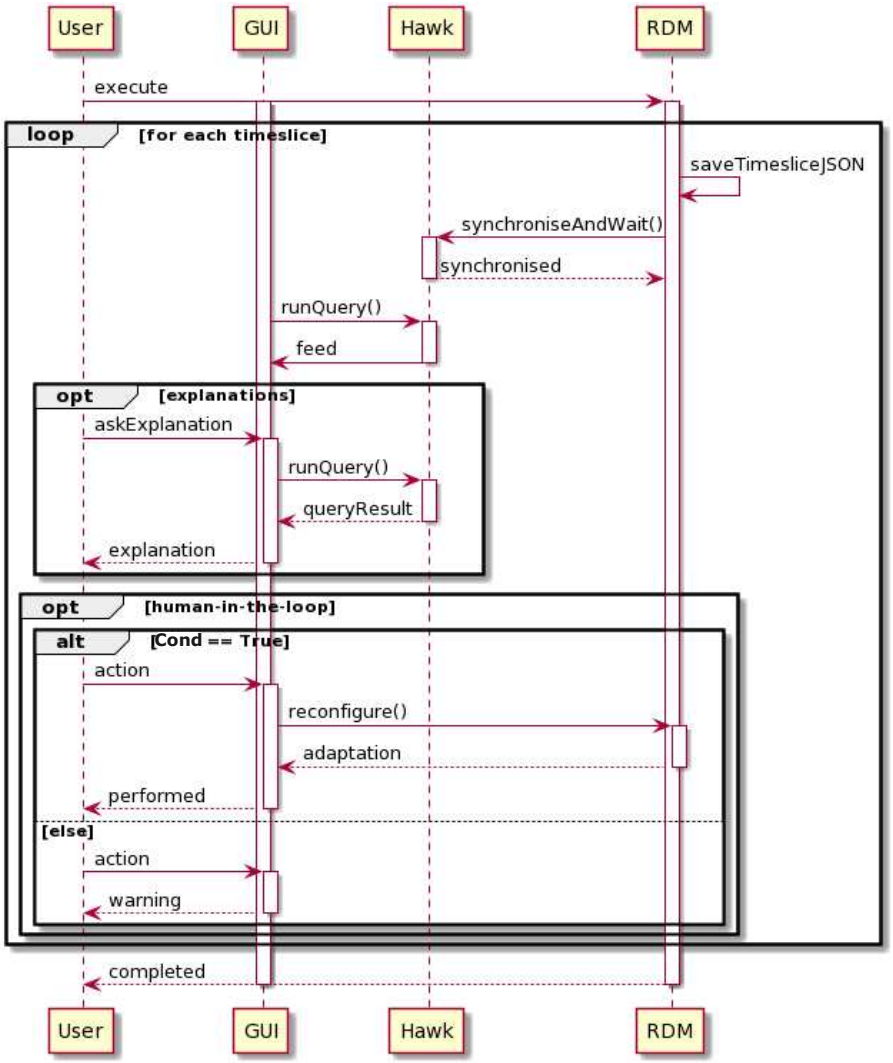


Figure 6.10: UML sequence diagram for interaction between components (RDM case study, level 3)

can be made using the effectors/controls in the GUI. A reconfiguration in the system will be made in compliance of a developer defined condition.

c. Results

The information provided can be used to generate visual explanations such as those in Figure 6.9, which summarise the behaviour of the RDM SAS as it runs. It is observed that initially, the satisfaction levels of the NFRs `Minimization of Cost (MC)`, `Maximization of Reliability (MR)`, and `Maximization of Performance (MP)` are in general over their Service Level Agreements (SLAs), with some values below their thresholds, but this noise is considered to be normal for the system. Later, from time slice 324 (See Figure 6.9,

Maximization of Reliability), a period of consecutive and unexpected data packet losses while using the `MST topology` reduces the observed reliability of the system. Data packet loss may represent link failures in a RDM, which can be caused by problems with the equipment (e.g. failures in a switch or router, or power failures). Despite the new detected conditions and based on the initial RDM configuration, the selected topology continues to be MST (See Figure 6.9, Maximization of Reliability: time slice 324). Specifically, under the current context, initial stakeholder preferences are not suitable anymore as they continue favouring the use of a topology that does not contribute to improve the satisfaction level of MR, which is mainly under its tolerance threshold (See Fig. 6.9, Maximization of Reliability: time slices 324 - 646). The preferences should be eventually reassessed and updated to assign higher importance to NFRs with poor satisfaction levels (e.g. MR, the reliability of the system) and to improve the selection of the topology in the RDM SAS.

Complementing the scenario presented above, through the integration of the human-in-the-loop based on the RDM components, the user is able to explore the history and steer the decision-making. History-aware explanations are presented to the SAS developer through the GUI. Under the current runtime context, special attention is paid to: (i) the NFRs satisfaction levels from time slice 324 onwards, (ii) the current preferred topology (MST), and (iii) the current preferences about the NFRs. These explanations help the developer refine their "hypotheses" or mental models about the current state of the system.

Next, based on the information provided by the graphical explanations, the user is allowed to potentially improve the current behaviour of the system. If the user considers that the system is not fulfilling the intended behaviour or that an external action could improve its performance, a reconfiguration can be made using the effectors/controls in the GUI. In order to reconfirm the external action selected by the user, Figure 6.11 shows a *pre-adaptation explanation* of how relevant the effectors "increase the priority of `MR`" can be (the + button under MR). After the change is applied, Figure 6.9 shows how the satisfaction level of `MR` increases from time slice 646 onward as a result. The satisfaction levels of `MC` and `MP` went down, but they still met their SLAs.

The experiment has covered how some unforeseen dynamic contexts may affect negatively the NFR satisfaction levels when initial assumptions, e.g. stakeholder preferences, are not updated in response. The "NFRs *without* update of preferences" series in Figure 6.12 shows this behaviour in the satisfaction levels of the NFRs from time slice 324 to time

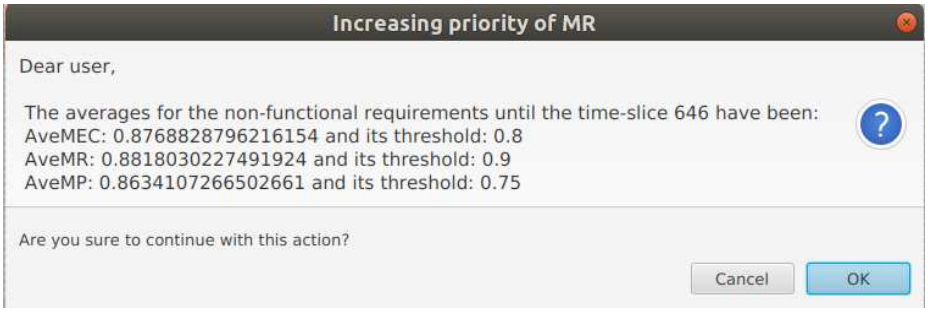


Figure 6.11: Textual pre-adaptation explanation from the system at time slice 646, when user shows interest in increasing priority of `MR`.

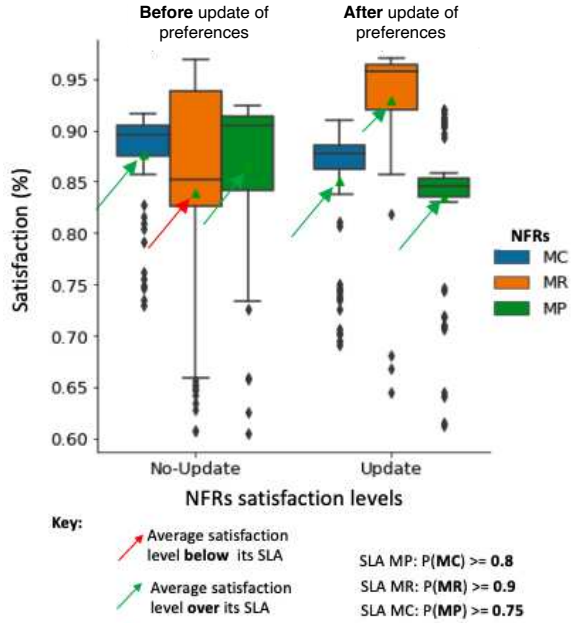


Figure 6.12: NFRs average satisfaction levels before and after human interaction

slice 646. The average satisfaction level of MR is always below the SLA given in the initial stakeholder preferences, despite the new detected context, and the RDM SAS continues favoring the MST topology as shown in Figure 6.9. In contrast, by including the human-in-the-loop in the decision-making of the RDM SAS, it is possible to improve the general performance of the system and the NFR trade-offs. Going back to Figure 6.12, "NFRs *with* update of preferences" shows this new behaviour from time slice 646 onwards. It can be seen that after the user intervened, the average satisfaction level of MR started meeting its SLA. There is also a slight reduction on the satisfaction levels of MC and MP, but they still met their SLAs.

6.3 Discussion

This chapter has focused on the implementation of Levels 1, 2 and 3 of the proposed research roadmap (Section 5.2) for enabling history-aware explanations in the RDM SAS case study. Using *temporal models* it is possible to access historical information for the explanation generation and presentation discussed in Section 2.2. Users (specially, SAS developers) can gain insights about the system's decision-making and can interact with RDM either passively or actively.

Level 1, (forensic history-aware explanations), focused in enabling black-box analysis after the system finished its execution. The main infrastructure was constructed to keep track of RDM SAS decision-making process based on it logs. The collected information was reshaped and stored in a TM conforming to the proposed trace metamodel of Figure 5.2 (Section 5.1.1) and its extension for RDM, Figure 6.3. Through the temporal query language it is possible to extract information from the TM. It is an evolving graph representation that depicts changes in metrics and relationships between entities of the previous mentioned metamodel. This level acts as a base from the upper levels.

The experimentation on Level 1 (Section 6.2.3) showed the feasibility of the approach for forensic analysis. However, query execution took longer as the history grew. This is expected as the queries need to visit more points in time when the history expands. In order to tackle this behaviour, timeline annotation (defined in Section 5.1.2) can be an option. It consists of creating tags in certain versions of the TM that the query will visit instead of the whole history. This approach requires to create annotations at runtime which logically is part of Level 2.

Level 2 enabled the analysis of the RDM SAS at runtime, extracting explanations while the system is running. In order to facilitate this interaction, Hawk's Thrift-based API was used. RDM notified Hawk when there is a new version of the Log and it will be added to the TM considering the copy-on-write capability. The user was able to run queries on demand or a predefined query can be programmed to run periodically. When using timeline annotations, the user defines specific situations of interest that will be tagged in the TM when they occur during RDM SAS execution. Once a query is performed, either on-demand or periodically, it will only visit the tagged time-points.

The experiment of Section 6.2.4 showed the functionality of Level 2. The system was able

to extract explanations related to the specific situation of interest (i.e., the long term effects) at runtime and periodically. Timeline annotations reduced querying times considerably from representing the 14 % of execution time to the 1.5% which showed the feasibility of the approach.

Level 3 built on the previous two levels, for constructing and presenting explanations and adding the functionality for an external entity to provide feedback to the RDM system at runtime. Thus, switching from passive runtime monitoring to active runtime monitoring. Through the information provided by the presented explanations, the users (i.e. developers) are able to understand the system and feel confident about making changes to the system at runtime through the use of effectors displayed in the developed GUI.

Through the experimentation in Level 3 (Subsection 6.2.5), it has been shown how human stakeholders, as external entities, are able to evaluate and update the parameters of a SAS based on live explanations of the SAS behaviour, participating in the tradeoffs between the NFRs in a SAS. The explanations presented in the experiment are based on the evolution of a metric (e.g., NFR average). However, explanations based on relationships between metrics and events spanned over time can be obtained by exploiting the full potential of the TMs as shown in the previous Sections 6.2.3 and 6.2.4. Changes in the system are only allowed to take place if they meet defined criteria or conditions defined by the developer. For the seek of experimentation, the condition defined by the developer is always set to true in this implementation. An example of criteria to be met can be that an update can only take place when the system is not performing critical tasks. Another option could focus on the human capabilities for performing a task as the work presented in [92, 93]. The authors use the Opportunity-Willingness-Capability (OWC) model to define when a user fulfils defined conditions for interacting with a SAS. In this thesis, an explanation is deemed to be useful if the recipient understands the system's behaviour and feels confident to interact with it either passively or actively. However, the evaluation of the explanations reception and the impact of human interaction in SAS (for example through the OWC model) are outside of the scope of this work. Evaluating how users understand the explanations and the full impact of their interactions with SAS (for example, through the OWC model) is outside the scope of this thesis.

This chapter has described how TMs are able to represent the RDM SAS history. TMs combined with the temporal query language, allow the generation and presentation of ex-

planations to users. From the classification of explanations from Section 2.2.1, the approach corresponds to the *post-hoc* type where the explanations are not part of the system decision-making. The scope of the explanations has been *local*, describing specific situations during the SAS execution. The targeted audience are *humans*, specifically domain experts and developers. Finally, both textual and graphical presentation methods have been designed, matching the targeted audience expertise.

The experimentation has shown the feasibility of the approach. Different aspects have to be taken into account as the computing resources. Using timeline annotations improves the system performance for presenting an explanation however it does not tackle the TMs' growing size for data intensive systems. Different optimisation strategies are required and will be tested in the following chapter. These strategies can include: *sampling*, for only storing logs at a certain rate; *time windows*, for focusing on the last n time-slices; or *event-oriented processing*, for storing logs only when certain events happen. These strategies would allow to further reduce the storage and processing overheads imposed by the addition of history awareness and will be explored in the next chapter.

Chapter 7

Scaling up Temporal Models through Event-Driven Monitoring for explanations

---

The work presented in this chapter has been adapted from the following publications:

[120] J. M. Parra-Ullauri, A. Garcia-Dominguez, N. Bencomo, S. Yang, C. Zheng, C. Zhen, J. Boubeta-Puig, and G. Ortiz. Event-driven temporal models for explanations - etemox: explaining reinforcement learning. *Software and Systems Modeling*, 2021. doi: 10.1007/s10270-021-00952-4.

[122] J. M. Parra-Ullauri, A. García-Domínguez, and N. Bencomo. From a series of (un) fortunate events to global explainability of runtime model-based self-adaptive systems. In *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 807–816, 2021. doi: 10.1109/MODELS-C53483.2021.00127.

[121] J. M. Parra-Ullauri, A. García-Domínguez, J. Boubeta-Puig, N. Bencomo, and G. Ortiz. Towards an architecture integrating complex event processing and temporal graphs for service monitoring. In *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, pages 427–435, 2021.

[165] C. Zheng, S. Yang, J. M. Parra-Ullauri, A. Garcia-Dominguez, and N. Bencomo. Reward-reinforced generative adversarial networks for multi-agent systems. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2021.

Part of the work in this chapter is also under review at MLSys 2023

7.1 Motivation

The previous chapter (Chapter 6), presented an MDE solution for traceability in SAS using TMs. Using the proposed approach, it was possible to present history-aware explanations about a running SAS, its decisions and behaviour. The approach allowed for explanations in both cases: interactive diagnosis (i.e. at runtime or during execution) and forensic analysis (i.e. after the system has finished its execution), based on the trajectory or history of the execution. These explanations can help someone monitoring the system to analyse and trace past actions, allowing to fix potential faults and fostering users' trust in the SAS. However, some trade-offs emerged when the system's history increased, particularly around scalability. Scalability issues in MDE can be split into the following categories [8]:

- Model persistence: storage of large models; ability to access and update such models with low memory footprint and fast execution time.
- Model querying and transformation: ability to perform intensive and complex queries and transformations on large models with fast execution time.
- Collaborative work: multiple developers checking out a part of their model and querying or editing it, as well as being able to commit their changes successfully.

In the previous chapter, through the use of time-line annotation (Section 6.2.4), it was demonstrated how the approach offers substantial benefits regarding model querying, but at the cost of disk space (model persistence). Nevertheless, these costs can be prohibitive when dealing with data-intensive systems, as it is the case of AI-based systems, where the volume and complexity of the data can grow considerably.

In this chapter, additionally to TMs, the proposed approach integrates the EDM technology, called Complex Event Processing (CEP) [98] for rapid detection of situations of interest. CEP is used to tackle the challenges associated with data-intensive systems and model persistence. It serves as a *real-time filter* that selects relevant points in time that need to be stored in the TGDB as runtime models. The criteria for storing the system's

history can be configured through event patterns on a CEP engine. For example, a certain data rate can be imposed, or the history may only keep points in time where certain conditions are met instead of the full history, saving memory resources and disk space.

This chapter applies an implementation of the proposed approach to a case study from the domain of AI-based SAS: Autonomous Airborne Base Stations (ABS). The section starts with the presentation of the ***ETeMoX*** (**E**vent-driven **Te**mporal **Mo**dels for e**X**planations) framework that combines TMs and CEP. Next, a description of the ABS system is presented. This is followed by an outline of the experimental setup of *ETeMoX* for implementing levels 1, 2 and 3 of the proposed research roadmap (Section 5.2) in the ABS SAS. Finally, the results are discussed.

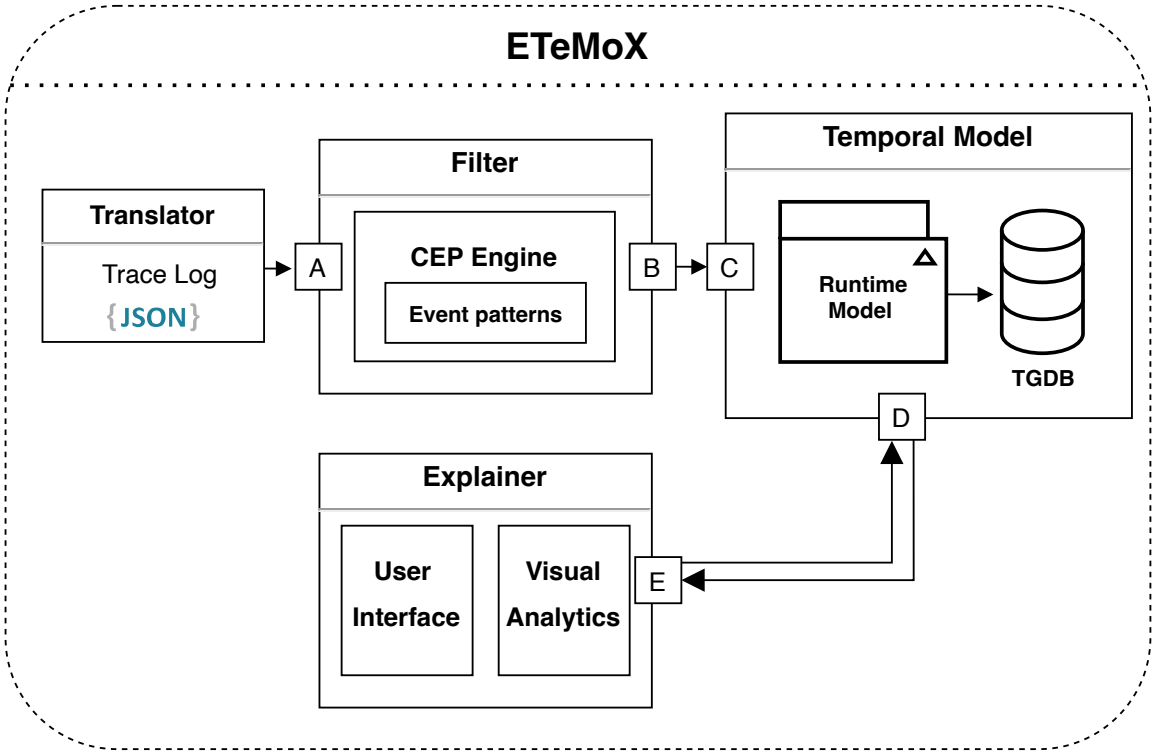


Figure 7.1: Event-Driven Temporal Models for Explanations (ETeMoX) architecture.

7.2 ETeMoX: Event-driven Temporal Models for eXplanations

This section presents the architecture of ETeMoX, which integrates CEP and TMs to support the generation of explanations for AI-based goal oriented systems. Based on the

categorisation from Section 2.2.1, the aim is to build an architecture for model-agnostic post-hoc explainability, using the benefits of EDM and MDE. Understanding what the system did requires the SAS to: i) track its own decision history, and ii) to explain those decisions to the users coherently. Both requirements are targeted by *ETeMoX* while also considering the scalability concerns mentioned in the motivation of this chapter. Figure 7.1 depicts the proposed architecture that is conformed by four main components: *Translator*, *Filter*, *Temporal Model* and *Explainer*. These will be described in detail next.

7.2.1 Translator component

The proposed implementation decouples the decision-making processes in the SAS system from the generation of the explanations (i.e., post-hoc explainability). The translator component receives data streams with execution traces. The traces (Logs) contain information related to the observations made by the SAS about its decisions, actions, states, rewards, and environment. The monitored system collects and exposes the data streams to the translator component through a message broker. An example of a broker is the open source Eclipse Mosquitto MQTT message broker [95]. This broker uses a publish-subscribe messaging pattern, where messages are published according to a set of topics and users subscribe to the topics of their interest. The log can follow structured (JSON / XML) or unstructured (plain text) formats: JSON has been selected for this implementation. This JSON log containing unprocessed data is converted into the data format required by the CEP engine, and then inserted into the Filter component for processing ("A" in Figure 7.1).

