From a Series of (Un)fortunate Events to Global Explainability of Runtime Model-Based Self-Adaptive Systems

Juan Marcelo Parra-Ullauri
SEA research group, EPS
Aston University
Birmingham, UK
j.parra-ullauri@aston.ac.uk

Antonio García-Domínguez
SEA research group, EPS
Aston University
Birmingham, UK
a.garcia-dominguez@aston.ac.uk

Nelly Bencomo
Department of Computer Science
Durham University
Durham, UK
nelly.bencomo@durham.ac.uk

***Abstract*—Self-adaptive systems (SAS) increasingly use AI-based approaches for their flexible decision-making, which often appear to users as "black boxes". These systems can exhibit unexpected and surprising behaviours that may violate imposed constraints. Runtime models (RTMs) have been used for SAS management in order to provide capabilities needed to explain reasons why the system present the current emergent behaviour. Existing work on explanations derived from RTMs have focused on justifying why the system has presented a specific behaviour at a given time. Nevertheless, we argue that a more general scope is required for understanding the entire evolution of the system, rather than understanding the behaviour for a given instance or situation. From the point of view of Explainable AI (XAI), the latter type of explanations are called *global explanations*, whereas understanding a single decision refers to *local explanations*. Global explanations tend to promote trust on the system in question, while local explanations tend to promote trust on a specific decision. In this paper, we propose the use of event graph models to construct global explanations from evolving RTMs. Event graphs allow the representation of the system behaviour as a state-time diagram, by indicating the *occurrence of events* and their relationships. RTMs are incrementally queried to look for situations of interest (i.e. events), using Complex Event Processing (CEP) in order to analyze and correlate real-time events and therefore, derive conclusions. The approach is applied to a AI-enhanced SAS in the domain of mobile communications. The encouraging results show that event graphs allow the system to present a summarised overview of the system's behaviour, promoting understandability and trustworthiness.**

***Index Terms*—Runtime Models, Global Explainability, Self-Adaptive Systems, CEP, Event Graph Models, XAI**

I. Introduction

The complexity of real-world problems requires modern software systems to be able to autonomously adapt and modify their behaviour at run-time, based on their observations to cope with uncertain and dynamic environments [1]. In order to tackle different complexities of these self-adaptive systems (SAS), runtime models (RTMs) have been widely used [2]. RTMs underpins self-awareness, by allowing systems to abstract their own state and behaviour in a way that is amenable for automated adaptation strategies ([2], [3]). SAS can exhibit unexpected and surprising behaviours due to their inherent complexity ([4], [5]), which is exacerbated by the ubiquity of Artificial Intelligence (AI)-based SAS [6].

RTMs provide abstraction, analysis and reasoning capabilities needed to explain why the system shows a given emerging behaviour [7]. Explainability in SAS 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 [8]. Moreover, explainability can be key to enable understanding to promote the widespread adoption of SAS [8]. Current works on explanation of SAS based on RTMs have focused on the justification of the system's behaviour for a specific point or points in time ([9], [10], [8]). Nonetheless, we argue that a more general scope may be required to understand the entire system's reasoning rather than just a single decision for a given point in time. An example is the case of developers trying to comprehend how a system operates over time, and to understand the logic and reasoning that lead to all possible outcomes. According to the Explainable Artificial Intelligence (XAI) and Explainable Reinforcement Learning (XRL) community ([11], [12]), the latter type of explanations are called *global explanations*, whereas understanding a single decision refers to *local explanations* [13]. While a global approach leads to users trusting a system, a local one leads to trusting a specific decision [12]. On the other hand, event graphs can conveniently model complex system behaviour as *state-time diagrams*, by indicating the occurrence of events and their relationships [14]. Furthermore, event graph models can be constructed and visualized for system analysis, extracting insights understandable to humans ([15], [16]).

In this paper, we propose the use of event graph models (which are built from the evolution of runtime models overtime) to represent global explanations in a human-understandable way, based on the timeline of the running system. For this purpose, runtime models are queried to look for situations of interest (i.e., events) using Complex Event Processing (CEP). CEP is an event-driven monitoring approach [17] that can be used for extracting valuable information from systems at runtime. Meaningful situations

(called complex events) can be specified by performing stepwise correlation over streams of events [18]. These events are analyzed and processed to build an event graph of the system's behaviour during execution, by focusing on specific events triggered by state changes. In this paper, we present a prototype of the system in a RL-based SAS to determine the feasibility of the approach, while tackling XRL. The results show that with the use of event graphs it is possible to represent the system as a whole (global explanations) based on events of interest. The approach can be used by developers and operators to gain insights about the system. These explanations could be used to prove or disprove hypotheses posed on the system behaviour upon demand. A main contribution of the paper is a demonstration of the potential for a new thread of research that lies at the intersection of modeling and CEP to support XAI.

The rest of the paper is structured as follows. Section II presents the foundations that underlie this research. Section III describes our envisioned proposal, and Section IV presents the case study and its discussion. Section V compares this work with other similar ones. Finally, Section VI presents the conclusions and future work.

