Temporal Models for History-Aware Explainability

Juan Marcelo Parra-Ullauri
Antonio García-Domínguez
Luis Hernán García-Paucar
Nelly Bencomo
j.parra-ullauri@aston.ac.uk
a.garcia-dominguez@aston.ac.uk
garciapl@aston.ac.uk
n.bencomo@aston.ac.uk
SEA research group, EPS, Aston University
Birmingham, United Kingdom

ABSTRACT

On one hand, there has been a growing interest towards the application of AI-based learning and evolutionary programming for self-adaptation under uncertainty. On the other hand, self-explanation is one of the self-* properties that has been neglected. This is paradoxical as self-explanation is inevitably needed when using such techniques. In this paper, we argue that a self-adaptive autonomous system (SAS) needs an infrastructure and capabilities to be able to look at its own history to explain and reason why the system has reached its current state. The infrastructure and capabilities need to be built based on the right conceptual models in such a way that the system's history can be stored, queried to be used in the context of the decision-making algorithms.

The explanation capabilities are framed in four incremental levels, from forensic self-explanation to automated history-aware (HA) systems. Incremental capabilities imply that capabilities at *Level n* should be available for capabilities at *Level n* + 1. We demonstrate our current reassuring results related to *Level 1* and *Level 2*, using temporal graph-based models. Specifically, we explain how *Level 1* supports forensic accounting after the system's execution. We also present how to enable on-line historical analyses while the self-adaptive system is running, underpinned by the capabilities provided by *Level 2*. An architecture which allows recording of temporal data that can be queried to explain behaviour has been presented, and the overheads that would be imposed by live analysis are discussed. Future research opportunities are envisioned.

CCS CONCEPTS

• **Software and its engineering → System modeling languages**; **Integration frameworks**; **Model-driven software engineering**.

SAM '20, October 19–20, 2020, Virtual Event, Canada

ACM ISBN 978-1-4503-8140-6/20/10…$15.00
https://doi.org/10.1145/3419804.3420276

KEYWORDS

self-explanation, temporal graphs, runtime models, graph databases

ACM Reference Format:
Juan Marcelo Parra-Ullauri, Antonio García-Domínguez, Luis Hernán García-Paucar, and Nelly Bencomo. 2020. Temporal Models for History-Aware Explainability. In *12th System Analysis and Modelling Conference (SAM '20), October 19–20, 2020, Virtual Event, Canada.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3419804.3420276

1 INTRODUCTION

Systems are increasingly expected to learn and adapt themselves to changing environmental conditions, and cope with uncertainty and external threats [31, 33]. Early solutions to self-adaptation adapt according to monitored changes based on knowledge that was known at design time, providing limited reasoning and reflection capabilities [9]. Modern solutions learn new information during execution and provide estimations about the future to support better-informed decision-making [6, 26, 30].

These more advanced self-adaptive systems (SAS) can expose behaviour that end-users may not understand [9]. Further, these users may cease to use the system due to lack of trust[34]. Therefore, providing understandable explanations for surprising behaviour is relevant for SAS. Lately, there have been increasing interest about the right to explanation [20, 36].

The explanations of the decision-making of a SAS that we support in this paper are based on the history of the execution of the system, i.e. the system explains its behaviour based on what the running system has observed in the past. In later iterations of the SAS, the same history-based explanations can become an additional source of information for its decision processes, in addition to its use of current observations and future projections. However, the required integration of history-awareness capabilities into the decision-making process can be complex, and therefore we propose ideas to do it in a gradual fashion.

Based on the above, this paper has three main contributions:

(i) a 4-level spectrum of reflective capabilities for a self-aware self-adaptive system (from forensic self-explanation to autonomous history-aware decision-making), which acts as our research roadmap;

(ii) a description of the forensic analysis layer (*Level 1*); and

(iii) a scenario for the level beyond forensic analysis (live history-aware explanation), with an evaluation of our current implementation of that level.

We use the term self-explanation to describe the capability of the system to answer questions based on its past behaviour. The answers explain the reasons why a decision was made to reach a particular system state. They can also prove or disprove hypotheses on the system behaviour. Explanations should be readable by and available to humans but also machines. Specifically, explanations should be available to different stakeholders such as end-users, developers, external systems, or the SAS itself. As part of the contributions described above, we present in this paper the data model and the progress made so far on building a querying infrastructure for this purpose. Initial results of these ideas were discussed in the workshop paper in [14].

The structured of the paper is: Section 2 presents the foundations that underlie our research, in terms of reflective self-adaptation and storage and retrieval of historic data. Section 3 presents the spectrum composed by the four envisioned levels of reflective capabilities that a SAS may offer. Section 4 presents the Remote Data Mirroring SAS which we use as our case study. Section 5 presents the latest updates on *Level 1* since our initial work. Section 6 introduces a scenario around *Level 2* of the spectrum for RDM, and Section 7 evaluates our current implementation of *Level 2*. Finally, Section 8 concludes the paper and presents several research avenues.

2 RESEARCH BASELINE

This section introduces the research that underpins the work, in the areas of self-adaptive systems and management of historical data.

2.1 Reflective, Self-aware Self-adaptation and Self-explanation

Self-awareness can be seen as the capability of a system to acquire and access knowledge about its own state and its environmental context [6, 8]. Such knowledge allows for better understanding and reasoning about its adaptive behaviour. Self-awareness is seen as a low level of abstraction of self-adaptivity [33], allowing for improvement of the self-adaptivity of a system. Self-awareness of a computing system can be related to different specific capabilities such as goal-awareness [38], requirements-awareness [34] or time-awareness [3]. Time-awareness is the use of knowledge of historical and perhaps future phenomena [3]. Time-awareness requires node-level memory, and capabilities for time series modeling and/or anticipation. History-awareness is implied in time-awareness.

Existing work tends to leave history-awareness implicit in the formal model [8]. We argue that explicit representation of history-awareness (i.e. time-awareness) will help to reason about the impact that past history has on the decision process.

Leaving history-awareness as something implicit to the model also means implementing the storage and retrieval of this past history as an *ad hoc* effort, which changed from SAS to SAS. In some cases, past history had been "compressed" to the point that it was not recoverable: the user could not see what the system had based its decisions upon.

