Runtime Monitoring in Adaptive systems

Avishan Sharafi[1]

Paderborn University, Computer Science Department, Germany

Abstract. By changing an environment, some adverse conditions may happen, which does not allow the system to satisfy its high-level goals anymore. These adverse conditions, called obstacles, can be resolved through countermeasures. Therefore, the system analyst performs obstacle analysis cycles to identify obstacles, assess their likelihood and criticality, and finally propose proper countermeasures to solve them. In this way, the system can be adapted to the new environment, and satisfaction rates of its high-level goals stay higher than target levels. At requirements engineering time, experts estimate the satisfaction rates of probabilistic obstacles and goals. However, these estimations can be inaccurate since environment properties or assumptions might have changed, or some variables might be hard costly to estimate at this time. Thus,monitoring the actual satisfaction rate of obstacles at system run-time helps obtain more information and better estimations. Based on this idea. This paper proposes an obstacle-driven run-time adaptation approach that defers obstacle resolution to system run-time. In the proposed approach, first, an AND/OR goal refinement tree is built to define systems' goals and obstacles. Then, a formal characterization of satisfaction rates of goals and obstacles is provided in terms of observed states and behaviors. Next, by monitoring probabilistic obstacles at run-times, the satisfaction rates of goals are obtained by uppropagation through obstacle/goal refinement tree. Whenever a goal's satisfaction rate decreases below its target level, more appropriate countermeasures are chosen instead of current ones to increase the satisfaction rate of the system's goals above the required degree.

Keywords: Adaptive systems · Models at Run time

1 Introduction

These days, most software systems are deployed in unpredictable environments like disaster management, autonomous vehicles, and unmanned underwater vehicles (UUVs), which are used for oceanic surveillance to monitor pollution levels. These are examples of problem domains where these systems must resolve the uncertainties during operation to guarantee their goals. For achieving a run-time system adaptation, the paper does 4 steps, which are Monitor, Analysis, Plan, and Execute cycles. The Monitor step collects data from the running system, such as performance metrics and configuration characteristics. The Analyze step determines whether a change is required or not. This decision is based on data analysis. The Plan step provides the actions which are needed to apply to meet the system goals.
In the requirement engineering (RE) phase, probabilistic requirements often emerge. Due to the adverse conditions at system runtime, probabilistic requirements may not be satisfied. Generally, they need some properties to be met at least X % of cases. The contribution of this paper focuses on runtime adaptation mechanisms. Using this mechanism, the authors ensure that despite environment changes at runtime, the minimal thresholds required by probabilistic requirements are still met. An obstacle is a precondition that is a barrier to system goal satisfaction. For identifying and overcoming obstacles, the authors used the AND/OR goal refinement graph, called the goal model. Obstacle analysis cycles have 3 steps: (i) obstacles are systematically identified, (ii) the probability and criticality of the identified obstacles are assessed (iii) Obstacles are resolved through countermeasures which will introduce to the system as a new goal in the goal model.

2 Background

In this section, the background knowledge that is necessary for understanding runtime monitoring in adaptive systems is presented.

2.1 Goal-oriented system modeling

A goal is a prescriptive statement of intent to be satisfied by agents forming the system. Goals can be classified in several ways. The presented article (1) focuses on behavioral goals.

A goal model AND/OR graph An AND/OR graph is a graphical representation of the reduction of problems (or goals) to conjunctions and dis-junctions of sub-problems (or sub-goals). Leaf goals are assigned to specific system agents. Parallelograms and hexagons, respectively, represent goals and agents. Figure 1 shows a goal model fragment for a flood detection system (1).
An AND/OR goal refinement tree is built to define systems' goals, and obstacles.

Obstacle in graph An obstacle is a precondition to the non-satisfaction of a corresponding goal. Figure 2 shows an Obstacle model fragment with a countermeasure for a flood detection system (1).

Goal in graph The goal of the system will be satisfied by agents forming the system. Goals can be classified in several ways. The present article focuses on behavioral goals.

Definition 1 (Behavioral goal). *A behavioral goal reflects a system's behavior. The goal is satisfied (or not) by what the system does when it runs.*

A behavioral goal has two types Achieve or Maintain. The specification pattern for Achieve goals is $\Box(C \rightarrow \Diamond T)$, where C and T refer to a current and a target condition, respectively. The specification pattern for Maintain goals is $\Box(C \rightarrow G)$, where G refers to a "good" condition.

