– Preprint –
Accepted for publication at Cyber Resilience Workshop 2022
Final published version available at: https://doi.org/10.1109/CNS56114.2022.10092919

Towards Cost-Benefit-Aware Adaptive Monitoring for Cyber-Physical Systems

Michael Vierhauser*, Rebekka Wohlrab†, Stefan Rass*‡
* LIT Secure and Correct Systems Lab, Johannes Kepler University Linz, Austria
† Chalmers | University of Gothenburg, Gothenburg, Sweden
‡ Institute of Artificial Intelligence and Cybersecurity, Universitaet Klagenfurt, Austria

***Abstract*—Cyber-Physical Systems (CPS) are becoming ubiquitous in many different domains, for example, in the form of Unmanned Aerial Vehicles (UAVs), (semi-)autonomous systems, or robotic applications. Given that CPS frequently operate in a safety-critical context, and interact and collaborate with humans, ensuring that these systems behave as intended and adhere to their specified security and safety requirements at runtime is essential. Providing automated support for monitoring is a fundamental part of collecting information about the system, and facilitating subsequent analysis and reasoning. However, while advances have been made, particularly in self-adaptation and self-management, the monitoring aspect is often neglected, resulting in suboptimal data collection that does not consider associated monitoring costs, changing environments, or varying benefits of the collected data. Such benefits range from accountability in the aftermath of a security incident, up to proactive defense against risks if connected to alarming. In this paper, we outline our initial concept for cost-benefit-aware adaptive runtime monitoring, for (but not limited to) safety and security requirements. As part of this work, we identify relevant monitoring aspects and create a Cost-Benefit-Aware Adaptive Model (CBAAM), conceptualizing the costs and benefits of adaptive monitoring. This is also especially relevant for security, saving resources, e.g., budget and computational. We further present an architecture for defining and executing these adaptations and discuss our research roadmap and next steps.**

***Index Terms*—Runtime Monitoring, Cost-Benefit Analysis, Self-Adaptation, Adaptive Monitoring**

I. Introduction

Cyber-Physical Systems (CPS) are becoming ubiquitous, with examples ranging from autonomous systems [1] such as Unmanned Aerial Vehicles (UAVs) [2], [3], to Cyber-Physical Production Systems, to name but a few. As these systems typically operate in safety-critical environments, exhibiting tight integration between hardware and software, and interacting with humans, it is crucial that these systems adhere to their specified requirements and behave as intended. For example, Robot Operating System (ROS) based systems require additional protection outside ROS since as of version 1 of ROS, security mechanisms are only gradually being integrated, with ROS2 and SROS. However, legacy systems or implementations under tight resource constraints may still run in "plain" ROS, hence lacking native, built-in security. While such systems are typically configured towards a correct functionality, keeping the correct behavior under changing environmental conditions, possibly induced by hostile entities, requires additional effort. Therefore, providing means to *monitor*, *analyze*, and dynamically *adapt* systems is key for ensuring correct, efficient, and above all, secure and safe operation as part of self-adaptive systems [4] that adapt system behavior and structure accordingly. To enable self-adaptation, it is crucial to ensure that relevant properties are monitored, so that adaptations can be adequately planned and executed. Insofar, as it concerns the system's basic functionality, sensors and actors in robotics are built to exactly adapt to the environment according to the robot's functional specification. Our focus in the following thus revolves around how to adapt to unexpected or unintended changes in the environment. While most self-adaptive systems do provide sophisticated mechanisms for analyzing and adapting its behavior and structure, the actual *monitoring* aspect itself, in many cases, remains largely static and inflexible. To collect the right amount of data at the right point in time, adaptive monitoring [5] is required to adapt the monitoring infrastructure alongside the System under Monitoring (SuM). This not only ensures that relevant security and safety constraints can be checked, but also that only required data is collected, reducing bandwidth, energy consumption, and computation power (the latter being of particular importance, if cryptographic protections shall be implemented, which, for some public-key primitives, can be expensive in several regards).

Existing adaptive monitoring approaches focus on system states to reduce the number of monitors, or adjust periodic or event-driven monitoring tasks [6], [7]. A widely adopted technology in the security domain are security incident and event monitoring (SIEM) systems [8], primarily found in enterprise environments, generally consisting of many sensors that continuously measure system performance characteristics and compile them into up-to-date pictures about the system's "health state". Furthermore, predictive analytics is often leveraged to receive early warnings about the system potentially running into race conditions, and logging of monitored events is crucial for forensic investigations and accountability (particularly relevant in instances where the system has no or weak immanent security, such as ROS in its early versions [9]). Combining this with model- and logic-based methods enables the application of control theory here to secure a system at runtime [10].