The knowledge base of the MAPE loop [21] can be used to maintain historical data and knowledge used by the system for informed adaptation. As a structured explicit knowledge is needed, the authors of [28] propose an extension of the context representation for the MAPE-K loop integrating the history of planned actions as well as the expected effects over time. Their analysis and planning phases can therefore compare measured and expected context metrics. The work is demonstrated on a cloud elasticity manager case. Authors of [7] propose stochastic game analysis and latency awareness, a kind of time-awareness, for proactive self-adaptation. In [27] the authors tackle the problem of tracking historical changes as well. To do that, they use causal relationships between requirements and their corresponding adaptations. In our own case, we propose the explicit use of temporal graph databases as a representation for trace-based models to enable self-explanation in interactive diagnosis or forensic analysis based on a generic meta-model that supports the structure for execution traces of SAS.

In regard to accessing the history to support reasoning and explanation, Welsh et al. [41] argue that an SAS needs to garner confidence not only in its users by explaining its behaviour during execution, but also in its developers by explaining "surprises" during testing and maintenance. The authors specifically use requirements-awareness and monitoring to enable the explanation capability in adaptive systems. By extending the goal models with a claim-refinement model, the aspects of the systems that will be monitored are defined. According to the current state of these aspects, they can explain why the system has adapted its behavior. Authors in [27] tackle a very related issue, i.e interactive diagnosis.

There is a class of self-adaptive systems which explicitly use models at runtime as abstractions of their state to implement reflective capabilities [5]. Rather than transforming structured logs into a temporal graph, Reynolds et al. propose automatically collecting a provenance graph of the evolution of these reflective models [32]. For every change, the agent involved, the activity the agent was performing, and the entity affected are recorded. In relation with our 4-level hierarchy in Section 3, the approach would be at *Level 1* (forensic history-aware explanation), since its case study focused on investigating issues after the system ran. The approach allows for direct integration with systems already using reflective models at runtime, but integrating it with other systems will require more work than the approach presented in this paper, which reuses existing logs. On the other hand, the approach already applies time windows to the collected information, simplifying the pruning of history no longer of interest. It also provides for capturing other types of activities in the system that impact its reflective models, beyond the decision making which is the focus of our trace metamodel (Section 5).

2.2 Storage and retrieval of historic data

History-awareness requires an efficient manner to represent and query the 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. However, this has been changing as systems have become more complex. 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. For

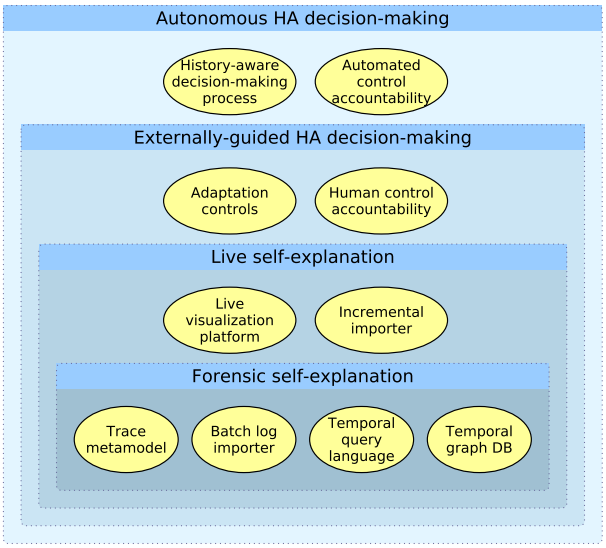

Figure 1: Required components by levels of reflective capabilities, from forensic self-explanation to history-aware (HA) systems

instance, the Google Cloud platform is known to track memory usage and recommend VM changes[1]. Another example is the Elastic indexing platform, which has recently gained machine learning capabilities for outlier detection capabilities in historical data[2].

Analyzing sequences of values over time is not new: for over 20 years, there has been considerable work on time-series data mining [10], which attempts to extract knowledge by looking at the shape of the data. The 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 is more complicated than a sequence of numbers: in its most general form, the configuration of a system is a complex entity that changes over time. Tracking this history requires a fitting data structure: thankfully, there has been a recent push towards adding temporal capabilities to graph databases, with the ability to efficiently store and navigate the history of an entire labelled attributed graph. Two examples include Greycat [18] and ChronoGraph [17].

Beyond storage, tracking the history of a SAS requires a data model, a query language, and enabling the SAS to feed the temporal graph database. In our prior work [16], we demonstrated a first version of a solution that integrated a SAS with our Hawk indexing framework for transparent forensic (after-the-fact) self-explanation. The solution took the raw JSON logs of the decision process over time and shaped them into a sequence of dedicated trace data models, which were turned into a Greycat temporal graph and exposed through a dedicated time-aware query language. This initial prototype is the base of the further proposals made in this paper.

3 LEVELS OF HISTORY-AWARE EXPLANATION CAPABILITIES IN SAS

Ideally, systems should be able to access their adaptation history and adjust future adaptations taking into account past results of previous adaptations. However, building this capability into a system can be costly and hard to evaluate.

Rather than an all-or-nothing situation, we argue that it is easier, safer and more rewarding in the short-term, to do it in stages or *levels*. A first stage can build upon the capabilities for the next one. The base level can focus on the basic capabilities for forensic analysis, and the next stages can adapt the MAPE-K functions to the use of history. A second stage focuses on Monitor and Knowledge as the SAS is running, a third stage provides Execute, and the final stage updates the Analysis and Plan functions.