II. Background

A. Why do we need Explanations?

Explanations provide a key capability to shape the human understanding of the environment, especially when their perceptions diverge from their expectations [19]. 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 [9]. Explaining the decision-making of a system is becoming increasingly important to enhance collaboration, and to increase confidence [20]. This is ratified by the General Data Protection Regulation (GDPR) law, which enshrines the right to explanation [21]. Considering this, explanation-aware computing has received growing interest due to the ubiquity and complexity of AI-based systems, creating the notion of explainable AI (XAI) [11] to gain insight into the "black boxes" associated with AI.

There are different arguments in favor of explanations in AI. Adadi et al. stated four main ones in [13]: *explain to justify* decisions impacting people, *explain to control* that the AI stays within an envelope of good behaviour, *explain to discover* knowledge from the learned behaviour, and *explain to improve* the system by discovering its flaws. Besides the motivation for explaining a system, it is also important to understand who or what is going to consume the explanations. An explanation could target humans (end-users and developers) or machines (external systems that interact with the SAS, or the SAS itself). If the consumer is the system itself, it is called a *self-explanation*. 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 using explanations to refine their internal models and reasoning processes [6]. In the present work, we focus on explanations to control and explanations to improve, targeting developers and SAS-knowledgeable users. 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 [22].

B. Types of explanations in XAI and XRL

There are different dimensions for classifying the approaches used in XAI and XRL [12], [13]. In this section we focus on two dimensions: the scope of the explanations, and the method for presenting the explanations.

(i) *depending on the scope of the explanations*, an approach can be global or local [12]. Local explanations focus on why the AI-system made a certain decision for one or a group of instances (points in time) [12], whereas global explanations focus on the whole AI-system and provide an understanding of the overall decision process. A global explanation aims to provide a general understanding of how the system works [13].

(ii) *depending on the presentation method*, different ways to communicate the explanations have been recognised [23]. Text-based explanations are the most common type presented in the form of natural language or logs ([24], [25]). Visual representations such as graphs, plots, or heatmaps among others are also used to depict explanatory information ([26], [27]). Other approaches include expressive motions and indicators, as well as text-to-speech for explaining robots [23].

The proposed approach aims to provide global explanations that describe the overall system behaviour by tracking the evolution of events on runtime models. The selected way for presenting and communicating explanations are through visual representations, specifically event graph models.

C. Tracking the Evolution of Runtime Models

Storage solutions for evolving models have traditionally been divided into two types: file-based approaches such as Git [28], or dedicated versioned model repositories such as the Eclipse Connected Data Objects project [29]. Traditional version control systems (VCS) by themselves do not provide any support for scalable querying [7]. Barmpis et al. proposed Hawk, a *model indexer* that mirrors model repositories stored in traditional VCS into graph databases, simplifying and speeding up queries [30].

In regard to using the runtime models of a SAS for explainability, different approaches have been proposed. Mouline et al. presented a metamodel for interactive diagnosis of adaptive systems which combines design-time and run-time concerns [31]. 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. Reynolds et al. proposed in [10] 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. Different from provenance graphs, in our previous work [8] we proposed temporal graphs and runtime models (i.e., temporal models) to provide history-aware explanations. We proposed an execution trace metamodel for linking the SAS goals and decisions to its observations and reasoning. These works have 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 reasoning is also required (i.e., global explanations).

D. Event-driven Monitoring, Complex Event Processing and Event-Graph Models

In order to explain AI "black boxes", their behaviour should be observed. Event-driven monitoring is a common approach to gain insights about a system [32]. This approach 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 [33]. Concerning how to process those events in real time, complex event processing (CEP) is a common choice ([34], [35], [36]). CEP [17] 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 [34]. These situations 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 events types [36]. Event patterns are deployed to a CEP engine, i.e., the software that allows the incoming data streams to be analysed in real time according to the defined patterns [18]. Each CEP engine provides its own event processing language (EPL) for defining the patterns to be deployed.

On the other hand, event graphs are a way of graphically representing discrete-event simulation models [37]. System dynamics are characterised by the events that change the state of the system, and the logical and temporal relationships among these events [14]. An event graph consists of nodes and directed edges. Nodes represent events (changes in state), and edges represent the temporal and logical relationships between pairs of events [37]. Fig. 1 shows the basic building blocks of an event graph. An event node n will change the system state from s to $f_n(s)$. Each directed edge $e_{od} = (n_o, n_d)$ indicates that if the condition (i) is true when the origin event n_o occurs, the destination event n_d is to be scheduled after a time delay t. If the event n_d is always scheduled, the edge condition is omitted, and the edge is called an unconditional edge. In the present work, we show how event graphs constructed from complex events triggered by CEP could be used to provide a global explanation for the behaviour of a SAS.

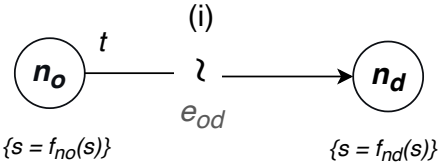

Fig. 1: Basic building blocks of an event graph: events (nodes n) and relationships (edges e) [14].

