Bug Finding in Dräger's Medical Devices: Field Experience with LoLa4D

Normann Decker[2], Martin Leucker[1], Torben Scheffel[2], Michael Schulz[2], and Daniel Thoma[1]

[1] Universität zu Lübeck, Lübeck, Germany
{leucker,thoma,scheffel}@isp.uni-luebeck.de
[2] Drägerwerk AG & Co. KGaA

Abstract. This paper reports on the experience gained by using the runtime verification tool LoLa4D for bug finding in Dräger's medical devices in the field for over thirteen years.

1 Introduction

Dräger (Drägerwerk AG & Co. KGaA) is a German technology company headquartered in Lübeck that develops, manufactures, and services products for medical and safety applications worldwide. Its portfolio spans life-support and anesthesia workstations, ventilators, patient monitoring systems, and related medical accessories, as well as gas detection, respiratory protection, and firefighting equipment for industrial and public-safety customers. Across these domains, Dräger's reputation rests on designing dependable devices for critical environments—operating rooms, intensive-care units, and emergency response—where reliability and traceability are non-negotiable.

Medical devices are among Dräger's core products, and they undergo rigorous verification and validation before shipment. Extensive bench tests, simulated clinical workflows, environmental stress evaluations, and formal release checks collectively drive a very low residual defect rate. Inevitably, however, a small number of issues only surface in practical use "in the field" where real-world variability exceeds what any test lab can comprehensively reproduce. These field issues fall into several recurring categories. Some arise from environmental conditions that deviate from the intended operating envelope (temperature, humidity, airflow, electromagnetic interference). Others emerge gradually through the ageing of components and normal wear, particularly in long-duty cycles. A portion reflect unmet maintenance requirements: filters not replaced on time, out-of-calibration sensors, or firmware left on an outdated revision. Finally, a small subset are uncaught software bugs that escaped pre-release testing. Importantly, certain problems are not intrinsic product defects but are instead contingent on uncontrolled customer-side conditions: for example, running a device outside its specified operating conditions or integrating it with third-party systems in unsupported ways. In all cases, quick, evidence-based diagnosis is essential: clinicians and technicians need to understand what happened, why, and how to prevent recurrence without compromising patient care.

Bug Finding To enable precise, retrospective analysis of behavior in the field, Dräger devices perform extensive logging. Log files are plain-text; each entry is a single line appended as events occur. Although line-oriented, the messages are structured: fields such as timestamp, subsystem, severity, identifiers, and payload can be parsed into columns, giving the logs an effective relational shape. This design keeps the logs both human-readable and machine-processable, allowing technicians to skim them quickly while also supporting automated analysis at scale.

When a problem is reported, Dräger follows a disciplined debugging model anchored in logs. A field technician's description, e.g. symptoms, context, timing, environmental observations, serves as the starting point. A tester or domain expert who understands the device's architecture converts that narrative into a concrete hypothesis: "Given these symptoms, the most plausible explanation is X, which would leave witness Y in the logs." The first validation step is to search for that witness: a distinctive message, code, or invariant violation that, if present, materially supports (or refutes) the hypothesis.

For many issues, a single log line suffices as a witness. Dräger uses a dedicated search tool that lets testers define patterns describing the line they are looking for based on regular expressions, typically combinations of keywords, field values, or simple predicates, for example "search for `AlarmManager: queue overflow` in the related device.". This approach is fast, reproducible, and easy to communicate. By having multiple regular expressions like that, the problem description and domain knowledge of the person analyzing the log, the person sooner or later determines the root cause of the problem, being it a bug, misuse or some external effect. Yet single-line search also has inherent limitations. It can produce false positives when a message template appears in benign contexts, and it struggles with failure modes that are fundamentally temporal or causal, i.e. where meaning emerges only from the order and timing of multiple events. Additionally, it may happen that a crucial single-line log message might be missing so that a simple pattern search approach based on single-line log messages fails entirely. To some degree LoLa4D may be able to compensate for such omissions that - more often than not - are only identified in retrospect.

Complex failures often unfold as sequences: a sensor signal degrades, a calibration check is skipped, a watchdog timer stretches, a buffer grows, and only then an alarm triggers. No single line is definitive; the pattern is the story told by several entries in relation to one another. Moreover, checks regarding timing constraints like that something occurred in a certain time span, are crucial to finding bugs, especially in regards to devices with hard real time requirements. In the current approach, checking time stamps is an additional manual effort. Recognizing this need for higher-order analysis, Dräger and the Institute of Software Engineering and Programming Languages of the University of Lübeck initiated a collaboration in 2012 that led to the development of the tool LoLa4D. Since then, LoLa4D has been used to express and detect complex temporal patterns across Dräger's device logs, turning raw line streams into verifiable, causal narratives about system behavior.

In this paper, we describe LoLa4D and the experience gained with using it in an industrial context.

2 LoLa4D

LoLa4D is custom implementation of the Lola approach pioneering stream runtime verification [1]. Lola is a lightweight runtime verification tool and specification language designed for monitoring system executions by evaluating stream-based properties over time. Instead of checking correctness only after the fact, Lola runs alongside a system (or over its logs) and continuously computes derived streams—such as counters, rates, or temporal predicates—from input signals like events, sensor values, or log fields. Specifications are written as declarative equations that define these output streams, making complex temporal relationships (ordering, windows, thresholds, absence/presence patterns) both expressive and readable. Because Lola processes inputs in a single pass with bounded memory where possible, it suits online monitoring and post-hoc analysis of large traces alike.

