Towards Unified Field-Testing and Monitoring for Safe and Secure Robotic Applications

Marco Stadler* Stefan Biffl† Michael Vierhauser‡ Johannes Sametinger*
*LIT SCSL / Inst. of Business Informatics – Software Eng., Johannes Kepler University Linz, Austria
Email: {firstname.lastname}@jku.at
†CDL-SQI, Institute of Information Systems Engineering, TU Wien, and CDP, Vienna, Austria
Email: {firstname.lastname}@tuwien.ac.at
‡Department of Computer Science, University of Innsbruck, Austria
Email: {firstname.lastname}@uibk.ac.at

Abstract*—Problems and failures that emerge in Cyber-Physical Systems (CPSs), particularly in robotic applications, may originate from various sources, including software bugs, security incidents, hardware malfunctions, and human errors. As robotic systems are deployed in various domains and application contexts, such as manufacturing sites, shop floors, agriculture, and autonomous vehicles, ensuring their safe and secure operation is a crucial aspect. While high-fidelity simulations are frequently used to validate system behavior and to perform tests, the "simulator-to-reality gap" presents significant challenges, requiring additional field testing to validate a system under realistic conditions. As simulations alone are insufficient for performing comprehensive testing and for ensuring adherence to both functional and non-functional requirements, real-world field testing helps to alleviate these issues. However, compared to well-established unit testing approaches, field testing typically is still a rather *ad hoc* process, with insufficient support from tools and frameworks. Field tests often heavily rely on human observations, hence risking overlooking critical issues. There is a pressing need for structured, guided field-testing processes combined with adaptive runtime monitoring to capture the data required for effective error diagnosis and analysis. This paper introduces initial concepts for the *Smart Unified Runtime Monitoring Infrastructure for Guided Field-Testing* (SMURF) framework designed for robotic applications, combining structured test execution with automated, adaptable monitors, to ensure the efficient collection of data required for post-test analysis. Building on prior efforts in drone field-testing frameworks, we extend our scope to identify essential features for testing and monitoring ROS-based systems. Future work shall further refine this process and implement a practical framework to support developers and testers in achieving reliable, safe, and secure robotic operations.

***Index Terms*—Field Testing, Robotic Application, Security, Safety**

I. Introduction

Robotic systems operate in a wide variety of different application domains to perform diverse tasks. Examples range from robotic systems that move goods or perform manufacturing tasks on a shop floor [1], small Uncrewed Aerial Systems (sUAS) that perform search-and-rescue operations [2], to autonomous vehicles operating on roadways [3]. In these cases, Cyber-Physical Systems (CPSs) operate in an emerging and sometimes partially unknown environment, closely interacting with human roles, e.g., workers on the shop floor or operators of sUAS. CPSs in general, and robotic applications in particular, offer significant benefits and advancements across these diverse domains, but face challenging issues and failures, which can be attributed to a variety of different sources, including *endemic factors* such as software-related bugs, hardware-related malfunctions, or human and operator errors [4], [5], as well as *intrusive factors* like the unprecedented surge in cyber-attacks targeting CPSs and robotic systems [6]. Therefore, ensuring the correct and safe operation of CPSs has become a critical concern and reinforces the need to test these systems thoroughly. Commonly, such tests are performed through high-fidelity simulations – to validate system behavior in a safe environment, without the risk of damaging the environment, or causing harm to human operators or bystanders [7]. However, simulations alone are incapable of covering the full spectrum of test scenarios required, and there is still a well-recognized "simulator-to-reality gap" between virtual hardware and how real systems behave in the physical world [8]. Therefore, simulations are typically complemented with "real-world" field tests, where hardware and software are tested in a close-to-realistic environment to ensure that common scenarios are thoroughly tested under realistic conditions.

Human observations may easily overlook or miss important information. We are convinced that a combination of structured test execution is needed, complemented by automated and adaptable runtime monitoring. Data collected during runtime can be used to discover problems, root causes, and ultimately to detect and fix problems. In this paper, we describe initial ideas and concepts towards a *Smart Unified Runtime Monitoring Infrastructure for Guided Field-Testing*, a holistic process and infrastructure that shall guide developers and testers through the testing and operations process of robotic applications based on the *Robot Operating System* (ROS) which serves as a platform for a wide variety of different applications [9].