Figure 1 shows our four envisioned levels of reflective capabilities that a SAS should offer, as well as their required core components. Each level requires the capabilities of the levels underneath. Sections 5 to 7 present the current state of the first two levels. At the end of the paper, there is further discussion about the last two levels. The levels are:

- ***Level 1* (forensic history-aware explanation)**: this level operates very much like a "black box". 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 converted into a *temporal graph database* conforming to a reusable *trace metamodel*. Users can then study its history with a *temporal query language*. This level is useful for either post-mortem analysis after an unexpected behaviour, or for internal evaluation during development.
- ***Level 2* (live history-aware explanation)**: this level allows users to evaluate past observations, decisions and performance on a running system without having to stop it. As such, it requires an *incremental importer* that loads periodically the latest state of the decision algorithms into the temporal graph database, adding one more timepoint to its history. To keep storage and memory costs manageable, the history of the temporal graph may be *bounded* to a specific *time window*. The temporal graph 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 on whether there is a restricted and finite number of possible system states, or not.
  A *live visualization/query 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.
- ***Level 3* (externally-guided and history-aware decision-making with explanation capabilities)**: if we consider the self-adaptive system as an autonomic element in MAPE-K, *Level 2* has provided the Monitor function, using the evolving models as a sensor from which to build the temporal graph (the (K)nowledge base). However, the Analyze, Plan,

[1] http://archive.is/mQ2k7
[2] http://archive.is/oDDh9

and Execute functions are still pending. The goal of *Level 3* is to implement the Execute function by providing effectors (e.g. input parameters or some type of configuration facility) that allow external entities to perform the Analyze and Plan functions. These external entities can be either a human or another software system. 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.

- ***Level 4* (autonomous history-aware decision-making with explanation/reasoning 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 [33, 39]. For example, one way to use the history would be to be able 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.
  Another way to use the history would 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. Those could be factored in the perceived utility levels of the available options. The system could double check the long-term performance of those decisions, and establish a confidence level on its own prediction model.
  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).

4 CASE STUDY: THE REMOTE DATA MIRRORING SAS

In order to evaluate the current results of the proposed gradual approach to history-aware self-adaptation, an existing SAS case study was selected, the Remote Data Mirroring (RDM) system from our prior work [4, 15, 30]. The RDM SAS is composed of data servers and network links. It must replicate and distribute data in an efficient manner by minimizing consumed bandwidth and providing assurance that distributed data is not lost or corrupted [19]. RDM uses an R-POMDP (Requirements runtime model based on Partially Observable Markov Decision Processes [25]). Its overall structure is shown in Figure 2.

R-POMDP runs over timeslices, just like regular POMDP. At each timeslice, the SAS monitors the Ranges of Energy Consumption (REC) and Number of Concurrent Connections (NCC), partially

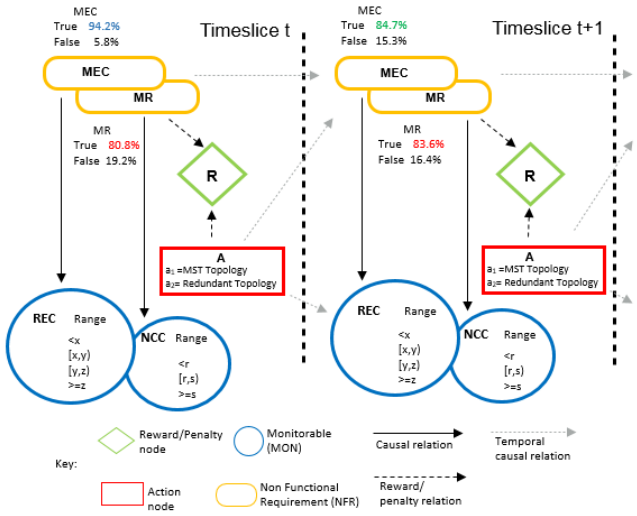

Figure 2: RDM case study, SAS that protects aginst data loss by storing copies on servers. Configuration: 2 topologies(MST, RT), 2 NFRs (MR, MEC), 2 monitoring variables (REC, NCC).

observable measures to estimate whether the corresponding Minimize Energy Consumption (MEC) and Maximize Reliability (MR) non-functional requirements (NFRs) are being met. Based on these estimations of the current system and a reward table (e.g. "a reward Y is given for action X if NFRs are estimated to be satisfied or not"), the SAS decides whether to choose a Minimum Spanning Tree (MST) network topology, or a Redundant Tree (RT) topology. MST is more efficient in terms of energy consumption, whereas RT is more reliable.

In contrast to reactive control methods [24], R-POMDP considers future evolutions (i.e. projections into the future) of the satisficement of the NFRs to decide the next action $a \in A$, i.e., to reason about long-term effects of immediate actions [37]. These future evolutions are represented by a belief over possible states or belief tree. The root node of the tree is the belief b_0 which represents the current state of the running system, i.e. the current level of satisficement of the NFRs. From there, the R-POMDP uses lookahead search [23] to approximate the optimal discounted reward value $V^*(b_0)$. The result is an approximately optimal policy for the current belief b_0 [37]. Accordingly, the system then executes the first action of the policy, $\pi(b_0)$.

Other proactive approaches like CobRA [1] and PLA [26] also predict future system states. Unlike R-POMDP, they assume full observability of the system's state (e.g. by using Markov Decision Processes [26]), so they cannot model the uncertainty that may arise from imprecise sensors in a real system. They do not explicitly model the levels of satisficement of the non-functional requirements. Proactive approaches can be deceiving at first while improving the behaviour in the long term. This kind of situations may require explanations.

5 FORENSIC SELF-EXPLANATION

According to Section 3, the first step to achieve automated history-aware self-adaptation is to offer "black box" capabilities (*Level 1*).