Most of these methods do not provide adequate support for determining "optimal" monitors and lack the ability to specify the costs associated with monitoring or to impose relevant safety restrictions on specific monitors. For SIEM systems, the cost factor is not necessarily a primary concern, conditional on the protected value being more than the investment for security. However, in large-scale infrastructures, the optimal placement of sensors for monitoring becomes a nontrivial issue, with sophisticated solution methods [11], [12].

When the monitors are in place, their activation and data delivery is again a matter of resource consumption and hence optimization to merit their use for security. To enable effective and efficient monitoring, the monitoring behavior needs to be automatically adapted. This in turn requires us to first *(1) identify and document relevant properties and data values that need to be monitored*. Secondly, means to *(2) capture the costs associated with these properties* (e.g., energy consumption, or bandwidth [13], [14]). These properties then are to be linked to the respective analysis tasks performing runtime checks, with each check providing a certain benefit when performed. Besides support for runtime checks, properties may also need to be monitored to keep users informed about the state of the system [15], [16]. From a human stakeholder perspective, properties might be more or less relevant to monitor, and some data needs to be updated more frequently (e.g., the location of a UAV during a rescue mission) in the user interface (UI). Hence, adaptive monitoring needs to *(3) factor in user input regarding monitoring preferences*. To automatically optimize adaptive monitoring, we propose *"Cost-Benefit-Aware Adaptive Monitoring"*. Cost-benefit analysis is frequently used in self-adaptive systems, e.g., to plan adaptions [17], [18], [19]. More importantly, cost-benefit analysis is crucial to argue for a security mechanism, since security mainly preserves existing values rather than generating additional revenues. Therefore, security based on monitoring and incident detection, automatically selecting when, what, and how frequently important system properties should be monitored, can provide valuable economic arguments for such a security system inside a CPS. This is especially important when "light" hardware is used, possibly preventing the use of cryptographic protection that requires strong computational resources to leverage public key cryptography [20], [21] to protect the system.

In this paper, we present a simple cost-benefit analysis method to dynamically adapt monitoring behavior. We first present a model conceptualizing costs and benefits of adaptive monitoring (Sect. II) and describe an initial approach for defining and executing these adaptations (Sect. III). We then report results from a preliminary evaluation (Sect. IV) and lay out our research roadmap (Sect. V).

II. Conceptualizing Monitoring Costs and Benefit

Monitoring is an integral part of the Monitor-Analyze-Plan-Execute over a shared Knowledge (MAPE-K) loop architecture for self-adaptive systems [22], [23], and general intrusion detection system (IDS). Our work here covers intrusion detection as one possible purpose of monitoring, but is more general, in the sense that it may raise alarms or serve predictive analytical purposes (such as SIEM systems based on IDS would offer), but additionally, facilitate self-adaptation depending on the current conditions or state of the system. That is, our monitoring methods are a form of self-enforcing security, differentiating it from IDS and SIEM in other contexts.

One key requirement of monitoring is that *appropriate information* should be collected at the right point in time, and that data is collected *frequently enough* to ensure runtime analysis, without negatively affecting system performance or blocking resources (the latter extends to waste of human time in case of false alarms). Both the required information and the frequency are dependent on the context of the system.

For example, when a UAV performs a mission, certain requirements need to hold before the UAV can take off. A sufficient number of satellites must be available, and the UAV's flight controller must be properly initialized and fully calibrated. The respective runtime checks need to be performed, and the mission commander needs to be informed when a UAV tries to take off before these requirements are met. Subsequently, once the UAV has commenced its mission and flies in close proximity to other UAVs, other runtime checks become important, requiring different data to be collected. In this case, collecting and checking GPS location, altitude, or sensor data from individual UAVs has more benefits than the previously already satisfied initialization/calibration data. Runtime information regarding satellite fixes might still be beneficial, as a poor signal can still occur, as well as active attacks by jamming or injections of adversarial signals. However, a poor signal is less likely to occur in this state, it is less valuable to monitor satellite fixes on a frequent basis. On the contrary, jamming signals are more important to detect and counteract, as are general unexpected injected signals that are not corresponding to normal operation. In this context, *adaptive monitoring* has been defined as "the ability a monitoring system has to modify its structure and/or behavior, in order to respond to internal and external stimuli" [5]. There is a need for *"more complete, flexible, reusable, and generic software engineering solutions for supporting adaptive monitoring"* [5].

