Towards Runtime Monitoring for malicious behaviors detection in Smart Ecosystems

Emilia Cioroaica*, Felicita Di Giandomenico†,
Thomas Kuhn *, Francesca Lonetti†, Eda Marchetti†, Jasmin Jahic*, Frank Schnicke*
* *Fraunhofer IESE*, Kaiserslautern, Germany
† *ISTI CNR* , Pisa, Italy
*{emilia.cioroaica, thomas.kuhn, jasmin.jahic, frank.schnicke}@iese.fraunhofer.de
†{felicita.digiandomenico, francesca.lonetti, eda.marchetti}@isti.cnr.it

***Abstract*—A Smart Ecosystem reflects in the control decisions of entities of different nature, especially of its software components. Particularly, the malicious behavior requires a more accurate attention. This paper discusses the challenges related to the evaluation of software smart agents and proposes a first solution leveraging the monitoring facilities for *a) assuring conformity* between the software agent and its digital twin in a real-time evaluation and *b) validating decisions* of the digital twins during runtime in a predictive simulation.**

***Index Terms*—Smart Ecosystems, Virtual Evaluation, Monitoring, Malicious Behaviors Detection, Digital Twins**

I. Introduction

Smart Ecosystems (SES) are the next major concept that enable adaptation of intelligent systems we use daily. When systems are part of a SES, they become much more dynamic as they get quick downloads of new software components and runtime updates of existing functionalities. Originally, SES extend the concept of System Of Systems (SoS) [1], because they aggregate not only systems and system components but also actors such as developers, organizations and users [2]. SES aggregate multiple parties that dynamically integrate into groups or migrate between groups in order to achieve common tactical goals that lead to fulfillment of higher level strategic ones. For example, in case of highway ecosystems, through the downloads of smart software agents, autonomous vehicles can start cooperation and drive together in vehicle platoons (tactical goal) that enable them to reduce fuel consumption (strategic goal). In an ecosystem, cooperation of systems is possible through involvement of actors, including developers that provide software updates during runtime.

Within an ecosystem, software and hardware components are provided by different actors that may have not only collaborative goals, but undeclared competitive goals as well. In this setting, malicious behavior may be injected to harm ecosystem participants. Even without access to privileged system functions and instructions, malicious code may be seriously affecting system performance and behavior.

In literature, there are several examples of how discovered vulnerabilities of hardware resources can be specifically exploited with malicious intentions. Indeed, Seaborn and Dullien [3] have shown that specially designed software that executes on hardware with known vulnerabilities can lead to serious system failures that manifest into security threats. Additionally, the dynamic random access memory (DRAM) row hammer effect can be repeatedly induced by software exploiting the hardware structure of DRAM modules. In this case, repetitive and specially crafted memory patterns can cause the memory cells to leak their charges and to interact electrically among each other [4].

Semantic attacks through which software components can show faulty behavior on purpose are further examples. For instance, malicious behavior of a downloaded software smart agent can, occasionally refuse to detect pedestrians at a crossing while controlling an autonomous vehicle and put citizens into dangerous situations. In this case, such a behavior can be caused by *logic bombs* [5] that remain dormant in the host system for a certain amount of time. Logic bombs will be activated when an event happens or certain conditions are met. This makes malicious behavior difficult to detect during admission of smart agents into a smart ecosystem as it may be triggered by events of software components deployed on the system at a latter point in time.