III. Proposal: Event Graph Models for Global Explainability

The previous section considered the available work on XAI and XRL, tracking of model evolution, and complex event processing. This section presents our proposal combining these approaches to generate global explanations for SAS based on runtime models. The solution proposed is based on the definition of the *black box explanation problem* by Guidotti et al. [38], to provide explanations through an interpretable and transparent model [38]. The model should be able to mimic the behaviour of the black box while being understandable by humans. The main objective of our approach is to provide global explanations to developers and knowledgeable users in the SAS domain that describe the overall system behaviour based on tracking the evolution of runtime models, helping to control and to improve SAS. More specifically, we propose the use of event-driven monitoring over the evolving runtime models of the SAS to build and maintain an event graph (the visual representation shown in Section II-B) that would explain its behaviour shown over time.

We propose using CEP as the main technology to detect temporal and causal dependencies between events, in order to gain insights from events as they occur during execution. Figure 2 depicts the proposed approach. It starts by detecting simple events identified from the runtime models. These events are filtered to detect the situations of interest. Afterwards, the events are correlated using event patterns to group and summarise the simple events. The results are finally displayed as event graphs for visual global explainability. Further details are listed below:

1) Simple Events: Simple events are occurrences of particular low-level patterns of something that happened at a point in time [36]. These low-level occurrences can then be processed and analysed to produce higher-level occurrences/events. 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. A simple event has a certain type and form. The form of an event may contain fields, which are similar to class attributes in object-oriented languages [39]. A simple event does not provide information about the previous state. Furthermore, it does not have a duration, existing only at a specific point of time [16]. In this approach, simple events are the changes that take place in the runtime model, such as the update of an instance of a

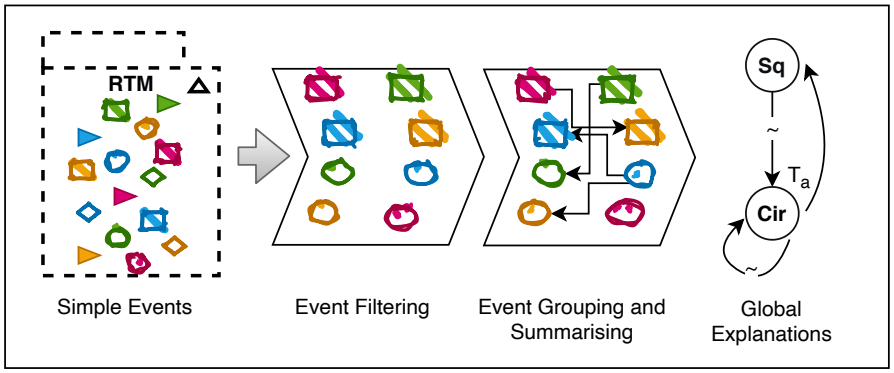

Fig. 2: Global Explainability using Event Graphs

component. Whenever a change occurs in the model such as creation and deletion of an instance, a simple event is detected.

2) Event Filtering: Before an explanation is constructed, the simple events, identified as changes in the runtime model, are classified. They produce complex events about the changes in the high-level state of the SAS. The filtering can be performed using different approaches, e.g., through incremental graph pattern matching approaches as those from Ehmes et al. [39], or through the use of EPL patterns in a CEP engine. For this implementation, we use CEP for processing and correlating events that match certain filtering criteria and help to provide an understanding of the mechanism on which the SAS based its decision making. The filtering patterns are the criteria for relevant situations that will be the focus of interest for the explanations. These patterns are implemented and provided to the CEP engine by developers at design-time and can be updated at runtime. When patterns are detected, the engine automatically generates complex events that are sent to the next stage for further processing. For example the creation and deletion events are identified, tagged and let through to the next stage for grouping and creation of the summary.

3) Event Grouping and Summarising: This stage enables reaching the higher level of abstraction needed to provide the summarised information and to convey explanations about the system behaviour. The previous filtered events are grouped by considering defined representative features such as its type and time of occurrence. As in the event filtering stage, this process is performed by using CEP. Event patterns containing the grouping criteria are deployed to the CEP engine. These patterns, when detected, generate higher-level complex events that contain the information required to present an explanation, which is performed in the next stage. For example, all the creation events followed by a deletion event in the next point in time are grouped as temporary occurrences.

4) Global Explanations through Event Graphs: Event graphs are used to create a representation of the system based on events of interest. This representation can offer an approximation of how the system works, providing a global explanation. By defining event patterns that analyse temporal correlations between individual events, knowledge can be derived in the form of complex events. Using complex events,

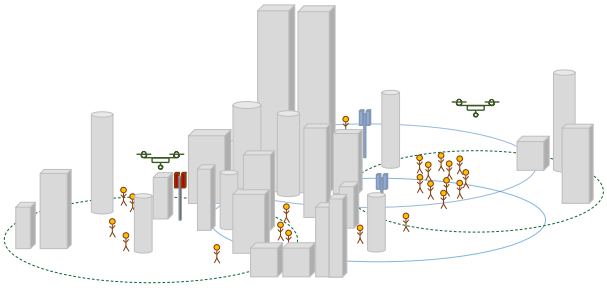

Fig. 3: Overview of the ABS (green) SAS from [40]

the aforementioned event graph is constructed with nodes as events (e.g. creation and deletion), and edges to indicate their temporal relationships. Afterwards, the event graph can be visualised to facilitate the developer's comprehension of the system focused on events of interest (e.g. temporary occurrences).

