Modeling Self-adaptation - A Possible Endeavour?

Emil Vassev(✉)

Lero–The Irish Software Research Centre, University of Limerick, Limerick, Ireland
emil.vassev@lero.ie

Abstract. Self-adaptive systems have the capability to autonomously modify their behavior at runtime in response to changes in their internal structure or execution environment. Therefore, often self-adaptation emerges as a means to solve problems related to performance or security, to increase efficiency, or to react to various hazards. Basically, self-adaptation may emerge to solve a whole spectrum of various problems or hazards occurring in the execution environment, which implies that behavior modeling for self-adaptation requires intrinsic knowledge of the system context.

A new approach to modeling self-adaptation compliant to system goals is presented in this paper. In this approach, KnowLang, a knowledge representation language for self-adaptive systems, is used to model self-adaptive behavior. Special KnowLang policies are at the core of this approach. Ideally, KnowLang policies are specified to handle specific situations by pursuing a specific goal. A policy exhibits a behavior via actions generated in the environment or in the system itself. Specific probabilistic beliefs and generic conditions determine what specific actions shall be executed. Context properties are intrinsically embedded in the self-adaptive behavior, which makes that behavior context-reactive. To demonstrate the novelty of this approach, the paper elaborates on a self-adaptive behavior of an autonomous vehicle modeled with KnowLang.

Keywords: Self-adaptation · Autonomy · Knowledge representation
KnowLang

1 Introduction to Self-adaptation

The idea of a system that evolves and autonomously finds a solution to a problem, or suggests new ways to solve a problem, is both visionary and very challenging. Without any doubt, the term "adaptive" identifies one of the most challenging topics we currently explore in technology. It identifies systems with the property of being able to autonomously react to situations occurring during its lifetime. The question arises as to whether such a behavior is feasible, implementable, or even desirable. Note that self-adaptive systems must be aware of their physical environment and whereabouts, as well as of their current internal status. This ability enables software intensive systems to sense, draw inferences, and react by exhibiting self-adaptation.

A common understanding about the process of self-adaptation is the ability of a system to autonomously monitor its behavior and eventually modify the same according to changes in the operational environment, or in the system itself. The paradigm requires

P. Cong Vinh et al. (Eds.): ICCASA 2017/ICTCC 2017, LNICST 217, pp. 60–68, 2018.
https://doi.org/10.1007/978-3-319-77818-1_6

that the system engages in various interactions where important structural and dynamic aspects of the environment are perceived. Therefore, it is of major importance for a self-adaptive system to be able acquire and structure comprehensive knowledge in such a way that it can be effectively and efficiently processed, so such a system becomes aware of itself and its environment.

One of the biggest concerns related to self-adaptive systems is how to prove that the autonomous self-adaptive behavior will not cause more safety hazards. Autonomous cars have already appeared on our streets and unfortunately due to some severe accidents they appear to be not as secure as we had hoped them to be. This paper tackles the question of achieving self-adaptive behavior through knowledge representation and awareness reasoning, at least in a certain context, with maximized safety guarantees that will help us to establish trust in autonomous systems.

2 Correct Self-adaptation

Self-adaptation is often related to non-determinism and thus, its correctness proof, if even possible, is a tedious task. As we have discussed before, 100% safety is not possible [1]. A possible approach could be to work on *probabilistic guarantees* through probabilistic model checking, which is considered a powerful technique for formally verifying quantitative properties of systems that exhibit stochastic behavior [2]. Probabilistic behavior may arise, for example, due to failures of unreliable components, a dynamic environment, etc. The problem is that non-determinism leads to *unforeseen behavior*, which basically, cannot be model-checked. It can be eventually simulated (to some extent) through a random generation of the simulated conditions and verified via testing.