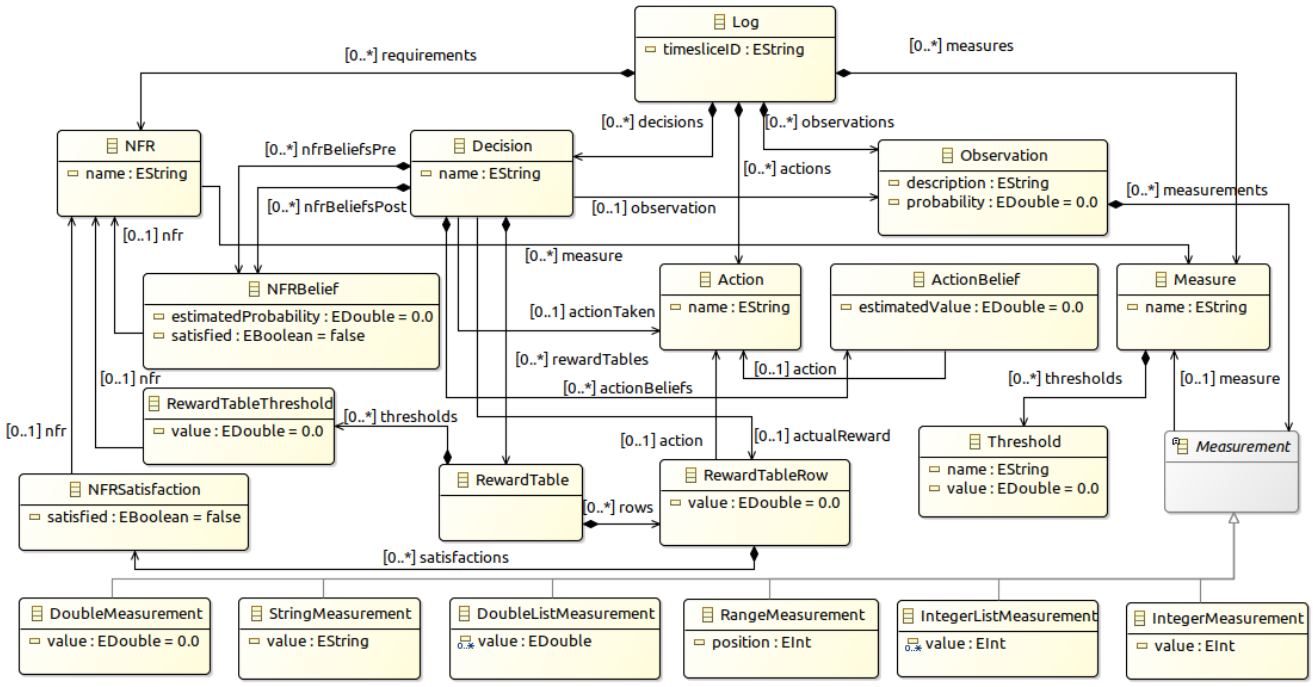

Figure 3: Execution trace metamodel for a decision-based self-adaptive system (updated since [16]).

This section shows the latest version of the four elements required for *Level 1* (as shown in Figure 1), initially developed in our previous work [16]:

- **Trace metamodel**: the state of the SAS at each timeslice was reshaped into the metamodel based on the Eclipse Modeling Framework[3] (EMF) of figure 3, essentially a custom type system for a specific domain. The metamodel has been improved since our previous work, based on community feedback and seeking generalisation. The trace metamodel is similar to the traceability information models used in the traceability community [29], but rather than relating artefacts to each other, it links the goals and decisions of the system to its observations and reasonings. It is divided into two parts: one specific to R-POMDP, and one reusable across other goal-oriented SAS types.
  The top half is the most general one. The information of a timeslice is contained inside a Log instance, which groups together the NFRs to satisfy, the observable Measures, the Decisions to be made and the Actions to choose from. The Measures are divided into ranges across Thresholds, and the Decisions are based on specific Observations that result in different types of Measurements of specific Measures. These observations help the decision making process derive an ActionBelief in the estimated value (i.e. expected utility) of each action.
  The bottom half is specific to reinforcement-based decision processes like R-POMDP, and contains the RewardTables used to make a decision, which is a lookup table made up of RewardTableRows. The lookup key is the truth value of the NFRSatisfactions 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, MEC may be considered to be satisfied if the estimated probability is higher than 70% (as stated by the requirements specifications).
- **Temporal graph DB (TGDB)**: Section 2.2 already mentioned Greycat and ChronoGraph. The current version of our approach is based on Greycat, based on its LevelDB backend. Our original work was based on the RocksDB backend. However, we concluded LevelDB offered better performance across operating systems.
- **Batch log importer**: our implementation of the RDM SAS generates JSON logs, and can be told to produce either one JSON with all timeslices from a run (potentially very large), or one JSON per timeslice. 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 3. The Git repository can then be indexed by Hawk into a Greycat TDB. This has noticeably better performance than the batch importer in prior works, which used Subversion repositories [12].
- **Temporal query language**: Hawk already had its own query language, the Epsilon Object Language [22] (conceptually, 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. In a more recent work [13], the time-aware EOL dialect was further extended with primitives

[3] http://archive.is/OLeFq

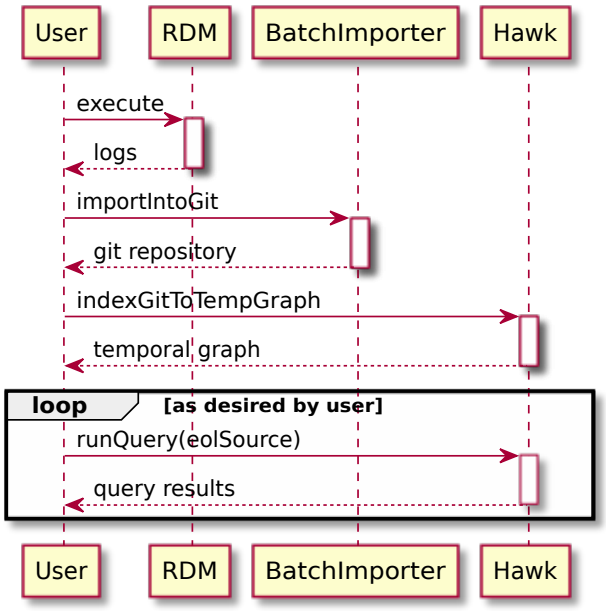

Figure 4: UML sequence diagram for interaction between components (RDM case study, *Level 1*)

inspired on Dwyer's work on temporal specification patterns, 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.

Figure 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 the BatchImporter, which produces a Git repository with one version of an XMI conforming to the metamodel in Figure 3 per timeslice. The user then tells Hawk to turn that into a temporal graph, which can be queried as needed.

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 GBs 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, we 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 is part of our future work.

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, but it would need a different "bottom half" replacing the current NFR / RewardTable concepts. For these reasons, we are considering separating this metamodel into two in the future.

6 LIVE SELF-EXPLANATION: BEYOND FORENSIC ANALYSIS

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.