7.2.2 Filter component

This component performs the transformation, processing, analysis and routing of data from the Translator component to the Temporal Model component. The main element in this component is a CEP engine for event capture, analysis and response (for more details, see Section 4.2.2). The Esper CEP engine[1] has been selected, as it is mature, has an active user community and is known to scale well with demand. Esper offers a Domain Specific Language (DSL) for processing events, the Esper Event Processing Language (EPL). The Esper EPL is similar to SQL but extended with temporal, causal and pattern operators, as well as data windows. Esper processes and correlates the simple events coming from the

[1] https://www.espertech.com/esper/

Translator component, aiming to detect in real time situations of interest that will match the filtering criteria. Compared to the CEP engines mentioned in Chapter 4, Esper is widely regarded as having a more user-friendly experience due to its simple and intuitive query language that enables developers to quickly and easily define event patterns and queries. On the other hand, Flink and InfoSphere can be more challenging for developers who are unfamiliar with distributed computing, as it has a steeper learning curve.

In a CEP engine, situations of interest are described through *event patterns* deployed into a *rule engine* (Figure 4.1, Section 4.2.2). Developers define the focus of interest, i.e. the subset of the data that will be recorded in the TM. Event patterns are implemented in Esper EPL and deployed to the Esper engine. When the filtering conditions are met (i.e. pattern matches are detected), the engine automatically generates complex events that collect the required information, and sends them to the Temporal Model component. The communication from the Filter component to the Temporal Model component ("B" to "C" in Figure 7.1) is performed using a message broker similar to the one employed by the Translator component.

7.2.3 Temporal Model component

The incoming complex events containing the log information about the state of the system are reshaped into the trace core metamodel of Figure 5.2 (Section 5.1.1) for linking the system goals and decisions to its observations and reasoning. This metamodel is generic for goal-oriented SAS and can be extended for specific domains as shown in the previous chapter (Section 6.2).

Within the Temporal Model component, the runtime model based on the core metamodel will then be used to update the TGDB, creating a new snapshot at the current point in time: all relevant versions are kept. A model indexer is used to automatically compare the runtime model as an object graph against the current version of the temporal graph. It creates a new version which only updates the temporal graph where needed, for efficient storage. Specifically, *ETeMoX* uses the previously mentioned Eclipse Hawk, which operates on Greycat temporal graphs. By using TGDBs, it is possible to track the evolution of certain metrics at each node, as well as the changes in the relationships of the various entities in the system, or their appearance and disappearance.

7.2.4 Explainer component

This component is where the explanations are constructed and presented. The explainer component can run a query on the TGDB using time-aware query language of Section 5.1.2, an extension of the Epsilon Object Language (EOL) to define temporal patterns that traverse the history of a model. The result of this query contains the information that will be used to construct the explanations. These explanations could be presented in textual or graphical ways, e.g. plots of various kinds, yes/no answers, or specific examples of matches of a certain temporal pattern.

In relation to the explanation phases defined in Section 2.2, *ETeMoX* tackles the first two: i) the *explanation generation* is the construction of the causally connected TGDB (performed on the previous component), and ii) the *explanation communication* is the extraction of the information using the temporal query language (what information will be provided) and the presentation of explanations, whether textually or graphically (how will it be presented).

In order for an explanation to satisfy its recipient, it needs to be expressed in a way that is easy to understand for that recipient. Therefore, a rigid system for which developers or domain expert have defined explanations with no awareness of the needs and expectations of the recipients may be not convenient for users with different backgrounds. The Explainer component in *ETeMoX* allows users to specify their own custom queries over the historic behaviour of the system, helping the users to complete their mental model of how the system works, or test hypotheses about its behaviour. This is done by forwarding the queries to the query engine in the Eclipse Hawk model indexer through the Hawk API (the "D-E" communication in Figure 7.1).

7.3 Experimental Study: ABS SAS

The Airborne Base Station SAS (ABS SAS) uses Reinforcement Learning (RL), an AI technique, to autonomously decide where to move in order to provide connectivity to as many users as possible (in a self-adaptive way) [165]). RL is a popular AI method used in support for self-adaptation [63]. RL can learn the effectiveness of adaptation actions through interactions with the system's environment [115].

Mobile connectivity requires that an adequate network of base stations has been set in

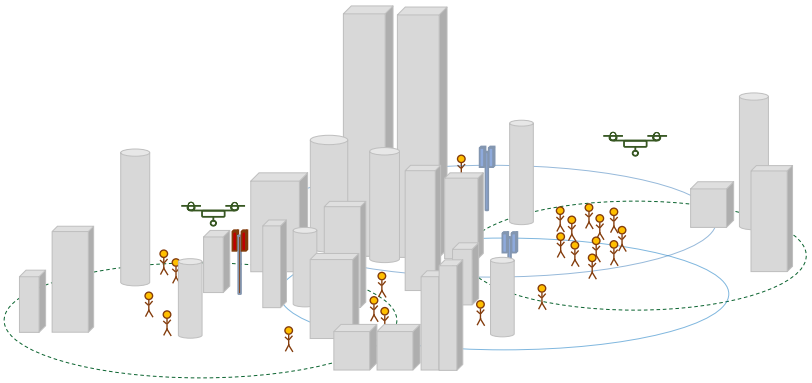


Figure 7.2: Overview of the ABS (green) SAS from [165]

place. When these networks cannot meet unexpected spikes with respect of user demand (e.g. due to a large concentration of users in one place, or due to failures in nearby communication towers), a swarm of ABS can act as a backup [165]. The goal of the system under study (the ABS SAS from [165]) is to precisely control the location of the ABS in relation to the locations of the users and the other stations, trying to serve as many users as possible while ensuring high-signal strength and minimising interference among ABSes [165].

The 5G Communications System Model performs the necessary calculations to estimate the Signal-to-Interference-plus-Noise Ratio (SINR) and the Reference Signal Received Power (RSRP) [165]. The SINR and RSRP values measure the signal quality of the communications between the ABS and the mobile stations. SINR and RSRP thresholds are used to determine whether a station can be considered to be "connected" or not [165]. Figure 7.2 shows the ABS system that can potentially be deployed upon a failure of the base stations (red in Figure 7.2), which cause communication difficulties for public safety and emergency communications [165]. The developers of the system are interested in studying the reasons why the SAS acted as it did, both regarding single decisions and regarding its overall performance.

In order to test the model agnosticism offered by ***ETeMoX***, three variants of the underlying RL algorithm have been used: Q-Learning, State-Action-Reward-State-Action (SARSA) and Deep Q-Network (DQN) and will be described next.

7.3.1 Reinforcement Learning

Influenced by behavioural psychology [116], RL is an ML approach where software agents learn actions based on their ability to maximise defined rewards in a trial-and-error fashion.

As shown in Figure 7.3, an RL agent interacts with the environment at discrete time steps (t). The agent initiates the learning process by performing a random action (a_t) that leads to a certain environmental state (s_t). The reward (r_t) corresponding to this state is assigned depending on how desirable the outcome is. After several iterations, the agent will learn a certain policy (π) (a function that maps states to actions), and will update the *value function* $V(s)$ or *action-value function* $Q(s, a)$ in order to maximise a cumulative reward, aiming to select an optimal action in every situation in order to achieve a long-term goal [147].

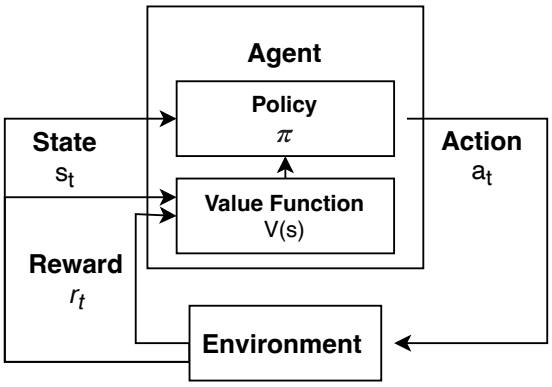


Figure 7.3: Reinforcement Learning

In RL, there are two main categories of algorithms: value-based algorithms and policy-based algorithms. The value-based algorithms focus on finding a $V(s)$ function that assigns state-action pairs a reward value. These reward values can then be used in a π. The policy-based algorithms, however, directly focus on finding an optimal π [147]. The value-based algorithms focus on trying to maximise the action-value function described as the Bellman Optimality Equation [147]:

$$Q^*(s, a) = \mathbb{E}\{r_{t+1} + \gamma * max_{a'} Q^*(s_{t+1}, a') | s_t = s, a_t = a\} \tag{7.1}$$

where $\mathbb{E}$ means that goal is to maximise the expected sum of future rewards characterised by the hyperparameter γ, which is the *discounting factor* that refers to a planning horizon [147]. Common examples of value-based functions and the ones used for the experimentation in this chapter are Q-learning, State-Action-Reward-State-Action (SARSA) and Deep Q-Networks (DQN) [147].

Q-learning

Q-Learning is an RL algorithm where an agent uses an action-value function $Q(s, a)$ to evaluate the expectation of the maximum future cumulative reward. This reward r_t is obtained from different executions of an action a_t in a given state s_t [147], which provides agents with the capability of learning to act with the aim of maximising the global reward [147].

$$\text{New } Q(s,a) = Q(s,a) + \alpha \left[R(s,a) + \gamma \cdot \max Q'\left(s', a'\right) - Q(s,a) \right] \tag{7.2}$$

Traditional Q-Learning uses a simple lookup table for calculating the maximum expected future rewards for an action at each state. It is often referred to as the Q-table, as it is a way of representing the Q-values (or Action-Values) in the Value function V_s [147]. Equation 7.2 is used to update the Q-table, where the α is the learning rate to determine how much of the sum of immediate rewards will be used. γ is the discount factor to determine the importance of future rewards and $R(s, a)$ is the reward of the action at state s_t. $Q'(s', a')$ is the new Q value in next time step; s' is next sate of environment; a' is the next action that ABSes is planning to take.

SARSA

SARSA is an RL algorithm very similar to Q-Learning [147]. The main difference between the SARSA and Q-Learning algorithms is the policy (π) type. The Bellman Optimality equation for SARSA presented in Equation 7.3, α is the learning rate, γ is the discount factor and $R(s, a)$ is the reward of the action at state s.

$$\text{New } Q(s,a) = Q(s,a) + \alpha \left[R(s,a) + \gamma \cdot Q'\left(s', a'\right) - Q(s,a) \right] \tag{7.3}$$

The most important difference between Q-learning and SARSA is how $Q(s, a)$ is updated after each action. Although the update of $Q(s, a)$ in SARSA is quite similar to Q-learning, both algorithms have different ways of choosing actions. SARSA uses the behaviour policy (meaning, the policy used by the agent to generate experience in the environment randomly) to select an additional action a_{t+1}, and then uses $Q(s_{t+1}, a_{t+1})$ (discounted by γ) as the expected future return in the computation of the update action and state value [147]. Q-learning does not use the behaviour policy to select an additional action a_{t+1}. Instead,

it estimates the expected future returns in the update rule as maximum action and state value. In other words, it tries to evaluate the policy while following the old policy, therefore it is seen as an off-policy algorithm. In contrast, SARSA uses the same policy all the time, hence it is seen as an on-policy algorithm.

DQN

Developed by DeepMind in 2015, DQN has produced some breakthrough applications able to solve a wide range of Atari games even more efficiently than humans [107]. DQN attempts to learn (i.e. maximise) an action-value function or Q-function [147]. In contrast to Q-Learning and SARSA, DQN avoids using a lookup table by instead predicting the Q-value of the current or potential states and actions using artificial neural networks (NN) or deep learning networks [147]. This Q-function (see Equation 7.1) provides the expected discounted reward that results from taking an action a_t in the state s_t while following a policy π.

For most problems, it is impractical to represent the Q-function as a table containing values for each combination of s and a as in the case of Q-Learning and SARSA. Instead, DQN introduces a function approximator, such as a NN with parameters θ, to estimate the Q-values, thus: $Q^*(s,a) \approx Q(s,a;\theta)$ [107]. Applying the Bellman equation, the network is then trained to minimise the loss L using the parameters θ such as:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\approx U(D)}[(y_i - Q(s,a;\theta))^2]$$
$$\text{where} \tag{7.4}$$
$$y_i = r + \gamma * max_{a'} Q(s',a';\theta_{i-1})$$

In this equation, y_i represents the TD (temporal difference) target, and $y_i - Q$ is called the TD error. For the calculation, DQN stores the transition tuple $(s_t; a_t; r_t; s_{t+1})$ in a replay buffer (i.e. experience replay) [107]. This also stabilises the algorithm since samples taken for training the NN are drawn uniformly from the replay buffer ($U(D)$) and the gradient is estimated in typical mini-batch fashion using these samples, thus de-correlating it. Furthermore, DQN uses the concept of a target network, which is only updated occasionally to make the learning target (predicted q-values) stationary [53].

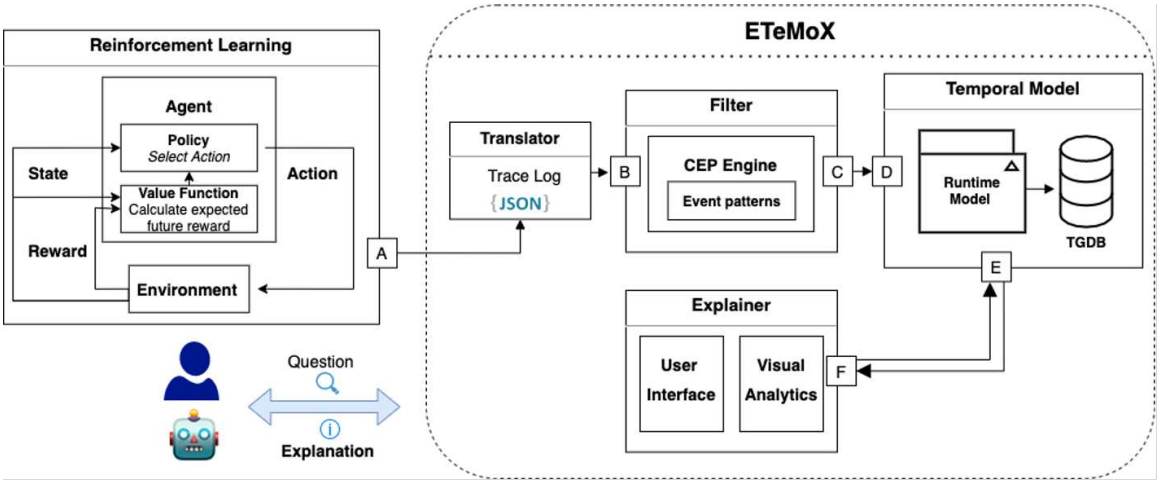


Figure 7.4: *ETeMoX* for explaining RL

7.3.2 *ETeMoX* for explaining the ABS SAS: step by step

Currently, generality is the biggest challenge for RL. Although many RL methods can be seen as performing well, it is difficult to apply them for generalisation purposes due to unforeseen situations [120]. Further, the traditional perception of RL methods is often viewed as black boxes. Without the proper tools, it is challenging to understand the behaviour of complex RL methods to solve general issues, especially when combining multiple neural networks for evaluating value functions during learning stages.

The different RL algorithms have been extended to expose their made decisions and observations in a trace log to *ETeMoX* at each simulation step as shown in Figure 7.4. In order to use *ETeMoX*, a user requires: i) an RL system that exposes its decision-making traces, ii) a parser to translate these traces into the metamodel in Figure 7.5, iii) a set of event patterns that define the filtering criteria (and their deployment to the Esper CEP engine), iv) a Hawk instance indexing the translated traces into a temporal model, and v) a set of temporal queries that extract the history-aware explanations from the temporal model. The RL-specific metamodel (Figure 7.5) imports elements from the core package (Figure 5.2). In particular, the RL package provides a specialised *RLAgent* which keeps track of the *RLState* that can be observed in the environment, an *RLDecision* which tracks the *QValue*s of each available action, and an *RLObservation* which tracks the current state before the action was taken, and the current *Reward* values.

The detailed step-by-step guidelines are as follows:

1. The proposed post-hoc approach is designed to be as least intrusive as possible for the

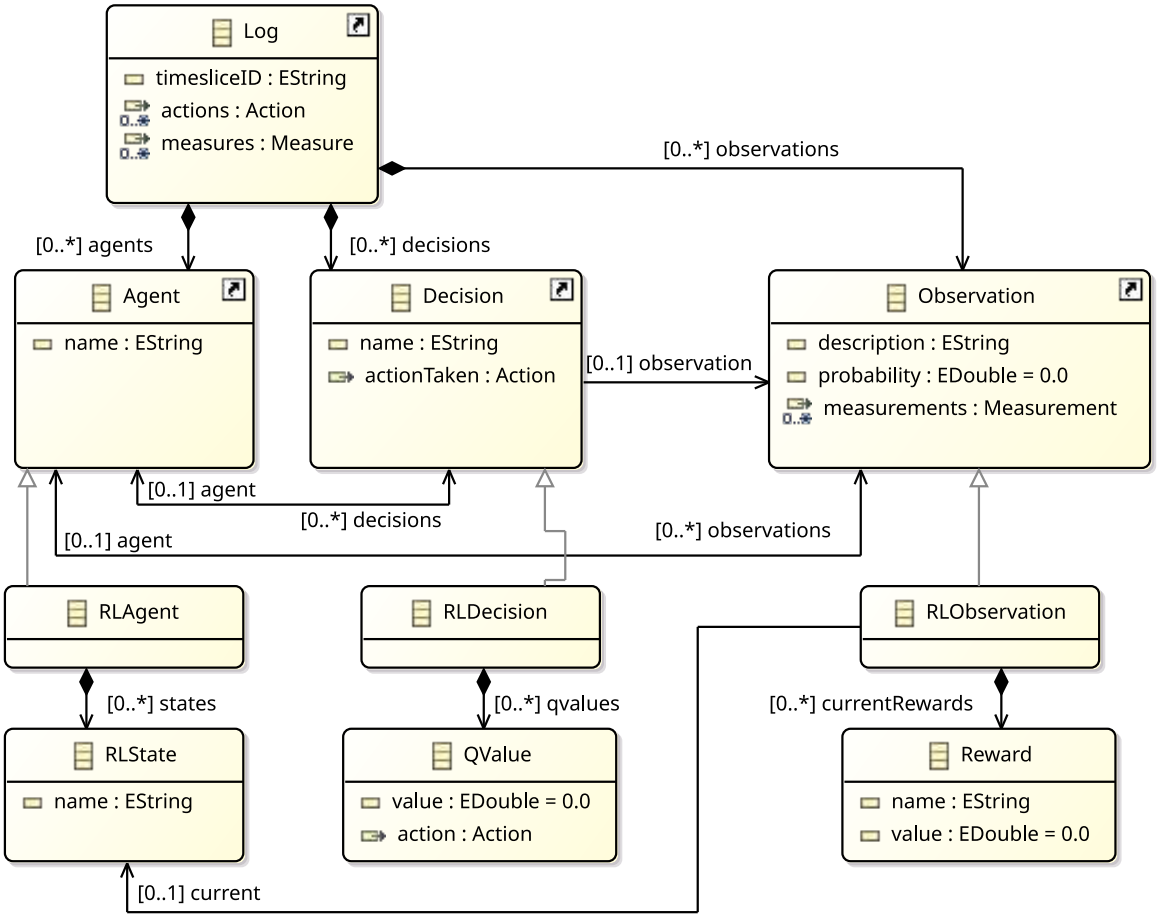


Figure 7.5: Class diagram for the RL extensions to the core metamodel used to record system history. Imported core elements are marked with an arrow.

RL agent to be explained. The first step is to collect observations from the system's decision-making. The available information about the agent's states, rewards, actions, and environment is exposed to the system through logs, which may be structured or not.

2. Once this log is constructed, the next step is to feed it to the architecture. MQTT is used as the core communication protocol, with MQTT clients in the different components talking to a central MQTT broker. The trace log is published to a MQTT topic to which the *translator* component is subscribed.

3. In order to handle the incoming data (i.e. trace log), *ETeMoX* requires the *Translator component* to parse the log. In the current implementation, this parser has been manually defined at design time, and is specific for each case study. However, different techniques for log and stream pre-processing are being studied for automatically creating these translators, such as the one proposed by Corral-Plaza et al. in [39].

4. When the parser that processes the log is ready, the next step is the creation of the event patterns needed by the *Filter component*. These EPL patterns will contain the criteria to curate the data, based on events of interest. Some predefined filtering criteria can be reused across projects (e.g., sampling at a certain rate). Also, problem-specific event patterns of interest can be added as needed. Afterwards, the filtered data is sent to the *Temporal Model component* over MQTT.

5. Next, the filtered information is stored in a casually-connected and efficient way in an Eclipse Hawk instance. This instance needs to be configured to use the execution trace metamodel of Figure 7.5 o structure the graph, and to have a Greycat temporal graph database as backend.

6. Once the information is structured as a TM, it is possible to extract information for explanations using EOL queries from the *Explainer component*. Depending on the requirements, some predefined temporal queries can be reused, or new domain-specific queries can be formulated. Information for explanations can be extracted after-the-fact or at runtime.