To demonstrate our envisioned solution for extracting global explanations of a SAS, a substantial case study is presented in the next section.

IV. Case Study

A. System under study: Airborne Base Stations SAS

This section presents a SAS from the domain of mobile communications. The case study (Fig. 3) shows airborne base stations (ABS) that can potentially be deployed upon a failure of the base stations (red in Fig. 3), which cause communication difficulties for public safety and emergency communications [40]. The SAS uses *Reinforcement Learning (RL)* to move the ABS autonomously for providing connectivity to as many users as possible [40]. RL is an AI approach where agents learn actions based on their ability to maximise a defined reward in a trial-and-error fashion [41]. In RL, an agent is trained to select actions to interact with the environment that maximise the cumulative reward resulting from those interactions [12]. RL is usually introduced as a Markov Decision Process (MDP), as it satisfies the Markov property of sensation, action, and goal [40].

For the current implementation we have selected Q-Learning as the RL algorithm. In Q-Learning, an agent uses an

action-value function Q to evaluate the expectation of the maximum future cumulative reward. The reward $r_t = Q(s_t, a_t)$ is obtained from executing an action a_t at a given state s_t, which provides agents with the capability of learning to act with the aim of maximising the global reward [41].

The ABS SAS performs the necessary calculations to estimate the Signal-to-Interference-plus-Noise Ratio (SINR) and the Reference Signal Received Power (RSRP) towards its goal of maximising rewards (users connected). The SINR and RSRP values measure the signal quality of the communications between the ABS and end-user devices (e.g., mobile phones). SINR and RSRP thresholds are used to determine whether a station is considered to be "connected" or not [40].

B. Experimentation: Scenario

In this case study, the developers of the ABS SAS are interested in studying the reasons why the system acted as it did, both regarding single decisions and its overall performance. For this purpose, a simulation of the SAS was run. It consisted of a training run of 10 episodes and 2000 steps for 2 ABSes with 1050 users scattered on a X-Y plane. The observations and decisions made by the system during this process are reshaped into the trace metamodel, as proposed in our previous work [8]. The trace metamodel links the system goals and decisions to its observations and reasoning processes. An object diagram with an instance of the runtime model at a certain step in the simulation is shown in Fig. 4. The LOG contains DECISIONs and OBSERVATIONs for `ABS 1` at `Episode 9` and `Step 199`. The possible ACTIONs are linked to their ACTIONBELIEFs that represent the estimated values (Q-values), which maximise the cumulative MEASURE (`Global reward`) at the given MEASURE (`State`).

As part of its use of RL, the system changes between *exploration* and *exploitation* states. Exploration means trying to discover new features of the environment by selecting a sub-optimal action. On the other hand, exploitation is when the agent chooses the best action according to what it already knows [42]. The developers want to gain a general idea of how the system changes between these two states: to do so, the process depicted in Fig. 2 is followed.

C. Experimentation: Setup

1) Simple Events: These low-level occurrences represent changes in the runtime model. To keep track of these changes, each instance of the model is analysed. Examples of a simple event are a DECISION being taken, or an ACTION being performed.

2) Event Filtering: For the present prototype, we used CEP to filter simple events focusing on decisions based on their type (exploration or exploitation). Using the Esper[1] CEP engine, we defined event patterns for detecting situations of interest and producing complex events about them. In order to find when a decision was performed using exploration or using exploitation, it is required to track the actual action taken and the Q-values (i.e., the ACTIONBELIEFs) 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 is considered to be taken by exploration. Considering the object diagram of Fig. 4, 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. Listing 1 shows the Esper EPL pattern for finding this situation. At each point in time, the Q-value of the selected action (`drone.qtable.action`) is compared to `maxValue()`, the maximum Q-value of the available actions.