From these examples it is evident that ahead of time evaluation of software components that join an ecosystem is not sufficient. A novel approach we have introduced in [6] proposes evaluation of a software smart agent during runtime without executing its behavior that may contain malicious code, but by executing it's digital twin (DT) in a simulated environment. Digital twins are downloaded on the systems together with the smart agents and are executable abstractions of the software components fed with real-time data. Evaluation of the DT's behavior is performed in *linked predictive simulation* in interaction with abstraction models of hardware components that enables execution of the digital twin behavior faster than the wall clock. If the digital twin shows a correct behavior in a simulated environment, trust in the smart agent is build over time. If the behavior of the digital twin is considered not trustworthy, then execution of the smart agent on real ECU (Embedded Control Unit) is stopped and a fail-over behavior is executed instead. Because the trustworthiness of the smart agent is judged in advance by executing the behavior of its digital twin in a simulated environment, conformity between the digital twin's behavior and smart agent behavior needs to be assured. One way of assuring conformity is through monitoring of their behavioral executions.

The main objective of this paper is to assess correctness of the run time behavioral execution of software components. This is achieved by monitoring the behavioral conformity between a real time execution of a smart agent and its digital twins execution in a simulated environment. In the context of linked predictive simulation, we consider the valid behavior of the digital twin being the contract against which the behavior of the smart agent is assessed. Therefore, we introduce the concept of runtime monitoring for malicious behaviour detection in smart ecosystems. Specifically, the behavior of the digital twin becomes the specification against which the behavior of the system is validated. Our solution represents a first attempt of using monitoring data coming from combined simulated and real world software executions to promptly detect malicious behaviors.

In what follows: Section II presents an overview of the main challenges when monitoring smart agents in smart ecosystems; Section III presents a first attempt to provide an architecture for mitigating malicious behavior during runtime through the use of digital twins; Section IV presents the related work and Section V summarizes the work in this paper and presents on-going and future work.

II. Monitoring Challenges

When smart agents are deployed on safety critical systems, such as autonomous vehicles, their malicious behaviours need to be promptly detected. Particularly, it is important to assure confidence in the smart agent's behavior before it is admitted to control critical components of the ecosystem. In traditional embedded systems, assurance of correct behavior at-design-time is usually performed through rigorous testing, verification and validation activities in order to prevent software component failures under specified operational conditions [7].

In SES, smart agents bind to embedded systems of host systems and dynamically interact with other components. This raises manifold challenges for the monitoring system that keeps track of the agents behavior and their trustworthiness:

1) *Unknown interaction with the Hardware Platform.* When smart agents interact with the hardware for the first time, different timing behavior between the hardware platform and the smart software agent may yield different results. This can lead to unexpected behavior of the software smart agent.
2) *Integration of third party systems or components.* An autonomous system is an open system, characterized by runtime integration of third party systems or system components. This aspect may affect the execution of a smart agent that needs to interact with other software applications, some being dynamically loaded smart agents as well.
3) *Changing environmental conditions.* Smart ecosystems realize autonomous systems that adapt to changing environmental conditions. A smart software agent therefore must perform dynamic adaptation and be able to interact with unknown software components.
4) *Changing operational context.* The operational context of ecosystem participants may change dynamically, either because the system joins a different ecosystem, or because the system operates in an environment that it was not designed for. This changes the smart agent behavior, as it has to adapt to new environmental conditions.
5) *Non-determinism.* The nature of advanced smart agent implementing machine learning may be non-deterministic. During their lifetime, these agents implement a continuous learning process that affects their behavior. Deviations from expected values may be due to the learning process, and not to malicious behavior.

III. Mitigation of Malicious Behavior

To prevent the injection of malicious behavior in smart ecosystems, one of the major goals is its detection and mitigation of its negative effects. For this purpose, here we extend our preliminary architectural proposal presented in [6] useful for the evaluation of software behavior in virtual environments. We propose a platform for mitigation of malicious behaviors able to : i) create a virtual environment such that smart agents cannot easily distinguish between simulations and the real world; ii) testing malicious behaviors that occur with very low probability in real but controlled testing environments, so to improve their early detection.