To create flexible and adaptive monitors, we introduce *Cost-Benefit-Aware Adaptive Monitoring* (CBAAM), which explicitly captures what runtime data needs to be collected and what the respective costs and benefits of monitoring are. Fig. 1 provides an overview of the three major elements: The restrictions that need to be ensured (e.g., due to security and safety regulations), the runtime monitor, and the costs and benefits of monitoring.

- **Runtime Monitors:** Monitoring employs the notion of *Monitorable Properties* (e.g., a UAV's location or mission status) to be collected and subsequently analyzed. Requirements related to these properties may fail at runtime [24], [25], e.g., due to unforeseen interactions between system components, due to unexpected behavior of agents interacting with the system, or due to environmental conditions. This can affect both functional and non-functional requirements, e.g., quality constraints targeting performance or reliability [26], [27], [28]. Requirements are typically formalized in a high-level formal language and then translated into a pattern of events, i.e., *Runtime Checks* [24]. For example, runtime checks can be used to ensure that UAVs keep a minimum separation distance at all times.

A particular challenge are integrity checks of sensor data, which are up to cryptographic protection only during transit of data, but the verification of the correctness (not necessarily the same as or implied by authenticity), typically requires redundant measurements and sensory. Since redundancy in turn induces additional costs during design- and run-time, optimizing the monitoring w.r.t. a balance between security and redundancy adds positively to the economic aspect when securing a CPS. Generally, we assume $m \in \mathbb{N}$ properties $p_1, \dots, p_m$ whose monitoring shall be optimized in terms of frequency (checks per time unit) which are collected in a set P. For example, redundant sensor queries to assure correctness of information, but also for the preservation of security and safety properties, such as bounds on pressure (e.g., when robots are to pick up and place parts), or safety distances to be adhered to. Corresponding to these properties, we let $n \in \mathbb{N}$ runtime checks exist on one or more of the properties, all modeled as predicates $\psi_i : A_i \to \{0, 1\}$ for $i = 1, 2, \dots, n$; where $A_i \subseteq P$ is the set of properties relevant for the check.

- **Costs & Benefits:** Manually defining monitoring frequencies on property-level, and for each potential state a system can transition, can quickly become both time-consuming and error-prone. In the case of the UAV example, potentially dozens of different runtime properties from the flight controller (GPS data, altitude, vehicle status), as well as system-specific properties, such as mission status, queued waypoints, or distance to other UAVs and obstacles are collected. Therefore, specifying individual monitoring frequencies can easily result in a combinatorial explosion of combinations of properties to be monitored in certain system states. This is exacerbated as certain runtime checks require several properties, e.g., checking for mission progress that relies on global mission properties, and individual UAV data which introduces additional dependencies between properties and associated runtime checks.

To tackle adaptive monitoring on a more abstract level, specifying respective costs and benefits helps to reduce the effort and overhead of defining adaptations for individual properties. Cost-benefit analysis has been successfully used to define adaptation strategies and perform adaptation in self-adaptive systems [17], [29], [18]. This work extends these concepts to the security of CPS. As part of our adaptive monitoring approach, we seek to adapt the monitoring framework itself, optimizing the monitors at runtime. Therefore, we link monitorable properties to their associated costs, and runtime checks to their respective benefits when a check is performed. Costs in CPS are typically related to e.g., energy consumption and bandwidth usage [13], [30], [17], whereas benefits represent the value and importance of a runtime check and the requirement it stems from. For security, the scope is respectively extended, which we do by introducing predicates that express the correct or incorrect functionality of a system, according to its specifications. This can, among other things, mean that safety distances are undercut, or sensitive information is accessed without permission.

- **Safety Envelope & User/System Restrictions:** The third part is concerned with restrictions constraining how frequently

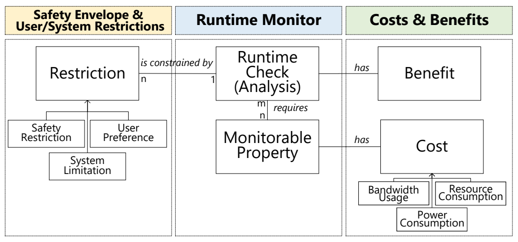


Fig. 1: Cost-Benefit-Aware Adaptive Model

runtime checks *can*, *should*, and *must* be executed. Such restrictions can stem from safety requirements constraining the lower bounds of checks (e.g., a safety requirement requiring GPS data to check potential violations of no-fly zones at least once every second). Also, the upper bound can be constrained by system limitations (e.g., that barometric altitude is updated at most every 250 milliseconds from the internal flight controller). These limitations have a direct impact on the cost-benefit optimization and provide ranges for monitorable properties and, in turn, the respective runtime checks that rely on these properties. Additionally, users might be involved in operating and monitoring the system, e.g., using a UI to "monitor" UAVs executing a mission. To facilitate human supervision, users may want to specify/update how often properties are collected or how often runtime checks are performed in certain states of a mission (within the mentioned limitations) [31], [32].