Lero, the Irish Software Research Center is currently tackling a project where a special test bed (Test Bed for Adaptive Systems, or TBAS) is under R&D. TBAS targets testing of self-adaptive systems under simulated conditions in both virtual and physical testing environments. With TBAS we shall be able to efficiently test adaptive behavior by validating self-* objectives through evaluation of the system's ability to perceive both the internal and external environments and react to changes. With TBAS, we target the evaluation of features that manifest the system's awareness about situations and conditions, and the system's ability to self-adapt to those situations and conditions when adaptation is required. The foundation of TBAS is the KnowLang Framework [3] and based on this we are developing two test platforms:

(1) A fully virtual simulation environment (virtual TBAS) where multiple virtual adaptive entities (VAEs) can be tested both individually or/and as an "intelligent swarm" whereby VAEs interact not only with the environment, but also internally. Each VAE incorporates a KnowLang Reasoner along with a knowledge base (KB) operated by that reasoner. The virtual TBAS runs as a transparent distributed system comprising multiple interconnected machines, each capable of running hundreds of VAEs.
(2) A test platform based on WiFi/Bluetooth-communicating, mini-computerized and robotized platforms, yet capable of running a fully-functional VAE. Such a robotized platform (Lero Robotics Platform - LRP) is autonomously controlled by the

hosted VAE and is equipped with a GPS and a variety of plug-in sensors such as: light detectors, microphones, smoke detectors, motion detectors, humidity detectors, high-speed thermometer, barometer, etc. LRP is to be designed to be pluggable into different motion platforms (wheels-based, propeller-based, and water-motion-based) and actuator platforms (e.g., mechanical arms).

3 Modeling Self-adaptation with KnowLang

KnowLang employs special knowledge structures and a reasoning mechanism for modeling autonomic self-adaptive behavior [4]. Such a behavior can be expressed via KnowLang policies, events, actions, situations and relations between policies and situations (see Eqs. 1 through 10). Policies (Π) are at the core of autonomic behavior. A policy π has a goal (g), policy situations (Si_π), policy-situation relations (R_π), and policy conditions (N_π) mapped to policy actions (A_π) where the evaluation of N_π may eventually (with some degree of probability) imply the evaluation of actions (denoted $N_\pi \overset{[Z]}{\rightarrow} A_\pi$) (see Eq. 6). A condition (n) is a Boolean expression over an ontology (see Eq. 2), e.g., the occurrence of a certain event. Policy situations Si_π are situations (see Eq. 7) that may trigger (or imply) a policy π, in compliance with the policy-situations relations R_π (denoted by $Si_\pi \overset{[R_\pi]}{\rightarrow} \pi$), thus implying the evaluation of the policy conditions N_π (denoted by $\pi \rightarrow N_\pi$) (see Eq. 6). Therefore, the optional policy-situation relations (R_π) justify the relationships between a policy and the associated situations (see Eq. 10).

$$\begin{aligned} &\Pi := \{\pi_0, \pi_1, \ldots, \pi_m\}, m \geq 0 \quad (policies) \\ &A_\pi \subset A \qquad (A_\pi - policy\ actions; A - the\ set\ of\ all\ actions) \\ &Si_\pi \subset Si \qquad (Si_\pi - policy\ situations) \\ &R_\pi \subset R \qquad (R_\pi - policy - situation\ relations) \end{aligned} \tag{1}$$

$$n := be(O)\ (Boolean\ expression\ over\ ontology) \tag{2}$$

$$N_\pi := \{n_0, n_1, \ldots, n_k\}, k \geq 0\ (policy\ conditions) \tag{3}$$

$$s := be(O)\ (state) \tag{4}$$

$$g := \langle \Rightarrow s' \rangle | s \langle \Rightarrow s' \rangle\ (goal) \tag{5}$$

$$\begin{aligned} &\pi := < g, Si_\pi, [R_\pi], N_\pi, A_\pi, map(N_\pi, A_\pi, [Z]) > \quad (policy) \\ &N_\pi \overset{[Z]}{\rightarrow} A_\pi \qquad (N_\pi implies\ the\ evaluation\ of\ actions\ A_\pi) \\ &Si_\pi \overset{[R_\pi]}{\rightarrow} \pi \rightarrow N_\pi \qquad (Si_\pi\ trigger\ \pi) \end{aligned} \tag{6}$$