ROS-based systems require the ability (1) to be thoroughly tested in the field, and (2) to collect relevant runtime data at the same time. Collected runtime data can be used for performing error diagnosis and analysis of failed tests as a foundation for ensuring functional, safe and secure behavior. Based on previous work in creating a structured information system for performing system analysis and testing [10]–[12], and drone field tests [13], we broaden our scope and identify challenges for providing effective testing/monitoring support for ROS-based systems and key capabilities of a testing process and describe our conceptual solution. The remainder of this paper is organized as follows. Section II provides background and basic motivation for structured testing and discusses current issues and challenges pertaining to testing and monitoring robotic applications. Section III introduces the concept of a flexible adaptive testing and runtime monitoring infrastructure and how this can be used in the context of ROS-based systems. Finally, Section IV lays out our ongoing and planned research.

II. Background & Motivation

Defining, selecting, preparing, and executing test cases for any CPS and robotic application requires in-depth knowledge of the system, its use cases, and application scenarios. In "traditional" software testing, test cases can often be generated automatically, while testing both hardware and software is much more complex and labor-intensive, particularly, when systems are tested in the field [14].

Even with meticulous planning, a certain level of uncertainty and adaptability must be accounted for. For instance, environmental factors like wind or lighting conditions are not always fully reproducible, requiring testers to maintain some flexibility in addressing these variables. We have identified key features (**F1** - **F5**) needed for a comprehensive testing/monitoring framework.

F1 – Structured definition of test cases allows us to reuse and re-execute test cases when, for example, new functionality is added. Based on our own experience, in the domain of sUAS testing [7], most field tests make heavy use of *ad hoc* approaches such as checklists or spreadsheets with key testing aspects being sketched out. In our previous work, we laid out the concept of an "information-system guided" testing process, driven by specific use cases [13].

F2 – A comprehensive yet easy notation, e.g., a graphical notation is needed to capture all relevant parts of the engineering, development, testing and subsequently the operation phase of a system. To integrate field-testing activities into the software engineering process, it is essential not only to model and describe test cases but also to capture actors, artifacts, workflows, and relevant resources.

F3 – The integration of runtime monitors and runtime data into the testing process is a key objective of this work. We observed that solely collecting testing data from the user has several drawbacks. First, it puts additional burden on the testers during field testing to, e.g., confirm that a robot or drone has reached a certain test location, or has performed an action as specified. Second, it limits the data that can be collected, potentially losing important system data.

F4 – Facilitating traceability from test scenarios, to test steps, to runtime information helps to diagnose problems, identify root causes of a failure or potential security incident, and apply fixes or updates. *F3* facilitates dynamic data collection, for example, when a certain security event occurs (cf., Listing 1). To make optimal use of this data after a test has been executed, related events, test steps, and runtime data need to be grouped with trace links.

F5 – Testing, monitoring, and analysis processes need to be integrated into the life cycle, sharing information and complementing each other. In this scenario, monitoring data and observed deviations can inform the creation of new/updated test cases. The test cases make use of runtime monitors to enhance field-testing activities and aid in error diagnosis and quality assurance of the system.

While individual solutions have been explored in the past, for example, Behavior-driven development (BDD) to automate (parts) of the testing process [15], current (security) solutions remain limited in scope, typically focusing on isolated issues without supporting a cohesive, life cycle-wide framework. These approaches do not address a comprehensive framework that guides developers, testers, and quality assurance throughout the life cycle of a CPS. In the following, we present our initial idea for such a framework and lay out future research.

III. The Smart Unified Runtime Monitoring Infrastructure for Guided Field-Testing

Based on the features we deem as crucial for a holistic testing and monitoring approach, we have derived an initial version of a conceptual infrastructure for combining field-testing and monitoring of robotic applications. As part of the different phases, we leverage existing methods for defining use cases, creating scenarios and analyzing test results, and combine them in an integrated life cycle.

A. Proposed Framework

Fig. 1 provides a high-level overview of our envisioned *Smart Unified Runtime Monitoring Infrastructure for Guided Field-testing (SMURF)*. Following the DevOps paradigm, where development and operation are intertwined, we also aim to leverage capabilities from both development/(field-)testing and operation/monitoring to complement information collection and analysis capabilities. The former is used to iteratively test and validate the System under Monitoring (SuM) during development while the latter is used in production to monitor the system. The field test life cycle (①, ②, and ③ in Fig. 1) is structured into three core phases:

• **Test Definition**: During the test definition phase, we use Use Case Descriptions [16] to define both normal course, and alternate scenarios for different field tests that can be executed.