7. The final step is to construct an explanation from the queries. The specific way this is done depends on the requirements and the targeted audience. For instance, textual

explanations (logs or natural language) or visual ones (graphs, plots or heatmaps) could be used.

The complete user manual that explains how to apply *ETeMoX* to an RL system can be found in the GitLab project at [118]. An implementation of the proposed approach for the different RL algorithms (i.e., Q-learning, SARSA and DQN) and the first three levels of the research roadmap for history-awareness with explanatory capabilities in SAS (Section 5.2), is presented next.

7.3.3 Level 1

Accorcing to Section 5.2, the first step to achieve automated history-awareness in SAS is to offer *forensic* capabilities. This section shows a description of this level's implementation for the ABS SAS. It also describes the scenario that motivates the need for explanations in the ABS SAS. Finally, the experiment results are presented.

a. Scenario

Providing developers and users the tools for using explanations to monitor and analyse the performance of RL based SAS systems is key for V&V and understanding. In the ABS SAS case study, the locations of users and the signal interference levels from ABS keep changing. Presenting the evolution of this change can help developers to understand if the ABS SAS is progressing towards its ultimate goal. In the SAS, the RL agents analyse and update their decision-making criteria based on the rewards received at each time step: it is key for the explanatory system to keep track of these rewards.

ETeMoX is capable of tracking both the individual and global rewards on every time step in the system's training history, and present an average reward by episode. Without any filtering, the results would match exactly what the system experienced, but at high storage and processing costs. CEP is used to tackle this issue. Event patterns defined using EPL allow the sampling of the data at a certain data rate, instead of storing the entire system history. Analysing the accuracy of the sampled data will help to answer if the information stored can provide similar conclusions (i.e., explanations) as storing the entire history.

Furthermore, in a multi-agent system, explaining collaborative aspects can help to understand whether agents in the ABS SAS learn to coordinate to achieve the global goal or

not. This can be difficult without the support of a monitoring tool like *ETeMoX*, considering that each RL agent has only access to its own local observations and is only responsible for choosing actions from its own action-state values. A challenge in mobile wireless communication is the hand-off or handover process. This is the process of providing continuous service by transferring a data session from one cell to another [165].

In the presented collaborative system, ABS agents are assumed to have ideal communication among themselves and cannot occupy the same position (state) at the same time. Therefore, considering these conditions, handovers should be kept at a minimum, demonstrating effective communication and stability within the system. In the present implementation of the ABS SAS, a handover could be considered when a user is connected to one ABS and then transferred to a different ABS in a short period of time. Explaining these situations would help to developers to discover moments when handovers are tacking place and to validate if the system is behaving as expected.

b. Implementation

This subsection describes the implementation of *ETeMoX* for explaining the situations presented in the scenario. The experiments focused on i) tracking the evolution of a metric, and ii) explaining the multi-agent collaborative behaviour.

Evolution of a metric: Regarding the tracking of the evolution of a metric, three Esper EPL event patterns that apply various sampling rates to update the TM (every 10, 100 and 500 steps of the simulation) were applied. Listing 2 shows the event pattern for indexing the runtime model into the TM every 10 steps. Finally, the log about the state and observations of the system when this criteria is met is recorded for further analysis.

An object diagram with an instance of the runtime model at a certain step in the simulation is shown in Figure 7.6. The *Log* contains *Decision*s and *Observation*s for `ABS 1` at `Episode 9` and `Step 199`. The possible *Action*s are linked to their *ActionBelief*s that represent the estimated values (Q-values), which maximise the cumulative *Measure*: `Global reward` at the given *Measure*: `State`. Having recorded the history of the system so far, at any time the TM can answer queries from the Explainer component. The Hawk GUI is used to extract the information needed to build the required explanations after the system has finished its execution (i.e., forensic explanations).

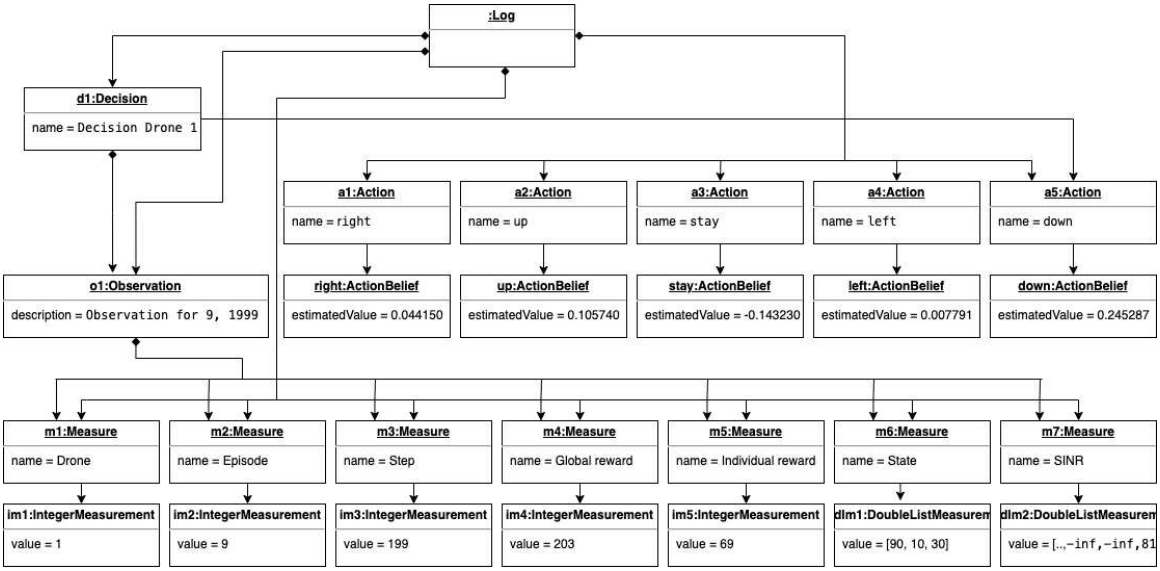


Figure 7.6: Runtime model object diagram

Multi-agent collaboration: Regarding multi-agent collaboration, in order to find handover situations in the different RL algorithms, a query was run on TMs containing the whole history of each RL approach. Algorithm 1 describes the logic followed in the query, which was implemented in the temporal query language supported by Hawk. A user (U) is connected to an ABS (D) when its received SINR ($\text{SINR}_{u,d}$) is above a defined threshold α^{SINR} [165]. Therefore, to find handovers over the history, it is necessary to analyze the SINR between each (user, ABS) pair at every simulation step. For example, a handover of user $u1$ from ABS 1 to ABS 2 happens when at step t, $\text{SINR}_{u1,1} > \alpha^{\text{SINR}}$ (the SINR between user $u1$ and ABS 1 is above the threshold), and then at step $t + x$ (where x is a certain time window, measured in numbers of steps) the $\text{SINR}_{u1,2} > \alpha^{\text{SINR}}$ (the SINR between user $u1$ and ABS 2 is above the threshold) and also $\text{SINR}_{u1,2} > \text{SINR}_{u1,1}$ (user $u1$ is better connected to ABS 2 than to ABS 1).

c. Results

This sections presents the evaluation of the results of using *ETeMoX* to explain the evolution of a metric and the multi-agent collaboration in the ABS SAS case study. Q-Learning, SARSA and DQN were tested under the same conditions. A training run consisted of 10 episodes and 2000 steps for 2 ABS agents with 1050 users scattered on a X-Y plane. As mentioned, *ETeMoX* follows a post-hoc approach that decouples the running RL system

Algorithm 1 Query to detect handovers. L is the current runtime log, T the set of timeslices in L, U the users, D the ABSes, $\text{SINR}_{u,d}(t)$ the link measurement between $u \in U$ and $d \in D$ at timeslice t, α^{SINR} the threshold for the SINR, and x a defined time window.

Result = {}
for each $u \in U$ **do**
  for each $d \in D$ **do**
    $T_B = \{t \in T | \text{SINR}_{u,d}(t) > \alpha^{\text{SINR}}\}$
    for each $t_b \in T_B$ **do**
      if $(\text{SINR}_{u,d}(t_b + x) < \text{SINR}_{u,d+1}(t_b + x) \wedge \text{SINR}_{u,d+1}(t_b + x) > \alpha^{\text{SINR}})$ **then**
        Add $(t_b, u, \text{SINR}_{u,d}(t_b), \text{SINR}_{u,d+1}(t_b + x))$ to **Result**
      end if
    end for
  end for
end for
Result: Sequences showing handover transitions.

from the generation of explanations. In that sense, the experiments were performed using two machines dedicated to different purposes: one performing the training of the different RL algorithms, and the other running *ETeMoX*. The RL algorithms ran on a virtual machine in the Google Cloud Platform[2]: specifically, an *a2-highgpu-1g* machine with 2vCPUs running Debian GNU/Linux 10 with 13GB RAM and an NVIDIA Tesla K80 GPU, using the ABS SAS simulator, Anaconda 4.8.5, matplotlib 3.3.4, numpy 1.19.1, paho-mqtt 1.5.0, pandas 1.1.3, and pytorch 1.7.1. The machine running *ETeMoX* was a Lenovo Thinkpad T480 with an Intel i7-8550U CPU with 1.80GHz, running Ubuntu 18.04.2 LTS and Oracle Java 1.8.0_201, using Paho MQTT 1.2.2, Eclipse Hawk 2.0.0, and Esper 8.0.0.

Evolution of a metric: The proposed architecture was able to sample the incoming data produced by the ABS SAS for each RL algorithm. Table 7.1 shows the costs of storing the TM using each approach. The full history of the system consisted of 40 000 model versions (10 episodes × 2 000 iterations × 2 ABS agents). Depending on the sampling data rate selected, the size of the TM showed a linear decrease, going from approximate 130 MBs for the full history, to less than 1MB when sampling the history each 500 time steps.

In order to test the accuracy of the results, a temporal query was ran on the different TMs to find the averages from each training episode and see how they evolved. Figure 7.7 show the results for a) Q-Learning, b) SARSA, c) DQN. Additionally, a t-test [146]

[2] https://cloud.google.com/

Approach	Model Versions	Q-Learning	SARSA	DQN
Full history	40000	126.00	129.00	162.00
History sampled r=10	4000	15.00	16.00	24.00
History sampled r=100	400	1.70	1.80	1.90
History sampled r=500	80	0.95	0.46	0.77

Table 7.1: TM size in MBs

Approach	Q-Learning	SARSA	DQN
History sampled r=10	0.95	0.94	0.88
History sampled r=100	0.37	0.62	0.39
History sampled r=500	1E-4	2E-3	5E-3

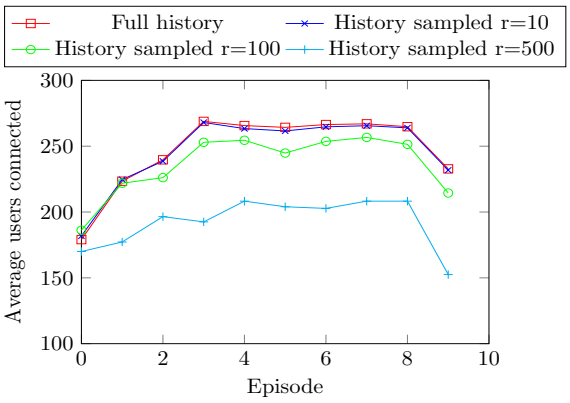
Table 7.2: T-test results

was used to compare the means of each group. A t-test is an inferential statistic used to determine if there is a significant difference between the means of two groups and how they are related [146]. The results using the full history are compared to those from doing sampling at different rates. Table 7.2 shows the p-values for the null hypothesis H_0 defined, as there is no statistically significant difference between the sample sets. Anything with $p < 0.05$ is classed significant [146]. Thus, only the null hypothesis for the sample sets corresponding to the history sampled with data rate of 500 is rejected. Therefore, they are significantly different to the base sample set (full history).

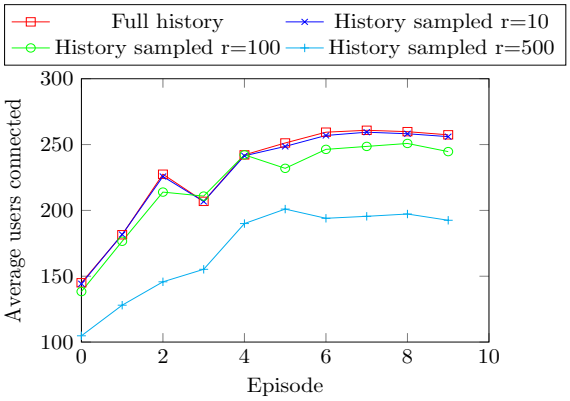
Furthermore, the costs for retrieving the information that built these visual explanations were also evaluated. Results are shown in Table 7.3. The query execution times also presented a linear decrease. Running the query in the TM corresponding to the full history took up to 43.23 seconds while running the query on the TM with the smaller size took between 0.08 and 0.09 seconds.

Approach	Model Versions	Q-Learning	SARSA	DQN
Full history	40000	42.91	43.23	41.95
History sampled r=10	4000	4.68	4.78	4.63
History sampled r=100	400	0.34	0.34	0.38
History sampled r=500	80	0.09	0.09	0.08

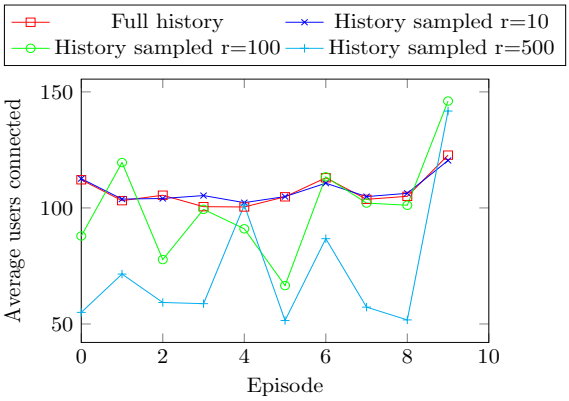
Table 7.3: Query execution times in Seconds



(a) Q-learning



(b) SARSA



(c) DQN

Figure 7.7: Evolution of a metric

Listing 7.1: Excerpt of output from Algorithm 1 about handovers on the system's history (SARSA).

```
1 [ ...
2 EolMap {SINR_1=80.76815264, ABS=1, user_id=477, SINR_2=81.39525019, episode
      =3, step=912, step_handover=915, ABS_to_handover=2},
3  ...
4 EolMap {SINR_2=80.52090196, ABS=2, user_id=925, SINR_1=81.20204141, episode
      =2, step=1231, step_handover=1234, ABS_to_handover=1},
5  ... ]
```

Multi-agent collaboration: In order to study the multi-agent collaboration behaviour in the SAS, the experiment for explaining situations when handovers took place, was performed. The goal of this experiment was to prove the hypothesis of the developer that under ideal conditions and ideal communication between the ABS agents, the system should reduce the number of handovers. A temporal query was implemented on the different TGDBs containing the full history. The selected SINR threshold was 40 ($\alpha^{\text{SINR}} = 40$ in Algorithm 1), and a time window of 3 time steps ($x = 3$ in Algorithm 1) was suggested by the developer to consider as the transition time. 1 784 handovers were found for Q-Learning, 590 for SARSA and 82 176 for DQN. An excerpt of the query results is presented in Listing 7.1. Line 2 indicates that a handover from ABS 1 to ABS 2 happened on SARSA on the episode 3 between time steps 912 and 915, when the user 477 was initially connected to ABS 1, and after 3 time steps was connected to ABS 2. A similar situation happened on line 4, but in this case there was a handover from ABS 2 to ABS 1 at episode 2 between time steps 1231 and 1234.

Due to the nature of the query, the execution times increased compared to previous queries. They were: 917s for Q-Learning, 1132s for SARSA and 7914s for DQN. This is because for each time slice (model version), it was needed to check how the SINRs for each user $u \in U$ changed over the defined time window. Thus, it was necessary to check across all 10 episodes (each spanning 2000 time steps) the SINRs for each of the 1050 users corresponding to each of the 2 ABSes. This produced $10 \times 2000 \times 2 \times 1050 = 42\,000\,000$ situations to check. Considering the previous, the situations found were very rare, representing only $4.2 \times 10^{-5}\%$ for Q-Learning, $1.4 \times 10^{-5}\%$ for SARSA. The previous mentioned is inline with that in this controlled experiment, the handover situations are expected to be minimum. However, in DQN, although the situations still represented a very small percentage $1.9 \times 10^{-3}\%$, further studies about collaborative tasks are needed.

7.3.4 Level 2: Live History-aware explanations in ABS SAS

The previous section presented the implementation of *ETeMoX* to enable forensic analysis and obtain explanations after the system had finished its execution. This section focuses on the Level 2 implementation of the proposed research roadmap (Section 5.2) for the ABS SAS. The main goal is to allow the extraction of explanations when the system is running, while considering storage and processing concerns.

a. Scenario

An important aspect for RL-based SAS developers is to analyse the agent's learning process and how the initial conditions affect it. As part of its use of RL, the agent changes between *exploration* and *exploitation* states. In the exploration state, the agent is trying to discover new features of the environment by selecting a sub-optimal action. In the exploitation state, the agent chooses the best action according to what it already knows [147]. The developers may want to gain a general idea of how the system changes between these two states. In order to find when a decision was performed using exploration or using exploitation, *ETeMoX* tracks the actual action taken and the Q-values (i.e. the *ActionBelief*s) for each possible action at given state. On one hand, when the action performed has the maximum Q-value then it could be said that the decision was taken by exploitation. On the other hand, if the action taken does not have the maximum Q-value, the action was taken by exploration. Considering the object diagram of Figure 7.6, where the *Action* selected (represented by the reference from *d1* to *a5*) was *down*. It can be seen that it is the one with the maximum estimated value: thus, it can be concluded that the decision was performed by exploitation. Additionally, the experiment aims to analyze how these types of actions affected the overall goal of the system (connecting as many users as possible).

b. Implementation

In order to evaluate the effect that a domain-specific filtering pattern could have on costs and accuracy, an Esper EPL pattern was developed to only capture in the TGDB the moments when a decision was performed using exploration. Listing 3 shows the Esper EPL pattern for finding this situation. At every point in time, the Q-value of the action selected (`drone.qtable.action`) is compared to the `maxValue()`, the action with the maximum

Q-value. If these values do not match a decision was taken by exploration and the log about the state and observations of the system at that point in time is recorded. For this experiment, two TMs were created in parallel. One using the sampled data, and another containing the full history of the system. In this last one, the temporal query of Listing 4 was run for validation. The temporal query written in EOL follows the same logic of the Esper EPL pattern, but traverses the full history. It looks for the Q-value of the action selected in each decision (`actionTakenValue`) and compares it with the maximum Q-value (`maxAB`). It returns a sequence of situations where the criteria is met. Finally, it reports a count of these situations.

In order to analyse the impact of the actions taken by exploration or exploitation, further Esper EPL event patterns following Algorithm 2 were defined. From the different instances T of TM M, actions marked as E_R (exploration) or E_T (exploitation) are analysed and classified into the respective group. Depending on the impact of an action on the reward for the subsequent time point (it produces an increment, a decrease, or no change), the counters $cI, c0, cD$ are incremented. The produced results are used to build global explanations in the form of an event graph [27].

Algorithm 2 EPL pattern to detect the impact of actions taken by exploration and exploitation. M is the current runtime model, T the set of instances of M, A the type of actions either exploration or exploitation, R the rewards (users connected), $cI, c0, cD$ counters for the type of impact on rewards (Increased, no impact, Decreased) of action A

Result = {}
cI, c0, cD = 0
for each $t \in T$ **do**
    for each $a \in A$ **do**
        if $\mathrm{R}_a(t) = \mathrm{R}_a(t+1)$ **then**
            Add $(t, cI, c0++, cD)$ to **Result**
        else if $\mathrm{R}_a(t) < \mathrm{R}_a(t+1)$ **then**
            Add $(t, cI++, c0, cD)$ to **Result**
        else if $\mathrm{R}_a(t) > \mathrm{R}_a(t+1)$ **then**
            Add $(t, cI, c0, cD++)$ to **Result**
        end if
    end for
end for
Result: Sequences showing the impact of actions taken by exploration or exploitation in the rewards over the time.

RL Approach	Model Versions	TGDB (MB)	Exploration (%)	Exploitation (%)
Q-Learning	562	8.80	1.41	98.60
SARSA	3195	18.00	7.99	92.01
DQN	3126	21.00	7.82	92.19

Table 7.4: Results and costs of filtering history with the exploration pattern.

c. Results

This experiment focused on finding situations where the action performed by the ABS SAS differs from the one that it currently thinks is best. An EPL query deployed in the CEP engine filters the history, letting through only the time steps where the ABS SAS was using exploration rather than exploitation. Table 7.4 shows a summary of the results of applying this filtering using the exploration EPL pattern. Both SARSA and DQN presented similar results, showing the system using exploration 8% of the time. In the case of Q-Learning, exploration was done during 1.41% of the time steps. The results of the EPL query selected the same time steps as a temporal query (EOL query) on the TGDB with the full history containing exploration events.