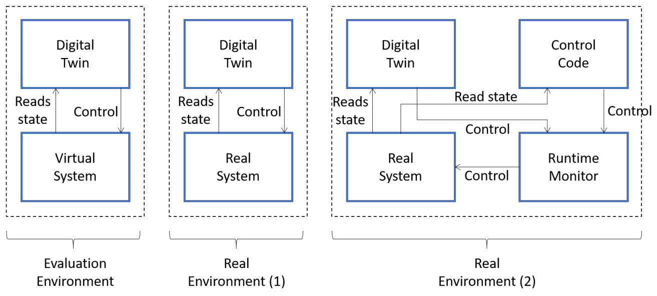

Fig. 1. Mitigation concept

Fig. 1 illustrates the concept of our mitigation platform. In particular, in *(Real Environment (1)*), we illustrate the situation already analyzed in [6] where the real environment is under direct control of the digital twin that has been previously evaluated in a simulated environment. Note that, in Fig. 1, the *Real System* is the ECU (Embedded Control Unit) showed in Fig 1 of [6] in charge of executing the control function of the software smart agent.

We extend our previous work as shown in *(Real Environment (2)*) of Fig. 1, where we assume that a smart agent is accompanied by a digital twin which provides an abstracted behavior of the smart agent. As in the figure, an additional component namely the *Runtime Monitor* ensures conformance between the DT behavior (specification) and the control component (realization). In this way, the control component of the smart agent may implement much more complex behavior than the DT; as long as it stays within the DT specification, it will be considered safe. The runtime monitor does not need application knowledge regarding the control component implementation; it needs to compare the DT specification to the control component behavior. It can therefore be a trusted and validated component inside the system. If it detects a deviation, it may switch to a predefined safe behavior that, e.g. disconnects the malicious control function and stops the autonomous vehicle.

Our evaluation environment evaluates whether the DT, which is a specification of the smart agent, provides a valid and safe behavior at all times. This is achieved by linking the DT to a virtual system (Evaluation Environment). This virtual system may be replicated many times. During multiple executions of the DT behavior, if the DT contains any malicious code, it is more likely to discover it inside one of the evaluation environments than in the real system. The *Runtime Monitor* judges the DT behavior and evaluates whether the DT did show malicious behavior or not.

DT that shows a proper behavior in the virtual environment will build up trust, and over time will be allowed to control the real system as well. The ecosystem however needs to take care that any situation, which includes behavior, time, and external factors is evaluated first in the virtual environment, and that the number of those evaluations is big compared to the controlled situation instances in the real system.

This assumption is however only feasible, if the DT is unable to detect whether it operates inside an evaluation environment, or whether it operates in a real environment. This detection cannot be prevented for generic software behavior. The DT behavior therefore must be limited to, for example, value tables that output expected value ranges for permutations of input values and historical inputs.

The proposed solution is a possible reply to the challenges listed in Section II. In particular, with reference to challenge (1) it can manage unexpected behavior of the smart agent by exploiting the time elasticity and by monitoring the events of behavioral execution both of the digital twin and of the software smart agent. Concerning challenge (2), the integration of third party systems or components is improved by using clear interfaces of communication of input and output data between the software smart agent and other interacting applications during the monitoring activity. Monitoring data can be exploited for offline or online analyses of root cause for malicious decisions. To cope with challenges (3) and (4), sensor data describing the current state of the system and of the environment are collected at the beginning of the simulation, so to easily allow detection of changes in operational contexts and environmental conditions. Finally, monitoring data collected after executing the digital twin in a simulated environment can be used to dynamically generating the specification of the software behavior of the smart agent so to mitigate the non-determinism, which is the major aspect of challenge (5). In the next subsection, we present the monitoring concept and briefly overview the main monitoring functionalities.

A. Monitoring

Existing run-time and offline monitoring solutions [8] address two main challenges: i) finding very powerful, concise and unambiguous specification languages able to express the properties to be validated; ii) provide efficient mechanisms to assess the on line or offline conformity of the system against these properties or contracts.