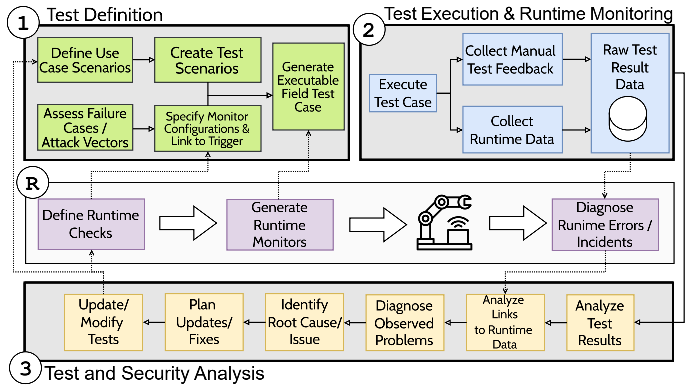
1
Test Definition
Define Use Case Scenarios
Create Test Scenarios
Generate Executable Field Test Case
Assess Failure Cases / Attack Vectors
Specify Monitor Configurations & Link to Trigger
2
Test Execution & Runtime Monitoring
Execute Test Case
Collect Manual Test Feedback
Collect Runtime Data
Raw Test Result Data
R
Define Runtime Checks
Generate Runtime Monitors
Diagnose Runtime Errors / Incidents
Update/ Modify Tests
Plan Updates/ Fixes
Identify Root Cause/ Issue
Diagnose Observed Problems
Analyze Links to Runtime Data
Analyze Test Results
3
Test and Security Analysis

Fig. 1: High-level overview of SMURF.

This enables the test steps to be captured in a structured fashion to thoroughly consider each use case while enabling the framework to later convert these user specifications into executable tests (cf. *F1*). To add further detail, these scenarios can be enriched using additional modeling tools or Domain-Specific Languages, incorporating information about each use case, such as stakeholders to be involved, or additional artifacts to consider. Earlier work in this area has shown that use cases, combined with a BDD can be used to specify test cases for different application scenarios [13].

However, while this has proven useful to capture simple drone testing cases, particularly for more complex ones, additional concepts and features, are required to cover the full spectrum of tests, and, more importantly, collect all necessary information during test execution. Specifically security-related test cases, need to be captured and documented in a structured and easy-to-use manner (cf. *F2*). Security scenarios focus on recording security-relevant data and outlining how the monitoring infrastructure can adapt during security incidents. This adaptation is facilitated by mapping the monitoring endpoints to triggers (e.g., detection of abnormal behavior). By integrating these two pillars – normal-course scenarios and failure/security scenarios – the infrastructure generates an executable field test case utilized in the second phase.

• **Test Execution & Runtime Monitoring**: Once respective test cases are created, they need to be executed in the field. This is done as part of the actual field test, where test data is collected. To capture all relevant information, manual test data alone is insufficient. For instance, determining if a robot reached its designated position in the test scenario or if a drone took off correctly based on the tester's observations requires additional validation. Particularly when test cases do not go as planned, i.e., a failure scenario is activated, and undesired behavior is observed, additional data needs to be collected to ensure that an occurring problem can actually be properly diagnosed, its root cause identified, and ultimately fixed. Therefore, it is crucial that runtime data is collected from the monitored systems, along with manual test feedback (cf. *F3*). The collected data is stored in a result repository, which serves as the basis for post-test analysis. Particularly for ROS-based systems, existing solutions can be leveraged, for example, rosbag [17] capturing and visualizing test executions.

The monitoring operations life cycle (Ⓡ in Fig. 1) is used as part of the field test life cycle but produces monitors to be used in production.

Listing 1: Instance of the *Given.. When... Then...* scenario template.

```
Given: a robot receives commands on the velocity
    from a control node.
When: a command with an abnormal or unexpected
    parameter (e.g., extreme velocity) is detected.
Then: increase monitoring frequency of velocity
    topic (/cmd_vel) to capture movement tampering.
```

The *Test Definition* phase generates runtime monitors. We can leverage these ready-to-use and iteratively tested runtime monitors to be used in production to adhere to system requirements. For the next development field test life cycle, we can incorporate the results (data, incidents, errors, etc.) into the field test life cycle.

• **Test and Security Analysis**: In the final phase, the test result data is used to verify whether all previously defined scenarios were successfully completed. Because each scenario is linked to its corresponding monitoring endpoint, this verification can be conducted efficiently (cf. *F4*). The analysis allows diagnosis of issues within the system and tracing them back to their root causes. Consequently, the system can be updated or fixed based on the test findings. Following an update, the test cases can be adjusted to reflect the new system structure, and the testing cycle restarts at phase one (cf. *F5*).

B. *Implementation Concept*

To establish our unified field-testing and monitoring infrastructure, we present the following conceptual prototypical implementation.