Definition 2 (AND/OR graph). *An AND/OR graph is a graphical representation of the reduction of problems (or goals) to conjunctions and disjunctions of subproblems (or subgoals) (8).*

A goal model is an AND/OR graph showing how goals contribute positively or negatively to each other. Leaf goals are assigned to specific system agents. Figure 1 shows a goal model fragment for a flood detection system (6). Goals and agents are represented by parallelograms and hexagons, respectively.

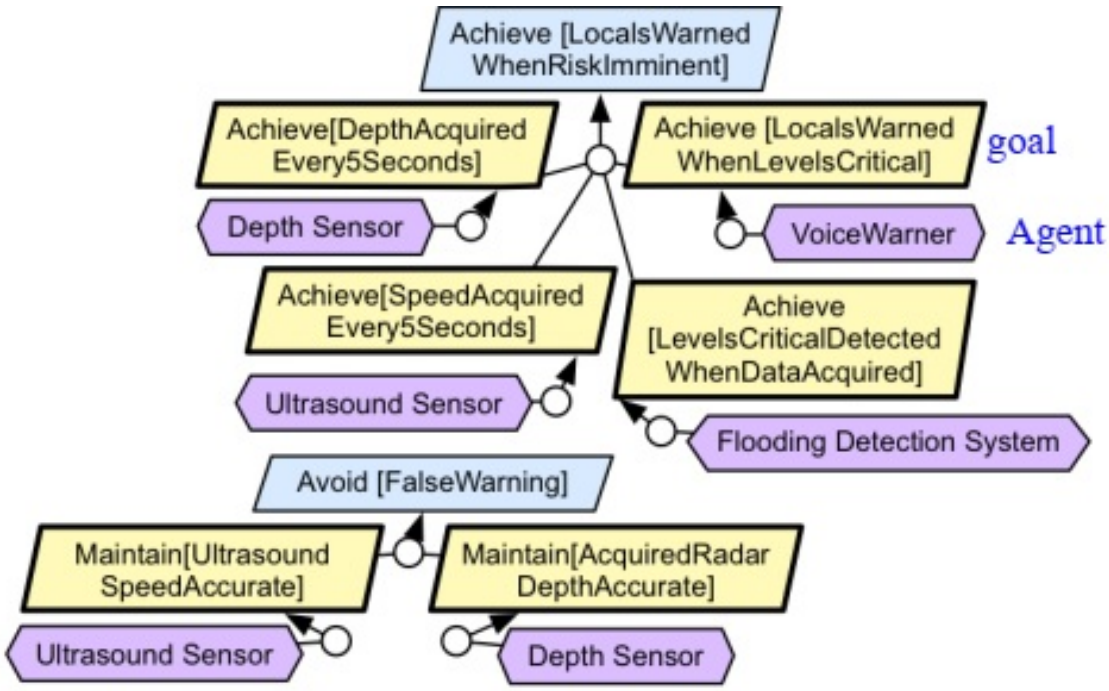

Fig. 1: Goal model fragment for a flood detection system

2.2 Obstacle Analysis

Obstacles are also formalized in MTL. The specification pattern for an obstacle to an Achieve goal is $\Diamond(C \wedge \Box\neg T)$. For an obstacle to a Maintain goal the pattern is $\Diamond(C \wedge \neg T)$.

Obstacles can be resolved by using countermeasures that help to reduce Obstacles' likelihood or consequences. Figure 2 illustrate two obstacle trees connected to their corresponding goals (obstacles are depicted by left parallelograms). Adding countermeasures to obstacles in a goal model increases the completeness of the model. The integration either adds a new goal to the model or replaces the obstructed goal with another one.

2.3 Model Checking

In computer science, model checking is an automatic and exhaustive method for checking whether an infinite behavior of a finite-state mathematical model of a system satisfies some desirable specifications. Model checking is typically associated with hardware or software systems, where the requirements contain liveness requirements (such as avoidance of live-lock) and safety requirements (such as avoidance of states representing a system crash) (3).

Atomic proposition A system model has different variables, like voltage. An atomic proposition p is a statement about the state variable, e.g. p := "voltage > 5". In what follows, AP will denote a set of atomic propositions.

System model: a transition system A Transition System (TS) shows a system's states and relationships. Figure 3 shows a transition system.