Listing 1: Esper 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
```

3) Event Grouping and Summarisation: As mentioned in Section II-D, complex events can be fed back to the system to derive complex events based on other complex events. The same CEP engine (Esper) is used to group and correlate events. In this experiment, we focus on how the system chooses between exploration and exploitation, tracking how often it chooses each option, how often it stays within the same state, and how often it changes from one state to the other. Aiming to characterise this behaviour, a *hierarchy of complex events* approach with 2 levels and 4 patterns was required. Complex events marked as "Exploration" E_R or "Exploitation" E_T from the event filtering stage are analysed on an event window for the subsequent time point as shown in Fig. 5. If an action performed by exploration at t_i is followed by another action performed by exploration at t_{i+1} ($E_R \Rightarrow E_R$), a counter is incremented. The same process is applied for the different scenarios: exploration followed by exploitation ($E_R \Rightarrow E_T$), exploitation followed by exploitation ($E_T \Rightarrow E_T$), and ex-

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

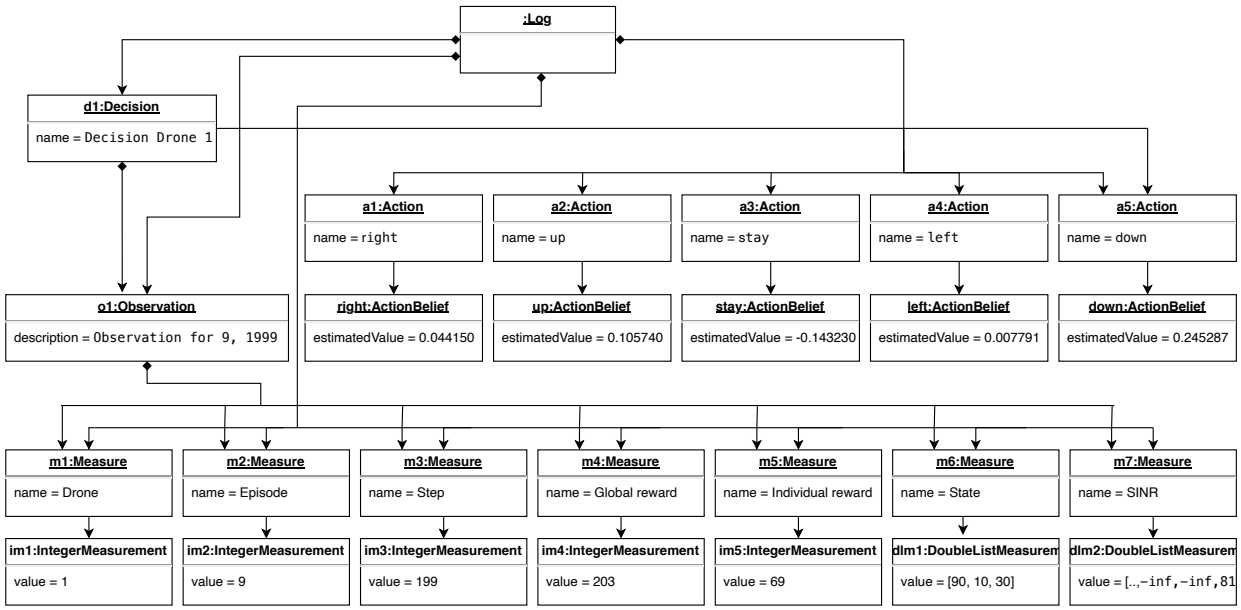

Fig. 4: Runtime model object diagram based on the trace metamodel from [8]

ploitation followed by exploration ($E_T \Rightarrow E_R$).

Additionally, we wanted to analyze how these types of actions affected the overall goal of the system (connecting as many users as possible). With this purpose, further Esper EPL event patterns following Algorithm 1 were defined. From the different instances T of runtime model M, actions marked as E_R or E_T are analysed and classified in 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.

4) Global Explanations through Event Graphs: We focus on events that occur over the entire execution of the case under study: here whether the system acted based on exploration (event E_R) or exploitation (event E_T). As shown in Fig. 6, the event graph represents how frequently the system acts one way or the other, and how often it changes between the two behaviours. Additionally, we were 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 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:**

Algorithm 1 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.

  - 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.

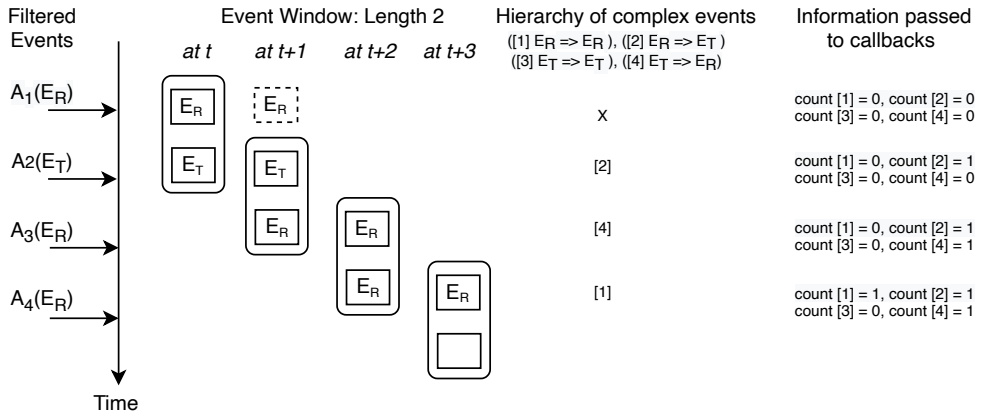

Fig. 5: Hierarchy of complex events. A_i = Actions marked as exploration (E_R) or exploitation (E_T) from the filter stage.

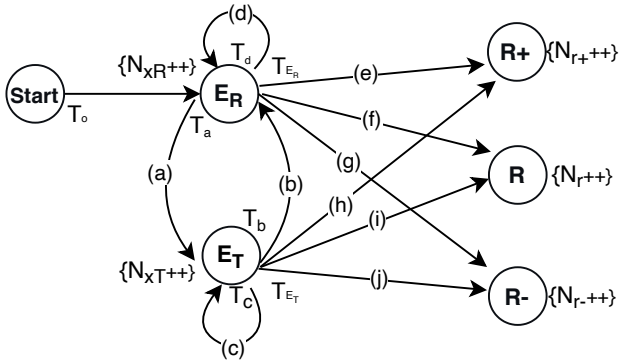

Fig. 6: Global explanations: exploration & exploitation in RL

- $\{T_d\}$ = 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.

D. Results

For this experiment a Lenovo Thinkpad T480 with an Intel i7-8550U CPU at 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 was used. As mentioned, the experiment consisted of a training run over 20 000 iterations for 2 ABSes. This produced 40 000 versions for the runtime model (20 000 for each ABS). 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 is chosen by exploitation (E_T), and a 9.02% (325 out of 3 605) chance that the system keeps 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 of 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, 555 decreased 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.

E. Discussion

For this initial case study, we knew in advance that the ABS SAS changed between two behaviours (exploration and exploitation), and sought to build an event graph that approximated the way in which it changed between the two events. These results can help to prove hypothesis about the systems behaviour. With the global explanation presented, 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 hyper-parameters defined at design time in the RL agent (e.g. in this case, a parameter for exploiting 90% of the time).

The changes in the runtime model used by the ABS SAS were turned into simple events, filtered into complex events representing the state that the system was in, and further grouped into complex events representing the transitions. 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. The case shown above is an example of how event filtering, grouping, and summary capabilities can be used with the runtime models of a SAS to complete an approximated model of its behaviour from a global point of view. The level of detail of the explanation could vary according to the needs of the reader of the explanation. More events could be required, such as for example which events typically follow after $R+$, R and $R-$. The results are encouraging for a first study, but it has required manual intervention: the states and transitions were manually postulated by the users, as well as the hierarchy of events, and the associated Esper EPL queries had to be manually coded.