Therefore, an additional set of $n \in \mathbb{N}$ constraints defining the least or maximum frequency of checking a property needs to be defined. These are specified as constraint pairs $\underline{r}_i, \overline{r}_i$ such that the frequency $x_i \in [0, 1]$ of checking property P_i lies in the range $0 \leq \underline{r}_i \leq x_i \leq \overline{r}_i \leq 1$ for $i = 1, \dots, n$.

Bounding the frequency in the unit interval is done w.l.o.g., since x_i is taken relative to a unit of time: that is, frequency 0 corresponds to no checks at all, while 1 means continuous monitoring. The values 0 or 1 for the bounds render the respective upper or lower bound trivial, if only one of the two (or no constraint) is imposed.

III. CBAAM Approach Overview

Based on the aforementioned challenges we have created an initial architecture, specifically augmenting the M (monitoring) part of the MAPE-K loop [33] using our CBAAM model. Fig. 2 provides an overview of the four main elements.

- **Monitoring Configuration:** As described in Sect. II, three sources of information, pertaining to runtime checks, restrictions, and cost/benefits are required to successfully establish and deploy adaptive monitors. At design time, before the system is put into operation, an initial set of these properties can be captured, which is updated and later extended as the system evolves. This information is part of the Knowledge Base and serves as the input for the respective adaptive monitoring process at runtime. Additionally, a user interface can be used, e.g., a dedicated monitoring dashboard for inspecting and modifying the configuration at runtime, adding new properties, and triggering monitor optimization.
- **Cost Manager:** Once an initial configuration is specified, it is handed over to the Cost Manager. Before adaptive monitors

can be generated, the monitoring configuration needs to be processed and checked for consistency to ensure that no conflicting information is provided, e.g., that only unique properties are specified, or that all relevant information regarding the runtime checks is present (such as required properties and restrictions). The configuration is then transformed into a cost-benefit matrix serving as the input for the optimizer. This information is to be considered as highly sensitive for itself, and is here assumed to be under special protection and integrity checking (yet another monitoring aspect, i.e., "self-monitoring").

• **Cost-Benefit Optimizer:** For the optimization of the monitoring frequencies, we rely on a game-theoretic approach [34].

We assume a hypothetical zero-sum game between two players: player 1 is the defender, seeking to optimize the frequencies $x_1, \dots, x_n$ of n properties to perform the runtime checks $\psi_1, \dots, \psi_n$. Player 2 is either "nature" (an irrational adversary) or a rational entity (an attacker) with – here unknown – incentives. In either case, player 2 modifies properties whose changes we can detect. This gives it the corresponding strategies $A_1, \dots, A_n$ (we adopt a one-to-one correspondence here, since we can only protect against what we can anticipate). The modeling then results in a square matrix $\mathbf{A} \in \{0,1\}^{n\times n}$, with row strategies being runtime checks ψ_i, and column strategies being changes of the properties in the sets A_j. The game's saddle point value $val(\mathbf{A})$ is then a worst-case lower-bound on the payoff (provable from the properties of Nash equilibria in zero-sum games [34]), as the optimal "expected success" (= *probability*) to detect an unwanted change, under all possible event patterns covered by the properties being checked.

Computing $val(\mathbf{A})$ is a matter of solving a linear program (1). In this problem, we have $\mathbf{A} = (a_{ij}) \in \mathbb{R}^{n\times n}, v = val(\mathbf{A}) \in \mathbb{R}$ and $\mathbf{x} = (x_1, \dots, x_n)$:

$$\left.\begin{array}{lrclcl} \max_{\mathbf{x}} v & & & & & \\ \text{s.t.} & v & \leq & \sum_{j=1}^{n} a_{jk} x_j, & & k = 1, \dots, n \\ & \sum_{i=1}^{n} x_i & = & 1, & & \\ 0 \leq \underline{r}_i & \leq & x_i & \leq & \overline{r}_i, & j = 1, \dots, n. \end{array}\right\} \quad (1)$$

where the last two sets of constraints are the conditions imposed on the minimum required and maximum possible check frequency. The linear program has a feasible solution if $\sum_{i=1}^{n} \underline{r}_i < 1$. This solution is itself a sensitive part of the system configuration and hence also needs to be put under monitoring. This part is solvable by, for example, hardware protection (e.g., smartcards) and standard cryptographic techniques, such as key-dependent checksums or digital signatures, as long as there is no change to the linear program itself. If so, then more sophisticated mechanisms for updating may need to be introduced. For example, sanitizable signatures that allow only a designated party to change parts of a digital document, here the linear program specification, without invalidating cryptographic signatures attached to it. We leave the details here aside, as the cryptographic protection of the linear program and its solution is not the core concern of this work (we refer to previous work in this space [35] for cryptographic possibilities).