For the *Test-Definition*, we combine the well-established *Given–When–Then* BDD-notation [15] with the Use Case notation to define concise scenarios with additional monitoring/testing related information attached. LISTING 1 contains an instance of a security scenario specification using the *Given... When... Then...* template. This specification is further enriched by modeling other components of the SuM's environment. This can be achieved, for example, using the UML modeling tool "UMLet". UMLet facilitates modeling both the stakeholders and the process steps involved in a scenario, making the life cycle also accessible to non-technical users. An example of an UMLet model for the scenario outlined in LISTING 1 can be found in Fig. 2. In case of a failure/attack during execution there is a direct link to the stakeholders and processes involved. For this running use case, a stakeholder involved is the *Shop Floor Worker*.

The resulting monitoring configuration consolidates these two worlds into a single configuration. We then map the configurations with the respective monitoring endpoints. For instance, in case of LISTING 1, the monitoring property of interest is the `/cmd_vel` ROS topic, which is then mapped to the scenario to generate the respective Python/C++ ROS monitors (i.e., the executable).

The monitor is then executed in the respective system environment, e.g., the target ROS network. During this runtime part of the field test, data on the monitoring properties of

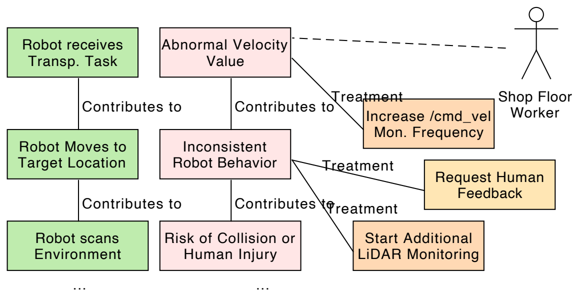
Robot receives Transp. Task
Abnormal Velocity Value
Shop Floor Worker
Contributes to
Contributes to
Treatment
Increase /cmd_vel Mon. Frequency
Robot Moves to Target Location
Inconsistent Robot Behavior
Treatment
Request Human Feedback
Contributes to
Contributes to
Treatment
Robot scans Environment
Risk of Collision or Human Injury
Start Additional LiDAR Monitoring
...
...

Fig. 2: Root cause analysis of one of the security scenarios. This includes the expected normal course scenario (green nodes), potential deviations and undesired behavior (red nodes), and resulting adaptations in data collection (orange nodes).

interest is collected and persisted. For the security/failure scenarios, this phase also involves the simulation of attacks, to check whether the field test configuration triggers a correct reaction of the monitors. For the security use case velocity scenario, this means attacking the system by publishing randomly seeded threshold-exceeding velocity commands into the ROS network.

Upon finishing the field test, the *Test and Security Analysis* begins by checking the gathered monitoring data and confirming that the SuM responded to the field tests as expected. If this is not the case, it is possible to dig deeper into the gathered data. Since there is a link in each field test configuration and monitoring property involved, one can simply filter based on the configuration specification and trace an error back to its root cause. For the running use case, we could filter our raw data based on the knowledge, that the `/cmd_vel` ROS topic is involved, and inspect collected data published on this topic. Once the cause for the unexpected behavior is found, the system can be updated/fixed and the field test repeated to check whether the changes made indeed solved the problem.

Once the SuM passes all field tests, the runtime monitors generated as part of the infrastructure can be deployed in the production setting.

IV. Roadmap

We are currently working on creating an end-to-end implementation of SMURF. In conjunction, we intend to conduct an exhaustive evaluation: The ROS Gazebo simulation software and tangible ROS-based hardware (TurtleBot 4) are employed to implement a case study using ROS 2 [9]. Additionally, we intend to enhance SMURF's automation capabilities and provide a more structured/formal way of specifying test cases and missions. For example, robotic mission pattern by Menghi *et al.* [18] can be a viable aspect to be incorporated in our scenarios. Our objective is to fully automate the process. By utilizing the user-friendly field test modeling in Phase 1, we can generate runtime monitors and map the data in the analysis phase to the data, as well as automatically involving the relevant stakeholders in a specific use-case scenario. This also includes further extending the aspect of behavior-driven requirements, which has been explored for robotic applications [19].

Lastly, beyond simply collecting data for field testing, we aim to enable true self-adaptation within the system. For example, the system can enforce specific security policies whenever a monitor detects anomalous data, thereby initiating adaptive responses in real-time.

V. Conclusion

In this paper, we presented our initial efforts towards combining field testing and monitoring for safe and secure robotic applications, identified five essential features a potential solution must provide, and demonstrated how our novel approach can be implemented in a proof-of-concept framework. We further outline ongoing research on security, test specification, and automated adaptive monitor generation. Our next steps include the completion of the implementation with an automation and self-adaptation process.

Acknowledgments

This work has partially been supported by the LIT Secure and Correct Systems Lab, funded by the State of Upper Austria.