In order to compare the impact on accuracy of custom EPL-based filters in comparison with uniform sampling, the same temporal query from Section 7.3.3 was performed to find the reward averages for each episode on the different TGDBs for each RL-algorithm. Figure 7.8 shows the results for each approach. A similar behaviour to the one presented in the previous experiment is exhibited. Less data (model versions) create less precise results, as it is the case of Q-Learning. Although for SARSA and DQN similar number of model versions were found (3195 and 3126), the results show a significant variability for the case of DQN.

Regarding the impact of actions taken by exploration or exploitation, this experiment tried to find events that occur over the execution of the case under study. The aim was to find whether the system acted based on exploration (event E_R) or exploitation (event E_T). As shown in Fig. 7.9, the event graph represents how frequently the system acts one way or the other, and how often it changes between the two behaviours. Additionally, the experiment was interested on the effects of these events on the system's overall goal (rewards). These E_R and E_T events produced the subsequent events; $R+$ the event of an increase in the reward, R the event where the reward stayed the same as in the previous

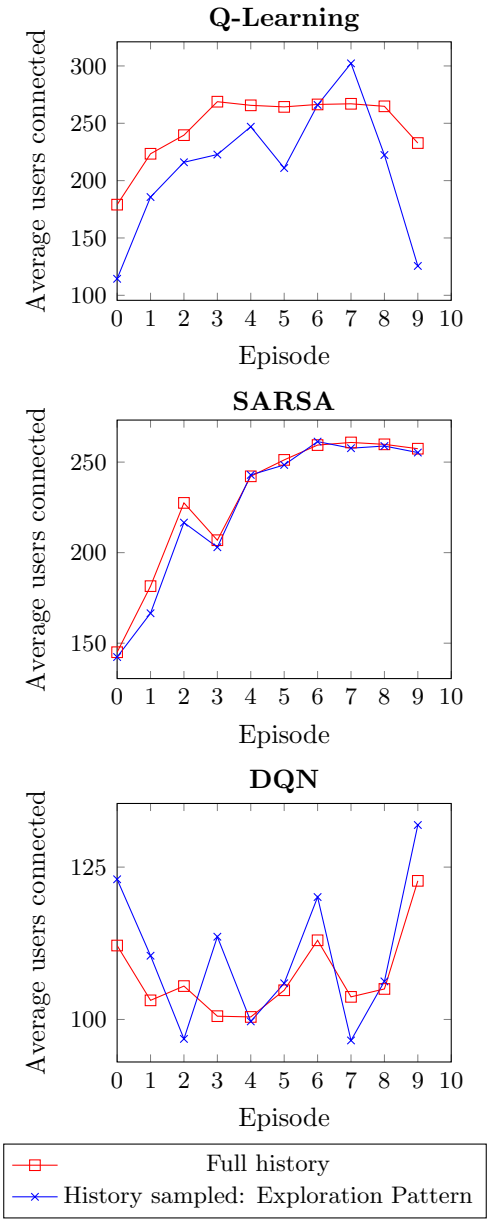


Figure 7.8: Reward averages by episode on exploration pattern

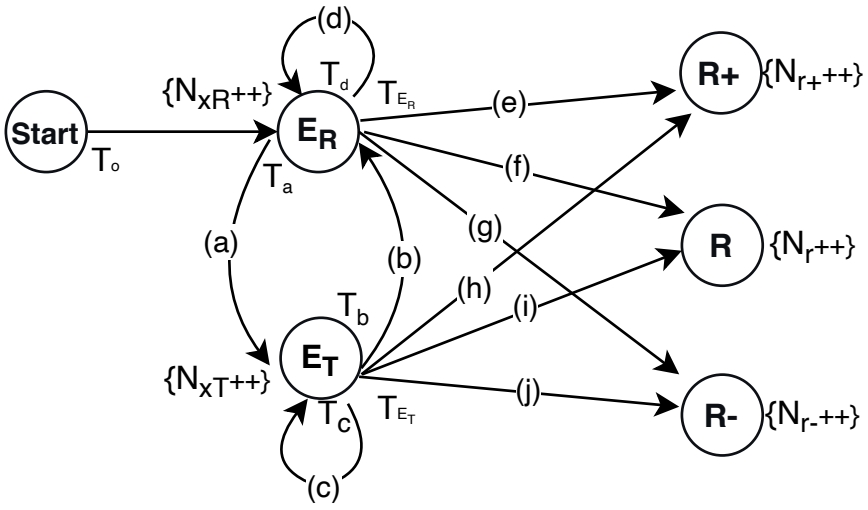


Figure 7.9: Global explanations: exploration & exploitation in RL

time point and, $R-$ the event of a decrease in the reward. The features of the summarised final event graph are:

- **Events:**
  - Start: initial event.
  - E_R: action chosen by exploration.
  - E_T: action chosen by exploitation.
  - $R+$: reward increased.
  - R: reward stayed the same
  - $R-$: reward decreased.
- **State Variables:**
  - N_{xR}: number of E_R events detected.
  - N_{xT}: number of E_T events detected.
  - N_{r+}: number of $R+$ events detected.
  - N_r: number of R events detected.
  - N_{r-}: number of $R-$ events detected.
- **Parameters:**

- $\{T_o\}$ = time delay between $Start$ and E_R.
- $\{T_a\}$ = sequence of time delays between E_R and E_T.
- $\{T_b\}$ = sequence of time delays between E_T and E_R.
- $\{T_c\}$ = sequence of time delays between E_T and E_T.
- $\{T_d\}$ = sequence of time delays between E_R and E_R.
- $\{T_{E_R}\}$ = sequence of time delays between E_R and $[R+, R, R-] \in \{T_o, T_a, T_d\}$
- $\{T_{E_T}\}$ = = sequence of time delays between E_T and $[R+, R, R-] \in \{T_b, T_c\}$
- (a): $(U(0,1) < p_a)$ where p_a = probability of a $E_R \Rightarrow E_T$ transition and $U(0,1)$ a random value chosen in the $[0,1]$ range following a uniform distribution.
- (b): $(U(0,1) < p_b)$ where p_b = probability of a $E_T \Rightarrow E_R$ transition.
- (c): $(U(0,1) < p_c)$ where p_c = probability of a $E_T \Rightarrow E_T$ transition.
- (d): $(U(0,1) < p_d)$ where p_d = probability of a $E_R \Rightarrow E_R$ transition.
- (e): $(U(0,1) < p_e)$ where p_e = probability of a $E_R \Rightarrow R+$ transition.
- (f): $(U(0,1) < p_f)$ where p_f = probability of a $E_R \Rightarrow R$ transition.
- (g): $(U(0,1) < p_g)$ where p_g = probability of a $E_R \Rightarrow R-$ transition.
- (h): $(U(0,1) < p_h)$ where p_h = probability of a $E_T \Rightarrow R+$ transition.
- (i): $(U(0,1) < p_i)$ where p_i = probability of a $E_T \Rightarrow R$ transition.
- (j): $(U(0,1) < p_j)$ where p_j = probability of a $E_T \Rightarrow R-$ transition.

In total, 36 395 of the actions (90.99%) were chosen by exploitation (the system was in the E_T state), whereas 3 605 actions (9.01%) were taken by exploration (the system was in the E_R state). The simulation began with a "Start" event and always transitioned to E_R first, starting with exploration. From E_R, it was observed from the complex events that there was a 90.98% (3 280 out of 3 605) chance that the subsequent action was chosen by exploitation (E_T), and a 9.02% (325 out of 3 605) chance that the system kept exploring. Likewise, from E_T it was observed that there was a 90.96% (33 104 out of 36 395) chance to stay in E_T, and a 9.04% (3 291 out of 36 395) chance to transition to E_R. Moreover, from the 36 395 actions chosen by exploitation, 5 159 produced an increase ($R+$) on the rewards, 5 101 produced a decrease ($R-$) on the rewards and, 26 135 kept the same (R) reward from the previous time point. Correspondingly, after an action taken by exploration there was a 14.18% chance that the number of users connected increased, a 14.01% chance that it decreased, and a 71.81% that it stayed the same. In case of the 3 605 actions chosen by exploration 474 increased the reward, 555 decreased it and 2 576 kept the same reward. Furthermore, there was a chance of 13.15% that an action taken by exploration led to an increase of the reward, a 15.4% that it led to a decrease and a 71.45% chance that the reward stayed the same.

7.3.5 Level 3

The previous sections in this chapter have demonstrated how TMs combined with CEP can enable history-aware explainability in AI-based SAS while making efficient use of computational resources. The experiments of Sections 7.3.3 and 7.3.4 depicted how explanations can be obtained after the system has finished its execution or at runtime. The conveyed information can help developers to validate and improve their systems and fix potential faults for a next round of the SAS execution. In other words, the explanations were used for passive monitoring. In this section, an active approach in line with the Level 3 of the research roadmap (Section 5.2) is presented. This level aims to allow external entities to interact with SAS at runtime using the information provided by the history-aware explanations. Different to 6.2.5, this Section targets another software system as the consumer of the explanations instead of humans. A SAS can be part of a system-of-systems deployment and explanations can be useful to promote collaboration between systems. Furthermore, a different system monitoring a SAS can potentially provide feedback based on information

that is not available to the SAS internal decision-making process (e.g., situations in the past). An example of this scenario is presented in the following subsections.

a. Scenario

The initial conditions or *hyperparameters* of the AI approaches used in a SAS need to be carefully tuned to obtain state-of-the-art performance during execution [53]. The search for the best hyperparameter configuration is a sequential decision process in which initial values are set, and later adjusted, through a mixture of intuition and trial-and-error, to optimise an observed performance on typically the hold-out validation set, i.e. to maximise the accuracy or minimise the loss [76]. This process often requires expensive manual or automated hyperparameter searches in order to perform properly on an application domain [164].

Traditional hyperparameter optimisation (HPO) approaches include manual search, grid search, and random search [53]. However, a noticeable limitation is the high cost related to algorithm evaluation for complex models, which makes the tuning process highly inefficient, computationally expensive, time-consuming, requires domain-specific knowledge and commonly adds extra algorithm developing overheads to the RL agent decision-making processes [164, 76, 53].

The present section proposes the use of *ETeMoX* to quickly detect temporal and causal dependencies between events on the fly, in order to gain insights from events as they occur during the execution of the RL agent, which is crucial for HPO [53]. The previously mentioned approach enables a scalable platform allowing the use of short- and long-term memory to reason about the RL agents' historical behaviour for HPO. This knowledge is conveyed to an external system for further processing. The external entity provides feedback to the SAS based on its historical behaviour, aiming to find the hyperparameter combination that produces the best SAS performance.

b. Implementation

Figure 7.10 describes the implementation of *ETeMoX* for Level 3 of the RL-based ABS SAS using DQN algorithm. An external entity is able to gain insights about the SAS historical behaviour by running queries over the stored TM. For this experiment, the external entity is a *Graph Listener* or *GL*. This system runs periodically a query in the TM to obtain information about the evolution of a metric: in this case, the value of the reward function

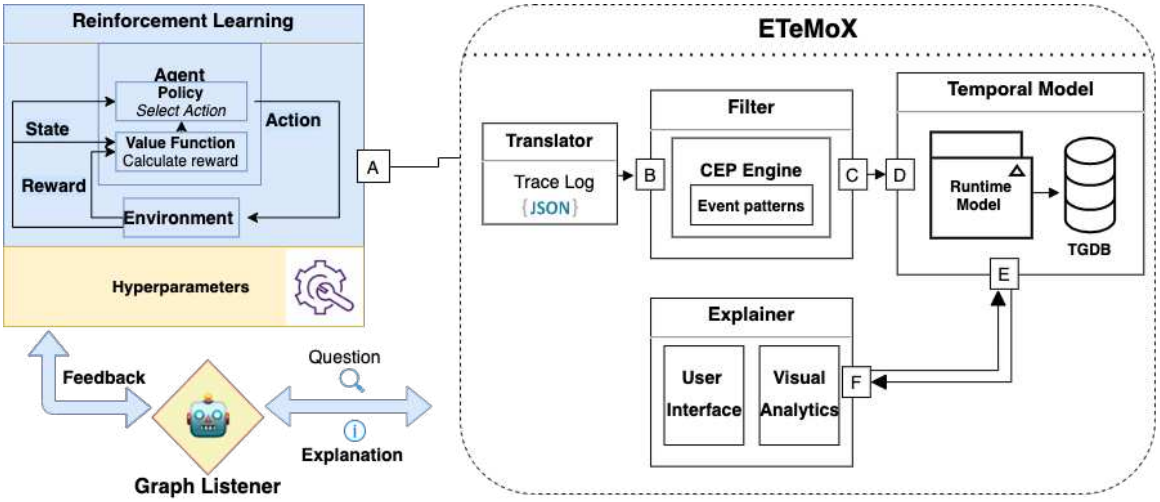


Figure 7.10: *ETeMoX* for enabling feedback from external entities

$\mathcal{R}$. In order to provide advantageous feedback, the efficient continuous monitoring of internal operations and parameters of the RL agent and its interactions with the environment over time are required. *ETeMoX* proposes the use of CEP for short-term analysis and TMs for navigation through the system history to provide feedback to the RL-based SAS.

In RL, the N-dimensional hyperparameter configuration space is defined as $\Lambda = \Lambda_1 \times ... \times \Lambda_N$, and a vector of hyperparameters is denoted by $\lambda \in \Lambda$. Let us denote the RL algorithm as Φ and Φ_λ as the algorithm instantiated to a vector of hyperparameters λ. Let us define the objective function to maximise the value of a reward function $\mathcal{R}$. The HPO problem can be defined as, given an algorithm Φ, the environment E and time T, finding the optimal hyperparameter vector λ^* that maximises the reward value achieved by Φ such as:

$$\lambda^* = \arg\max_{\lambda \in \Lambda} \mathcal{R}(\Phi_\lambda, E, T) \tag{7.5}$$

where $\mathcal{R}(\Phi_\lambda, E, T)$ measures a reward value generated by the algorithm Φ under a configuration of the λ hyperparameter while interacting with the environment E at time T.

For the current implementation, the aim is to study the impact of optimising the discount factor as the key element in the Bellman equation (Equation 7.4) for the proposed case-study. The discount factor determines how much the RL agents care about rewards in the distant future, relative to those in the immediate future [147]. In this regard, the

hyperparameter vector is expressed as follows:

$$\lambda(\gamma, \kappa)$$

where γ represents the hyperparameter to be optimised (i.e. discount factor) and κ the hyperparameters that remain fixed.

With these preliminaries, different experiments using the proposed framework were performed under the same scenario. It consisted of a training round (i.e., a single lifetime) of 100 episodes and 1000 steps for a set of 4 ABS with 1050 users scattered on a X-Y plane. The ABS try to maximise the number of users connected by performing actions (i.e. moving on different directions) and calculating the SINR to users on a collaborative fashion.

The main goal of the *GL* as an external entity is to try to solve Equation 7.5 using the information provided by *ETeMoX*. The history-aware epsilon-greedy logic for HPO described in Appendix 1 and 2 was implemented in GL and *ETeMoX*. In order to choose the correct moment to provide feedback to the SAS, the GL asks *ETeMoX* for moments when the value of the reward function $\mathcal{R}$ is stable over a period of time ($rewardWinAvg$). When the GL has discovered that the SAS is on an stable state, in terms of the value of reward function, a reconfiguration of the hyperparameters is suggested: $return\ \gamma$. Listing 5 shows the implementation that attempts to detect stable conditions on time windows of 3 episodes ($w = 3$). Every pattern `AvgByEpisode`, followed (`->`) by two subsequent `AvgByEpisode` and a `EpiWinAVG` (which refers to the average in a time window), is analysed (`where` statement) in compliance of the boolean conjunction. When the condition is met (i.e., boolean conjunction = $True$), the engine automatically generates complex events that collect the required information to be stored in the TM.

In order to show the feasibility of the approach, two different experiments were defined and are described next.

History-Aware HPO vs traditional HPO: A comparison with traditional HPO techniques is performed. Specifically, the previously mentioned grid search and random search. For this experiment, hyperparameter tuning with these techniques was performed during the training processes. The experiments encompassed of a run of the DQN using each of the defined hyperparameter tuning techniques. In this context, the initial hyperparameters for each approach are described in table 7.5. From the literature [147], commonly used values

Approach	γ_o	Tuning criteria
Manual setting	0.9	static
Grid search	0.9	updated every 10 episodes
Random search	random	updated every 10 episodes
History-aware HPO	0.9	automated-tuning

Table 7.5: Initial configuration for experiment 1

for the discount factor are within the range of 0.9 and 0.99. A manual setting with static discounting factor has been included for comparison. For the case of grid search, in order to cover the hyperparameter-value space, the initial value of γ is set to 0.9 and decays over the time with a rate of 0.1. Random search starts randomly and the history-aware HPO starts in the centre of the value-space.

History-Aware HPO vs static hyperparameters A second experiment was performed to analyse the impact of the proposed approach on the performance of the system in comparison to keeping the hyperparameters static during the training. For this purpose, the training round was initialised with the same seeds, and the progression of the rewards over time was compared across the various hyperparameter configurations. The initial values of the discount factor that conformed the experiment were:

$$\gamma_o \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\} \tag{7.6}$$

c. Results

In this section, the evaluation of the results of using the proposed framework implementing the history-aware epsilon-greedy logic for HPO is presented. The RL-based SAS was trained using DQN under the same conditions for the different experiments.

History-Aware HPO vs traditional HPO The first experiment corresponded to a qualitative study of the performance of the ABS system using the proposed approach, comparing against traditional HPO techniques. Figure 7.11 shows the results. As it can be observed, the approach using feedback provided by the GL (red line) outperformed the different approaches, obtaining its maximum values from episode 42 onward. The periodic random search (blue line) fluctuates and its performance is closed to that of the

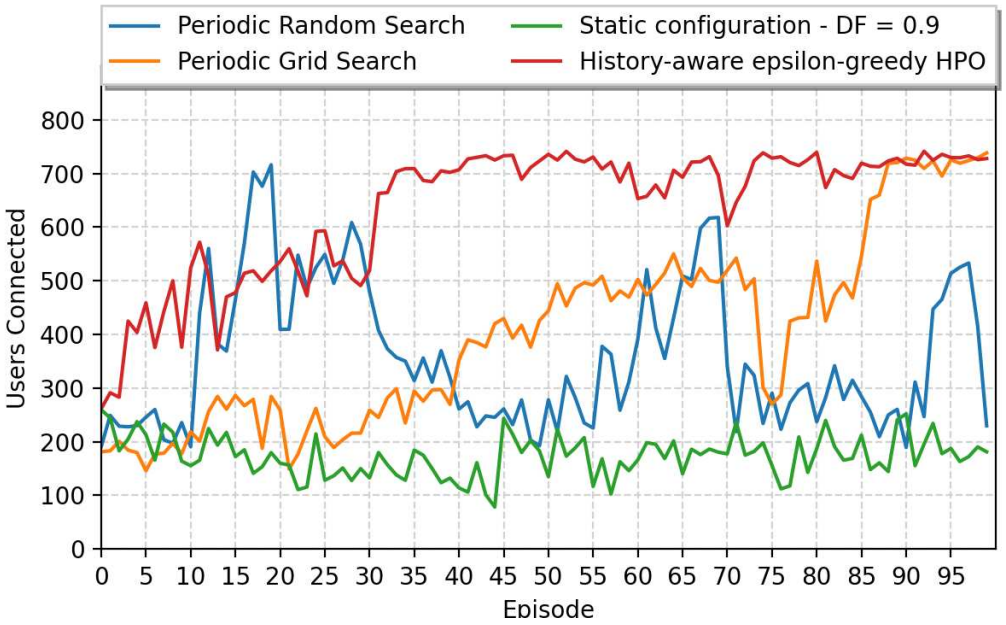


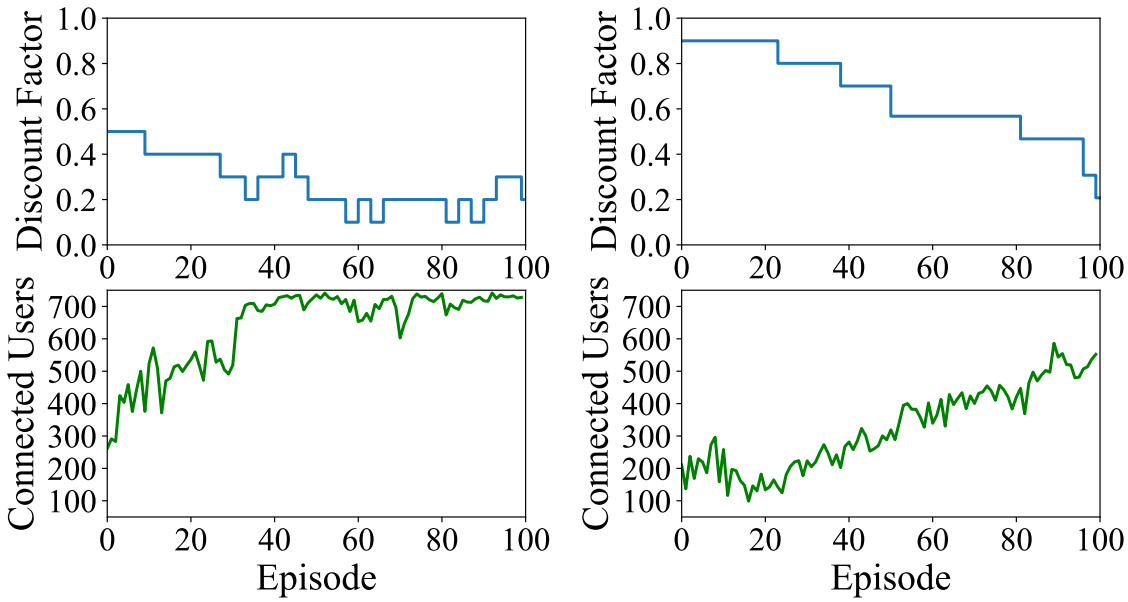
Figure 7.11: Comparison of hyperparameter tuning methods in DQN

static configuration (green line). It is interesting to note that periodic grid search (green line) achieved the same performance. However, the sharp dip at episode 70 to 80 shows a potential instability.