Further work will involve the gradual lifting of these restrictions by providing additional automated support for populating an event graph from the observed events on the runtime model. A first approximation is to follow a model-driven approach where a designer would model the basic structure of the event graph to be populated, describing the conditions on the runtime model that define each state and (potentially) transition. The approach would then generate the appropriate Esper EPL queries to produce the data needed to complete any randomized transitions, and evaluate how close the template event graph matches the actual system behaviour. Calculating this "closeness of fit" is an area where known approaches in the field of *conformance checking* within process mining could be adapted [43]. In this first step presented in this paper, the user receives support on collecting the data to complete an event graph and evaluate it.

We will also study the automatic derivation of the recurrent states of the system, by considering the most common situations that the runtime models can be in. In this regard, one option would be to study the similarity of the various versions of the runtime model against each other over time, and use this similarity to cluster the models at certain values. In order to manage the difficulty of clustering of complex entities like runtime models, we envision that users would be provided with an approach to specify the key model features to be used for clustering, and the system would do the rest given a certain target number of clusters. Those states could be later used by the previously mentioned modelling environment, giving the users some guidance to start the diagram.

The final step would be the reduction of the need for manual definition of the expected transitions themselves, allowing the approach to produce a first approximation of the most common conditions. In a way, this is similar to the problem of inferring a state machine from a collection of traces: while it is known to be NP-complete, there have been advances recently using SAT solvers that can manage longer traces by operating in an incremental manner [44]. This problem has also been considered in the search-based software engineering space, with approximations inspired on living organisms such as Avida-MDE [45].

V. Related Work

A. Explainable Reinforcement learning

Different approaches have been proposed for explaining AI algorithms. One popular approach is LIME (Local Interpretable Model-agnostic Explanations) [46]. LIME focuses on learning an interpretable model locally around a classifiers' prediction. This work shows its flexibility by explaining different AI techniques. However, it focuses on local explanations to understand a single prediction different from our work that focuses on global explanations. An approach that tackles global explainability is the one presented by Van der Waa et al. in [25]. They propose illustrating how the actions affect the total value of the policy allowing users to ask contrasting questions about why the RL algorithm followed a certain policy instead of an alternate simulated policy. Their approach focuses on simulating what if scenarios to explain the different possible outcomes which can be computationally costly and adds latency which is an advantage of using CEP (low latency). Cashmore et al. also used contrasting questions to design an approach for explainable planning. It allowed developers to see the consequences of forcing a particular action to be taken (rather than the one suggested by the algorithm) [24]: their work mentioned the risks in improperly interpreting the question, and the difficulties in formalising questions about plan structures. Further, the PIRL (Programmatically Interpretable Reinforcement Learning) framework proposed by Verma et al. in [47] focuses on generating interpretable and verifiable agent policies at runtime. For this purpose, the RL algorithm to be explained has to be modified, which contrasts

with our approach of monitoring the system in an post-hoc way using CEP.

B. Runtime Monitoring

In regard to the event-driven and CEP-based approaches for runtime monitoring, Fowler [48] proposed an event sourcing model that facilitates the traceability of the changes over time of the application state as an event sequence. However, event sourcing can be costly in terms of performance [49] since this model tracks every change leading up to a state. The present work describes an event-driven approach integrating CEP to both monitor event streams efficiently, and also deal with scalability problems. Moser et al. [50] used CEP technology to create a flexible monitoring system with support for causal and temporal dependencies between messages for WS-BPEL service composition infrastructures. The proposal by Moser addresses several requirements: unobtrusive platform agnosticism, integration with other systems, multi-process monitoring, and anomaly detection. Wang et al. [35] proposed an staged monitoring approach using CEP to process RFID data by devising RFID application logic into complex events. Additionally, Romano et al. [51] proposed the detection of contract violations through a quality of service (QoS) monitoring approach for cloud computing platforms. This approach integrated Content Based Routing (CBR) with CEP. None of the approaches shown above use CEP to provide explanations in AI. In our most recent work [18], we conducted a feasibility study on the combination of temporal models (TMs) and CEP for software monitoring. In particular, the proposed architecture was able to respond to meaningful events (using CEP) as well as flexibly access relevant linked historical data (using TMs) which motivated the present study. In this previous work, we focused on monitoring the QoS of a runtime model-based SAS while focusing on the system behaviour at specific points, which differs from the proposal in the present paper.