Each TS can be shown by a tuple consisting of 6 items, $< S, I, A, \delta, AP, L >$. The elements of this tuple are defined as follows:

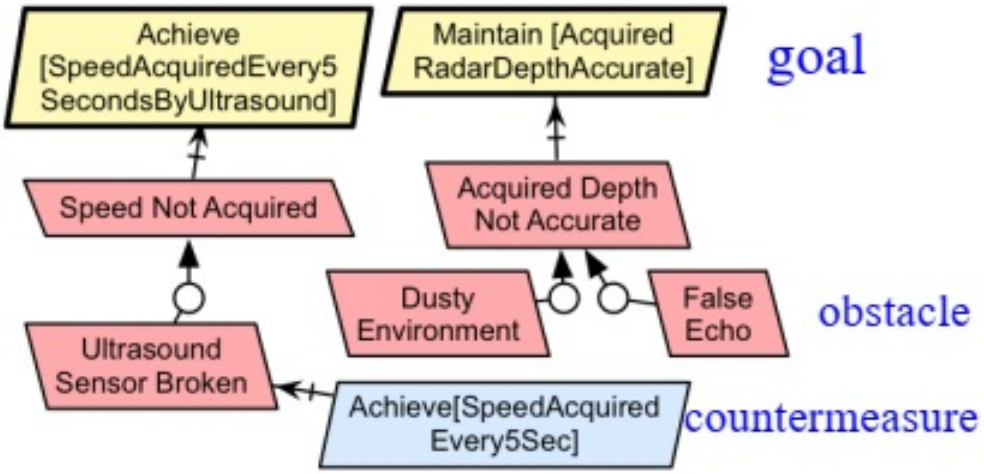

Fig. 2: Obstacle model fragment with a countermeasure

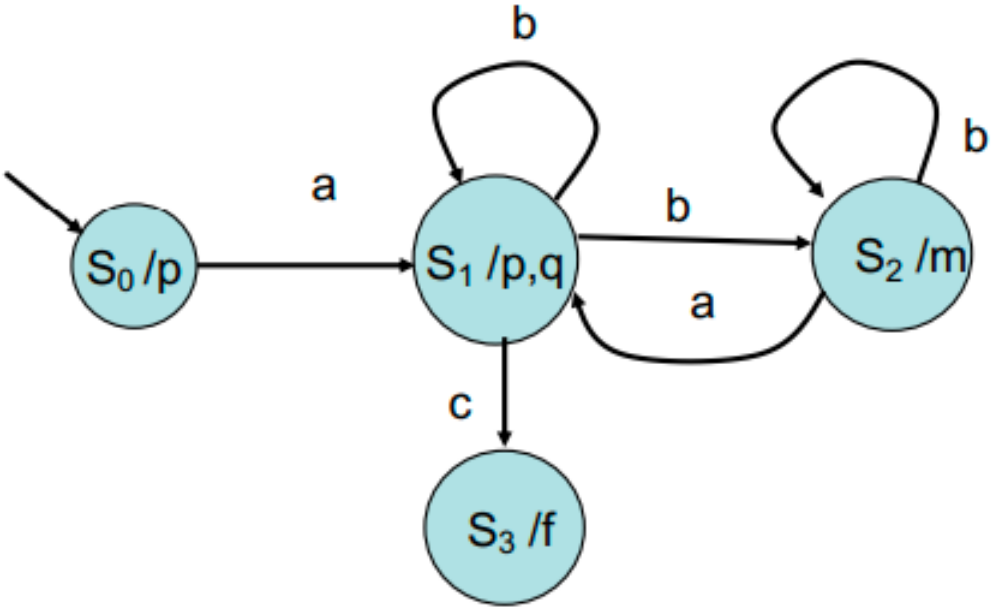

Fig. 3: A transition system

– S is a finite set of states
– $I \subseteq S$ is a set of initial states
– A is a finite set of inputs (or 'actions')
– δ is a transition relation
– AP is a set of atomic propositions on S
– L is a state labeling function. Intuitively, L(s) is the set of atomic propositions satisfied by state s

Definition 3 (Path). *A path is an (infinite) sequence of states in the TS.*

For example,$\delta = S0, S1, S2, S2, S2, S2\ldots$ is a path in the TS depicted in Figure 3.

Definition 4 (Trace). *A trace is the corresponding sequence of labels.*

For instance, $pp, qqqqq$... is the trace corresponding to δ.