The proposed approach allows to gain more insights about the HPO process by analysing the history stored in the TM in conformation of metamodels of Figures 5.2 (page 63) and 7.5 (page 114). Listing 6 describes an EOL temporal query ran using the Hawk GUI to obtain the evolution of the rewards based on hyperparameter tuning. For each version of the TM where feedback was provided (`Agent = 'HAWK'`), the values of (`episode, gamma, reward`) are obtained. Figure 7.12 (a) depicts the results. The extracted information shows that the maximum value of the reward function $\mathcal{R}$ was 727.055 at episode 72 with discounting factor γ=0.204. Therefore, under the configuration $\Phi_{\lambda(\gamma,\kappa)}$ the optimal value found for the HPO problem of Equation 7.5 is:

$$\lambda^* \leftarrow \lambda(\gamma = 0.204, \kappa) \tag{7.7}$$

History-Aware HPO vs static hyperparameters The second experiment included an exhaustive analysis of the performance of the RL algorithm using different initial values for the discount factor. The comparison includes the analysis of the reward value function $\mathcal{R}$ with and without the proposed approach for each system configuration $\Phi_{\lambda_t(\gamma,\kappa)}$. The



(a) Gamma = 0.5. (b) Gamma = 0.9.

Figure 7.12: Reward and discount factor evolution, starting at $\gamma = 0.5$ and $\gamma = 0.9$ using history-aware HPO.

boxplots of Figure 7.13 display the results. As evidenced, by using the proposed history-aware HPO (in red) the system was able to reach greater maximum values (the upper end of the whiskers) for each system configuration. Furthermore, the interquartile ranges (boxes) in each case had a greater upper quartile. Regarding the medians, that represent the middle of the set, they were also greater for each case except for $\gamma = 0.2$ and $\gamma = 0.3$. This can suggest two things: i) the optimal values of γ is within this range $0.2 < \gamma^* < 0.3$, which reinforce the result obtained in experiment 1, and ii) the variance in the data corresponds to that of the optimiser exploring the hyperparameter value-space with probability ϵ.

The best performance of the RL system using the history-aware HPO approach occurred when the initial value of the discounting factor was the centre of the hyperparameter value-space, $\gamma_o = 0.5$ with an average of connected users of 636.104 and a median of 702.886. In the same manner, the worst performance occurred with $\gamma_o = 0.9$ with an average of 309.818 connected users and a median of 323.774. The temporal-query of Listing 6 was run for further analysis. As shown in Figure 7.12 (b), after exploring the hyperparameter value-space, the optimiser was going towards the optimal value of γ as the rewards increased. Thus, the system would have needed longer to find the optimal value.

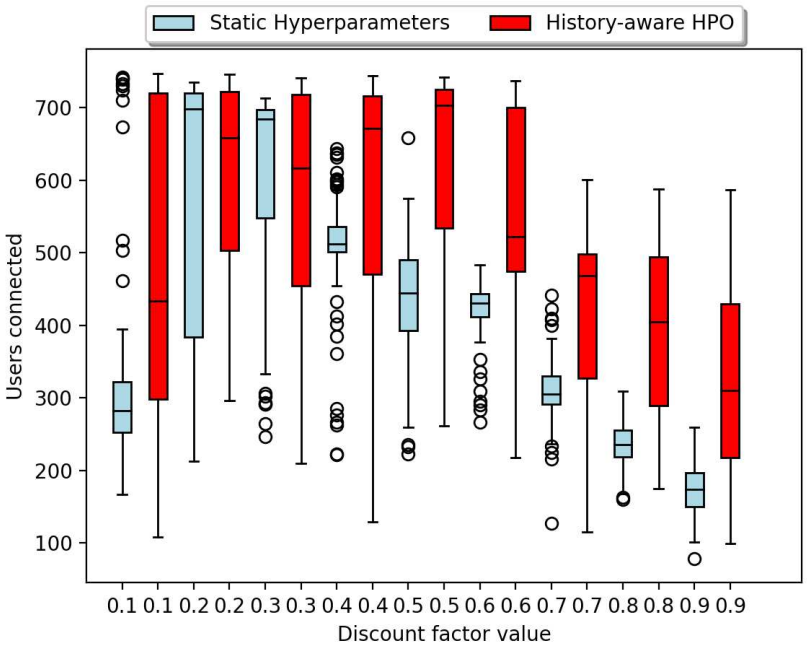


Figure 7.13: Comparison of history-aware hyperparameter optimisation vs static values

7.4 Discussion

This chapter has presented the implementation of Levels 1, 2 and 3 of the proposed research roadmap (Section 5.2) for enabling history-aware explainability in data-intensive AI-based SAS. By the introduction of an event-driven runtime monitoring approach (CEP), it is possible to analyse and correlate large amounts of data in real-time based on situations of interest or events while making use of parallel resources. CEP can be used as a real-time filter or for analysis of defined time-windows which enables short-term volatile memory capabilities. These capabilities combined with TMs and the temporal query language (presented in the previous Chapter 6), create a generalisable post-hoc approach for explainability that tackles the challenges in volume and throughput imposed by data-intensive systems as in the case of RL-based SAS.

Explanations are generated using CEP and TMs, and presented visually through graphs and plots in the case of Levels 1 and 2, and in the form of machine parsable logs for Level 3. The architecture outcome produced from this PhD work, *ETeMoX*, has been applied to an RL-based SAS for mobile communications. In order to test the model-agnosticism of the approach, three different RL algorithms were used. The experiments were performed during the training of the models, to help developers gain insights about the learning process while

they work on validating and improving their systems.

The objectives of the experiment of Level 1 (Section 7.3.3) were to keep track of the evolution of a metric throughout the history of the ABS SAS and to analyse the collaborative work in this multi-agent system after the system finished its execution. Different filtering criteria were defined and the results were evaluated. Costs of storing and retrieving the system's history as well as the accuracy of the explanations provided have been analysed. Uniformly sampling the history of the system every 10 time steps produced a good representation: a statistical t-test did not report significant differences in the query results compared to using the full history. This allows for extracting similar conclusions, while requiring less disk space (85% to 88% less) and taking less time to compute (88% less). Furthermore, the system was able to correlate, process and filter data at runtime, providing the ability to flexibly define filtering criteria for building a TM of complex events.

From the RL developer's point of view, retrieving historical information about the locations of the ABS, their SINRs, and how many users were connected at specific time steps provided a better understanding of how ABS agents interacted with the environment when using various RL systems. The interaction data that was collected during training and execution retained more information than just the learned policy: studying how these metrics evolve revealed interesting challenges encountered by the ABS SAS. By analysing the collaborations within the multi-agent system, the behaviour of each ABS provides an understanding about the reasons why the ABS SAS with the Q-Learning system had more connected users overall (i.e. maximum global reward) than SARSA and DQN. The Q-Learning ABS SAS had fewer handovers during simulation compared to the systems using SARSA and DQN.

Furthermore, the analysis of collaborations shows that the SARSA-based ABS SAS had a level of knowledge of their neighbour's position and capacity. In this case study, an ABS was rewarded if it increased its number of connected users, even if it reduced the number of users of other ABS agents. The total reward was not an implicit learning constraint for the ABS agents. Experiment 7.3.3 showed, in both SARSA and Q-Learning cases, that the number of handovers that happened during the simulation were minimal. The latter seems to demonstrate that the ABS agents were able to perceive the intention of neighbouring ABS agents in this controlled experiment. In contrast, in the case of DQN, multiple situations were found that show violations of the collaboration principle. Therefore, more analysis on

collaborative ABS SAS with DQN system is needed to have better understand how Q-value function works in this case study. Regarding Level 2 (Section 7.3.4), it was known in advance that the ABS system changed between two behaviours (i.e., exploration and exploitation) as part of the RL algorithms. The objective of the experiment was to discover situations where actions were taken by exploration and exploitation and to build an event graph that approximated the way in which it changed between the two events. The observed numbers of transitions between the two events and to themselves were counted, and these counts were used to populate the transition conditions in the event graph.

These results can help to prove hypothesis about the systems behaviour. With the presented global explanation, the user can discover how frequent and interrelated are these events. With this information, the developer was able to confirm that the system was acting as expected by contrasting the results with the hyperparameters defined at design time in the RL agent (e.g. in this case, a parameter for exploiting 90% of the time). Statistical and historical information can guide developers when working with AI systems to improve the learning performance: for example, by imposing a balance between exploration and exploitation if necessary. The exploration and exploitation query can give developers further insights about how to improve Q-Learning RL system performance by increasing the exploration time.

In relation to Level 3 (Section 7.3.5) the results from the conducted experiments showed the feasibility of the history-aware approach for HPO. Combining CEP and TMs offers both the short and long term memories required for hyperparameter tuning with reflective capabilities. An external entity, in this case a GL, is able to provide feedback to the ABS SAS at runtime for tuning its hyperparameters. The history-aware epsilon-greedy logic implemented in the GL allowed to explore the hyperparameter value-space with explicit long-term memory to remember good/optimal system configurations. The experiments provide valuable insights into the effects of the tuning of the discount factor and its influence on the stability of training and overall system performance (maximised cumulative rewards).

The presented work has helped the case study developers to gain deeper insights about the behaviour shown by the running system and the reasons for its decisions, and to allow external entities to provide feedback actively (e.g., Experiment 7.3.5.

7.4.1 Comparing *ETeMoX* with state-of-the-art techniques for explainability in SAS

This subsection contrasts the architecture produced by this PhD thesis (i.e., *ETeMoX*) with the state-of-the-art approaches for explainability in SAS mentioned in Section 2.2.2. Relevant approaches for explainability in RL (i.e., explainable RL or XRL) are also discussed. The research challenges to enable history-aware explainability stated in Sections 1.1.1 and 2.1.2 were i) the collection, organisation and storage of historical data produced by a SAS, ii) the query and extraction of information to provide explanations, and iii) the provision of explanations for different purposes and consumers. Based on these challenges, the features required by an architecture for history-aware explainability in SAS are defined below.

- **D1 On the Historical Analysis:** Can the architecture support the exploration of historical behaviour? The main focus of this PhD work is to demonstrate how the analysis of historical behaviour of a SAS can be used for conveying knowledge to external entities (either humans or machines) in the form of explanations.

- **D2 On SAS Model Agnosticism:** Can the approach be applied to different types of SAS? Generalizability is a key aspect that is targeted in this thesis. The use of the problem-independent execution trace metamodel (Section 5.1.1) enables a formal structure for storing and accessing the historical behaviour for different types of SAS.

- **D3 On the type of explanations:** Is the approach for explainability unobtrusive to the SAS decision-making? Being able to introduce explainability in SAS without the risk of impacting the decision-making process is important aspect targeted by this PhD work. Different to intrinsic explanations that are part of the SAS decision-making processes, post-hoc explainability aims to convey knowledge to users using surrogate models and processes.

- **D4 On the scope of the explanations:** Does the architecture support local and global explainability? Local explanations focus on data and provide individual explanations, helping provide trust on AI-model outcomes. A global explanation aims to provide a general understanding of how the AI-model works. This PhD work focuses on a complete solution that allows both types of explanation approaches.

- **D5 On the use of explanations for justifying:** Can the information extracted using the architecture be used for justifying SAS behaviour and decisions? Proving the reasons why a decision was made or an action was taken is one of the main scopes of explanation in SAS. These type of explanations refer to local explanations and can target different stakeholders, ranging from non-expert users to developers.

- **D6 On the use of explanations for discovering:** Can the information extracted using the architecture be used to discover knowledge from SAS? Asking for explanations is a helpful mechanism to learn new facts, collect information and consequently gain knowledge. Moreover, through the help of explanations, users can discover situations that only emerge at runtime.

- **D7 On the use of explanations for controlling and improving:** Can the explanations help users to refine their systems? Thanks to the information obtained through explanations, external entities can provide feedback to the SAS either after-the-fact or at runtime. Explaining a system through runtime monitoring to help improve it is one of the main objectives of *ETeMoX*.

- **D8 On the explanation generation:** Does the framework allow the collection and construction of explanations? This phase of explainability focuses on the methods that allow the construction of the explanation. What information is collected for an explanation, and how it is recorded and accessed, are aspects considered by this phase of explainability.

- **D9 On the explanation communication:** Does the architecture allow multiple presentation methods for explanations? This phase deals with what information will be provided to the explanation consumer and how will it be presented. As mentioned in Section 2.2, different presentation methods can be used to convey explanations (e.g., textual, graphical, motions among others).

- **D10 On the explanation reception:** Does the architecture have an awareness of the mental state of the targeted audience for providing explanations? This phase studies how well the explanation consumer understands the conveyed information. In order to assess this, typically, research relies on user studies, probabilistic models and subjective evaluations.

	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
Bencomo et al. [12]	✓	✗	✗	✗	✓	✓	✗	✓	✓	✗
Drechsler et al. [48]	✗	✓	✗	✗	✓	✓	✗	✓	✓	✗
Blumreiter et al. [19]	✗	✓	✗	✗	✓	✓	✗	✓	✓	✗
Li et al. [92]	✗	✗	✗	✗	✓	✓	✓	✗	✓	✓
Reynolds et al. [131]	✓	✓	✗	✗	✓	✓	✗	✓	✓	✗
Kordts et al. [87]	✗	✓	✗	✗	✓	✓	✗	✓	✓	✗
Diallo et al. [46]	✗	✗	✓	✗	✓	✓	✗	✓	✓	✗
Khalid et al. [80]	✗	✗	✗	✗	✓	✓	✗	✓	✓	✗
van der Waa et al. [151]	✗	✗	✓	✗	✓	✓	✗	✓	✓	✗
Juozapaitis et al. [77]	✗	✓	✓	✗	✓	✓	✗	✓	✓	✗
Madumal et al. [101]	✗	✓	✓	✗	✓	✓	✗	✓	✓	✗
Sequeira et al. [143]	✓	✓	✓	✗	✓	✓	✗	✓	✓	✗
Verma et al. [152]	✗	✗	✗	✗	✓	✓	✗	✓	✓	✗
ETeMoX	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗

Table 7.6: Comparison of explainability approaches for SAS

Table 7.6 compares *ETeMoX* with the state-of-the-art approaches for explainability in SAS using the criteria defined in the previously mentioned dimensions. The discussion of which of these works meet the defined criteria is presented next.

- **D1** On the Historical Analysis: Only few works in the literature have focused in the historical analysis for explainability in SAS. The interestingness elements approach proposed by Sequeira et al. in [143], collects data produced by the RL agent to present visual explanations. The authors conclude that the diversity of aspects captured by the different elements is crucial to help humans correctly understand an agent's strengths and limitations. Unfortunately, this approach lacks a formal structure to store the information, which is the advantage of using a MDE approach. Having this defined structure for representing decision traceability would allow algorithm developers to decouple the system to be explained from the rest of the explanation infrastructure. Reynolds et al. proposed in [131] automated provenance graphs to explain the behaviour of SAS based on runtime models. Provenance graphs relate the entities, actors and activities in the system over time, recording the reasons why the system reached its current state. However, the authors do not provide a querying system to traverse the history which is offered by the temporal query language in this thesis. Moreover, the authors do not consider or tackle scalability issues for storing and accessing history in data-intensive which is offered by *ETeMoX* with the use of

CEP.

- **D2** On SAS Model Agnosticism: Some works in the literature target this important aspect for an explainability approach that can be reused across domains [19, 77, 48, 143, 87, 101]. The approach more closely related to *ETeMoX* for explainability in SAS is the work proposed by Blumreiter et al. in [19]. The authors propose the Monitor-Analyze-Build and Explain (MAB-EX) framework for building self-explainable systems. This work proposes a framework that can be applied to different SAS. They claim that it can be extended to any type of systems. However, this is not demonstrated nor described. Moreover, the authors do not analyse the impact of the explanations' generation and communication that are important aspects to consider when generalising, which are tackled in this PhD work through data sampling (Section 7.3.3) and time-line annotation (Section 6.2.4).

- **D3** On the type of explanations: From the literature review, most of the works on explainability in SAS focus on intrinsic explanations or what they define as *self-explanations*. Post-hoc explainability is more recognised in the XRL community [77, 151, 143, 101]. However these approaches only target RL-loop based systems while *ETeMoX* can be used for different SAS like those based on the MAPE-K loop for example.

- **D4** On the scope of the explanations: Existing approaches for integrating explanatory capabilities into SAS have focused on local explanations, which help stakeholders understand specific decisions [12, 48, 19, 92, 131, 87, 46]. Global explainability aims at providing higher-level abstractions in order to convey a general understanding of the behaviour of the system [77, 151, 101]. Providing the capability of presenting both global and local explanations is a main contribution of this PhD work and *ETeMoX*. Using the historical behaviour of the SAS stored in an structured way allows the construction of explanations about specific situations as well as understanding the entire evolution of the system. To the authors knowledge, this is the first work that enables this capability specifically for SAS.

- **D5** On the use of explanations for justifying: From the literature, all the works revised cover this dimension. For example in [19], the system explains why a car is disallowed

to enter a a defined section because other cars are passing the obstacle in the opposite direction. In this PhD work, an example is the explanations of LTE in Section 6.2.3), where a decision apparently bad is justified because its long-term effect.

- **D6** On the use of explanations for discovering: For example, in [12] through the provided explanations the user can identify that using Bluetooth will lead to a greater fragmentation than using WiFi. Another example that depicts this dimension is the check-in scenario in [80] where the user discovered that there is a time threshold for registering. In this document, an example of explanations for discovering knowledge is the multi-agent collaboration experiment of Section 7.3.3, where the developer discovered that the ABS SAS using DQN produces more handovers than the SAS using SARSA or Q-Learning.

- **D7** On the use of explanations for controlling and improving: From the literature, the only work that tackles this dimension is [92] by Li et al. The authors proposed a probabilistic model to reason when to provide explanations for human-in-the-loop in SAS. Their target is to define when an explanation should be provided as a tactic of the SAS to support human interaction at runtime. However, their work does not specify how the user can interact with the SAS, which *ETeMoX* enables through the use of effectors as depicted in Sections 6.2.5 and 7.3.5. Moreover, through the experiment developed in Section 7.3.5, it is demonstrated how other systems can also consume explanations.

- **D8** On the explanation generation: Most of the works in the literature and *ETeMoX* focus on this phase. For example, in [87], the authors present a description language as well as a framework to support self-explaining ambient applications. *ETeMoX* supports this dimension through trace metamodels, TMs, and the temporal query language presented in Chapter 5.

- **D9** On the explanation communication: This phase is targeted by all the works in this literature review as well as by *ETeMoX*. For example, Maduma et. al., presented in [101] an approach for explainability based on causal lens. The authors used different presentation methods (e.g., text and graphs) in different case studies to depict the feasibility of the approach. Similarly, in *ETeMoX*, the presentation methods are

textual and graphical on predefined situations to be explained. Moreover, the user can query on-demand the system, extract the required information, and presented in the format of preference.

- **D10** On the explanation reception: As demonstrated by Li et al. in [92] probabilistic models can be used for assessing this dimension. This is the only work that analyses the user perspective as explanation consumer. The authors propose the use of the Opportunity-Willingness-Capability (OWC) model. This is a modelling approach used to quantify how much a participant who is performing a given task can affect systems. Only when the model is evaluated as true, explanations are provided. This is the least explored stage in explainability and it is out of the scope of this thesis.

ETeMoX, as an outcome of this PhD, is an open-source implementation that tackles most of the dimensions for an state-of-the-art approach for history-aware explainability in SAS. The code can be found the SEA research group repository `https://gitlab.com/sea-aston/etemox`.

Chapter 8

Conclusions

This PhD dissertation presented two comprehensive experimental studies that were designed to investigate how temporal models (TMs) (from model-driven engineering—MDE) combined with complex event processing (CEP) (from event-driven runtime monitoring—EDM) can enable explainability based on the historical behaviour of self-adaptive systems (SAS). Furthermore, this PhD work has studied how the use of runtime models extended with short and long-term memory can provide the abstraction, analysis and reasoning capabilities needed to support explanations when using AI-based SAS. Finally, enabling filtering capabilities on data-streams produced by SAS allows the effective use of parallel resources.