VI. Conclusion and Future Work

Existing approaches for integrating explanatory capabilities into SAS have focused on local explanations, which help stakeholders understand specific decisions. In this paper, we propose a different but complementary research direction, aimed at providing higher-level abstractions to underpin global explanations in order to convey a general understanding of the behaviour of the system. For this purpose, we propose an event-driven monitoring approach for explanation, which is based on the use of runtime models and the timeline of the running system. The approach helps users to understand the system behaviour in a summarised and understandable way based on events of interest. We have shown a proof-of-concept case study related to the training of a RL-based SAS.

In its current version, the approach requires the creation of an *event graph* with manual intervention, where the nodes are specified by a stakeholder as events of interest in the system (e.g., whether the RL algorithm is using exploration or exploitation), and the transitions are stochastic. The transition probabilities are efficiently extracted from the actual running system using Complex Event Processing: the system sends runtime model change events, and event patterns filter and group them into state events and then into state transition events. Further research into this area will focus on reducing the effort involved in creating and populating such an event graph, to improve its expressive power beyond stochastic transitions (i.e., including conditions on the runtime model). The first step will be designing a model-driven approach to describe the general structure of the event graph and the equivalence classes in the state of the runtime model, and produce the Esper EPL queries.

The model-driven approach offers different abstractions from which different research lines can be underpinned. For example, integrating results from the conformance checking field within process mining [43] would allow users to evaluate how well does the event graph representation fit later runs of the system. Further, clustering over selected features of the runtime model could extract the potential nodes of the event graph, by finding the higher-level states that the system is going through. Finally, finding the transitions between the states has similarities with the problem of learning discrete finite automata from traces [44].

In broader terms, we argue that there is potential in extracting approximated models from observed behaviour to explain the general inner workings of systems. Also, there are techniques such as CEP and formalisations that can prove to be helpful. Such approaches will need to balance the accuracy and conciseness of the models. It may have been possible to add more states and transitions to produce a more detailed event graph. However, this would result in a global explanation that may be more difficult to understand. Beyond a certain point, there is the risk of "overfitting" the observed behaviour due to machine learning and data mining practice. Studying this balance is also be part of our future work.

Acknowledgments

This work has been partially sponsored by The Lerverhulme Trust Grant No. RF-2019-548/9 and the EPSRC Research Project Grant No. EP/T017627/1.

References

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

[2] N. Bencomo, S. Götz, and H. Song, "Models@run.time: a guided tour of the state-of-the-art and research challenges," *Software and Systems Modeling*, vol. 18, no. 5, 2019, springer-Verlag.

[3] G. Blair, N. Bencomo, and R. B. France, "Models@ run.time," *Computer*, vol. 42, no. 10, 2009.

[4] P. Sawyer, N. Bencomo, J. Whittle, E. Letier, and A. Finkelstein, "Requirements-Aware Systems: A Research Agenda for RE for Self-adaptive Systems," in *Proceedings of RE'10*, Sep. 2010.

[5] N. Bencomo and A. Belaggoun, "A world full of surprises: bayesian theory of surprise to quantify degrees of uncertainty," *Companion Proceedings of the 36th International Conference on Software Engineering*, 2014.

[6] T. Roth-Berghofer, S. Schulz, D. B. Leake, and D. Bahls, "Explanation-aware computing," *AI Magazine*, 2007.

[7] A. García-Domínguez, N. Bencomo, J. M. Parra-Ullauri, and L. Garcia, "Querying and annotating model histories with time-aware patterns," in *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE, 2019.

[8] 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*, 2020.

[9] 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.

[10] 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.

[11] W. Samek, T. Wiegand, and K.-R. Müller, "Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models," *arXiv preprint arXiv:1708.08296*, 2017.

[12] E. Puiutta and E. M. Veith, "Explainable reinforcement learning: A survey," in *International Cross-Domain Conference for Machine Learning and Knowledge Extraction*. Springer, 2020.

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

[14] E. L. Savage, L. W. Schruben, and E. Yücesan, "On the generality of event-graph models," *INFORMS Journal on Computing*, vol. 17, no. 1, 2005.