Definition 5 (Word). *A word is a sequence of inputs*

For instance, $abbbbbb\ldots$ induces δ.

Linear Temporal Logic Consider Figure 3 which shows 4 stations in a transition system. Let us consider the following specification which we are interested in knowing if the system can satisfy it or not.

Specification: m is always followed by q.

There is some ambiguity about this statement because it does not specify that m is immediately followed by q or m at some point in the future will be followed by q. A new language, Linear Temporal Logic (LTL), is introduced to avoid these kinds of ambiguities. Therefore, LTL is a logic (a 'language') for describing the properties of transition systems. Figure 4 shows the relation between the transition system and Linear Temporal Logic.

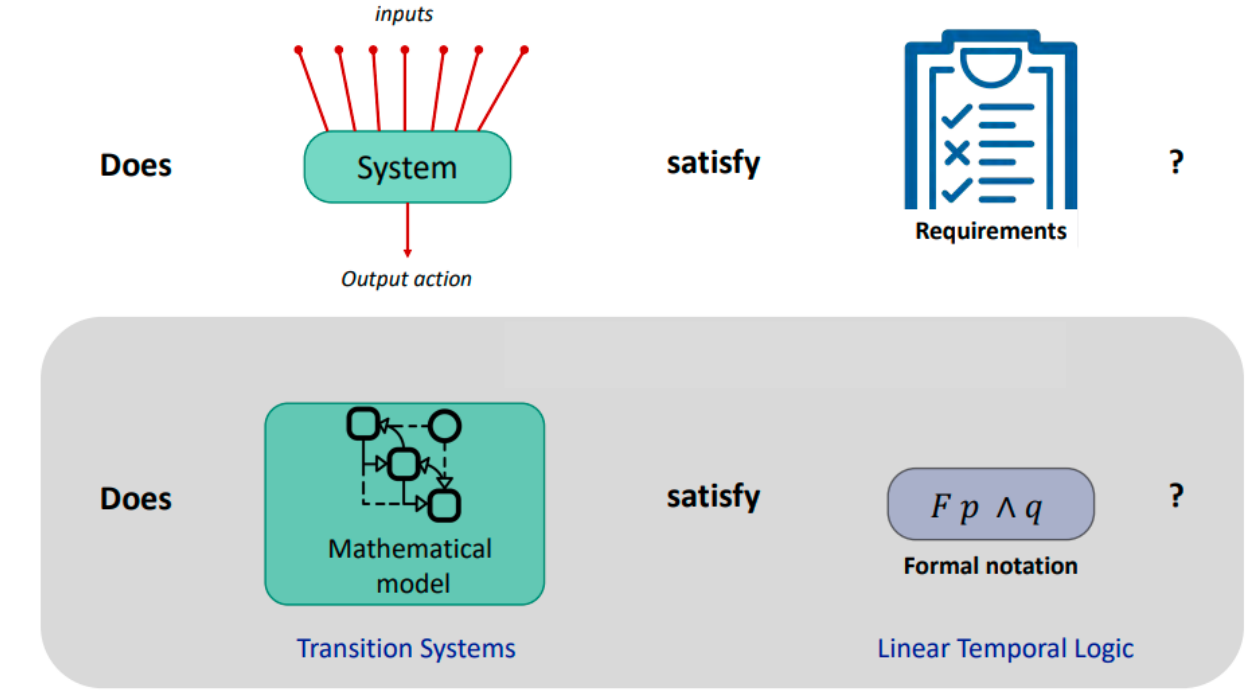

Fig. 4: The relation between transition system and Linear Temporal Logic

Metric temporal logic

Definition 6 (Metric temporal logic). *Metric temporal logic (MTL) is a special case of temporal logic. It is an extension of temporal logic in which temporal operators are replaced by time-constrained versions like until, next, since, and previous operators. It is a linear-time logic that assumes both the interleaving and fictitious-clock abstractions.***??**

A metric linear temporal logic (MTL) is used for formalizing behavioral goals to enable their analysis (6). Table 1 demonstrates some of the temporal operators and standard logical connectives of MTL.