• **Monitoring Executor:** Once the optimizer provides the property frequencies, this information needs to be distributed to the monitors. It is used to set the frequencies at which data is collected and distributed, e.g., to a rule engine performing the runtime checks. This optimization can be performed when the system is initially configured, but also when costs or benefits change at runtime or when user preferences are updated.

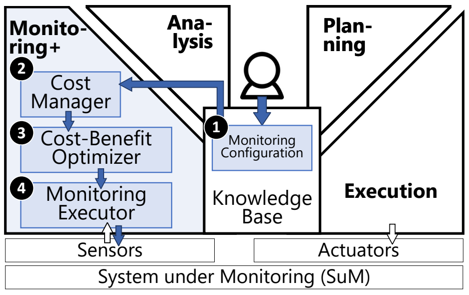


Fig. 2: Main components of the CBAAM MAPE-K Extensions

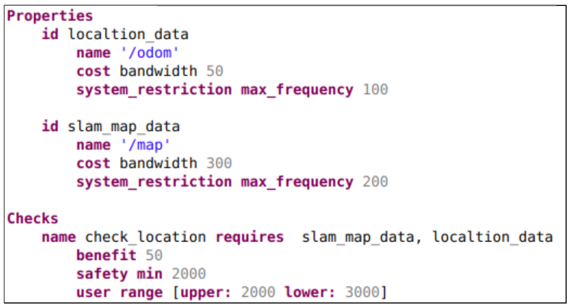
With the integration of CBAAM and its adaptation mechanism, we can automatically determine and update monitoring frequencies and automatically update the respective monitors.

IV. Prototype and Evaluation

As an initial end-to-end proof of concept, we created a complete software toolchain that allows defining the CBAAM Monitoring Configuration in an editor and automatically generates monitoring code for a Robot Operating System (ROS)-based [36] robot system, a Cost-Benefit Optimizer that determines optimal monitoring frequencies, and a rule engine that performs runtime checks.

To provide a simple way for specifying properties, runtime checks, restrictions, as well as costs and benefits, we designed a Domain Specific Language (DSL) using the Eclipse Xtext framework [37]. The editor (cf. Fig. 3) allows creating the respective *Monitoring Configuration*, specifying relevant properties, cost, and benefits as well as restrictions (specified in ms in the DSL) which are then transformed into a JSON representation. The *Cost Manager* and *Cost-Benefit Optimizer* are implemented in Python, and we use the `pulp` linear programming module for the optimization. The resulting frequencies provided by the optimizer are then used to generate monitors for a ROS-based application, controlling and monitoring TurtleBot3 robots [38]. ROS topic subscriptions and caching of the monitoring data is automatically generated and configured, and monitoring data is then distributed using Message Queuing Telemetry Transport (MQTT) and the ROS bridge package. Finally, for performing runtime checks, we use the Drools Rule Engine [39] that receives data via MQTT broker and executes checks (however, any other constraint engine, such as Complex Event Process could be employed).

To assess the feasibility of our approach, we have created a number of different monitoring configurations in the DSL and used them to perform the optimization and generate monitors for the TurtleBot. In total, we created 5 distinct configurations, each containing 10 properties, using 10 different ROS topics from the TurtleBot, and a corresponding runtime check for each



```
Properties
    id localtion_data
        name '/odom'
        cost bandwidth 50
        system_restriction max_frequency 100

    id slam_map_data
        name '/map'
        cost bandwidth 300
        system_restriction max_frequency 200

Checks
    name check_location requires  slam_map_data, localtion_data
        benefit 50
        safety min 2000
        user range [upper: 2000 lower: 3000]
```

Fig. 3: XText DSL for specifying the Monitoring Configuration.