$$Si := \{si_0, si_1, \ldots, si_n\}, n \geq 0\ (situations) \tag{7}$$

$$
\begin{aligned}
&si: =< s, A_{si}^{\leftarrow}, \left[E_{si}^{\leftarrow}\right], A_{si} > \quad (situation)\\
&A_{si}^{\leftarrow} \subset A^{*} \quad \left(A_{si}^{\leftarrow} - executed\ actions\right)\\
&\qquad\qquad (A^{*} - the\ set\ of\ all\ finite\ sequences\ with\ elements\ in\ A)\\
&A_{si} \subset A \quad \left(A_{si} - posible\ actions\right)\\
&E_{si}^{\leftarrow} \subset E^{*} \quad \left(E_{si}^{\leftarrow} - situation\ events\right)\\
&\qquad\qquad (E^{*} - the\ set\ of\ all\ finite\ sequences\ with\ elements\ in\ E)
\end{aligned} \tag{8}
$$

$$
R: = \left\{r_0, r_1, \ldots., r_n\right\}, n \geq 0 \ (relations) \tag{9}
$$

$$
\begin{aligned}
&r: =< \pi, [rn], [Z], si > \quad (relation)\\
&si \in Si,\ \pi \in \Pi,\ si \overset{[Z]}{\rightarrow} \pi
\end{aligned} \tag{10}
$$

Note that in order to allow for self-adaptive behavior, relations must be specified to connect policies with situations over an optional probability distribution (Z) where a policy might be related to multiple situations and vice versa. Probability distribution (Z) is provided to support probabilistic reasoning and to help the reasoner to choose the most probable situation-policy "pair". Thus, we may specify a few relations connecting a specific situation to different policies to be undertaken when the system is in that particular situation and the probability distribution over these relations (involving the same situation) should help the reasoner decide which policy to choose (denoted by $si \overset{[Z]}{\rightarrow} \pi$) (see Eq. 10). Hence, the presence of probabilistic beliefs (Z) in both mappings and policy relations justifies the probability of policy execution, which may vary with time.

Ideally, KnowLang policies are specified to handle specific situations, which may trigger the application of policies. A policy exhibits a behavior via actions generated in the environment or in the system itself. Specific conditions determine which specific actions (among the actions associated with that policy (see Eq. 6) shall be executed. These conditions are often generic and may differ from the situations triggering the policy. Thus, the behavior not only depends on the specific situations a policy is specified to handle, but also depends on additional conditions. Such conditions might be organized in a way allowing for synchronization of different situations on the same policy. When a policy is applied, it checks what particular conditions N_{π} are met and performs the mapped actions A_{π} ($map\left(N_{\pi}, A_{\pi}, [Z]\right)$) (see Eq. 6). An optional probability distribution Z may additionally restrict the action execution. Although specified initially, the probability distribution at both mapping and relation levels is recomputed after the execution of any involved action. The re-computation is based on the consequences of the action execution, which allows for reinforcement learning.

4 Case Study

Obviously, self-adaptive and autonomous systems that replace human beings should be carefully designed in concern with safety risks stemming from the autonomous behavior.

For example, autonomous vehicles should be designed towards maximizing the safety guarantee that no pedestrian would ever be injured while operating in autonomous mode.

4.1 Modeling Self-adaptation that Adds on Safety

In this case study, we used KnowLang to model and specify self-adaptive behavior that adds on that safety guarantee. To do so, we determined multiple situations that can be considered critical because they involve an autonomous vehicle in close proximity to pedestrians. For example, such situations are "approaching a crosswalk", "passing through a school zone", "crossing uncontrolled intersection", "approaching a failed traffic light", "passing a stopped vehicle", "approaching a car accident", etc. The following is an example of specifying with KnowLang the "approaching a crosswalk" situation (see Fig. 1).

```
CONCEPT_SITUATION ApproachingCrosswalk {
....
  SPEC {
    SITUATION_STATES
{eMobility.eCars.CONCEPT_TREES.Route.STATES.InCloseDistanceToCrosswalk}
    SITUATION_ACTIONS  {
      eMobility.eCars.CONCEPT_TREES.SlowDown,
      eMobility.eCars.CONCEPT_TREES.StopCar,
      eMobility.eCars.CONCEPT_TREES.UseEngineBreaking,
      eMobility.eCars.CONCEPT_TREES.DenySpeeding,
      eMobility.eCars.CONCEPT_TREES.TurnSteeringWheelRight,
      ....
    }
    ....
  }
}
```