Table 1: Temporal operator and Standard logical connectives of MTL language

Temporal operator	
Symbol	Meaning
$\bigcirc$	In the next state
$\Diamond$	Eventually
$\Diamond_{\leq d}$	Eventually before deadline d
$\Box$	Always in the future
$\Box_{\leq d}$	Always up to deadline d
W	Always in the future unless
Standard logical connectives	
Symbol	Meaning
$\wedge$	and
$\vee$	or
$\neg$	not
$\rightarrow$	implies

2.4 Runtime verification

"Runtime verification is a computing system analysis and execution approach based on extracting information from a running system and using it to detect and possibly react to observed behaviors satisfying or violating certain properties. (4)". In other words, Runtime verification checks whether an execution of a running system satisfies a given property.

2.5 Runtime Verification versus Model Checking

Runtime verification has many similarities with model checking, but there are important differences too as following (5):

(1) In model checking, *all executions* of a given system are examined to answer whether these satisfy a given property. This corresponds to the language inclusion problem. In contrast, runtime verification deals with the word problem which is far lower complexity than the inclusion problem.

(2) Model checking considers infinite traces, while runtime verification deals with finite traces.

(3) In model checking a whole model is given allowing to consider arbitrary positions of a trace, whereas runtime verification considers finite executions of increasing size.

The differences between runtime verification and model checking lead to introduce LTL3, the LTL considered in the article (5), as a linear time temporal logic which shares the syntax with LTL but deviates in its semantics for finite traces.

3 Overview Of the runtime monitoring in adaptive systems

The runtime monitoring in adaptive systems approach comprises 6 steps as follows (5).

(1) At RE time, an AND/OR goal refinement tree is built to define systems' goals, obstacles, and countermeasures. The LTL3 algorithm, which monitors the leaf obstacles, is built.

(2) States of monitor systems will be observed at runtime. The strategy is that a new virtual monitor for each leaf obstacle is started and existing monitors are updated.

(3) The monitored satisfaction rate of leaf obstacles is up propagated through obstacle/goal refinement trees up to high-level goals.

(4) Approach compares the monitored satisfaction rates which are obtained for those goals with their required degree of satisfaction (RDS). If the mentioned ratio falls below the goal's RDS, more appropriate alternative countermeasures will be selected among those which are available to increase the amount of goals' satisfaction rates and make them above of RDS.

(5) The approach updates the goal/obstacle model by integrating the new current countermeasures and updating the propagation in Step 4.

(6) Eventually, the software will be updated automatically according to countermeasures which are selected before .

4 Monitoring Probabilistic Obstacles

As mentioned before, experts estimate the satisfaction rates of probabilistic obstacles and goals at RE time. However, these estimates can be inaccurate since environment properties or assumptions might have changed, or some variables might be hard costly to estimate at this time. Monitoring the actual satisfaction rate of leaf obstacles at system runtime helps filling the gap between actual rates and their estimations.

As we know, probabilistic goals might be satisfied only partially. Then, a precise characterization in terms of observed states and behaviors enables the monitoring of their satisfaction rates.

An Achieve goal $\Box(C \rightarrow \Diamond T)$ requires all possible system states to satisfy $(C \rightarrow \Diamond T)$. Since, it might be possible that the goal be satisfied partially, Then a state s has a probability that the behaviors starting from it satisfy $(C \rightarrow \Diamond T)$.

The state probability of a non-probabilistic formula φ in state s, denoted by $Pr(S, \varphi)$. $Pr(S, \varphi)$ is defined as the ratio of the number of possible behaviors from s satisfying φ over the number of possible behaviors from s. The notation $P^s_{\geq x}(\varphi)$ denotes the statement "the state probability of φ in s is greater than x, that is $Pr(S, \varphi) > X$

The satisfaction rate of an Achieve goal $\Box(C \rightarrow \Diamond T)$ is the lowest state probability of $(C \rightarrow \Diamond T)$, for any possible state s. Note that the \"$\Box$" goal prefix requires a lower bound as we focus on the lowest chance of goal satisfaction.

A goal $\Box(C \rightarrow \Diamond T)$ with satisfaction rate x states that the system satisfies the formula in at least x percent of cases. This may be written as $\Box P_{\geq x}(C \rightarrow \Diamond T)$ where the assertion $\Box P_{\geq x}(\varphi)$ is satisfied by a behavior if all states S along this behavior satisfy $P^{s}_{\geq x}(\varphi)$ The preceding definitions are similar for Maintain goals.

An obstacle $(C \wedge \Box\neg T)$ states that there is one future state at least that satisfies $(C \wedge \Box\neg T)$. A state has a probability that behaviors starting from it satisfy $(C \wedge \Box\neg T)$.