property (we randomly assigned 1-2 properties to a check). Additionally, we specified restrictions (based on topic frequencies and user preferences) and assigned costs and benefits to each topic (guided by the size of the messages from each topic). We executed our framework in a Gazebo simulation environment, simulating the robot and using Simultaneous Localization and Mapping (SLAM) navigation, to verify that runtime data was indeed collected properly. Each run (for each configuration) was executed 3 times, lasting 10 minutes with an additional 60 seconds warm-up time. For each run, we collected and validated the generated frequencies and number of messages (i.e., properties) that were monitored and set to the runtime checks for evaluation and we report average values over the three runs. Furthermore, we performed an initial baseline run, not restricting any of the property frequencies, resulting in an average of 220,000 properties being monitored within the 10-minute run. As some ROS topics provide data at a frequency of $> 10\,\text{Hz}$, this is not surprising, and causes a large number of messages (and in turn resources being used). With our adaptive monitoring approach, we recorded between $\sim 25,000$ and 126,000 messages per run (depending on the configuration), significantly reducing the number of messages, and the resources used for monitoring when configured with different costs/benefits. Additionally, if a property is not part of any runtime check (i.e., its benefit is 0), it is automatically excluded from the monitoring configuration during optimization and does not need to be deactivated manually. We validated the results (messages and generated frequencies) from all runs to ensure that the optimization process worked indeed as desired.

The results show that CBAAM enables an effortless specification of costs and benefits using our DSL, that linear optimization can be performed with no significant delay, and that monitoring code can be automatically generated.

V. Research Roadmap and Conclusion

Our proof of concept implementation has confirmed that specifying properties, runtime checks, and their associated costs & benefits can be used to automatically optimize the frequency of data collection and subsequent runtime checks. While the prototype covers an end-to-end implementation, our ongoing work focuses on three areas to extend our CBAAM framework:

• **Dynamic Cost-Benefit Estimation:** Currently, costs and benefits are statically collected in our Monitoring Configuration based on estimates (e.g., used bandwidth or severity of a runtime check failure). Approaches such as time series prediction [40] have been used to forecast quality of service (QoS) attributes over time, the entirety of these techniques are known as predictive analytics and predictive maintenance. This information can be used to (1) dynamically adjust the benefits of certain runtime checks, e.g., which are less likely to fail, and re-optimize at runtime based on the collected data. Furthermore, collected runtime information can also be used to (2) increase accuracy of defined costs (e.g., by analyzing bandwidth or CPU consumption, and adjusting costs at runtime). The game-theoretic model herein offers the additional appeal of allowing for online learning, since if the system will always "best-adapt" its monitoring frequencies to the observed system status (i.e., the actions of the adversarial player 2 in terms of the game model), it is known that the process will converge to the true optimum (fictitious play [41]). This offers a lightweight method of solving the linear program even with very limited resources, at the cost of generally slow convergence to the optimum. Thus, starting from an optimum that is pre-computed at design time, and adapting this online during run-time can unify the advantage of only light computational requirements for the devices and adaptivity of the cost-benefit tradeoff for security.

• **Monitoring Configuration Consistency:** To ensure scalability and provide support for a large number of properties, runtime checks, and restrictions, automated validation of the monitoring configuration is required. This ensures that no conflicting properties exist, or that no invalid ranges or restrictions are defined. By using the Xtext framework, we can specify additional constraints on the monitoring configuration using the object constraint language (OCL) [42], [43]. This will allow instant validation while creating the configuration and immediate feedback to the user when an invalid configuration is specified in the DSL. Furthermore, providing context assist, e.g., by automatically collecting available properties from a ROS instance alongside frequency restrictions, or extracting properties from runtime checks further automates the task of creating and maintaining the Monitoring Configuration.

• **User Feedback & Runtime Updates:** Finally, we work on active user engagement at runtime. While the configuration currently is static, we are planning on instantiating the configuration at runtime providing a runtime (monitoring) model of the SuM that is used to provide feedback to the user about the system and runtime checks. Users can adjust their preferences at runtime. For example, runtime information about the location of a UAV and mission data may be more important in certain contexts and hence should be updated more frequently. This information then feeds back into the CBAAM configuration and triggers an update and re-optimization of the frequencies.

Once fully implemented, we are committed to making our framework publicly available on GitHub.

Acknowledgments

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, and the Linz Institute of Technology (LIT-2019-7-INC-316).

References

[1] A. Stocco, M. Weiss, M. Calzana, and P. Tonella, "Misbehaviour prediction for autonomous driving systems," in *Proc. of the ACM/IEEE 42nd Int'l Conf. on Software Engineering*. ACM, 2020, pp. 359–371.

[2] J. Cleland-Huang, M. Vierhauser, and S. Bayley, "Dronology: an incubator for cyber-physical systems research," in *Proc. of the 40th Int'l Conf. on Software Engineering: New Ideas and Emerging Results Track*. IEEE, 2018, pp. 109–112.