In order to meet these demands, the presented implementation of *Level 2* introduces two new components: a *live visualization platform* and an *incremental importer*. The temporal graph is kept up to date as the SAS runs. This section introduces a case study where the *Level 2* infrastructure is used to answer queries while the system is running. The current implementation of the components of *Level 2* is also described.

6.1 Scenario: illustrating proactiveness to users

If a user sees that the performance of the running system is deteriorating even if momentarily, they may become anxious about its long-term viability. RDM may present cases like this. RDM is proactive and estimates the future trajectory of the system; as such it may decide to make a decision that may be perceived as negative in the short-term but, which will proof to be positive in the long-term. Like the RDM, other proactive approaches, such as the those mentioned in Section 4 (CobRA or PLA), may present these initially "surprising" situations.

One way to explain this behaviour to the user while on-the-fly (i.e. when the system is running) is to illustrate RDM's proactiveness with cases where a seemingly "bad" decision taken by the SAS turned out to be a good one in the long run. Assuming we keep track of the history of the system, Algorithm 1 can find the examples that could then be presented to the user on demand, and close to a simple plot of estimated requirement satisfaction levels over time. This takes the monitoring beyond a passive set of listings and figures, to allow users ask questions or request examples of particular relevant nuances of the SAS.

The main idea of the example in the scenario is to find a timeslice where the satisficement level of a NFR is below its threshold (e.g. MR ≥ 0.9), and the action suggested by the SAS under this context results in a further reduction in the next timeslice. However, as the decision action is further kept in the following timeslices, the satisficement gradually increases until it reaches and even exceeds its threshold. This is an example of proactive adaptation [2], the type of reasoning used in the RDM case study based on R-POMDPs. In contrast to reactive systems [11], the RDM SAS can predict what is the likely impact of the current decision action.The RDM SAS uses look-ahead search on a tree [42] to take into account the likelihoods

of future sensor observations and their effects on NFR satisficement belief levels.

Algorithm 1 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 satisficement of the NFR at timeslice t, and α^{NFR} the threshold for the NFR.

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

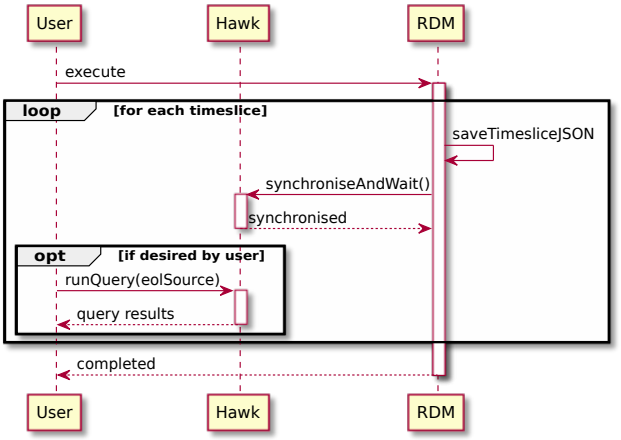

Figure 5: UML sequence diagram for interaction between components (RDM case study, *Level 2*)

6.2 Implementation of the *Level 2* components

In the implementation offered in this paper, the infrastructure related to *Level 2* takes advantage of the fact that Hawk can be run as a network service [12], with the ability to run queries at any time via its Thrift-based API[4]. Therefore, both RDM and Hawk can be running at the same time.

The query to detect proactive adaptations (see Algorithm 1), was implemented in the Hawk time-aware dialect of EOL supported by our Hawk tool. The EOL query, the incremental importer and the query tool are available from our Gitlab project[5].

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 3. This in-memory model can be given directly to Hawk, while significantly reducing overheads in comparison to our initial implementation. Previously in [14], the implementation serialised the model back into an XMI file that the standard EMF support in Hawk would use. The back-and-forth saving and loading on disk used to introduce noticeable slowdowns in the simulation.

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.

The resulting communication between the various components for the *Level 2* version of the RDM case study is shown in Figure 5. Hawk is assumed to be running and set up, having registered the trace metamodel and the folder 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.

The latest version of Hawk introduces timeline annotation, which 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. 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.

7 LIVE SELF-EXPLANATION: EVALUATION AND DISCUSSION

Previously, Section 6 described a scenario where live self-explanation support is needed for a developer to further understand the behaviour of a SAS. It also discussed the changes that were required in our prototype in order to support that capability. This section evaluates our prototype, studying both the results of the queries and the overheads introduced by our current implementation of the new time-awareness capabilities. Specifically, we study the performance of the system with two different history-aware techniques to extract explanations: i) run an EOL query that revisits the whole history at every time slice, and ii) use timeline annotation to mark situations of interest and directly jump to them while querying, as introduced in [13].

7.1 Experimental setup

For the experiments, two 2000-timeslice simulation runs were conducted, using the Eclipse Hawk model indexing server [12] (v2.1.0 nightly from July 6th, revision `eeffd8f`), which runs in the background to build the temporal graph. We used a Lenovo Thinkpad T480 with an Intel i7-8550U CPU with 1.80GHz, running Ubuntu 18.04.5 LTS, Oracle Java version 1.8.0_201 and 15.6GB RAM, allocating 8GB to Hawk (`-Xmx8g`).