In our approach, we consider the valid behavior of the digital twin being the specification or contract against which the behavior of the system is validated. However, to the best of our knowledge, there is not so much recent literature addressing monitoring of combined simulated and real world software executions as well as the conformity among them. This paper goes in this direction providing an approach exploiting monitoring for evaluating the digital twin behavior in a simulated environment and then assuring conformity between the software agent and its digital twin in a real-time evaluation. In this section, we consider three main monitoring aspects: monitoring in context of linked predictive simulation; monitoring for validity and monitoring for conformity.

a) Monitoring in context of linked predictive simulation: An important aspect for runtime monitoring remains the definition of parameters against which the behavior of the smart agent is judged as being trustworthy or not. This trustworthiness does not only cover the results that an agent or control component provides; it must also consider timeliness. Monitors therefore need to consider the tuple *(decision, Δt)*, Δt being the decision time. If a result is not provided within the safe time interval Δt, the software execution must be considered potentially malicious and countermeasures need to be activated.

b) Monitoring for validity: Monitoring the validity of a decision can be done by performing a look-ahead simulation and assessment of the impact of virtually triggered decision events in the simulation environment. This approach requires internal simulation models: *a)* of the components with which the smart agent interacts, *b)* of the system itself, *c)* of the environment. The simulation models need to be abstract enough for enabling efficient execution of the digital twins and accurate enough for being effective in the scope of the evaluation. Fig. 2 depicts our concept for a monitor that validates the decision of a digital twin in a simulated environment. The events triggered during the execution of the digital twin of the smart agent are notified at its interface and propagated to digital twins of interacting components. These digital twins are internal simulation models of interacting system components fed with predicted real-time data of system's sensors and actuator. The behavior of the system including its future state is evaluated by an evaluation engine that observes the effects of triggered events on the orchestrated digital twins. The evaluation engine provides an alarm that triggers the fail-over behavior, or provides a list of events in the order they occurred.

c) Monitoring for conformity: Building trust on a particular behavior execution of a smart agent requires an a-priory knowledge of its intentions. This knowledge can be achieved by evaluating its specifications in advance in a

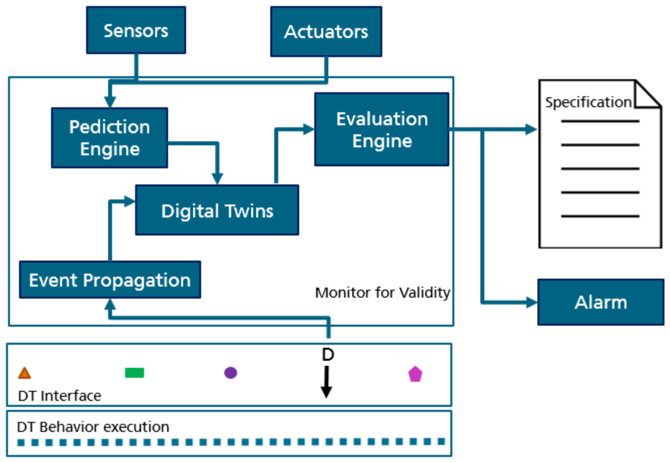

Fig. 2. Monitoring for Validity

simulated environment. Digital twins are running specification of smart agent's behavior fed with real-time data obtained from querying the state of the system and the state of the environment at the moment when the linked predictive simulation starts. Conformity monitoring during linked predictive simulation starts immediately after the execution of the digital twin's behavior in a simulated environment. The results of the digital twin execution form the specification against which the behavior of the smart agent is evaluated.

IV. Related Work

Smart Ecosystems extend the concept of cyber-physcial systems (CPS). The authors of [8] present a review of techniques and tools for qualitative and quantitative monitoring of CPS, distinguishing two major contexts in which the monitoring of dynamic behaviours can be applied: run time verification during system execution with respect to some specified properties; model-based system design and development where the system behavior is simulated and evaluated in many scenarios.