The sample above is a simple specification where for clarity some details are missing, but the reader can conclude that the specified situation ApproachingCrosswalk is determined by the InCloseDistanceToCrosswalk state (specified somewhere else in the specification model) and by actions that can be undertaken once the autonomous vehicle has ended up in this situation (see Eq. 8 in Sect. 3). Note that the InCloseDistanceToCrosswalk state is specified as a Boolean expression (not shown here) that determines if the car enters a section of the followed route where a crosswalk is in close proximity.

In the next step, we formalized self-adaptive behavior in the form of policies (Π) (see Eq. 1 in Sect. 3) driving the autonomous vehicles in this situation. For example, we specified a policy that handles this situation in various conditions emphasizing damages or malfunction of the driving system, e.g., flat tires, malfunctioning steering wheel, malfunctioning brakes, etc. Hence, one possible self-adaptive behavior that emerged from this exercise as adding on car safety can be described as "*automatically deny car*

speeding, turning steering wheel right and stopping the car in the case of flat tire when the car is getting in close proximity to a crosswalk".

```
CONCEPT_POLICY SafeDriveAroundCrosswalk {
....
  SPEC {
    POLICY_GOAL {eMobility.eCars.CONCEPT_TREES.SafeCrosswalkPassing}
    POLICY_SITUATIONS
{eMobility.eCars.CONCEPT_TREES.ApproachingCrosswalk}
    POLICY_RELATIONS {eMobility.eCars.RELATIONS.Situation_Policy_1}
    POLICY_ACTIONS {
      eMobility.eCars.CONCEPT_TREES.SlowDown,
      eMobility.eCars.CONCEPT_TREES.StopCar,
      eMobility.eCars.CONCEPT_TREES.UseEngineBreaking,
      eMobility.eCars.CONCEPT_TREES.DenySpeeding,
      eMobility.eCars.CONCEPT_TREES.TurnSteeringWheelRight,
      ....
    }
    POLICY_MAPPINGS {
      MAPPING {
        CONDITIONS {eMobility.eCars.CONCEPT_TREES.Vehicle.STATES.FlatTire}
        DO_ACTIONS {
eMobility.eCars.CONCEPT_TREES.Vehicle.FUNCS.CarDenySpeeding,
eMobility.eCars.CONCEPT_TREES.Vehicle.FUNCS.CarTurnSteeringWheelRight,
eMobility.eCars.CONCEPT_TREES.Vehicle.FUNCS.CarStopCar }
        PROBABILITY {0.5}
      }
      MAPPING {
        CONDITIONS {eMobility.eCars.CONCEPT_TREES.Vehicle.STATES.FlatTire}
        DO_ACTIONS {
eMobility.eCars.CONCEPT_TREES.Vehicle.FUNCS.CarStopCar,
eMobility.eCars.CONCEPT_TREES.Vehicle.FUNCS.CarDenySpeeding,
eMobility.eCars.CONCEPT_TREES.Vehicle.FUNCS.CarTurnSteeringWheelRight }
        PROBABILITY {0.3}
      }
      MAPPING {
        CONDITIONS {eMobility.eCars.CONCEPT_TREES.Vehicle.STATES.FlatTire}
        DO_ACTIONS {
eMobility.eCars.CONCEPT_TREES.Vehicle.FUNCS.CarTurnSteeringWheelRight,
eMobility.eCars.CONCEPT_TREES.Vehicle.FUNCS.CarStopCar,
eMobility.eCars.CONCEPT_TREES.Vehicle.FUNCS.CarDenySpeeding }
        PROBABILITY {0.2}
      }
....
    }
....
  }
}
```

Fig. 1. Autonomous vehicles approaching a crosswalk.