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

Listing 1: Excerpt of output from Algorithm 1 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]]], ...]
```

The RDM SAS was configured to communicate directly with Hawk to update the temporal graph and then run the EOL implementation of Algorithm 1 after each timeslice. In the case of timeline annotation, the situation of interest to be monitored was configured in the server in advance. No other processes were running in the system. The Greycat DB grew to 5MB with the first technique and 14MB with timeline annotations.

7.2 Query results

Listing 1 shows an excerpt of the examples found by the query with both techniques (with and without timeline annotation), 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 1597. 29 situations were found during the 2000 timelices 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 behaviours, which otherwise could be found unreasonable at first.

Based on the defined concept of self-explanation earlier in this paper, 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 it after *Level 3* of the spectrum is reached.

7.3 Performance results

Figure 6 includes the execution times for the simulation run 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

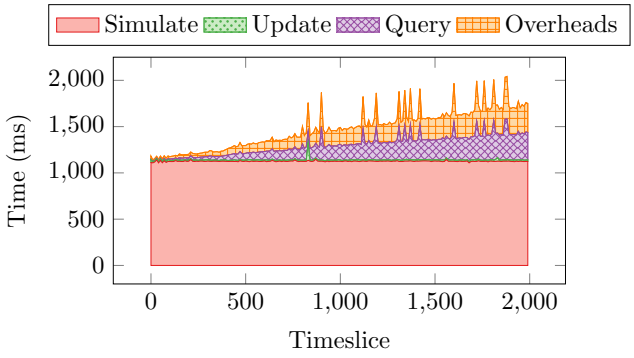

Figure 6: Stacked area plot with execution times for RDM SAS simulations in milliseconds, by timeslice and phase, for the queries running without annotations.

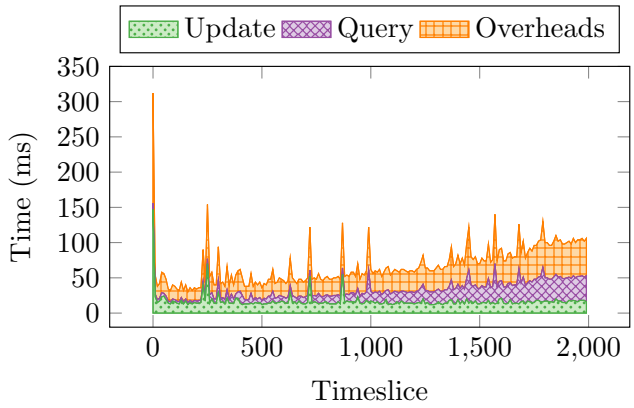

Figure 7: Stacked area plot with execution times for RDM SAS simulations in milliseconds, by timeslice and phase, for the queries running with annotations. "Simulate" times are excluded due to small values in the other series, being the same as in Figure 6.

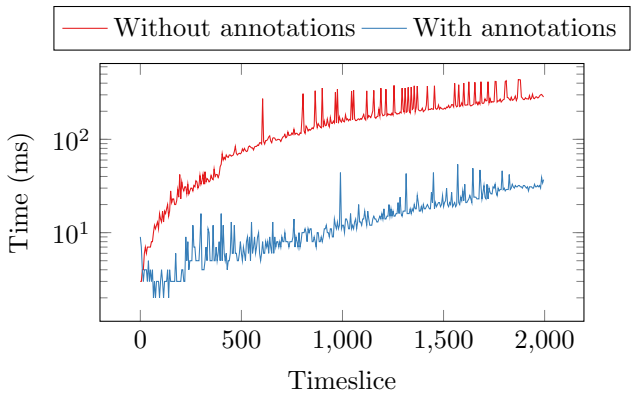

Figure 8: Raw server-side execution times of EOL query implementing Algorithm 1 in milliseconds, by timeslice.

history of the temporal graph. Query times came to represent up to 14% of the execution time in average. This 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 6.1. In other words, when the system decided to change (i.e. adapt) the topology and this action ended in a reduction of the satisficement level of the NFR. Figure 7 shows the different stages, except for the simulation time that is the same as in Figure 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. We can conclude with timeline annotation, that the reduction in the system's performing time can be kept to 2–3% due to overheads. This is a significant improvement from our previous version of our *Level 2* implementation in [14], where the addition of temporal graphs made a 1000-timeslice simulation to go from 18.25 minutes to 94.65 minutes (an increment over 400%).

Figure 8 shows the raw execution times for the EOL implementation of the query in Algorithm 1 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 and 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.

7.4 Next steps

While the original solution (without timeline annotation) worked as expected, the total simulation time raised to 43.54 minutes from the original 37.49 minutes which could be considered still an important overhead. Fortunately, the timeline annotation capability improved the performance impact to 38.36 minutes. A different approach, which we plan to pursue, would be to run the processing steps in a concurrent fashion, while the time-awareness could run at a slower pace than the main decision-making. Queries took longer as the history grew, but different optimization strategies will be tested in the future. Some new strategies we are considering to apply include: *sampling*, for only storing logs at a certain rate; *time windows*, for focusing on the last n timeslices; or *event-oriented processing*, for storing logs only when certain events happen. These strategies would allow us to further reduce the storage and processing overheads imposed by the addition of history awareness.

Beyond increasing efficiency and improving the queries, future work will also look into the completion of *Level 2* with dedicated visualisations, and the design of new queries that look into system features desired by users (e.g. NFR satisficement status) and developers (e.g. stability or predictability). Work on *Level 3* will continue, with the extension of a decision-making process to allow hints from an external entity (initially a human) using effectors. Once these adaptation controls are in place, work on the *Level 4* reflective and self-aware SAS would start.

8 CONCLUDING REMARKS AND FUTURE WORK

We have given our vision for developing reflective, self-adaptive systems, proposing a spectrum of capabilities, starting from *Level 1* (forensic "black box" self-explanation after the system has finished running) and ending at *Level 4* (autonomous history-aware decision-making). An existing SAS in the Remote Data Mirroring domain has been extended up to *Level 2* (live self-explanation), which has been reported in this paper. We have proposed the explicit use of temporal graph databases as a representation for trace models to support self-explanation, interactive diagnosis and forensic analysis. We have presented a generic meta-model to structure execution traces of SAS, based on the meta-model. A SAS system was extended to run arbitrary commands between timeslices. Two utilities are offered to convert JSON-based logs into a reusable metamodel to finally obtain a temporal history-based graph model. Moreover, queries can be designed and implemented in the EOL query language supported by the Hawk indexer to study long-term effects of adaptations. The case study has been evaluated, where the temporal graph was updated and queried.

