Cyber-Physical Systems Design for Runtime Trustworthiness Maintenance Supported by Tools

Torsten Bandyszak[1], Nazila Gol Mohammadi[1], Mohamed Bishr[1], Abigail Goldsteen[2], Micha Moffie[2], Bassem I. Nasser[3], Sandro Hartenstein[4], Symeon Meichanetzoglou[5]

[1]paluno – The Ruhr Institute for Software Technology, University of Duisburg-Essen, Germany
{torsten.bandyszak, nazila.golmohammadi, mohamed.bishr}@paluno.uni-due.de
[2]IBM Research - Haifa, Israel
{abigailt, moffie}@il.ibm.com
[3]It-Innovation Center, School of Electronics and Computer Science, University of Southampton, UK
bmn@it-innovation.soton.ac.uk
[4]Department of Economics, Brandenburg University of Applied Sciences, Germany
sandro.hartenstein@fh-brandenburg.de
[5]Foundation for Research and Technology Hellas, Greece
simosme@ics.forth.gr

Abstract. The trustworthiness of cyber-physical systems is a critical factor for establishing wide-spread adoption of these systems. Hence, especially the behavior of safety-critical software components needs to be monitored and managed during system operation. Runtime trustworthiness maintenance should be planned and prepared in early requirements and design phases. This involves the identification of threats that may occur and affect user's trust at runtime, as well as related controls that can be executed to mitigate the threats. Furthermore, observable and measureable system quality properties have to be identified as indicators of threats, and interfaces for reporting these properties as well as for executing controls have to be designed and implemented. This paper presents a process model for preparing and designing systems for runtime trustworthiness maintenance, which is supported by several tools that facilitate the tasks to be performed by requirements engineers and system designers.

Keywords: Trustworthiness Requirements, Runtime Monitoring, Threats, Mitigation, Adaptation, Trustworthiness-by-Design

1 Introduction

Cyber-Physical Systems (CPS) comprise humans as well as software and hardware components, are distributed and connected via the Internet, and are able to dynamically adapt to context changes [1]. These software-intensive systems often perform critical tasks, and the consequences of software failures become tangible in terms of threats to the physical environment. For instance, security vulnerabilities that may be

exploited by malicious attacks are severe threats to CPS that control critical infrastructures such as Vehicle-to-Vehicle communication systems [7]. Hence, decisions to trust these systems heavily depend on their trustworthiness, which is especially crucial for software components. We consider trustworthiness as an objective system property that can be measured in terms of several software quality attributes and corresponding metrics (cf. [4]).

In requirements engineering and design, trustworthiness requirements need to be analyzed so that their satisfaction can be monitored and controlled at runtime [14]. It is crucial to explicitly document assumptions about the operational context of CPS in the development, so that possible changes in the context can be predicted, especially if these systems are designed to be long-living [12]. This involves the identification of threats, i.e., undesired situations that may occur during system operation and negatively affect the trustworthiness. A CPS needs to be aware of such situations, and adapt by applying certain controls to mitigate the threats. To this end, capabilities of monitoring and adaptation need to be considered in the requirements, and appropriate interfaces should be designed to allow for monitoring trustworthy system behavior and executing controls. Respective guidelines and tool support is necessary to support developers in performing these tasks. Existing approaches towards design for monitoring quality of software services, have a very technical focus, e.g., on implementing monitoring rules with assertions [5]. These approaches also often regard software independently of the environment, and are usually specific to a certain technology or modeling language. There is a gap in providing essential concepts, tasks, and guidance for preparing runtime trustworthiness maintenance.

In this paper, we build upon our previous work on trustworthiness-by-design [3], a risk-management approach for identifying threats and controls [6], and a runtime trustworthiness maintenance system [2, 11]. We elaborate on how these approaches and respective tools can be applied in order to support the design of adaptive CPS that can be monitored and maintained w.r.t. trustworthiness at runtime. To this end, we show a process model as generic guidance for developers, and introduce the tools that support the involved steps. Our approach supports the preparation of runtime trustworthiness maintenance by guiding the designer in identifying threats and associated controls, as well as observable and measureable system properties. Based on these essential concepts, respective monitoring and control interfaces can be designed and implemented. Eventually, the runtime trustworthiness maintenance system is configured to allow for analyzing threat activities by means of the system behavior observed from measureable properties. We illustrate our approach by showing its application to a Distributed Attack Detection and Visualization (DADV) system that supervises intrusions in a corporate network by means of sensors that attract malicious attacks.