[3] V. V. Klemas, "Coastal and environmental remote sensing from unmanned aerial vehicles: An overview," *Journal of Coastal Research*, vol. 31, no. 5, pp. 1260–1267, 2015.

[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 *Proc. of the 18th IEEE Int'l Requirements Engineering Conf.* IEEE, 2010, pp. 95–103.

[5] E. Zavala, X. Franch, and J. Marco, "Adaptive monitoring: A systematic mapping," *Inf. and Software Technology*, vol. 105, pp. 161–189, 2019.

[6] P. Casanova, D. Garlan, B. Schmerl, and R. Abreu, "Diagnosing unobserved components in self-adaptive systems," in *Proc. of the 9th Int'l Symposium on Software Engineering for Adaptive and Self-Managing Systems*. ACM, 2-3 June 2014, pp. 75–84.

[7] T. Brand and H. Giese, "Generic adaptive monitoring based on executed architecture runtime model queries and events," in *Proc. of the 13th Int'l Conf. on Self-Adaptive and Self-Organizing Systems*. IEEE, 2019, pp. 17–22.

[8] S. Kurowski and S. Frings, "Computational Documentation of IT Incidents as Support for Forensic Operations," in *Proc. of the 6th Int'l Conf. on IT Security Incident Management & IT Forensics (IMF)*. IEEE Computer Society Press, 2011, pp. 37–47.

[9] B. Breiling, B. Dieber, and P. Schartner, "Secure communication for the robot operating system," in *2017 Annual IEEE Int'l Systems Conf. (SysCon)*. Montreal, QC, Canada: IEEE, 2017, pp. 1–6.

[10] B. Ramasubramanian, A. Clark, L. Bushnell, and R. Poovendran, "Secure Control under Partial Observability with Temporal Logic Constraints," in *Proc. of the 2019 American Control Conf. (ACC)*. Philadelphia, PA, USA: IEEE, Jul. 2019, pp. 1181–1188.

[11] H. H. Müller and C. A. Castro, "Genetic algorithm-based phasor measurement unit placement method considering observability and security criteria," *IET Generation, Transmission & Distribution*, vol. 10, no. 1, pp. 270–280, Jan. 2016.

[12] N. Boumkheld, S. Panda, S. Rass, and E. Panaousis, "Honeypot Type Selection Games for Smart Grid Networks," in *Decision and Game Theory for Security*, T. Alpcan, Y. Vorobeychik, J. S. Baras, and G. Dán, Eds. Cham: Springer International Publishing, 2019, pp. 85–96.

[13] R. C. Mendez, D. Dresscher, and J. Broenink, "Power and energy communication services for control-software models," in *Proc. of the 3rd IEEE/ACM Int'l Workshop on Robotics Software Engineering*. IEEE, 2021, pp. 55–62.

[14] S. Parra, S. Schneider, and N. Hochgeschwender, "Specifying qos requirements and capabilities for component-based robot software," in *Proc. of the 3rd IEEE/ACM Int'l Workshop on Robotics Software Engineering*. IEEE, 2021, pp. 29–36.

[15] J. E. Fischer, C. Greenhalgh, W. Jiang, S. D. Ramchurn, F. Wu, and T. Rodden, "In-the-loop or on-the-loop? interactional arrangements to support team coordination with a planning agent," *Concurrency and Computation: Practice and Experience*, vol. 33, no. 8, p. e4082, 2021.

[16] Y. Lim, N. Pongsakornsathien, A. Gardi, R. Sabatini, T. Kistan, N. Ezer, and D. J. Bursch, "Adaptive human-robot interactions for multiple unmanned aerial vehicles," *Robotics*, vol. 10, no. 1, 2021.

[17] M. J. Van Der Donckt, D. Weyns, M. U. Iftikhar, and R. K. Singh, "Cost-benefit analysis at runtime for self-adaptive systems applied to an internet of things application," in *Proc. of the 13th Int'l Conf. on Evaluation of Novel Approaches to Software Engineering*. SCITEPRESS, 2018, pp. 478–490.

[18] I. Gerostathopoulos, C. Raibulet, and E. Alberts, "Assessing self-adaptation strategies using cost-benefit analysis," in *Proc. of 44th Int'l Conf. on Software Engineering: Companion Proceedings*. ACM, 2022.

[19] J. C. Moreno, A. Lopes, D. Garlan, and B. Schmerl, "Impact models for architecture-based self-adaptive systems," in *Proc. of the Int'l Conf. on Formal Aspects of Component Software*. Springer, 2014, pp. 89–107.

[20] B. Dieber, S. Kacianka, S. Rass, and P. Schartner, "Application-level security for ROS-based applications," in *Proc. of the 2016 IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*, Oct. 2016, pp. 4477–4482.