The satisfaction rate of an obstacle $(C \wedge \Box\neg T)$ is the highest state probability of $(C \wedge \Box\neg T)$ for any possible state s. An obstacle $(C \wedge \Box\neg T)$ with satisfaction rate x states that the system satisfies the formula in at most X % of cases.

We mentioned that the satisfaction rate of an obstacle $\Diamond$, φ is the upper bound among the state probabilities of φ therefore, at runtime we count the number of observed behaviors satisfying φ from states s. This number is equivalent to the corresponding state probability. The automate-based monitoring procedure for LTL3 determines at runtime whether φ is satisfied from s.

The monitored satisfaction rate of an obstacle or a goal is its actual satisfaction rate as observed in the running system. For an obstacle, $\Diamond$, φ, the monitored satisfaction rate is determined from the monitored state probabilities. The latter is obtained by monitoring the satisfaction of $\varphi$for all observed states.

4.1 Obstacle Base System Adaptation

A system should be adapted to a new environment at runtime when the current countermeasures cannot guarantee the system's high-level goals are kept above their required degree of satisfaction. Therefore the actual satisfaction rate of these goals at runtime must be determined from the monitored satisfaction rates of leaf obstacles. When the system's high-level goals fall below their RDS, alternative countermeasures are chosen and replace the current ones to maximize the satisfaction rate of these goals.
It is probable that a goal has multiple alternative obstruction sets. An obstruction superset for goal G is the set of all its obstruction sets. For obtaining a goal's monitored satisfaction rate we need to calculate its obstruction superset. We have an obstruction set for a goal that prevents the goal from being satisfied. For achieving this value, we have to up-propagation satisfaction rates through the goal/obstacle model, from the leaf obstacles/goal to the root obstacle/goal.

4.2 Running Development of Most Appropriate Countermeasurments

Our running software will be adapted to changes to be able to match the update goals when suitable countermeasures are integrated and selected in the goal model.An adaptor is a component responsible for adapting the running system.This component is responsible for following current selection of countermeasures.It identifies the countermeasures that must be added or removed.(i) added —that is, not found in the current selection but in the selection of the most appropriate ones; and (ii) removed —that is, no longer in the selection of the most appropriate ones.As we mentioned we have 2 procedure and adding and removing countermeasures. These procedures are used to: add, remove or replace a running component; update configuration parameters; activate hardware components; and so forth.In the adaptation process, the Adaptor calls the activation and deactivation procedures which are related to countermeasures to be able to add and remove them.The activation procedure is the procedure that is responsible for deploying the corresponding countermeasure in the running software system. On the other hand deactivation procedure is the procedure that is responsible for countermeasure removal.

For example, let us consider that the monitored satisfaction rate of the leaf obstacle Ultrasound-Sensor-Broken For5Sec increases, as shown in the dashed line in Figure. 5(1). This increase causes a decrease in the monitored satisfaction rate of the high-level goal Achieve[Locals-Warned-When-Risk-Imminent] (in solid line in Figure. 5) below its RDS. This decrease causes the countermeasure Achieve [Speed-Acquired-Every-5-Sec-By-Camera] to be selected as the most appropriate countermeasure. The activation procedure for Achieve [Speed-Acquired-Every-5-Sec By-Camera] is called and replaces the software component acquiring the speed by the camera-related component. As Figure. 5 shows the satisfaction rate of the high-level goal increases above its RDS after software adaptation (1).