As specified, the probability distribution gives an initial designer's preference about what actions should be executed if the system ends up in running the SafeDriveAroundCrosswalk policy. Note that at runtime, the KnowLang reasoner maintains a record of all the action executions and re-computes the probability rates every time when a policy has been applied and consecutively, actions have been executed. Thus, although the system will initially execute the sequence of actions CarDenySpeeding, CarTurnSteeringWheelRight, and CarStopCar (it has the higher probability rate of 0.5), if that policy cannot achieve satisfactory its SafeCrosswalkPassing goal with this sequence of actions, then the probability distribution will be shifted in favor of another sequence, which might be executed the next time when the system will try to apply the same policy. Therefore, probabilities are recomputed after every action execution, and thus the behavior changes accordingly.

4.2 Testing the Self-adaptation Model

As part of this exercise, tests were performed with the Lero's virtual TBAS (see Sect. 2) to simulate awareness emerging when the KnowLang Reasoner operates over the specified model for self-adaptation. The exercise was performed with the initial version of TBAS where a special host application ran the KnowLang Reasoner and the communication to it went through a command line where the reasoner was fed with simulated conditions. Note that the KnowLang Reasoner iterates over an *awareness control loop* [5] where all the states expressed in KnowLang are evaluated at any loop iteration, which leads to re-evaluation of all the goals and situations expressed with states. Therefore,

when a situation is determined through the evaluation of its states, eventually, a policy will be applied to tackle this situation.

In this test, we simulated the ApproachingCrosswalk situation along with the car's FlatTire state (see Sect. 4.1). Further, by providing feedback to the reasoner from the actions' realization, we enforced self-adaptation by switching the sequence of actions due to the probability re-distribution caused by this feedback. Therefore, the reasoner attempted to find the safest sequence of actions in case of flat tire. More simulated conditions, e.g., rain, snow, ice, etc., helped to determine that the safest sequence of actions is <CarTurnSteeringWheelRight, CarStopCar, CarDenySpeeding> (see Sect. 4.1). Note that this sequence was initially granted with the lowest probability rate of 0.2 (out of 1.0), but during the simulation process the reinforcement learning made it the one that is to be selected first in case of a flat tire.

5 Conclusion

An autonomous vehicle is loaded with AI and operates in a potentially nondeterministic environment. This lack of determinism and certainty is additionally extended by requirements, business conditions, and available technology. Therefore, if we want to construct reliable autonomous vehicles, we need to plan for uncertainty by capturing the autonomous and self-adaptive behavior exhibited while operating in the nondeterministic environment. Failure to do so may result in systems that are overly rigid for their purpose, an eventuality unsafe in their autonomy and adaptation.

This paper has presented the authors' experience with the KnowLang framework mastered to capture self-adaptation for autonomous smart vehicles. The approach demonstrated that self-adaptation needs to be properly handled and formalized, so it can be processed by contemporary formal verification techniques. For example, simulation is one possible method that can be helpful in verifying safety properties, via the formalization of non-desirable system states along with the formalization of behavior that will never lead the system to these states. The paper has presented simulation and testing with a KnowLang-based test bed.

Acknowledgements. This work was supported with the financial support of the Science Foundation Ireland grant 13/RC/2094 and co-funded under the European Regional Development Fund through the Southern & Eastern Regional Operational Programme to Lero - the Irish Software Research Centre (www.lero.ie).

References

1. Vassev, E.: Safe artificial intelligence and formal methods. In: Margaria, T., Steffen, B. (eds.) ISoLA 2016. LNCS, vol. 9952, pp. 704–713. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47166-2_49
2. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press, Cambridge (2008)
3. Vassev, E., Hinchey, M.: KnowLang: knowledge representation for self-adaptive systems. IEEE Comput. **48**(2), 81–84 (2015)

4. Vassev, E., Hinchey, M.: Knowledge representation for adaptive and self-aware systems. In: Wirsing, M., Hölzl, M., Koch, N., Mayer, P. (eds.) Software Engineering for Collective Autonomic Systems. LNCS, vol. 8998, pp. 221–247. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-16310-9_6
5. Vassev, E., Hinchey, M.: Awareness in software-intensive systems. IEEE Comput. **45**(12), 84–87 (2012)