The remainder of this paper is structured as follows: in Section 2, we introduce the underlying concepts that we developed in previous works, and discusses related work in the areas of alignment between requirements and design, and runtime monitoring. In Section 3 we present our approach towards preparing for runtime maintenance in requirements and design, and explain the tool support of the different steps. Section 4 provides an application example in order to illustrate our approach. Last, Section 5 summarizes the paper and provides an outlook on future work.

2 Fundamentals and Related Work

We consider trustworthiness as an objective system property that can be measured in terms of several software quality attributes and corresponding metrics. Hence, trustworthiness can be seen as an objective system property, while trust is the subjective perception of some trustor (cf. [4]). In [4] we presented a taxonomy of software quality attributes that contribute to trustworthiness (denoted as trustworthiness attributes), and the derivation of metrics that can be used in order to quantify them. In order to consider trust and trustworthiness throughout the system life cycle, we follow an approach that allows for tailoring and extending existing process models with trustworthiness-by-design guidelines and best practices denoted as capability patterns [3].

Regarding the maintenance of trustworthiness at runtime, a reference architecture and a prototype of a Trustworthiness Maintenance system that is capable of monitoring trustworthiness properties and mitigating threats at runtime is described in [2]. The fully operational trustworthiness maintenance tool is described in [11] in the context of CPS, including an updated architecture. This system consists of several components that are orchestrated to realize trustworthiness maintenance, based on an ontology that relates important key concepts of runtime trustworthiness maintenance. Fig. 1 depicts a simplified version of this ontology.

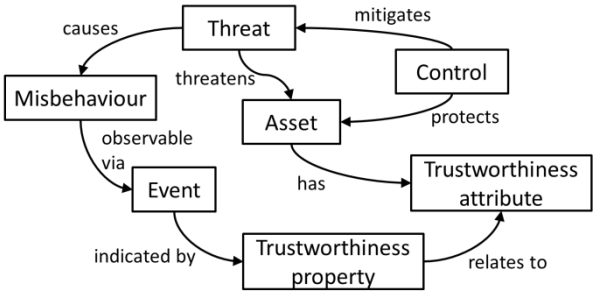

Fig. 1. Runtime Maintenance Concepts (based on [2])

A threat to trustworthiness may cause a misbehaviour at runtime, which is observable by means of events that the system or some observation probe reports. A threat undermines an asset's (i.e., system building block) value and reduces its trustworthiness [2], while a misbehaviour is similar to the notion of a failure (cf. [13]). A control can be executed to mitigate a threat and thereby protect an asset. Our risk-management approach for identifying threats and controls, which we took over from the SERSCIS[1] project, is described in [6]. Events are related to trustworthiness metrics and trustworthiness properties that are observable and more tangible than trustworthiness attributes, i.e., trustworthiness properties can be quantified as values of a certain type [2].

Related work can be found in the area of service monitoring, since the quality of software services is the basis for service-level agreements that need to be met during

[1] Project website: www.serscis.eu

system operation. For instance, Baresi and Guinea [5] use source code assertions to specify monitoring rules as constraints for the execution at runtime. There is also a framework that involves the automated extraction of behavioural properties (i.e., formal specifications of events) from service requirements, which are then used as input for a monitor [8]. However, these approaches are on a very technical level and require the use of certain languages and technologies, without explicitly considering general aspects that need to be accounted for when planning and designing trustworthy software-intensive systems that are maintainable at runtime.

De Miguel et al. [10] elaborate on the use of modelling languages throughout the whole lifecycle (i.e., from requirements modeling to runtime monitoring and management phases) of quality-aware systems, and conclude that suitable modeling techniques for describing service quality properties are needed to align the phases. Moisan et al. [9] present a framework supporting the design, deployment and runtime monitoring of video-surveillance systems, which utilizes feature diagrams both for design and runtime management. These models are used to describe possible feature configurations that can be adapted at runtime, but do not specifically consider the specification and mitigation of threats at runtime.