For an explanation to meet its purpose, it needs to be expressed in a way that is understandable for the recipient. In this PhD dissertation, the primary focus was on explanations targeting SAS developers and knowledgeable users or domain experts. These two groups of users are familiar with developing SAS (in the case of the former group) and using SAS. Hence, they are interested in understanding, diagnosing, and refining such systems in the given application contexts. Explanations shown by the running system help stakeholders who are observing the system's behaviour to analyse and trace actions that can help fix potential faults, convey knowledge to support better informed decision-making, and foster trust of end users.

A four-stage research roadmap for history-aware explainability was followed. This research roadmap was previously proposed in [59] and then refined in [119]. The stages consisted of 1) forensic history-aware explainability, 2) live history-aware explainability, 3) externally-guided history-aware decision making, and 4) autonomous history-aware self-

adaptive systems (i.e., self-explanation). This PhD research has focused on the first three stages of the research roadmap. The design, implementation and evaluation of these levels have been analysed through two SAS case studies: Remote Data Mirroring (RDM) and Airborne Base Stations (ABS).

RDM (Chapter 6) ensures data availability to avoid data loss by replicating (i.e., mirroring) data across servers. RDM is a MAPE-K-based SAS which uses Bayesian Learning and Partial Observable Markov Decision Processes for its self-adaptive decision-making [61]. RDM switches between two topologies, minimum spanning tree and redundant topology, while balancing three non-functional requirements (NFRs): energy consumption, reliability, and performance. Due to its proactive self-adaptation technique, this SAS can expose surprising behaviour for a developer that is monitoring the system. Long-term effects (LTEs) of immediate actions are an example of surprising behaviour. LTEs depict situations when an action with a negative immediate impact produces an increase in the SAS performance in the long term. The experimentation for Levels 1 and 2 focused on the explanation of these types of situations both after-the-fact and while the system is running. The proposed approach in TMs supports forensic analysis and interactive diagnosis based on explanations. Regarding level 3, through TMs and the temporal query language, users can query the history of the system to improve their understanding about the behaviour exposed by the SAS and provide feedback if required using effectors. The type of explanations presented, either textual or graphical, fit the audience (RDM developers), who are able to understand the data representations and can extract knowledge from their system.

Regarding the second case study, the ABS SAS (Chapter 7) uses Reinforcement Learning (RL) to move autonomously a set of base stations for providing connectivity to as many users as possible. The ABS SAS performs the necessary calculations to estimate the Signal-to-Interference-plus-Noise Ratio and the Reference Signal Received Power towards its global goal of maximising rewards (number of users connected). In order to test the generalizability of the proposed approach, three variants of the underlying RL algorithms are used: Q-Learning, SARSA and DQN. The experiments are performed during training to support developers in gaining insights about the learning process while they work on validating and improving their systems. Due to the amount of data that RL and the ABS SAS in particular generate, the initial approach based only on TMs did not scale. Consequently, CEP was included as a real-time filter to select relevant points in time to be recorded in the

TMs, instead of recording whole history. This architecture resulting from the combination of TMs and CEP was named Event-driven TEmporal MOdels for eXplanations (*ETeMoX*) which is an outcome of this PhD work. Accordingly, Levels 1, 2, and 3 of the proposed research roadmap were implemented to show the feasibility of the approach. Through the use of *ETeMoX*, developers were able to obtain explanations about both the evolution of a metric and relationships between metrics. They were also able to track relevant situations of interest that spanned over time (i.e. over time windows) while making efficient use of computational resources. Finally, in Level 3, it is shown how an external entity can benefit from explanations in order to provide feedback to the SAS.

From the developers' point of view, the RDM and ABS case studies presented different explainability challenges to be addressed. On one hand, RDM considered the satisfaction of NFRs for its decision-making which are not directly measurable nor easy to understand. Furthermore, its proactive self-adaptation nature can surprise an user observing the system. Therefore, providing the right tools for asking for explanations about situations that are connected over time is crucial for trustworthiness and for promoting the use of the system which is provided by TMs. On the other hand, the ABS SAS used RL for its decision-making which is a promising area of ML where autonomous agents learn through trial-and-error how to find good solutions to a problem. Thus, the underlying decision-making criteria may become opaque to users that interact with the system and who may require explanations about the system's reasoning. Furthermore, the developers wanted to study the collaborative behaviour of this multi-agent SAS which is not directly observable. *ETeMoX* provided an scalable solution that allows developers exploring, validating and improving their systems.

Based on the results of the experimental studies conducted in this PhD dissertation, it has been demonstrated how TMs can enable history-aware explainability in SAS. Additionally, CEP was used to tackle the challenges in volume and throughput posed by data-intensive systems. For example in the RL-based ABS SAS. Explanations obtained using the proposed approach were used for different purposes, such as explanations for justifying surprising behaviour from a SAS (e.g., LTE in RDM, § 6.2.3), explanations for runtime verification and validation (e.g, evolution of a metric, Sec. 7.3.3, and exploration vs exploitation in ABS Sec. 7.3.4), explanations for discovery (e.g. handovers in ABS Sec. 7.3.3), and explanations for controlling and improving the system decision-making (e.g, human-in-the-loop

in RDM Sec. 6.2.5 and hyperparameter optimisation in ABS Sec. 7.3.5). In relation to the explanation phases defined in [3], this work tackles the first two: i) the *explanation generation* is the construction of the TM, and ii) the *explanation communication* is the extraction of the information using the temporal query language and the presentation of explanations either textually or graphically.

8.1 Answering the Research Questions

The studies conducted to investigate the research questions in this PhD dissertation demonstrate the feasibility of the approach towards the targeted three stages of the research roadmap on history-explainability. In Section 1.1.2, three main research questions were presented. After the research carried out in this PhD dissertation, the answers to these questions are as follows:

- *RQ1: How can model-driven engineering and runtime monitoring enable scalable and structured data storage?*

  The integration of model-driven engineering (MDE) and runtime monitoring (RTM) has been investigated in this PhD thesis to enhance the explanatory capabilities of SAS systems based on their historical behaviour. However, storing the historical data of a running SAS can be costly, both in terms of storage space and performance. To overcome this challenge, the thesis proposed different approaches for efficiently processing large amounts of data generated by a SAS, such as logs. These approaches were based on RTM and enabled the extraction of valuable information in the form of explanations while following a defined structure based on models from MDE. By combining the strengths of MDE and RTM, the thesis demonstrated the potential to provide useful insights into the behaviour of a running SAS system and explain any issues that may arise. This work represents an important contribution to the field of software engineering and has the potential to benefit organisations that rely on SAS systems for critical operations.

  - RQ1.1: What is the effectiveness of the proposed solution which combines Temporal Models from model-driven engineering and Complex Event Processing from runtime monitoring for history-aware explainability?

The proposed solution, which combines Temporal Models (TMs) based on runtime models and a temporal graph database with Complex Event Processing (CEP), is effective for history-aware explainability. This solution allows efficient storage of historical data by only saving changes in the model to save disk, using a copy-on-write approach. The structured format defined by the problem-independent trace metamodel (from Section 5.1.1) enables the extraction and storage of information for constructing explanations. By defining a translator component from system logs to this metamodel, the infrastructure and queries written against the trace metamodel can be reused. Additionally, the trace metamodel enforces a consistent level of abstraction across multiple algorithms, facilitating comparisons among various SAS decision-making algorithms in the same domain. However, TMs alone may not be able to scale for data-intensive SAS (e.g., RL systems), and a complete solution for history-aware explainability, such as the proposed *ETeMoX*, integrates CEP to handle vast amounts of data at runtime. CEP is used for detecting matches to event patterns that need to be stored, instead of keeping the entire history.

  - RQ1.2: How does the proposed approach compare to existing approaches?
    Compared to existing approaches, such as those presented in Section 7.4.1 the proposed solution offers a more efficient and scalable approach to history-aware explainability. Different dimensions have been analysed and compared against state-of-the-art techniques to provide explanations in SAS. These dimensions are related to the approaches followed by the existing works regarding their scope, form, construction methods, objective, and target. The open-source architecture outcome of this thesis *ETeMoX* meets the defined criteria in 9 out 10 dimensions. The detailed comparative information can be found in Section 7.4.1.

- *RQ2: How can the exploration of the stored SAS history support developers wishing to improve or validate their systems?*
  The ability to perform queries on a given data storage is crucial for analysing historical information, which is particularly important for providing explanations to users and stakeholders in cases where a SAS exhibits unexpected or surprising behaviour (related to RQ1). By utilising execution-tracing features, it becomes possible to analyse the

system's history and monitor decision-making performance against available evidence at various points in time. The proposed methodology in this PhD work for structuring, processing, and analysing historical data provides developers with a valuable tool to better understand, validate, and refine their systems. This methodology can also aid in identifying potential issues that may arise in the future and provide insights into the root causes of these issues. Overall, the methodology proposed in this work represents an important contribution to the field with the potential to benefit developers, users, and stakeholders alike.

- RQ2.1: How useful is the proposed approach, which uses post-hoc explanations extracted during runtime monitoring, in enabling developers or external entities to monitor and refine SAS systems?

  The proposed approach, which uses post-hoc explanations extracted during runtime monitoring, is useful in enabling developers or external entities to validate and refine SAS systems. By promoting users' understanding of the system's decision-making and behaviour, explainability is key to discovering the system's flaws. To this end, the monitoring system should be capable of unveiling potential anomalies in the monitored system. The proposed approach uses a defined structure (i.e., TMs) that allows the exploration of stored past history through reusable Epsilon Object Language (EOL)-based queries. The EOL was extended with time-aware primitives and the concept of history, enabling the use of temporal query language and timeline annotations to tag specific moments in history where an event of interest happened. Furthermore, CEP can be used to analyse specific events at runtime while they occur in time or concrete time windows, allowing for the reuse of queries or event patterns across different domains.

- RQ2.2: What insights can be gained to improve the system's behaviour and decision-making using the proposed approach?

  Experiments 6.2.4 and 7.3.4 demonstrate how explanations can be used to validate developers' hypotheses and gain insights to improve the system's behaviour and decision-making. For instance, in Experiment 6.2.4, the proactive self-adaptation approach of the RDM using POMDPs when an action is performed towards a future reward instead of an immediate one can be confirmed. In Ex-

periment 7.3.4, developers of the ABS SAS can validate if the system behaves as expected regarding the initial hyperparameter, such as the exploration and exploitation rate. By using the proposed approach, developers can refine their systems if required, while users can gain confidence in the system and potentially influence its improvement passively or actively.

- *RQ3: How can external entities using history-aware explanations influence the SAS decision-making in an informed way?*

  The use of history-aware explanations is a powerful tool for enhancing collaboration between a SAS and external entities, such as consumers or recipients of explanations. This PhD thesis discussed that by utilising the extracted information from the SAS's historical data, users and stakeholders can interact with the system either passively or actively, thereby enabling them to make more informed decisions. In addition, by providing explanation-driven feedback, users and stakeholders can help to improve the performance and functionality of the SAS. The use of history-aware explanations can also facilitate communication and understanding between the SAS and its users, which is especially important in cases where the SAS is used for critical operations or decision-making.

  - RQ3.1: How can a history-aware architecture enable a bi-directional communication approach between stakeholders and a SAS?

    This PhD work advocates for the ability of a SAS to explain its behaviour and communicate how it reached its current state, especially if the SAS learns from its past execution. External entities that receive these explanations may feel that the SAS is not taking their decision-making criteria and priorities into account, but they can provide valuable feedback to enhance the SAS performance. To enable this two-way communication, a SAS should provide access to historic data, track its decision-making reasons over time, and offer capabilities called "effectors" to allow external entities to steer its decision-making process.

  - RQ3.2: How can feedback from external entities through explanations be integrated into the SAS decision-making process to enhance its performance?

    The experiments conducted at Level 3 (Sections 6.2.5 and 7.3.5 of the research roadmap) demonstrate how external entities can provide useful feedback to the

SAS based on received explanations. In Section 6.2.5, explanations were used to introduce human-in-the-loop, and the feedback provided allowed the SAS to prioritise a non-functional requirement and improve performance. In the experiment of Section 7.3.5, an external system used the information provided by explanations to provide feedback that updated the RL-based SAS initial conditions. The results show that the proposed history-aware architecture, *ETeMoX*, significantly improves SAS performance through the feedback provided by external entities.

8.2 Contributions Revisited

To summarise the major contributions of this PhD dissertation are as follows:

i *The study, design and implementation of the first three levels of the gradual approach for a spectrum of reflective capabilities for history-aware self-adaptive systems* (Section 5.2). This research roadmap was revised, analysed and adapted from [59]. Level 1, "Forensic-history aware explanations", focuses in conveying knowledge to different stakeholders after the SAS has finished it execution. The system's history is stored in a TM and analysed through the temporal query language. The features developed for this level are reused in the subsequent levels. Level 2, "Live history-aware explanations", describes the passive analysis of the SAS online while the system is in execution. It exploits the storing and querying capabilities of Level 1 but requires extra features to extract and present explanations at runtime. Level 3, "Externally guided history-aware decision making with explanation capabilities" allows external entities to interact with the SAS at runtime. The explanation consumers (humans or machines) are able to perform changes in the SAS through a set of effectors. These levels were implemented and analysed in two SAS case studies considering different dimensions, such as performance overheads, storage space impact and feasibility.

ii *A novel approach for combining runtime models and temporal databases that produces TMs (Section 5.1), which records in an structured fashion the goals of a SAS, its decisions, its observations and its reasoning over time.* TMs based on the problem-independent execution trace metamodel (Section 5.1.1) are proposed and used as key enablers for efficient and reusable storage and analysis of a SAS's history.

iii *An evaluation of a set of temporal assertions that enable access to the stored system's history to extract information that will conform explanations.* The queries developed for each experiment in Chapters 6 and 7 allowed the extraction of information of situations of interest spanned over time. Explanations about specific points in time, a set of time points (i.e., time-windows) and the whole history were extracted using the time-aware model querying language.

iv *A novel generalizable architecture based on Temporal Models and event-driven runtime monitoring, called* ETeMoX *(presented in Chapter 7), for the extraction of history-aware explanations from data-intensive SAS in a post-hoc manner.* In order to tackle the computational constrains such as storage capacity and time to response, TMs were combined with the event-driven approach (CEP) for runtime monitoring based on situations of interest (i.e., events). The criteria for storing the SAS history can be configured through event patterns on a CEP engine. For example, a certain data rate can be imposed, or the history may only keep points in time where certain conditions are met instead of the full history, saving memory and disk resources. The *ETeMoX* framework and the user manual are available at [118].

v *Two comprehensive experimental studies and accompanying analyses, designed to investigate the proposed research questions about effects of history-aware explanations, that establish:*

- Temporal models are a convenient solution for storing the history of SAS. The provision of temporal assertions based on the temporal query language allows access and exploration of the SAS history.
- Temporal models combined with event-driven runtime monitoring allows leveraging the challenges in volume and throughput posed by data-intensive systems in a resource-aware manner.
- Explanations based on the historical behaviour of SAS permit developers and other stakeholders the validation and verification about the system's runtime decision-making processes.
- The consumers of explanations, humans or machines, can benefit from the information received for internal reasoning, or for providing feedback, online or offline, to the system based on the information acquired.

8.3 Limitations, Direction and Future Work

The two experimental studies presented in Chapters 6 (pages 74 to 102) and 7 (pages 103 to 144) were designed to answer the three research questions detailed in Chapter 1. After the research carried out during this PhD work, the feasibility of proposed approach based on Temporal Models for History-Aware Explainability in SAS has been demonstrated. Using the post-hoc explainability approach, SAS developers and knowledgeable users are able to explore the system's history and obtain local explanations about specific decisions taken by the SAS, and global explanations that promote a general understanding of the behaviour of the system. The explanations, whether textual or graphical, allow stakeholders to discover and justify SAS decisions, verify systems requirements and goals, control, and improve the system runtime performance based on the obtained information.

Based on the carried out experiments, the goals of this PhD dissertation have been successfully met in the context of the research roadmap and experimentation. However, there are still some limitations and threats to validity identified during the carried out research. They are defined as technical limitations and theoretical limitations that open exciting future research directions.

8.3.1 Technical limitations and possible research directions

From the technical point of view, the approach presented in this PhD dissertation focuses on the use of historical information to convey knowledge about SAS behaviour. This implies that the decision-making history has to be tracked and stored in an structured and effective manner. Regarding structure, in the current implementation, the trace logs produced from a SAS need to be reshaped in order to comply with the problem-independent metamodel. Defining this log-to-model parser requires user expertise which can be a deterrent for lay-users. In terms of scalability, the TMs used in this PhD work are based on the Greycat TGDB. Greycat uses a copy-on-write approach to store only the nodes and edges changed between time-points, saving disk space. Nonetheless, when the history increases the disk space required also increases. CEP was introduced to select only situations of interest that will conform explanations instead of the whole history. These situations can be a specific point in time or situations spanning over the time. The approach scaled well, however, it can be further improved. In terms of filtering criteria, CEP time window capabilities could

be exploited for focusing on the last n versions to only keep a time window of the history in the TMs. Moreover, enabling the capability of forgetting time-points that are no longer of interest to the TMs would help keeping resource consumption bounded and requires further studies.

The information that constructs the explanations is obtained through querying the stored information in TMs using the temporal query language. The query language allows exploring the history back and forth looking for situations of interest spanned over time. Some of the developed queries can be reused across different case studies. For example *evolution of a metric* was used in experiments of Sections 6.2.5, 7.3.3, and 7.3.5, or *the exploration vs exploitation* query of Section 7.3.4 that is recurrent in every RL-based system. These queries can be found in the *ETeMoX* project repository[1]. However, developing domain-specific queries requires expert knowledge. Initial guidelines on how to develop temporal queries using EOL can be found in the Eclipse Hawk project website[2] but further studies would be required depending on the application domain and scope.

The presented explanations targeted domain experts and developers. The textual and graphical presentation methods suited the audience, providing the users the tools to validate and improve their systems. The Hawk user interface (GUI) was used for Level 1 for forensic analysis in both case studies. For Levels 2 and 3 a dedicated GUI was developed. This GUI allowed the users to implement queries to be executed at runtime (e.g., in Level 2). This GUI can be further developed to meet users requirements. Additionally, the queries used for different consumers can be explored for providing the right level of abstraction depending on the user' level of expertise. Using the defined GUI the user can provide feedback to the SAS using effectors (as shown in Level 3). These effectors can vary depending on the monitored SAS. For example, instead of updating hyperparameters and NFR priorities as shown in the experiments of Sections 6.2.5 and 7.3.5, the effectors can force an action to be taken by the SAS. Moreover, the exploration of different explanations presentation techniques for different users and purposes is possible future research line. Finally, the current implementation is designed to be used in a central control node for a SAS. It would require further efforts to be applied to the case of a distributed SAS such as the ones discussed in [161].

[1] https://gitlab.com/sea-aston/etemox/-/tree/main/hawk/EOLQueries
[2] https://www.eclipse.org/hawk/advanced-use/temporal-queries/

8.3.2 Theoretical limitations and possible research directions

In relation to theoretical limitations, two main areas of research were defined as out of the scope of this PhD dissertation. The first one corresponds to human factors in the generation and reception of explanations. The focus of this PhD work was to describe, develop and test an architecture that would enable post-hoc explainability using MDE and EDM. The validation of the provided explanations was performed by the SAS developers of the different case studies. Therefore, further studies using other SAS scenarios and developers about how explanations are understood, are required. The type of explanations presented, either textual or graphical, make the targeted audience able to understand the data representations and extract knowledge from their system. However, in order to fully analyse the role of history-aware explanations for the SAS community, additional user studies would be required. These studies should consider different aspects such as the SAS developer's level of expertise (e.g., senior or beginner), the underlying system architecture (e.g., following or not following a MAPE-K feedback loop), the type of explanations (i.e., local or global), and the presentation methods (i.e., textual, graphics) and follow ethical guidelines for user studies.

Furthermore, this work has focused on explanations for developers and knowledgeable users. However, there are other stakeholders that can also be affected by SAS. As mentioned in [70] and [162], there are different target audience profiles, each one with a different technical background. They include: developers, experts (knowledgeable users), non-technical users, executives and regulatory agencies. Therefore, it is important to define the intended target audience and the pursued goal of the generated explanations (e.g., trustworthiness, informativeness, fairness, etc.). As such, more studies related to the explanation requirements from different actors in different scenarios are necessary.

A different area of research yet to be explored corresponds to Level 4 of the proposed research roadmap, *"Autonomous history-aware decision-making with explanation capabilities"* 5.2. The recipient of the explanations could be the system itself (i.e., *self-explanations*). With self-explanation support, there is potential for the system to use its own history as another input to underpin its own decision-making. For example, the SAS could use the history for the identification of similar situations in the past and the consequent evaluation of the long-term performance of the decisions that were made at those times. This

evaluation could be factored into the SAS' perceived utility levels of the available options for adaptation. The system could double-check the long-term performance of those decisions and establish a confidence level on its own prediction model. Decisions made on top of self-explanation could be tracked for automated control accountability. Furthermore, if the SAS is able to explain itself, this knowledge and information could be also useful for different users and other type of systems.

The studies conducted within this PhD dissertation explore the synergy of MDE and EDM for enabling explainability in SAS. This work provides a foundation on which the open research questions mentioned above can be explored, which are all viable avenues for future research into trustworthy and reflective SAS.