We have also presented an architecture, which allows recording of temporal data that can be queried to explain behaviour. Our work so far is based on the use of cases supported by Bayesian learning and Markov processes, which naturally fit in with the idea of time series modelling. However, we are currently working on how to include other learning techniques, and explore links from deeper nuances of decision-making data with the time series modelling. These techniques build up certain data structures during their learning, such as the Q-tables in the Q-Learning model-free reinforcement learning algorithm [40]. Combining internal data (like the Q-table) with an understanding of the theory behind the algorithm, we can explain some aspects of their decision-making. However, the understanding is partial, and further, it still would need to be linked to abstractions related to the understanding by end users or at the level of requirements for explainability [35]. We argue that the temporal graph models can provide further support for more comprehensive explainability. Further, as self-adaptation needs to be proactive and not just reactive, we are working on further conceptual models to support proactive decision-making [7] by anticipating adaptation actions based on history.

Acknowledgements: The work was partially funded by the Leverhulme Trust Research Fellowship RF-2019-548 and the EPSRC Research Project Twenty20Insight (Grant No. EP/T017627/1).

REFERENCES

[1] K. Angelopoulos, A. V. Papadopoulos, V. E. S. Souza, and J. Mylopoulos. 2016. Model Predictive Control for Software Systems with CobRA. In *2016 IEEE/ACM 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. 35–46. https://doi.org/10.1109/SEAMS.2016.012

[2] H. Bai, S. Cai, N. Ye, D. Hsu, and W. S. Lee. 2015. Intention-aware online POMDP planning for autonomous driving in a crowd. In *2015 IEEE Conference on Robotics and Automation (ICRA)*. 454–460. https://doi.org/10.1109/ICRA.2015.7139219

[3] T. Becker, A. Agne, P. R. Lewis, R. Bahsoon, F. Faniyi, L. Esterle, A. Keller, A. Chandra, A. R. Jensenius, and S. C. Stilkerich. 2012. EPiCS: Engineering Proprioception in Computing Systems. In *IEEE 15th International Conference on Computational Science and Engineering*. 353–360. https://doi.org/10.1109/ICCSE.2012.56

[4] N. Bencomo and L. H. Garcia Paucar. 2019. 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)*.

[5] Nelly Bencomo, Sebastian Götz, and Hui Song. 2019. Models@run.time: a guided tour of the state of the art and research challenges. *Software & Systems Modeling* 18, 5 (Oct. 2019). https://doi.org/10.1007/s10270-018-00712-x

[6] Javier Cámara, Kirstie L. Bellman, Jeffrey O. Kephart, Marco Autili, Nelly Bencomo, Ada Diaconescu, Holger Giese, Sebastian Götz, Paola Inverardi, Samuel Kounev, and Massimo Tivoli. 2017. *Self-aware Computing Systems: Related Concepts and Research Areas*. Springer International Publishing, Cham, 17–49. https://doi.org/10.1007/978-3-319-47474-8_2

[7] Javier Cámara, Gabriel A. Moreno, and David Garlan. 2014. Stochastic Game Analysis and Latency Awareness for Proactive Self-adaptation. In *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2014)*. ACM, New York, NY, USA, 155–164. https://doi.org/10.1145/2593929.2593933

[8] Tao Chen, Rami Bahsoon, and Xin Yao. 2018. A Survey and Taxonomy of Self-Aware and Self-Adaptive Cloud Autoscaling Systems. *ACM Comput. Surv.* 51, 3, Article 61 (June 2018), 40 pages. https://doi.org/10.1145/3190507

[9] Betty H. C. Cheng, Rogério de Lemos, Holger Giese, et al. 2009. *Software Engineering for Self-Adaptive Systems: A Research Roadmap*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1–26. https://doi.org/10.1007/978-3-642-02161-9_1

[10] Philippe Esling and Carlos Agon. 2012. Time-series data mining. *Comput. Surveys* 45, 1 (2012), 12. https://doi.org/10.1145/2379776.2379788

[11] D. Fox, W. Burgard, and S. Thrun. 1997. The dynamic window approach to collision avoidance. *IEEE Robotics Automation Magazine* 4, 1 (March 1997), 23–33. https://doi.org/10.1109/100.580977

[12] Antonio Garcia-Dominguez, Konstantinos Barmpis, Dimitrios S. Kolovos, Ran Wei, and Richard F. Paige. 2019. Stress-testing remote model querying APIs for relational and graph-based stores. *Software & Systems Modeling* 18, 2 (June 2019), 1047–1075. https://doi.org/10.1007/s10270-017-0606-9

[13] Antonio Garcia-Dominguez, Nelly Bencomo, Juan Marcelo Parra-Ullauri, and Luis Garcia-Paucar. 2019. Querying and annotating model histories with time-aware patterns. In *Proceedings of the 22nd International Conference on Model Driven Engineering Languages and Systems*. ACM, Munich, Germany. https://doi.org/10.1109/MODELS.2019.000-2

[14] A. Garcia Dominguez, N. Bencomo, J. M. Parra Ullauri, and L. H. Garcia Paucar. 2019. Towards History-Aware Self-Adaptation with Explanation Capabilities. In *IEEE 4th Workshop on Foundations and Applications of Self* Systems (FAS*W)*.

[15] Luis Garcia-Paucar, Nelly Bencomo, and Kevin Fung Yuen. 2019. ARRoW: Automatic Runtime Reappraisal ofWeights forSelf-Adaptation. *34th ACM/SIGAPP Conference on Applied Computing. Limassol, Cyprus* (2019).

[16] Antonio García-Domínguez, Nelly Bencomo, and Luis H Garcia Paucar. 2018. Reflecting on the past and the presentwith temporal graph-based models. In *Proceedings of MODELS Workshops*, Vol. 2245. CEUR-WS.org, Denmark, 46–55.

[17] Martin Haeusler, Thomas Trojer, Johannes Kessler, Matthias Farwick, Emmanuel Nowakowski, and Ruth Breu. 2018. ChronoGraph: A Versioned TinkerPop Graph Database. In *Data Management Technologies and Applications (Communications in Computer and Information Science)*, Joaquim Filipe, Jorge Bernardino, and Christoph Quix (Eds.). Springer International Publishing, 237–260.