References

[1] T. Blender and C. Schlegel, "Dynamic allocation of service robot resources to an order picking task considering functional and non-functional properties," in *Proc. of the 4th Int'l WS on Robotics Software Engineering*, 2022, pp. 25–32.

[2] T. Chambers, J. Cleland-Huang, and M. Vierhauser, "Self-adaptation of loosely coupled systems across a system of small uncrewed aerial systems," in *Proc. of the 12th ACM/IEEE Int'l WS on Software Engineering for Systems-of-Systems and Software Ecosystems*, 2024, pp. 37–44.

[3] C. Gao, G. Wang, W. Shi, Z. Wang, and Y. Chen, "Autonomous driving security: State of the art and challenges," *IEEE Internet of Things Journal*, vol. 9, no. 10, pp. 7572–7595, 2022.

[4] F. Zampetti, R. Kapur, M. Di Penta, and S. Panichella, "An empirical characterization of software bugs in open-source Cyber–Physical Systems," *Journal of Systems and Software*, vol. 192, p. 111425, Oct. 2022.

[5] M. Zhang, C. Li, Y. Shang, H. Huang, W. Zhu, and Y. Liu, "A task scheduling model integrating micro-breaks for optimisation of job-cycle time in human-robot collaborative assembly cells," *International Journal of Production Research*, vol. 60, pp. 4766–4777, 2022.

[6] C. Kumar, S. Marston, and R. Sen, "Cyber-physical systems (cps) security: State of the art and research opportunities for information systems academics," *Communications of the Association for Information Systems*, vol. 47, no. 1, p. 678–696, 2020.

[7] A. Agrawal, B. Zhang, Y. Shivalingaiah, M. Vierhauser, and J. Cleland-Huang, "A requirements-driven platform for validating field operations of small uncrewed aerial vehicles," in *Proc. of the 31st Int'l Requirements Engineering Conf.* IEEE, 2023, pp. 29–40.

[8] E. Salvato, G. Fenu, E. Medvet, and F. A. Pellegrino, "Crossing the reality gap: A survey on sim-to-real transferability of robot controllers in reinforcement learning," *IEEE Acc.*, vol. 9, pp. 153 171–153 187, 2021.

[9] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, p. eabm6074, 2022.

[10] S. Biffl, D. Hoffmann, E. Kiesling, K. Meixner, A. Lüder, and D. Winkler, "Validating production test scenarios with cyber-physical system design models," in *Proc. Conf. on Business Inf.* IEEE, 2023, pp. 1–10.

[11] S. Biffl, S. Kropatschek, K. Meixner, D. Hoffmann, and A. Lüder, "Configuring and validating multi-aspect risk knowledge for industry 4.0 information sys." in *Proc. Conf. on Adv. Inf. Sys. Eng.* IEEE, 2024.

[12] R. Caldas, J. A. P. n. García, M. Schiopu, P. Pelliccione, G. Rodrigues, and T. Berger, "Runtime verification and field-based testing for ros-based robotic systems," *IEEE Trans. Softw. Eng.*, vol. 50, no. 10, p. 2544–2567, Oct. 2024.

[13] M. Vierhauser, K. Meixner, and S. Biffl, "Scenario-based field testing of drone missions," in *Proc. IEEE Int'l Conf. on Software Engineering and Advanced Applications*, 2024, pp. 1–8, arXiv:2407.08359.

[14] A. Arrieta, G. Sagardui, L. Etxeberria, and J. Zander, "Automatic generation of test system instances for configurable cyber-physical systems," *Software Quality Journal*, vol. 25, no. 3, pp. 1041–1083, 2017.

[15] D. Terhorst-North, "What's in a Story?" https://dannorth.net/whats-in-a-story/, 2007.

[16] T. Yue, L. C. Briand, and Y. Labiche, "Facilitating the transition from use case models to analysis models: Approach and experiments," *ACM Trans. on Softw. Eng. and Meth.*, vol. 22, no. 1, pp. 1–38, 2013.

[17] ros.org, "rosbag," https://wiki.ros.org/rosbag, 2025, [Last Accessed 01-01-2025].

[18] C. Menghi, C. Tsigkanos, P. Pelliccione, C. Ghezzi, and T. Berger, "Specification patterns for robotic missions," *IEEE Transactions on Software Engineering*, vol. 47, no. 10, pp. 2208–2224, 2019.

[19] M. Nguyen, N. Hochgeschwender, and S. Wrede, "An analysis of behaviour-driven requirement specification for robotic competitions," in *Proc. of the 5th RoSE Workshop.* IEEE, 2023, pp. 17–24.