To summarize, there is a need for generic guidelines that support requirements engineers and designers in preparing for runtime maintenance. Furthermore, regarding CPS the existing approaches mentioned above focus on software exclusively, without considering its relations to the physical environment.

3 Preparation for Trustworthiness Runtime Maintenance

In this section, we present the process of preparing for maintaining the trustworthiness of CPS at runtime. Fig. 2 gives an overview of the relevant steps that should be performed for designing adaptive trustworthy CPS, and the tools that support these tasks (indicated by grey rectangles and dashed lines). This process can be seen as an important capability and guidance to develop trustworthy CPS, and aims at reducing the effort required for runtime preparation. It can be added to common software engineering process models using the process model extension mechanism sketched in [3].

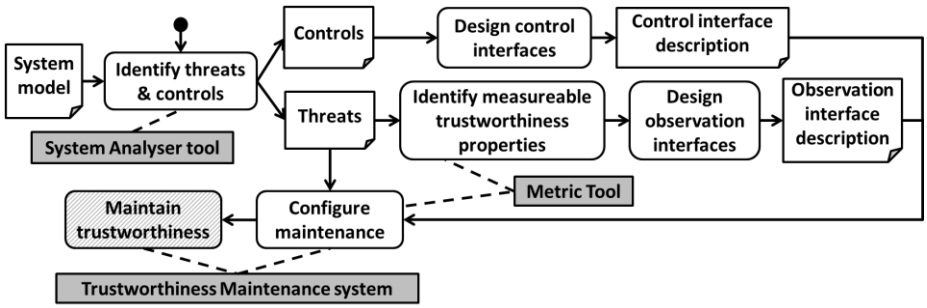

Fig. 2. Preparation for Trustworthiness Maintenance and Tools

The first step in Fig. 2 is the identification of potential threats that may occur at runtime and thereby impact the trustworthiness of the system, as well as related controls. These artifacts constitute trustworthiness requirements as outcome of a risk-based analysis. As input, a model of the CPS is needed, which specifies its main building blocks (assets) and their relations. This model involves software parts, as well as humans and technical assets such as network hardware, since threats emerge from the interplay between these assets, though they can be observed and also mitigated on the software. The *System Analyser* tool supports this task by automatically generating a list of threats and controls per asset, based on an ontology of generic threats for different asset types and their relations. In particular, a generic model specifies different (sub)types of assets, and comprises knowledge about potential threats related to these generic asset types. This allows for analyzing these threats on concrete models specifying a system under development. Similarly, related controls can be identified based on this comprehensive knowledge base.

Based on the list of threats and controls, the system designer has to identify observable system properties that can be interpreted as indicators for trustworthiness and need to be observed by the trustworthiness maintenance system. Since the activeness of threats may not be directly visible in the system behavior, it is essential to determine observable system characteristics that contribute to trustworthiness, and can be objectively measured (cf. Section 2). To this end, the taxonomy of trustworthiness attributes and corresponding metrics presented in [4] can be used as guidance, which forms the theoretical basis of the *Metric Tool*. This tool provides a comprehensive repository of metrics that are related to trustworthiness attributes and define trustworthiness properties as inputs for calculations.

Next, the monitoring of these trustworthiness properties, as well as the execution of controls has to be prepared. To this end, appropriate interfaces are designed in order to realize the runtime observation. For instance, a REST interface could be prepared by creating a respective interface design document. Implementing this interface will allow the *Trustworthiness Maintenance* system (cf. [2]) to receive sensor utilization statistics and thereby monitor the asset behavior w.r.t. trustworthiness. Furthermore, for each control, the designer defines an interface that can be used by the trustworthiness maintenance to invoke the control on a specific asset.

The final step of runtime trustworthiness maintenance preparation is configuring the *Trustworthiness Maintenance* system. This requires not only to connect to the interfaces to which the CPS periodically sends its trustworthiness property values as events, but also to specify a relation between these observed properties and threats. For each threat, misbehaviour symptoms that can be identified from the reported events are determined. These misbehaviours will indicate the activeness of threats at runtime, and are processed by the maintenance system to select controls (cf. [2]). With the help of the Metric Tool, rules and thresholds for identifying the occurrence of misbehaviors are defined.