Appendix

Chapter 6: Explaining SAS with Temporal Models

Algorithm 3 Query to detect proactive adaptation: the long term effects of immediate actions. L is the current runtime log, T the set of timeslices in L, $S^{\text{NFR}}(t)$ the level of satisfaction of the NFR at timeslice t, and α^{NFR} the threshold for the NFR.

Result $= \{L\}$
$T_B = \{t \in T | S^{\text{NFR}}(t) < \alpha^{\text{NFR}}\}$
for each $t_b \in T_B$ **do**
    if $S^{\text{NFR}}(t_b + 1) < S^{\text{NFR}}(t_b) \wedge$
    $\exists n \in \mathbb{N}_{>0}, \forall_j \in [1, n] \,|$
    $S^{\text{NFR}}(t_b + j + 1) > S^{\text{NFR}}(t_b)$ **then**
        Add (t_b, n) to **Result**
    end if
end for
Result: Sequences showing proactive adaptation.

Listing 1: EOL query to analyse the system history looking for LTEs at runtime using timeline annotation

```
return Decision.latest.all.first.nfrBeliefsPre
  .selectOne(blf | blf.nfr.name.contains('Reliability'))
  .whenAnnotated('shortTermNegativeAction')
  .ifUndefined(Sequence {})
  .versions.collect(v | v.intervalInformation())
  .reject(s | s.containsKey('noMatchAt'))
;

operation NFRBelief intervalInformation() : Map {
    var intervalStart = self.unscoped.next;
    if (intervalStart.isDefined()) {
      var actionTaken = self.actionTaken();
      var interval = intervalStart.sinceThen
      .before(w | w.actionTaken() <> actionTaken
        or w.estimatedProbability < w.prev.estimatedProbability
        );

      if (interval.isDefined()) {
          return Map {
            "matchAt" = self.versionInformation(),
            "versions" = interval.versions.collect(v|v.versionInformation())
          };
      }
    }
    return Map { "noMatchAt" = self.time };
}

operation NFRBelief versionInformation() : Map {
  return Map {
    "timesliceID" = self.eContainer.eContainer.timesliceID,
    "actionName" = self.eContainer.actionTaken.name,
    "nfrName" = self.nfr.name,
    "estimatedProbability" = self.estimatedProbability
  };
}

operation NFRBelief actionTaken(): String {
  return self.eContainer.actionTaken.name;
}
```

Listing 2: EPL pattern to sample the data every 10 steps

```
@public @buseventtype @Name("Sampler")
insert into Sampler
select drone as complexEventInfo
from pattern [every drone = DronesLog(drone.step%10=0)]
```

Listing 3: EPL pattern to select when the system performs an action based on *exploration*

```
@public @buseventtype @Name("Exploration")
expression selectedActionValue{
        droneLog => case drone.qtable.action
          when "east" then drone.qtable.position.east
          when "west" then drone.qtable.position.west
          when "south" then drone.qtable.position.south
          when "north" then drone.qtable.position.north
          when "stay" then drone.qtable.position.stay
        end}
expression maxValue{
        droneLog => max(drone.qtable.position.east,
                        drone.qtable.position.west,
                        drone.qtable.position.south,
                        drone.qtable.position.north,
                        drone.qtable.position.stay) }
insert into Exploration
select drone as Log
from pattern [every drone = DronesLog] as droneLog
where maxValue(droneLog) != selectedActionValue(droneLog)
                        and maxValue(droneLog) != 0
```

Listing 4: EOL query to select when the system performed an action based on *exploration*

```
var results : Sequence;

for (decision in Decision.latest.all) {
  var totalCount = decision.versions.size;
  var exploration = decision.versions
                    .select(v|v.isExploration());
  var totalExploration = exploration.size;

  results.add(Map {
    'total' = totalCount,
    'exploration' = totalExploration
  });
}
return results;

operation Decision isExploration() : Boolean {
  var maxAB = self.actionBeliefs.estimatedValue.max();
  var actionTakenName=self.actionTaken.name;
  var actions = self.actionBeliefs.action;
  var actionTakenValue = actions
                .selectOne(a|a.name = actionTakenName)
                .revRefNav_action.estimatedValue.first;
  return actionTakenValue <> maxAB and maxAB <> 0.0;
}
```

Listing 5: Esper EPL pattern to select when the system is on an stable condition on a defined time-window

```
@public @buseventtype @Name("isStableAVG")
insert into isStableAVG
select w.averageWindow as avg,
       w.episode as episode,
       'CEP' as agent,
       CAST(a3.drone_number as INT) as drone_number,
       a3.step as step,
       a3.gamma as gamma
from pattern [every a1 = AvgByEpisode ->
                    a2 = AvgByEpisode ->
                    a3 = AvgByEpisode ->
                    w = EpiWinAVG]
where
  w.episode = a3.episode and
  Math.abs(a1.avg-w.averageWindow) < 30 and
  Math.abs(a2.avg-w.averageWindow) < 30 and
  Math.abs(a3.avg-w.averageWindow) < 30
```

Algorithm 4 History-aware epsilon-greedy hyper-parameter optimisation: Optimising the discounting factor - Variables definition and initialisation

variables def
R, reward function
$reward$, reward by episode
$rewards$, array of rewards by episode
$rewardWinAvg$, reward average by time window
$maxRW$, maximum reward average by time window
γ, discounting factor
Γ, value of γ that produces $\max reward$
γ', updated discounting factor
cV, cutoff value
x, time window size
t_o, time window initial point
ϵ, probability of exploring γ-value space
edf, ϵ decay factor
gvf, γ variance factor
end variables def
$\gamma \leftarrow 0$
$x \leftarrow 3$
$t_o \leftarrow 0$
$reward \leftarrow 0$
$rewards \leftarrow size[x]$
$rewardWinAvg \leftarrow 0$
$maxRW \leftarrow 0$
$c \leftarrow 0$

Algorithm 5 History-aware epsilon-greedy hyper-parameter optimisation: Optimising the discounting factor - Process

```
episodes ∈ ℝ
for each e ∈ episodes do
    rewards ← add(reward_e)
    γ ← γ at e
    if e = t_o + x then
        rewardWinAvg = rewards/x
        if ∏_{i=t_o}^{t_o+x} |reward[i] − rewardWinAvg| < rewardWinAvg ∗ cV then
            if maxRW < rewardWinAvg then
                Γ ← arg max_γ R(γ)
                γ' ← Γ                                   ▷ Act greedily
                maxRW ← rewardWinAvg                     ▷ Update the maximum reward average
                initialise flag                          ▷ random integer number between 0 and 1
            else
                ϵ ← ϵ ∗ edf
                if n < ϵ then                            ▷ n uniform random decimal number between 0 and 1
                    switch flag do
                        case 0                           ▷ decrease discounting factor
                            γ' ← γ − gvf
                            if rewardWinAvg_e < rewardWinAvg_{e+x} then
                                continue
                            else
                                flag ← 1
                                γ' ← Γ
                            end if
                        case 1                           ▷ increase discounting factor
                            γ' ← γ + gvf
                            if rewardWinAvg_e < rewardWinAvg_{e+x} then
                                continue
                            else
                                flag ← 0
                                γ' ← Γ
                            end if
                else
                    γ' ← Γ                               ▷ Act greedily
                end if
            end if
        end if
        t_o = e
        γ ← γ'
        return γ
    end if
end for
```

Listing 6: EOL query to analyse the system history regarding the discounting factor and reward values

```
var results : Sequence;
for(agentVer in Agent.latest.all){
  var agentHawk = agentVer.versions
  .select(c|c.name='HAWK');
  for (agent in agentHawk){
    var episode = agent.eContainer
        .observations.first
        .measurement('Episode');
    var gamma = agent.eContainer
        .observations.first
        .measurement('Gamma');
    var average = agent.eContainer
        .observations.first
        .measurement('Reward by episode');
    results.add(Map {
            'gamma' = gamma,
            'reward' = average,
            'episode of change' = episode});
  }}
return results;
operation Observation measurement(n:String) {
var value;
 if (self.measurements.isDefined()) {
    value= self.measurements
    .selectOne(m|m.measure.name = n);
    }
  return value;
}
```

List of References

[1] A. Adadi and M. Berrada. Peeking inside the black-box: a survey on explainable artificial intelligence (xai). *IEEE access*, 6:52138–52160, 2018.

[2] E. Albassam, J. Porter, H. Gomaa, and D. A. Menascé. Dare: A distributed adaptation and failure recovery framework for software systems. In *2017 IEEE International Conference on Autonomic Computing (ICAC)*, pages 203–208. IEEE, 2017.

[3] S. Anjomshoae, A. Najjar, D. Calvaresi, and K. Främling. Explainable agents and robots: Results from a systematic literature review. In *18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), Montreal, Canada, May 13–17, 2019*, pages 1078–1088. International Foundation for Autonomous Agents and Multiagent Systems, 2019.

[4] M. Backmann, A. Baumgrass, N. Herzberg, A. Meyer, and M. Weske. Model-driven event query generation for business process monitoring. In *International Conference on Service-Oriented Computing*, pages 406–418. Springer, 2013.

[5] F. Barbon, P. Traverso, M. Pistore, and M. Trainotti. Run-time monitoring of instances and classes of web service compositions. In *2006 IEEE International Conference on Web Services (ICWS'06)*, pages 63–71. IEEE, 2006.

[6] L. Baresi and S. Guinea. Event-based multi-level service monitoring. In *2013 IEEE 20th International Conference on Web Services*, pages 83–90. IEEE, 2013.

[7] K. Barmpis and D. Kolovos. Hawk: Towards a scalable model indexing architecture. In *Proceedings of the Workshop on Scalability in Model Driven Engineering*, BigMDE '13, pages 6:1–6:9, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2165-5. URL `http://doi.acm.org/10.1145/2487766.2487771`.

[8] K. Barmpis and D. S. Kolovos. Comparative analysis of data persistence technologies for large-scale models. In *Proceedings of the 2012 Extreme Modeling Workshop*, XM '12, pages 33–38, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1804-4. doi: 10.1145/2467307.2467314. URL `http://doi.acm.org/10.1145/2467307.2467314`.

[9] K. Barmpis, S. Shah, and D. S. Kolovos. Towards Incremental Updates in Large-Scale Model Indexes. In G. Taentzer and F. Bordeleau, editors, *Modelling Foundations and Applications*, number 9153 in Lecture Notes in Computer Science, pages 137–153. Springer International Publishing, July 2015. ISBN 978-3-319-21150-3 978-3-319-21151-0.

[10] B. Baudry, C. Nebut, and Y. Le Traon. Model-driven engineering for requirements analysis. In *11th IEEE international enterprise distributed object computing conference (EDOC 2007)*, pages 459–459. IEEE, 2007.

[11] N. Bencomo and L. H. G. Paucar. Ram: Causally-connected and requirements-aware runtime models using bayesian learning. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 216–226. IEEE, 2019.

[12] N. Bencomo, K. Welsh, P. Sawyer, and J. Whittle. Self-explanation in adaptive systems. In *2012 IEEE 17th International Conference on Engineering of Complex Computer Systems*, pages 157–166. IEEE, 2012.

[13] N. Bencomo, A. Bennaceur, P. Grace, G. Blair, and V. Issarny. The role of models@ run. time in supporting on-the-fly interoperability. *Computing*, 95(3):167–190, 2013.

[14] N. Bencomo, R. B. France, B. H. Cheng, and U. Aßmann. *Models@ run. time: foundations, applications, and roadmaps*, volume 8378. Springer, 2014.

[15] N. Bencomo, S. Götz, and H. Song. Models@ run. time: a guided tour of the state of the art and research challenges. *Software & Systems Modeling*, 18(5):3049–3082, 2019.

[16] A. Benelallam, T. Hartmann, L. Mouline, F. Fouquet, J. Bourcier, O. Barais, and Y. L. Traon. Raising Time Awareness in Model-Driven Engineering: Vision Paper. In *2017*

ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS), pages 181–188, Sept. 2017. doi: 10.1109/MODELS.2017.11.

[17] A. Bertolino, A. Calabrò, F. Lonetti, A. Di Marco, and A. Sabetta. Towards a model-driven infrastructure for runtime monitoring. In *International Workshop on Software Engineering for Resilient Systems*, pages 130–144. Springer, 2011.

[18] S. Bhardwaj, L. Jain, and S. Jain. Cloud computing: A study of infrastructure as a service (iaas). *International Journal of engineering and information Technology*, 2 (1):60–63, 2010.

[19] M. Blumreiter, J. Greenyer, F. J. C. Garcia, V. Klös, M. Schwammberger, C. Sommer, A. Vogelsang, and A. Wortmann. Towards self-explainable cyber-physical systems. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 543–548. IEEE, 2019.

[20] J. Bocanegra, J. Pavlich-Mariscal, and A. Carrillo-Ramos. On the role of model-driven engineering in adaptive systems. In *2016 IEEE 11th Colombian Computing Conference (CCC)*, pages 1–8. IEEE, 2016.

[21] J. Boubeta-Puig, G. Ortiz, and I. Medina-Bulo. MEdit4CEP: A model-driven solution for real-time decision making in SOA 2.0. *Knowledge-Based Systems*, 89:97–112, Nov. 2015. ISSN 0950-7051. doi: 10.1016/j.knosys.2015.06.021.

[22] M. Brambilla, J. Cabot, and M. Wimmer. Model-driven software engineering in practice. *Synthesis Lectures on Software Engineering*, 1(1):1–182, 2012.

[23] T. Brand and H. Giese. Towards generic adaptive monitoring. In *2018 IEEE 12th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pages 156–161. IEEE, 2018.

[24] Y. Brun, G. D. M. Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Müller, M. Pezzè, and M. Shaw. Engineering self-adaptive systems through feedback loops. In *Software engineering for self-adaptive systems*, pages 48–70. Springer, 2009.

[25] A. Bucchiarone, J. Cabot, R. F. Paige, and A. Pierantonio. Grand challenges in model-driven engineering: an analysis of the state of the research. *Software and Systems Modeling*, 19(1):5–13, 2020.

[26] S. Burmester, H. Giese, and M. Tichy. Model-driven development of reconfigurable mechatronic systems with mechatronic uml. In *Model Driven Architecture*, pages 47–61. Springer, 2004.

[27] A. Buss. Basic event graph modeling. *Simulation News Europe*, 31(1), 2001.

[28] J. Cabot and M. Gogolla. Object constraint language (ocl): a definitive guide. *Formal Methods for Model-Driven Engineering: 12th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2012, Bertinoro, Italy, June 18-23, 2012. Advanced Lectures*, pages 58–90, 2012.

[29] R. Calinescu, R. Mirandola, D. Perez-Palacin, and D. Weyns. Understanding uncertainty in self-adaptive systems. In *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*, pages 242–251. IEEE, 2020.

[30] J. Cámara, R. De Lemos, C. Ghezzi, and A. Lopes. *Assurances for self-adaptive systems: principles, models, and techniques*, volume 7740. Springer, 2013.

[31] J. Cámara, K. L. Bellman, J. O. Kephart, M. Autili, N. Bencomo, A. Diaconescu, H. Giese, S. Götz, P. Inverardi, S. Kounev, and M. Tivoli. *Self-aware Computing Systems: Related Concepts and Research Areas*. Springer International Publishing, Cham, 2017. ISBN 978-3-319-47474-8. doi: 10.1007/978-3-319-47474-8_2.

[32] P. Carey. *Data protection: a practical guide to UK and EU law*. Oxford University Press, Inc., 2018.

[33] I. Cassar, A. Francalanza, L. Aceto, and A. Ingólfsdóttir. A survey of runtime monitoring instrumentation techniques. *arXiv preprint arXiv:1708.07229*, 2017.

[34] P. Cedillo, J. Gonzalez-Huerta, S. M. Abrahão, and E. Insfran. Towards monitoring cloud services using models@ run. time. In *MoDELS@ Run. time*, pages 31–40, 2014.

[35] B. H. C. Cheng, R. de Lemos, H. Giese, B. H. C. others", editor="Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee. *Software Engineering for Self-Adaptive Systems: A Research Roadmap*, pages 1–26. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. ISBN 978-3-642-02161-9. doi: 10.1007/978-3-642-02161-9_1.

[36] F. Ciccozzi, I. Crnkovic, D. Di Ruscio, I. Malavolta, P. Pelliccione, and R. Spalazzese. Model-driven engineering for mission-critical iot systems. *IEEE software*, 34(1):46–53, 2017.

[37] G. Cicotti, L. Coppolino, R. Cristaldi, et al. QoS Monitoring in a Cloud Services Environment: The SRT-15 Approach. In *Euro-Par 2011: Parallel Processing Workshops*, LNCS, pages 15–24, Berlin, Heidelberg, 2011. Springer. ISBN 978-3-642-29737-3.

[38] D. Cofer, A. Gacek, S. Miller, M. W. Whalen, B. LaValley, and L. Sha. Compositional verification of architectural models. In *NASA Formal Methods Symposium*, pages 126–140. Springer, 2012.

[39] D. Corral-Plaza, I. Medina-Bulo, G. Ortiz, and J. Boubeta-Puig. A stream processing architecture for heterogeneous data sources in the Internet of Things. *Computer Standards & Interfaces*, 70:103426, June 2020. ISSN 0920-5489. doi: 10.1016/j.csi.2020.103426.

[40] M. T. Cox. Metareasoning, monitoring, and self-explanation. *Metareasoning: Thinking about thinking*, pages 131–149, 2011.

[41] K. Cyras, R. Badrinath, S. K. Mohalik, A. Mujumdar, A. Nikou, A. Previti, V. Sundararajan, and A. V. Feljan. Machine reasoning explainability. *arXiv preprint arXiv:2009.00418*, 2020.

[42] M. Danilevsky, K. Qian, R. Aharonov, Y. Katsis, B. Kawas, and P. Sen. A survey of the state of explainable ai for natural language processing. *AACL-IJCNLP 2020*, 2020.

[43] R. De Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. M. Villegas, T. Vogel, et al. Software engineering for self-adaptive systems: A second research roadmap. In *Software Engineering for Self-Adaptive Systems II*, pages 1–32. Springer, 2013.

[44] N. Delgado, A. Q. Gates, and S. Roach. A taxonomy and catalog of runtime software-fault monitoring tools. *IEEE Transactions on software Engineering*, 30(12):859–872, 2004.

[45] A. J. Demers, J. Gehrke, B. Panda, M. Riedewald, V. Sharma, W. M. White, et al. Cayuga: A general purpose event monitoring system. In *Cidr*, volume 7, 2007.

[46] A. B. Diallo, H. Nakagawa, and T. Tsuchiya. Adaptation space reduction using an explainable framework. In *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*, pages 1653–1660. IEEE, 2021.

[47] W. Dou, D. Bianculli, and L. Briand. A Model-Driven Approach to Trace Checking of Pattern-Based Temporal Properties. In *2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 323–333, Sept. 2017. doi: 10.1109/MODELS.2017.9.

[48] R. Drechsler, C. Lüth, G. Fey, and T. Güneysu. Towards self-explaining digital systems: A design methodology for the next generation. In *2018 IEEE 3rd International Verification and Security Workshop (IVSW)*, pages 1–6. IEEE, 2018.

[49] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in property specifications for finite-state verification. In *Proceedings of ICSE'99*, pages 411–420, May 1999. doi: 10.1145/302405.302672.

[50] P. Esling and C. Agon. Time-series data mining. *ACM Computing Surveys*, 45(1):12, 2012. doi: 10.1145/2379776.2379788.

[51] T. Faison. *Event-Based Programming*. Springer, 2006.

[52] G. Feltrin, N. Popovic, and M. Wojtera. A Sentinel Node for Event-Driven Structural Monitoring of Road Bridges Using Wireless Sensor Networks, Jan. 2019.

[53] M. Feurer and F. Hutter. Hyperparameter optimization. In *Automated machine learning*, pages 3–33. Springer, Cham, 2019.

[54] J. Floch, S. Hallsteinsen, E. Stav, F. Eliassen, K. Lund, and E. Gjorven. Using architecture models for runtime adaptability. *IEEE software*, 23(2):62–70, 2006.

[55] F. Fouquet, B. Morin, F. Fleurey, O. Barais, N. Plouzeau, and J.-M. Jezequel. A dynamic component model for cyber physical systems. In *Proceedings of the 15th ACM SIGSOFT symposium on Component Based Software Engineering*, pages 135–144, 2012.

[56] R. France and B. Rumpe. Model-driven development of complex software: A research roadmap. In *Future of Software Engineering (FOSE'07)*, pages 37–54. IEEE, 2007.

[57] A. García-Domínguez, N. Bencomo, and L. H. Garcia Paucar. Reflecting on the past and the present with temporal graph-based models. In *CEUR Workshop Proceedings*, volume 2245, pages 46–55, 2018.

[58] A. Garcia-Dominguez, K. Barmpis, D. S. Kolovos, R. Wei, and R. F. Paige. Stress-testing remote model querying APIs for relational and graph-based stores. *Software & Systems Modeling*, 18(2):1047–1075, June 2019. ISSN 1619-1366, 1619-1374. doi: 10.1007/s10270-017-0606-9.