Other approaches [9] target offline monitors proposing solutions for identifying violations against policies expressed in first-order temporal logic as well as efficient and scalable parallelization of the logs analysis leveraging map-reduce approaches.

Regarding monitoring for validity, Blum et al. [10] propose a look-ahead simulation for evaluating the safety of a decision through a consequence engine. It utilizes internal simulation models of the robot, interacting actors and the environment. Our approach doesn't rely on selection of a set of predefined actions, but dynamically generates the actions exploiting the results of decision events. In particular, our validity monitoring activities involve selection of most probable environmental changes and iterations through multiple internal simulation models of the environment.

V. Conclusion

In our work, we are interested in the behavioral execution of software components and behavioral conformity between a real time execution of a smart agent on an ECU (embedded control unit) and its digital twin's execution in a simulated environment.

Therefore, we introduced the challenges of runtime monitoring for malicious behaviour detection in smart ecosystems and presented a first architecture solution using monitoring to promptly detect malicious behaviors. The proposed architecture is applicable for monitoring software smart agents and their digital twins that are received as black boxes.

The approach we are proposing has several limitations as well. It has been conceived for well defined interfaces and in situations where the agents cannot influence intermediate states in each other way except through the well defined interfaces.

As future work, we would like to extend the monitoring components so as to address not only the input/output communications but also intermediate states.

At the same time we are exploring the possibility of defining digital twins of smart agents by abstracting it's behavior for the scope of evaluation.

Acknowledgment

This work has received funding from ECSEL Joint Undertaking (JU) under grant agreement No 783119. The JU receives support from the European Union Horizon 2020 research and innovation program and Netherlands, Austria, Belgium, Czech Republic, Germany, Spain, Finland, France, Hungary, Italy, Portugal, Romania, Sweden, United Kingdom, Tunisia.

References

[1] "Office of the deputy under secretary of defense for acquisition and technology, systems and software engineering. systems engineering guide for systems of systems," *Version 1.0. Washington, DC, ODUSD(A and T)SSE*, 2008.

[2] K. Manikas, "Revisiting software ecosystems research: A longitudinal literature study," *Journal of Systems and Software*, vol. 117, pp. 84–103, 2016.

[3] M. Seaborn and T. Dullien, "Exploiting the dram rowhammer bug to gain kernel privileges," *Black Hat*, vol. 15, 2015.

[4] "Ars technica." [Online]. Available: https://arstechnica.com/information-technology/2016/10/using-rowhammer-bitflips-to-root-android-phones-is-now-a-thing/

[5] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE transactions on dependable and secure computing*, vol. 1, no. 1, pp. 11–33, 2004.

[6] E. Cioroaica, T. Kuhn, and B. Buhnova, "(do not) trust in ecosystems," in *Proceedings of the 41st International Conference on Software Engineering: New Ideas and Emerging Results*. IEEE Press, 2019, pp. 9–12.

[7] A. Romanovsky and F. Ishikawa, *Trustworthy cyber-physical systems engineering*. CRC Press, 2016.

[8] E. Bartocci, J. Deshmukh, A. Donzé, G. Fainekos, O. Maler, D. Ničković, and S. Sankaranarayanan, *Specification-Based Monitoring of Cyber-Physical Systems: A Survey on Theory, Tools and Applications*. Cham: Springer International Publishing, 2018, pp. 135–175.

[9] D. Basin, G. Caronni, S. Ereth, M. Harvan, F. Klaedtke, and H. Mantel, "Scalable offline monitoring," in *Runtime Verification*, B. Bonakdarpour and S. A. Smolka, Eds. Cham: Springer International Publishing, 2014, pp. 31–47.

[10] C. Blum, A. F. T. Winfield, and V. V. Hafner, "Simulation-based internal models for safer robots," *Frontiers in Robotics and AI*, vol. 4, p. 74, 2018. [Online]. Available: https://www.frontiersin.org/article/10.3389/frobt.2017.00074