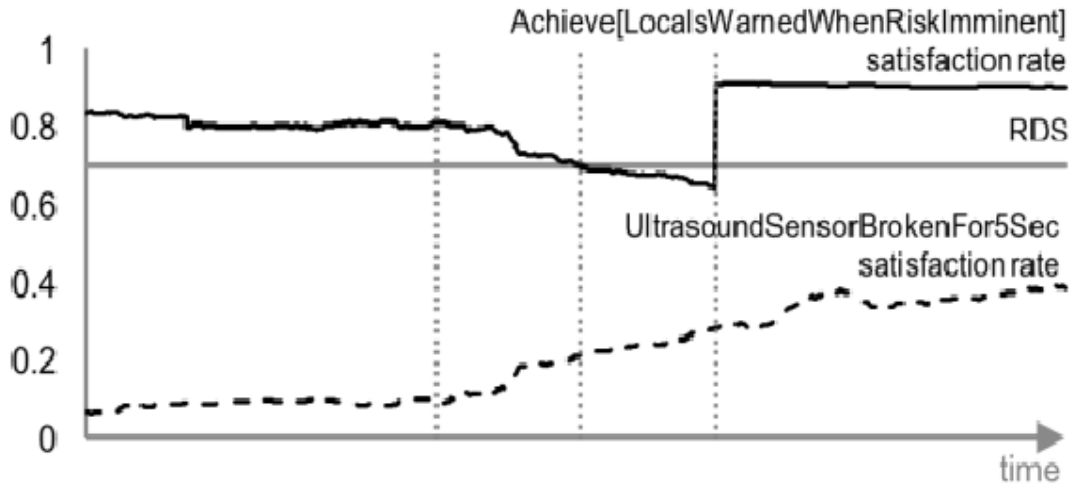

Fig. 5: Adaptation after an increase in the satisfaction rate of obstacle Ultra sound Sensor Broken For 5Sec

5 Conclusion

Modern software systems often operate in very dynamic environments. The dynamics of such systems introduce uncertainties that may be difficult or even impossible to anticipate before deployment. Hence, these systems need to resolve the uncertainties during operation. There are some adverse conditions which make barriers that prevent software systems to reach their goals. These adverse conditions, called obstacles, can be resolved through countermeasures. Therefore, Obstacle analysis is a goal-oriented form of risk analysis for requirements engineering (RE). Also, at requirements engineering time, experts estimate the satisfaction rates of probabilistic obstacles and goals. However, these estimates can be inaccurate since environment properties or assumptions might have changed, or some variables might be hard and costly to estimate at this time. Thus, monitoring the actual satisfaction rate of obstacles at system run-time helps obtain more information and better estimations. Based on meeting the system's goals under changing conditions, the research proposes to defer obstacle resolution to system run time. For monitoring the goal/obstacle satisfaction rate of a system, there are some steps that should be done, like monitoring, Analysis, Plan, and Execute cycles. The paper's contribution helps a) monitoring the satisfaction rate of probabilistic obstacles, b) establishing the severity of the obstacles' consequences, and c) by considering cost constraints, dynamically shifting to alternative countermeasures which helps to better meet the required satisfaction rate.

6 Research Evaluation

First, in the paper, the authors have not clearly mentioned how they calculated appropriate countermeasures if they used some sort of statistical calculations or not. They just mentioned that the complexity of this naïve approach is $O(2^n)$ where $O(2^n)$ is the number of countermeasures. Second, there are some ambiguities in the literature of the paper, which cause difficulties in understanding it. Third, although, in the beginning, they mentioned their approach is evaluated on fragments of an ambulance dispatching system, in the introduction, they refer to the "goal model fragment for a flood detection system" in figure 1, which they did not provide any result of flood detection. It would be nice if they could provide some sort of result for the "flood detection system". Moreover, they do not consider that each countermeasure has its own cost. Therefore to select the best countermeasures, considering the cost of choosing countermeasures could be a nice factor too.

Bibliography

[1] Cailliau, A, Lamsweerde, A: run-time monitoring and resolution of probabilistic obstacles to system goals. ACM Transactions on Autonomous and Adaptive Systems **14**(1), 1–40 (2019)

[2] Van Der Donckt, M Jeroen and Weyns, Danny and Iftikhar, M Usman and Singh, Ritesh Kumar:Cost-Benefit Analysis at Runtime for Self-adaptive Systems Applied to an Internet of Things Application,ENASE,478–490,2018

[3] Wikipedia Homepage, https://en.wikipedia.org. Last accessed 16 June 2022

[4] Wikipedia Homepage, https://en.wikipedia.org/wiki/Runtime_verification. Last accessed 17 June 2022

[5] Bauer, A, Leucker, Schallhart, C: Runtime verification for LTL and TLTL. AACM Transactions on Software Engineering and Methodology (TOSEM) **20**(4), 1–64 (2011)

[6] Van Lamsweerde, A: Requirements engineering: From system goals to UML models to software. tenth edn. Chichester, UK: John Wiley & Sons (2009)

[7] Wikipedia Homepage, https://en.wikipedia.org/wiki/Andor_tree. Last accessed 17 June 2022

[8] Wikipedia Homepage, https://en.wikipedia.org/wiki/Metric_temporal_logic. Last accessed 17 June 2022