[21] B. Dieber, B. Breiling, S. Taurer, S. Kacianka, S. Rass, and P. Schartner, "Security for the Robot Operating System," *Robotics and Autonomous Systems*, vol. 98, pp. 192–203, 2017.

[22] P. Arcaini, E. Riccobene, and P. Scandurra, "Modeling and analyzing mape-k feedback loops for self-adaptation," in *Proc. of 10th Int'l Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE, May 2015, pp. 13–23.

[23] J. Kephart and D. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[24] K. Mahbub and G. Spanoudakis, "A framework for requirents monitoring of service based systems," in *Proc. of the 2nd Int'l Conf. on Service Oriented Computing*. ACM, 2004, pp. 84–93.

[25] S. Fickas and M. S. Feather, "Requirements monitoring in dynamic environments," in *Proc. of 1995 IEEE Int'l Symposium on Requirements Engineering*. IEEE, 1995, pp. 140–147.

[26] L. Baresi, S. Guinea, R. Kazhamiakin, and M. Pistore, "An integrated approach for the run-time monitoring of BPEL orchestrations," in *Proc. of the Conf. on a Service-Based Internet*. Springer, 2008, pp. 1–12.

[27] A. Van Hoorn, J. Waller, and W. Hasselbring, "Kieker: A framework for application performance monitoring and dynamic software analysis," in *Proc. of the 3rd ACM/SPEC Int'l Conf. on Performance Engineering*. ACM, 2012, pp. 247–248.

[28] H. Eichelberger and K. Schmid, "Flexible resource monitoring of java programs," *Journal of Systems and Software*, vol. 93, pp. 163–186, 2014.

[29] J. Cámara, G. Moreno, and D. Garlan, "Reasoning about human participation in self-adaptive systems," in *Proc. of the 10th IEEE/ACM Int'l Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE, 2015, pp. 146–156.

[30] L. Baresi, D. Y. X. Hu, G. Quattrocchi, and L. Terracciano, "Neptune: Network-and gpu-aware management of serverless functions at the edge," in *Proc. of the 17th Symposium on Software Engineering for Adaptive and Self-Managing Systems*. ACM, 2022.

[31] R. Wohlrab and D. Garlan, "A negotiation support system for defining utility functions for multi-stakeholder self-adaptive systems," *Requirements Engineering*, pp. 1–20, 2022.

[32] R. Trestian, O. Ormond, and G.-M. Muntean, "Enhanced power-friendly access network selection strategy for multimedia delivery over heterogeneous wireless networks," *IEEE Transactions on Broadcasting*, vol. 60, no. 1, pp. 85–101, 2014.

[33] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[34] S. Rass, S. Schauer, S. König, and Q. Zhu, *Cyber-Security in Critical Infrastructures: A Game-Theoretic Approach*. SpringerNature, 2020.

[35] S. Rass and D. Slamanig, *Cryptography for Security and Privacy in Cloud Computing*. Artech House, 2013.

[36] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source robot operating system," in *Proc. of the ICRA Workshop on Open Source Software*, vol. 3. Kobe, Japan, 2009, p. 5.

[37] Eclipse , "Eclipse Xtext - Language Engineering Framework," https://www.eclipse.org/Xtext, 2022, [Last accessed: 25-05-2022].

[38] Robotis, "TurtleBot E-Manual," https://emanual.robotis.com/docs/en/platform/turtlebot3, 2022, [Last accessed: 25-05-2022].

[39] M. Proctor, "Drools: a rule engine for complex event processing," in *Proc. of the Int'l Symposium on Applications of Graph Transformations with Industrial Relevance*. Springer, 2011, pp. 2–2.

[40] A. Amin, L. Grunske, and A. Colman, "An automated approach to forecasting qos attributes based on linear and non-linear time series modeling," in *Proc. of the 27th IEEE/ACM Int'l Conf. on Automated Software Engineering*. IEEE, 2012, pp. 130–139.

[41] J. Robinson, "An Iterative Method of Solving a Game," *The Annals of Mathematics*, vol. 54, no. 2, p. 296, Sep. 1951.

[42] T. Arendt, G. Taentzer, and A. Weber, "Quality assurance of textual models within eclipse using ocl and model transformations," in *OCL@MoDELS*, 2013, pp. 1–12.

[43] J. Holtmann, J. Meyer, and M. von Detten, "Automatic validation and correction of formalized, textual requirements," in *Proc. of the IEEE 4th Int'l Conf. on Software Testing, Verification and Validation Workshops*. IEEE, 2011, pp. 486–495.