Later in runtime monitoring, the events reported by through the observation interfaces are processed by the *Trustworthiness Maintenance* system, whose architecture is based on control loops in autonomic computing (cf. [2]). This tool actually performs the runtime monitoring, analysis, and mitigation tasks w.r.t. trustworthiness. More

information on the runtime trustworthiness maintenance system, i.e., its components and functionality, can be found in [2, 11].

4 Application Example

In order to exemplify and illustrate our approach and the applications of the tools, we use a Distributed Attack Detection and Visualization (DADV) system. This system consists, among other components, of a number of sensors, i.e., virtual machine honeypots that simulate vulnerabilities and thereby attract attacks to corporate networks. Sensors need to be hypervised, and a sensor's trustworthiness should be monitored and managed at runtime, as compromised sensors constitute security vulnerabilities. Hence, this maintainability and adaptivity needs to be considered and built into the system design early in the development lifecycle. In the following, we will illustrate the preparation for runtime maintenance in this case example.

As explained in Section 3, the first step in the process consists of identifying threats and controls based on a system model including the main system components, such as a "Sensor" asset. The system model also includes e.g. the human operator of the DADV, as well as the organizations interested in the security of their corporate network. In the background, the assets depicted in the system model are mapped to generic asset types, which allows the *System Analyser* tool to generate a list of threats and associated controls. Among others, the threat "Unauthorized Communication" and the related control "Blacklisting" are derived for the sensor assets.

An observable property related to the threat mentioned above is the resource usage of a sensor asset. In particular, e.g. CPU, memory or network consumption properties are defined as parameters for performance metrics, which can be browsed in the *Metric Tool*. Consequently, respective observation interfaces for the automatic reporting of resource usage statistics in the form of events are designed. Similarly, control interfaces need to be designed for each of the identified controls. The DADV-specific control "Blacklisting" can be executed by shutting down a sensor. To this end, the sensor hypervisor component needs to provide a respective interface. In the DADV system, all observation and control interfaces are specified and implemented using REST.

Finally, the *Trustworthiness Maintenance* system needs to be configured to allow for actual runtime maintenance by establishing connections to the interfaces described above. Furthermore, the configuration comprises mapping trustworthiness properties to misbehaviours (cf. Section 2). Based on the current resource usage, which have been defined as relevant trustworthiness properties of a DADV sensor, the misbehaviour "Overloaded" can be determined. Rules and thresholds for identifying the occurrence of this misbehaviour include, e.g., current CPU usage above 10%. These thresholds are indicators for an unauthorized communication caused by a compromised sensor, since a DADV sensor's normal resource consumption is very low.

The DADV system is also used in [11] as an application example and initial evaluation of the *Trustworthiness Maintenance* system, in order to illustrate the actual maintenance of trustworthiness at runtime.

5 Conclusion and Outlook

In this paper, we presented a process model of essential steps that need to be considered when planning and designing adaptive CPS. This process model aims at guiding requirements engineers and designers to prepare for runtime maintenance w.r.t. trustworthiness. These systems should be monitored at runtime to assure that they remain trustworthy; consequently, threats to trustworthiness need to be analyzed, and respective controls need to be executed to mitigate them, if necessary. All this requires that threats, controls, and observable system trustworthiness properties are identified early in the development process, as well as observation and control interfaces designed and built into the system.

Future work will focus on the empirical evaluation of our tools. Especially the benefits for developers in terms of effort required for planning and designing observation and control interfaces should be investigated. Furthermore, extensions of our approach to take trust maintenance into account should be considered. It is important to understand how trustworthiness properties actually influence the subjective trust of the system users, so that systems can be designed to support monitoring and maintaining trust at runtime.

Acknowledgements. The research leading to these results has received funding from the European Union's FP7/2007-2013 under grant agreement no. 317631 (OPTET).