[18] Thomas Hartmann, Francois Fouquet, Matthieu Jimenez, Romain Rouvoy, and Yves Le Traon. 2017. Analyzing Complex Data in Motion at Scale with Temporal Graphs. In *Proceedings of the 29th Conference on Software Engineering & Knowledge Engineering (SEKE'17)*. 596–601. https://doi.org/10.18293/SEKE2017-048

[19] Minwen Ji, Alistair C Veitch, John Wilkes, et al. 2003. Seneca: remote mirroring done write.. In *USENIX Annual Conference*. 253–268.

[20] Margot E. Kaminski. 2019. The Right to Explanation, Explained,. *Berkeley Technology Law Journal* (2019). https://doi.org/10.15779/Z38TD9N83H

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

[22] Dimitrios S. Kolovos, Richard F. Paige, and Fiona Polack. 2006. The Epsilon Object Language (EOL). In *Model Driven Architecture - Foundations and Applications, Second European Conference, ECMDA-FA 2006, Bilbao, Spain, July 10-13, 2006, Proceedings*. 128–142. https://doi.org/10.1007/11787044_11

[23] Hanna Kurniawati, David Hsu, and Wee Sun Lee. 2008. Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces.. In *Robotics: Science and systems*, Vol. 2008. Switzerland.

[24] H. Kurniawati and V. Yadav. 2013. An Online POMDP Solver for Uncertainty Planning in Dynamic Environment. *Proc. Int. Symp. on Robotics Research* (2013).

[25] George E. Monahan. 1982. State of the Art—A Survey of Partially Observable Markov Decision Processes: Theory, Models, and Algorithms. *Management Science* 28, 1 (Jan. 1982), 1–16. https://doi.org/10.1287/mnsc.28.1.1

[26] Gabriel A. Moreno, Javier Cámara, David Garlan, and Bradley Schmerl. 2015. 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)*. ACM, New York, NY, USA, 1–12. https://doi.org/10.1145/2786805.2786853

[27] Ludovic Mouline, Amine Benelallam, François Fouquet, Johann Bourcier, and Olivier Barais. 2018. A Temporal Model for Interactive Diagnosis of Adaptive Systems. In *2018 IEEE International Conference on Autonomic Computing, ICAC 2018, Trento, Italy, September 3-7, 2018*. 175–180. https://doi.org/10.1109/ICAC.2018.00029

[28] Ludovic Mouline, Amine Benelallam, Thomas Hartmann, François Fouquet, Johann Bourcier, Brice Morin, and Olivier Barais. 2018. Enabling Temporal-aware Contexts for Adaptative Distributed Systems. In *Proceedings of the 33rd Symposium on Applied Computing (SAC '18)*. ACM, New York, NY, USA, 1433–1440. https://doi.org/10.1145/3167132.3167286

[29] Patrick Mäder and Jane Cleland-Huang. 2013. A visual language for modeling and executing traceability queries. *Software & Systems Modeling* 12, 3 (July 2013), 537–553. https://doi.org/10.1007/s10270-012-0237-0

[30] Luis H Garcia Paucar, Nelly Bencomo, and Kevin Kam Fung Yuen. 2017. Juggling Preferences in a World of Uncertainty. *RE NEXT, Lisbon.* (2017).

[31] Andres J Ramirez, Adam C Jensen, and Betty HC Cheng. 2012. A taxonomy of uncertainty for dynamically adaptive systems. In *7th Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE.

[32] Owen Reynolds, Antonio García-Domínguez, and Nelly Bencomo. 2020. Towards automated provenance collectionfor runtime models to record system history. In *Proceedings of SAM 2020*. https://doi.org/10.1145/3419804.3420262 To be published.

[33] Mazeiar Salehie and Ladan Tahvildari. 2009. Self-adaptive Software: Landscape and Research Challenges. *ACM Trans. Auton. Adapt. Syst.* 4, 2, Article 14 (May 2009), 42 pages. https://doi.org/10.1145/1516533.1516538

[34] P. Sawyer, N. Bencomo, J. Whittle, E. Letier, and A. Finkelstein. 2010. Requirements-Aware Systems: A Research Agenda for RE for Self-adaptive Systems. In *2010 18th IEEE International Requirements Engineering Conference(RE)*, Vol. 00. 95–103. https://doi.org/10.1109/RE.2010.21

[35] Pete Sawyer, Nelly Bencomo, Jon Whittle, Emmanuel Letier, and Anthony Finkelstein. 2010. Requirements-Aware Systems: A Research Agenda for RE for Self-adaptive Systems. In *Proceedings of the 2010 18th IEEE International Requirements Engineering Conference (RE '10)*. IEEE Computer Society, Washington, DC, USA, 95–103. https://doi.org/10.1109/RE.2010.21

[36] Andrew D Selbst and Julia Powles. 2017. Meaningful information and the right to explanation. *International Data Privacy Law* 7, 4 (12 2017), 233–242. https://doi.org/10.1093/idpl/ipx022

[37] Adhiraj Somani, Nan Ye, David Hsu, and Wee Sun Lee. 2013. DESPOT: Online POMDP planning with regularization. In *Advances in neural information processing systems*. 1772–1780.

[38] Gabriel Tamura, Norha M. Villegas, Hausi A. Müller, Laurence Duchien, and Lionel Seinturier. 2013. Improving context-awareness in self-adaptation using the DYNAMICO reference model. In *Proceedings of the 8th Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2013, USA, May 20-21, 2013*. 153–162. https://doi.org/10.1109/SEAMS.2013.6595502

[39] Norha M. Villegas, Gabriel Tamura, Hausi A. Müller, Laurence Duchien, and Rubby Casallas. 2013. *DYNAMICO: A Reference Model for Governing Control Objectives and Context Relevance in Self-Adaptive Software Systems*. Springer Berlin Heidelberg, Berlin, Heidelberg, 265–293. https://doi.org/10.1007/978-3-642-35813-5_11

[40] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine learning* 8, 3-4 (1992), 279–292. https://doi.org/10.1007/BF00992698

[41] Kristopher Welsh, Nelly Bencomo, Pete Sawyer, and Jon Whittle. 2014. Self-Explanation in Adaptive Systems Based on Runtime Goal-Based Models. *Trans. Computational Collective Intelligence* 16 (2014), 122–145. https://doi.org/10.1007/978-3-662-44871-7_5

[42] Nan Ye, Adhiraj Somani, David Hsu, and Wee Sun Lee. 2017. DESPOT: Online POMDP Planning with Regularization. *J. Artif. Int. Res.* 58, 1 (Jan. 2017), 231–266. http://dl.acm.org/citation.cfm?id=3176764.3176770