[59] A. García-Domínguez, N. Bencomo, J. M. Parra-Ullauri, and L. H. García-Paucar. Towards history-aware self-adaptation with explanation capabilities. In *2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS* W)*, pages 18–23. IEEE, 2019.

[60] A. García-Domínguez, N. Bencomo, J. M. Parra-Ullauri, and L. H. García-Paucar. Querying and annotating model histories with time-aware patterns. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 194–204. IEEE, 2019.

[61] L. Garcia-Paucar and N. Bencomo. Knowledge base k models to support trade-offs for self-adaptation using markov processes. *13th IEEE Conference SASO, Sweden*, 2019.

[62] S. R. Garzon and M. Cebulla. Model-based personalization within an adaptable human-machine interface environment that is capable of learning from user interactions. In *2010 Third International Conference on Advances in Computer-Human Interactions*, pages 191–198. IEEE, 2010.

[63] O. Gheibi, D. Weyns, and F. Quin. Applying machine learning in self-adaptive systems: A systematic literature review. *arXiv preprint arXiv:2103.04112*, 2021.

[64] A. Gómez, J. Cabot, and M. Wimmer. Temporalemf: A temporal metamodeling framework. In *International Conference on Conceptual Modeling*, pages 365–381. Springer, 2018.

[65] J. Greenyer, M. Lochau, T. Vogel, et al. Towards explainable controller synthesis and reinforcement learning for cyber-physical systems. *Explainable Software for Cyber-Physical Systems (ES4CPS)*, page 37, 2019.

[66] M. Haeusler, T. Trojer, J. Kessler, et al. ChronoSphere: a graph-based EMF model repository for IT landscape models. *Software and Systems Modeling*, 2019.

[67] T. Hartmann, F. Fouquet, G. Nain, B. Morin, J. Klein, and Y. Le Traon. Reasoning at runtime using time-distorted contexts: A models@ run. time based approach. In *Proceedings of the 26th International Conference on Software Engineering and Knowledge Engineering*, pages 586–591. Knowledge Systems Institute Graduate School, USA, 2014.

[68] T. Hartmann, F. Fouquet, et al. Analyzing Complex Data in Motion at Scale with Temporal Graphs. In *Proceedings of SEKE'17*, 2017.

[69] C. He and G. Mussbacher. Model-driven engineering and elicitation techniques: a systematic literature review. In *2016 IEEE 24th International Requirements Engineering Conference Workshops (REW)*, pages 180–189. IEEE, 2016.

[70] A. Heuillet, F. Couthouis, and N. Díaz-Rodríguez. Explainability in deep reinforcement learning. *Knowledge-Based Systems*, 214:106685, 2021.

[71] G. Heward, I. Müller, J. Han, J.-G. Schneider, and S. Versteeg. Assessing the performance impact of service monitoring. In *2010 21st Australian Software Engineering Conference*, pages 192–201. IEEE, 2010.

[72] F. Hilken and M. Gogolla. Verifying Linear Temporal Logic Properties in UML/OCL Class Diagrams Using Filmstripping. In *2016 Euromicro Conference on Digital System Design (DSD)*, pages 708–713, Aug. 2016. doi: 10.1109/DSD.2016.42.

[73] H. N. Ho and E. Lee. Model-based reinforcement learning approach for planning in self-adaptive software system. In *Proceedings of the 9th International Conference on Ubiquitous Information Management and Communication*, pages 1–8, 2015.

[74] C. Inzinger, W. Hummer, B. Satzger, P. Leitner, and S. Dustdar. Generic event-based monitoring and adaptation methodology for heterogeneous distributed systems. *Software: Practice and Experience*, (7), 2014. ISSN 00380644.

[75] M. Ji, A. C. Veitch, J. Wilkes, et al. Seneca: remote mirroring done write. In *USENIX Annual Conference*, pages 253–268, 2003.

[76] H. S. Jomaa, J. Grabocka, and L. Schmidt-Thieme. Hyp-rl: Hyperparameter optimization by reinforcement learning. *arXiv preprint arXiv:1906.11527*, 2019.

[77] Z. Juozapaitis, A. Koul, A. Fern, M. Erwig, and F. Doshi-Velez. Explainable reinforcement learning via reward decomposition. In *IJCAI/ECAI Workshop on Explainable Artificial Intelligence*, 2019.

[78] B. Kanso and S. Taha. Specification of temporal properties with OCL. *Science of Computer Programming*, 96:527–551, Dec. 2014. ISSN 0167-6423. doi: 10.1016/j.scico.2014.02.029.

[79] S. Kent. Model driven engineering. In *International conference on integrated formal methods*, pages 286–298. Springer, 2002.

[80] N. Khalid and N. A. Qureshi. Towards self-explainable adaptive systems (seas): A requirements driven approach. In *REFSQ Workshops*, 2021.

[81] R. Klar, A. Quick, and F. Soetz. Tools for a model-driven instrumentation for monitoring. In *Proceedings of the 5th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 165–180. Citeseer, 1992.

[82] F. Koetter and M. Kochanowski. A model-driven approach for event-based business process monitoring. *Information Systems and e-Business Management*, 13(1):5–36, 2015.

[83] D. Kolovos. What is model-driven engineering? `https://codebots.com/app-development/what-is-model-driven-engineering`, 2021.

[84] D. Kolovos and A. Garcia-Dominguez. The epsilon playground. In *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, pages 131–137, 2022.

[85] D. S. Kolovos, R. F. Paige, and F. Polack. The Epsilon Object Language (EOL). In *Model Driven Architecture - Foundations and Applications, Second European Confer-*

ence, ECMDA-FA 2006, Bilbao, Spain, July 10-13, 2006, Proceedings, pages 128–142, 2006. doi: 10.1007/11787044_11.

[86] S. Konno and X. Défago. Approximate QoS Rule Derivation Based on Root Cause Analysis for Cloud Computing. In *2019 IEEE 24th Pacific Rim International Symposium on Dependable Computing (PRDC)*, pages 33–3309, Dec. 2019. doi: 10.1109/PRDC47002.2019.00020. ISSN: 2473-3105.

[87] B. Kordts, B. Gerlach, and A. Schrader. Self-organizing and self-explaining pervasive environments by connecting smart objects and applications. *Technologies*, 10(1):15, 2022.

[88] C. Krupitzer, F. M. Roth, S. VanSyckel, G. Schiele, and C. Becker. A survey on engineering approaches for self-adaptive systems. *Pervasive and Mobile Computing*, 17:184–206, 2015.

[89] J. Küster. Model-driven software engineering foundations of model-driven software engineering. *IBM Research*, 2011.

[90] D. K. Lewis. Causal explanation. 1986.

[91] P. R. Lewis, A. Chandra, F. Faniyi, K. Glette, T. Chen, R. Bahsoon, J. Torresen, and X. Yao. Architectural aspects of self-aware and self-expressive computing systems: From psychology to engineering. *Computer*, 48(8):62–70, 2015.

[92] N. Li, J. Cámara, D. Garlan, and B. Schmerl. Reasoning about when to provide explanation for human-involved self-adaptive systems. In *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*. IEEE, 2020.

[93] N. Li, J. Cámara, D. Garlan, B. Schmerl, and Z. Jin. Hey! preparing humans to do tasks in self-adaptive systems. In *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 48–58. IEEE, 2021.

[94] S. W. Liddle. Model-driven software development. In *Handbook of Conceptual Modeling*, pages 17–54. Springer, 2011.

[95] R. Light. Mosquitto: server and client implementation of the MQTT protocol. *Journal of Open Source Software*, 2017.

[96] B. Y. Lim, A. K. Dey, and D. Avrahami. Why and why not explanations improve the intelligibility of context-aware intelligent systems. In *Proceedings of CHI 2009*. ACM, 2009.

[97] S. Liu, X. Wang, M. Liu, and J. Zhu. Towards better analysis of machine learning models: A visual analytics perspective. *Visual Informatics*, 1(1):48–56, 2017.

[98] D. C. Luckham and B. Frasca. Complex event processing in distributed systems. *Computer Systems Laboratory Technical Report CSL-TR-98-754. Stanford University, Stanford*, 28:16, 1998.

[99] A. Lukkarinen, L. Malmi, and L. Haaranen. Event-driven programming in programming education: A mapping review. *ACM Transactions on Computing Education (TOCE)*, 21(1):1–31, 2021.

[100] F. D. Macías-Escrivá, R. Haber, R. Del Toro, and V. Hernandez. Self-adaptive systems: A survey of current approaches, research challenges and applications. *Expert Systems with Applications*, 40(18):7267–7279, 2013.

[101] P. Madumal, T. Miller, L. Sonenberg, and F. Vetere. Explainable reinforcement learning through a causal lens. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 2493–2500, 2020.

[102] M. Mahmud, M. S. Kaiser, A. Hussain, and S. Vassanelli. Applications of deep learning and reinforcement learning to biological data. *IEEE Transactions on Neural Networks and Learning Systems*, 29(6):2063–2079, 2018. doi: 10.1109/TNNLS.2018.2790388.

[103] A. Mazak, S. Wolny, A. Gómez, J. Cabot, M. Wimmer, and G. Kappel. Temporal models on time series databases. *J. Object Technol*, 19(3):1, 2020.

[104] B. Meyers, H. Vangheluwe, J. Denil, and R. Salay. A Framework for Temporal Verification Support in Domain-Specific Modelling. *IEEE Transactions on Software Engineering*, 2018. ISSN 0098-5589, 1939-3520, 2326-3881. doi: 10.1109/TSE.2018.2859946.

[105] T. Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artificial intelligence*, 267:1–38, 2019.

[106] R. Mitchell, J. Michalski, and T. Carbonell. *An artificial intelligence approach*. Springer, 2013.

[107] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[108] G. A. Moreno, J. Cámara, D. Garlan, and B. Schmerl. Proactive self-adaptation under uncertainty: A probabilistic model checking approach. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, pages 1–12, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3675-8. doi: 10.1145/2786805.2786853. URL `http://doi.acm.org/10.1145/2786805.2786853`.

[109] O. Moser, F. Rosenberg, and S. Dustdar. Event driven monitoring for service composition infrastructures. In *International Conference on Web Information Systems Engineering*, pages 38–51. Springer, 2010.

[110] L. Mouline, A. Benelallam, F. Fouquet, et al. A temporal model for interactive diagnosis of adaptive systems. In *ICAC 2018*, 2018. doi: 10.1109/ICAC.2018.00029.

[111] R. Murch. *Autonomic computing*. IBM Press, 2004.

[112] S. Nordstrom, A. Dubey, T. Keskinpala, R. Datta, S. Neema, and T. Bapty. Model predictive analysis for autonomicworkflow management in large-scale scientific computing environments. In *Fourth IEEE International Workshop on Engineering of Autonomic and Autonomous Systems (EASe'07)*, pages 37–42. IEEE, 2007.

[113] Object Management Group. UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) 1.2 Beta1, Dec. 2018. URL `https://www.omg.org/spec/MARTE/1.2/Beta1/`. Archived at http://archive.is/UaK85.

[114] R. F. Paige, N. Drivalos, D. S. Kolovos, K. J. Fernandes, C. Power, G. K. Olsen, and S. Zschaler. Rigorous identification and encoding of trace-links in model-driven engineering. *Software & Systems Modeling*, 10(4):469–487, Oct 2011. ISSN 1619-1374. doi: 10.1007/s10270-010-0158-8. URL `https://doi.org/10.1007/s10270-010-0158-8`.

[115] A. Palm, A. Metzger, and K. Pohl. Online reinforcement learning for self-adaptive information systems. In *International Conference on Advanced Information Systems Engineering*, pages 169–184. Springer, 2020.

[116] T. Panch, P. Szolovits, and R. Atun. Artificial intelligence, machine learning and health systems. *Journal of global health*, 8(2), 2018.

[117] D. Papamartzivanos, F. G. Mármol, and G. Kambourakis. Introducing deep learning self-adaptive misuse network intrusion detection systems. *IEEE Access*, 7:13546–13560, 2019.

[118] J. M. Parra-Ullauri. ETeMoX event-driven temporal models for explanations, 2021. URL `https://gitlab.com/sea-aston/etemox`.

[119] J. M. Parra-Ullauri, A. García-Domínguez, L. H. García-Paucar, and N. Bencomo. Temporal models for history-aware explainability. In *Proceedings of the 12th System Analysis and Modelling Conference*, pages 155–164, 2020.

[120] J. M. Parra-Ullauri, A. Garcia-Dominguez, N. Bencomo, S. Yang, C. Zheng, C. Zhen, J. Boubeta-Puig, and G. Ortiz. Event-driven temporal models for explanations - etemox: explaining reinforcement learning. *Software and Systems Modeling*, 2021. doi: 10.1007/s10270-021-00952-4.

[121] J. M. Parra-Ullauri, A. García-Domínguez, J. Boubeta-Puig, N. Bencomo, and G. Ortiz. Towards an architecture integrating complex event processing and temporal graphs for service monitoring. In *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, pages 427–435, 2021.

[122] J. M. Parra-Ullauri, A. García-Domínguez, and N. Bencomo. From a series of (un) fortunate events to global explainability of runtime model-based self-adaptive systems. In *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 807–816, 2021. doi: 10.1109/MODELS-C53483.2021.00127.

[123] J. M. Parra-Ullauri, A. García-Domínguez, and L. G.-P. Bencomo, Nelly. History-aware explanations: Towards enabling human-in-the-loop in self-adaptive systems.

In *Proceedings of the 14th System Analysis and Modelling Conference*, 2022. To be published.

[124] L. H. G. Paucar, N. Bencomo, and K. K. Fung Yuen. Juggling Preferences in a World of Uncertainty. *RE NEXT, Lisbon.*, 2017.

[125] G. R. Poole David, Mackworth Alan. Computational intelligence: a logical approach.(1998). *Google Scholar Google Scholar Digital Library Digital Library*, 1998.

[126] E. Puiutta and E. M. Veith. Explainable reinforcement learning: A survey. In *International Cross-Domain Conference for Machine Learning and Knowledge Extraction*. Springer, 2020.

[127] F. Quin, D. Weyns, T. Bamelis, S. S. Buttar, and S. Michiels. Efficient analysis of large adaptation spaces in self-adaptive systems using machine learning. In *2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 1–12. IEEE, 2019.

[128] R. Rabiser, S. Guinea, M. Vierhauser, L. Baresi, and P. Grünbacher. A comparison framework for runtime monitoring approaches. *Journal of Systems and Software*, 125: 309–321, 2017.

[129] A. Ramirez, B. Cheng, N. Bencomo, and P. Sawyer. Relaxing claims: Coping with uncertainty while evaluating assumptions at run time. *MODELS*, 2012.

[130] G. Ras, M. van Gerven, and P. Haselager. Explanation methods in deep learning: Users, values, concerns and challenges. In *Explainable and Interpretable Models in Computer Vision and Machine Learning*, pages 19–36. Springer, 2018.

[131] O. Reynolds, A. García-Domínguez, and N. Bencomo. Automated provenance graphs for models@ run. time. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, 2020.

[132] W. N. Robinson. A requirements monitoring framework for enterprise systems. *Requirements engineering*, 11(1):17–41, 2006.

[133] T. Roth-Berghofer, S. Schulz, D. B. Leake, and D. Bahls. Explanation-aware computing. *AI Magazine*, 2007.

[134] S. Rougemaille, F. Migeon, C. Maurel, and M.-P. Gleizes. Model driven engineering for designing adaptive multi-agents systems. In *International Workshop on Engineering Societies in the Agents World*, pages 318–332. Springer, 2007.

[135] N. B. Ruparelia. Software development lifecycle models. *ACM SIGSOFT Software Engineering Notes*, 35(3):8–13, 2010.

[136] M. Sadeghi, V. Klös, and A. Vogelsang. Cases for explainable software systems: Characteristics and examples. In *2021 IEEE 29th International Requirements Engineering Conference Workshops (REW)*, pages 181–187. IEEE, 2021.

[137] M. Salehie and L. Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM transactions on autonomous and adaptive systems (TAAS)*, 4(2):1–42, 2009.

[138] W. Samek, T. Wiegand, and K.-R. Müller. Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models. *arXiv preprint arXiv:1708.08296*, 2017.

[139] C. Sansores and J. Pavón. An adaptive agent model for self-organizing mas. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 3*, pages 1639–1642, 2008.

[140] T. R. D. Saputri and S.-W. Lee. The application of machine learning in self-adaptive systems: A systematic literature review. *IEEE Access*, 8:205948–205967, 2020.

[141] P. Sawyer, N. Bencomo, J. Whittle, E. Letier, and A. Finkelstein. Requirements-aware systems: A research agenda for re for self-adaptive systems. In *2010 18th IEEE International Requirements Engineering Conference(RE)*, volume 00, pages 95–103, Sept. 2010. doi: 10.1109/RE.2010.21.

[142] D. C. Schmidt. Model-driven engineering. *Computer-IEEE Computer Society-*, 39(2): 25, 2006.

[143] P. Sequeira and M. Gervasio. Interestingness elements for explainable reinforcement learning: Understanding agents' capabilities and limitations. *Artificial Intelligence*, 288:103367, 2020.

[144] V. E. Silva Souza, A. Lapouchnian, W. N. Robinson, and J. Mylopoulos. Awareness requirements for adaptive systems. In *Proceedings of the 6th international symposium on Software engineering for adaptive and self-managing systems*, pages 60–69, 2011.

[145] D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro. *EMF: eclipse modeling framework*. Pearson Education, 2008.

[146] Student. The probable error of a mean. *Biometrika*, 6(1):1–25, 1908. ISSN 00063444. doi: 10.2307/2331554.

[147] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[148] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[149] M. Szvetits and U. Zdun. Systematic literature review of the objectives, techniques, kinds, and architectures of models at runtime. *Software & Systems Modeling*, 15(1): 31–69, 2016.

[150] M. Turilli and L. Floridi. The ethics of information transparency. *Ethics and Information Technology*, 11(2):105–112, 2009.

[151] J. van der Waa, J. van Diggelen, K. v. d. Bosch, and M. Neerincx. Contrastive explanations for reinforcement learning in terms of expected consequences. *arXiv preprint arXiv:1807.08706*, 2018.

[152] A. Verma, V. Murali, R. Singh, P. Kohli, and S. Chaudhuri. Programmatically interpretable reinforcement learning. In *International Conference on Machine Learning*. PMLR, 2018.

[153] M. Vierhauser, R. Rabiser, P. Grünbacher, K. Seyerlehner, S. Wallner, and H. Zeisel. Reminds: A flexible runtime monitoring framework for systems of systems. *Journal of Systems and Software*, 112:123–136, 2016.

[154] N. M. Villegas, G. Tamura, H. A. Müller, L. Duchien, and R. Casallas. *DYNAMICO: A Reference Model for Governing Control Objectives and Context Relevance in Self-Adaptive Software Systems*, pages 265–293. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. ISBN 978-3-642-35813-5. doi: 10.1007/978-3-642-35813-5_11. URL https://doi.org/10.1007/978-3-642-35813-5_11.

[155] M. Viswanathan and M. Kim. Foundations for the run-time monitoring of reactive systems–fundamentals of the mac language. In *International Colloquium on Theoretical Aspects of Computing*, pages 543–556. Springer, 2004.

[156] T. Vogel and H. Giese. Model-driven engineering of self-adaptive software with eurema. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 8(4):1–33, 2014.

[157] G. Waignier, A.-F. Le Meur, and L. Duchien. A model-based framework to design and debug safe component-based autonomic systems. In *International Conference on the Quality of Software Architectures*, pages 1–17. Springer, 2009.

[158] K. Welsh, P. Sawyer, and N. Bencomo. Towards requirements aware systems: Runtime resolution of design-time assumptions. In *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*, pages 560–563. IEEE, 2011.

[159] D. Weyns. Software engineering of self-adaptive systems: an organised tour and future challenges. *Chapter in Handbook of Software Engineering*, 2017.

[160] D. Weyns and T. Ahmad. Claims and evidence for architecture-based self-adaptation: A systematic literature review. In *European Conference on Software Architecture*, pages 249–265. Springer, 2013.

[161] D. Weyns, S. Malek, and J. Andersson. Forms: Unifying reference model for formal specification of distributed self-adaptive systems. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 7(1):1–61, 2012.

[162] A. F. Winfield, S. Booth, L. A. Dennis, T. Egawa, H. Hastie, N. Jacobs, R. I. Muttram, J. I. Olszewska, F. Rajabiyazdi, A. Theodorou, et al. Ieee p7001: a proposed standard on transparency. *Frontiers in Robotics and AI*, page 225, 2021.

[163] E. Wu, Y. Diao, and S. Rizvi. High-performance complex event processing over streams. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, 2006.

[164] T. Zahavy, Z. Xu, V. Veeriah, M. Hessel, J. Oh, H. van Hasselt, D. Silver, and S. Singh. Self-tuning deep reinforcement learning. *arXiv preprint arXiv:2002.12928*, 2020.

[165] C. Zheng, S. Yang, J. M. Parra-Ullauri, A. Garcia-Dominguez, and N. Bencomo. Reward-reinforced generative adversarial networks for multi-agent systems. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2021.

[166] P. Ziemann and M. Gogolla. An extension of OCL with temporal logic. In *Critical Systems Development with UML*, volume 2, pages 53–62, 2002.