References

1. Broy, M., Cengarle, M.V., Geisberger, E.: Cyber-Physical Systems – Imminent Challenges. In: Calinescu, R., Garlan, D. (eds.) Monterey Workshop 2012. LNCS 7539, pp. 1–28. Springer, Heidelberg (2012)
2. Gol Mohammadi, N., Bandyszak, T., Moffie, M., Chen, X., Weyer, T., Kalogiros, C., Nasser, B., Surridge, M.: Maintaining Trustworthiness of Socio-Technical Systems at Run-Time. In: Eckert, C. et al. (eds.): TrustBus 2014, LNCS 8647, pp. 1–12. Springer, Heidelberg (2014)
3. Gol Mohammadi, N., Bandyszak, T., Paulus, S., Meland, P.H., Weyer, T., Pohl, K.: Extending Development Methodologies with Trustworthiness-By-Design for Socio-Technical Systems (Extended Abstract). In: Holz, T., Ioannidis, S. (eds.): TRUST 2014, LNCS 8564, pp. 206–207. Springer, Heidelberg (2014)
4. Gol Mohammadi, N, Paulus, S., Bishr, M., Metzger, A, Könnecke, H., Hartenstein, S., Weyer, T., Pohl, K.: Trustworthiness Attributes and Metrics for Engineering Trusted Internet-Based Software Systems. In: Helfert, M., et al. (eds): Cloud Computing and Services Science, CCIS 453, pp 19–35. Springer, Heidelberg (2014)
5. Baresi, L., Guinea, S.: Towards Dynamic Monitoring of WS-BPEL Processes. In: Benatallah, B., et al. (eds.): ICSOC 2005, LNCS 3826, pp. 269–282. Springer, Heidelberg (2005)
6. Surridge, M., Nasser, B., Chen, X., Chakravarthy, A., Melas, P.: Run-Time Risk Management in Adaptive ICT Systems. In: ARES 2013, pp. 102–110. IEEE (2013)

7. Zalewski, J., Drager, S., Kornecki, A.J.: Threat Modeling for Security Assessment in Cyberphysical Systems. In: Proceedings of the Eighth Annual Cyber Security and Information Intelligence Research Workshop (CSIIRW '13), Art. no. 10. ACM (2013)
8. Mahbub, K., Spanoudakis, G.: A Framework for Requirements Monitoring of Service Based Systems. In: Proceedings of the 2nd International Conference on Service Oriented Computing (ICSOC '04), pp. 84–93. ACM (2004)
9. Moisan, S., Rigault, J., Acher, M., Collet, P., Lahire, P.: Run Time Adaptation of Video-Surveillance Systems – A Software Modeling Approach. In: Crowley, J.L., Draper, D., Thonnat, M. (eds.): ICVS 2011, LNCS 6962, pp. 203–212. Springer, Heidelberg (2011).
10. De Miguel, M.A., Massonet, P., Silva, J.P., Briones, J.: Model Based Development of Quality-Aware Software Services. In: Proceeedings of the 11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing (ISORC 2008), pp. 563–569. IEEE (2008)
11. Goldsteen, A., Moffie, M., Bandyszak, T., Gol Mohammadi, N., Chen, X., Meichanetzoglou, S., Ioannidis, S., Chatzidiam, P.: A Tool for Monitoring and Maintaining System Trustworthiness at Runtime. In: Proceedings of the 1st International Workshop on Requirements Engineering for Self-Adaptive and Cyber Physical Systems (RESACS), CEUR Workshop Proceedings (2015)
12. Daun, M., Tenbergen, B., Brings, J., Weyer, T.: Documenting Assumptions about the Operational Context of Long-Living Collaborative Embedded Systems. In: Proceedings of the 2nd Collaborative Workshop on Evolution and Maintenance of Long-Living Software Systems (2015)
13. ISO/IEC/IEEE: Systems and Software Engineering – Vocabulary. International Standard ISO/IEC/IEEE 24765, First edition (2012)
14. Alebrahim, A., Gol Mohammadi, N., Heisel, M.: Challenges in Rendering and Maintaining Trustworthiness for Long-Living Software Systems. In: Proceedings of the 2nd Collaborative Workshop on Evolution and Maintenance of Long-Living Software Systems (2015)