[15] M. Atzmueller, S. Bloemheuvel, and B. Kloepper, "A framework for human-centered exploration of complex event log graphs," in *Discovery Science*, P. Kralj Novak, T. Šmuc, and S. Džeroski, Eds. Cham: Springer International Publishing, 2019.

[16] D. Kranzlmüller, *Event graph analysis for debugging massively parallel programs*. na, 2000.

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

[18] 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*, ser. SAC '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3412841.3441923

[19] M. T. Cox, "Metareasoning, monitoring, and self-explanation," *Metareasoning: Thinking about thinking*, 2011.

[20] 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.

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

[22] S. Liu, X. Wang, M. Liu, and J. Zhu, "Towards better analysis of machine learning models: A visual analytics perspective," *Visual Informatics*, vol. 1, no. 1, 2017.

[23] 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*. International Foundation for Autonomous Agents and Multiagent Systems, 2019.

[24] M. Cashmore, A. Collins, B. Krarup, S. Krivic, D. Magazzeni, and D. Smith, "Towards explainable ai planning as a service," *arXiv preprint arXiv:1908.05059*, 2019.

[25] 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.

[26] P. Sequeira and M. Gervasio, "Interestingness elements for explainable reinforcement learning: Understanding agents' capabilities and limitations," *Artificial Intelligence*, vol. 288, 2020.

[27] 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.

[28] Software Freedom Conservancy, "Git," May 2021, date of last access: May 14th, 2021. Archived in http://archive.is/DD6qG. [Online]. Available: https://git-scm.com/

[29] Eclipse Foundation, "CDO Model Repository," date of last access: May 14th, 2021. Archived at http://archive.is/nYpNb. [Online]. Available: http://www.eclipse.org/cdo/

[30] K. Barmpis and D. S. Kolovos, "Towards Scalable Querying of Large-Scale Models," in *Proceedings of ECMFA'14*, 2014.

[31] L. Mouline, A. Benelallam, F. Fouquet, J. Bourcier, and O. Barais, "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*, 2018.

[32] R. Klar, A. Quick, and F. Sötz, "Tools for a Model-driven Instrumentation for Monitoring," in *Proceedings of TOOLS 1991*. Torino, Italy: Elsevier, Feb. 1991.

[33] 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*, vol. 7, 2007.

[34] 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*, vol. 89, 2015.

[35] F. Wang, S. Liu, P. Liu, and Y. Bai, "Bridging physical and virtual worlds: complex event processing for rfid data streams," in *International Conference on Extending Database Technology*. Springer, 2006.

[36] 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.

[37] A. Buss, "Basic event graph modeling," *Simulation News Europe*, vol. 31, no. 1, 2001.

[38] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, and D. Pedreschi, "A survey of methods for explaining black box models," *ACM computing surveys (CSUR)*, vol. 51, no. 5, 2018.

[39] S. Ehmes, L. Fritsche, and K. Altenhofen, "Grapel: Combining graph pattern matching and complex event processing," in *International Conference on Systems Modelling and Management*. Springer, 2020.

[40] 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.

[41] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[42] M. Coggan, "Exploration and exploitation in reinforcement learning," *Research supervised by Prof. Doina Precup, CRA-W DMP Project at McGill University*, 2004.

[43] C. dos Santos Garcia, A. Meincheim, E. R. Faria Junior, M. R. Dallagassa, D. M. V. Sato, D. R. Carvalho, E. A. P. Santos, and E. E. Scalabrin, "Process mining techniques and applications – a systematic mapping study," *Expert Systems with Applications*, vol. 133, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417419303161

[44] F. Avellaneda and A. Petrenko, "Fsm inference from long traces," in *Formal Methods*, K. Havelund, J. Peleska, B. Roscoe, and E. de Vink, Eds. Cham: Springer International Publishing, 2018.

[45] H. J. Goldsby and B. H. C. Cheng, "Automatically generating behavioral models of adaptive systems to address uncertainty," in *Model Driven Engineering Languages and Systems*, K. Czarnecki, I. Ober, J.-M. Bruel, A. Uhl, and M. Völter, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.

[46] M. T. Ribeiro, S. Singh, and C. Guestrin, "" why should i trust you?" explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016.

[47] A. Verma, V. Murali, R. Singh, P. Kohli, and S. Chaudhuri, "Programmatically interpretable reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2018.

[48] M. Fowler, "Event sourcing," 2005-12-12, last accessed on May 14th 2021. Archived at https://archive.is/U6Gsl. [Online]. Available: https://martinfowler.com/eaaDev/EventSourcing.html

[49] M. Overeem, M. Spoor, and S. Jansen, "The dark side of event sourcing: Managing data conversion," in *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2017.

[50] O. Moser, F. Rosenberg, and S. Dustdar, "Event Driven Monitoring for Service Composition Infrastructures," in *Web Information Systems Engineering – WISE 2010*, ser. Lecture Notes in Computer Science, L. Chen, P. Triantafillou, and T. Suel, Eds. Berlin, Heidelberg: Springer, 2010.

[51] L. Romano, D. De Mari, Z. Jerzak, and C. Fetzer, "A Novel Approach to QoS Monitoring in the Cloud," in *2011 First International Conference on Data Compression, Communications and Processing*, 2011.