LoLa4D is an implementation of the Lola approach tailored for Dräger. It is implemented in Scala and has a direct interface to the logfile databases used for Dräger medical device log analysis. Properties are specified in a Scala-based DSL (Domain Specific Language) enriching the Lola formalism with the expressiveness of a functional programming language. Field logs can span days and millions of lines. LoLa4D is designed to stream through large log databases efficiently, evaluating patterns in one pass where possible. Every finding is traceable back to concrete log lines and timestamps.

3 Runtime Verification of Logfiles at Dräger

The classical approach analyzing logfiles based on regular expressions used at Dräger requires significant manual effort and is heavily dependent on the person analyzing and combining the clues correctly. Therefore, Dräger started a research project 13 years ago to improve log analysis by using Runtime Verification methods on their logs to facilitate a greater degree of automatization and to allow for precise specifications of misbehaviour.

Dräger's Logfiles As mentioned before, each line of a device logfile follows a specific structure. A log entry comprises fields like the timestamp when an event occurred, the associated part of the system and content fields that describe the event in detail. Events include alarms being raised, exceptions having occured, or the user having switched between certain device modes. Not all these types of events represent a failure, most merely document the behavior of the device.

Use Cases There are many cases where the usage of Runtime Verification methods helped to efficiently find the corresponding problem in a logfile. We want to highlight the most interesting ones here. It is important to note that, while some of the use cases were found by clients using the devices in the field, in none of use cases a patient could have been harmed or has been in danger. Most have been found during clients maintaining the device or were cases where the device did something odd or unpleasant which the client reported later, but which had no severance that it could have stopped the client from treating a patient properly or confusing the client. All problems occurred many years ago and are all fixed today.

Unwarranted Flow-Sensor Calibration Alarm This is a behaviour that occurred multiple times in the field. The calibration of a sensor was successful every morning, however, an unwarranted sensor calibration alarm arose claiming that this did not happen. Therefore, a recognition of faulty alerting for flow sensor calibration was necessary as this faulty alarms can fool the user that a sensor calibration is needed and the device is not usable. No cause was being found in the logs for some weeks until the device requirement from risk management was converted directly into a Lola specification to measure the minimal time between two events. With this, a faulty behaviour was being found quickly and lead to the discovery of a hardware defect of the sensor as well as a bug in the self test software of the device.

Dubious Ambient Pressure Measurements In rare cases, the abortion of a reset of the sensor lead to measuring too low environmental pressure. A Lola specification was written which uses existing correct numerical log data to compare with every pressure measurement within five seconds after a sensor calibration in the log under scrutiny. Various suspicious measurements have been found which allowed to further investigate the surrounding log entries which finally lead to the realization that this is a general problem that can be fixed via a software solution.

Log Integrity It was reported that in the debug log, error messages were included stating that the sequence of the events in the log is compromised, even though after investigation, everything was fine and the events in the log have been in the correct order. A Lola specification was used to look back from certain events to check causality of the order of events. It was shown that this is an effect that occurs quite often by analyzing many logs with Lola. Even though this is annoying, it was considered acceptable at that time and fixed with a later software version. LoLa4D helped to analyze this issue quickly by automatically analyzing many logs for this specific behaviour.

Alive-flag Failures Here, the reason for the very rare alive-flag failures arising after the device's start phase was not obvious. Such failures can potentially lead to a device restart, so it was critical. After some investigation, a relation to the patient category and chosen language was suspected. Therefore a Lola specification was written that outputs those values in a clean and structured way when error cases are found in the log. This helped getting a much better idea what the problem could be and to find out that the failure at hand was not able to cause a device restart.

Battery Problem: Alarms Wrong alarms regarding the battery could fool the user, so they issue a battery switch which may lead to high costs for Dräger. Therefore, we checked of functional correctness of battery related alarms by checking consistency between alarm state and measured values. We wrote a Lola specification to implement alarm logic for checking of measured values fit to risen alarm.

Battery Problem: Loading Problems with faulty loading functionality occurred, which could lead to batteries not working when they are needed. A Lola specification was written to check the minimal loaded capacity after a certain amount of time. It confirmed the assumption that there are problems with the batteries and made it easy to quickly analyze logs for this misbehaviour.

False Alarm Suppression In rare cases, an alarm was suppressed when it should not have been. Alarms are important as they notify the user that something is not right with the device. A Lola specification was used to check whether an alarm marked as suppressed in the logfile has been suppressed correctly by specifying alarm logic (values and timing) in Lola.

4 Experiences

Over 13 years of use, LoLa4D has proven uniquely valuable in Dräger's field investigations: it consistently helps analyzing rare, potentially high-impact findings, that elude conventional techniques. For the use cases mentioned above, the investigation before using LoLa4D had been ongoing for weeks or even months. In contrast, the answer was found quickly after starting an analysis with LoLa4D, either because it provided a hint or directly lead to the cause of the anomaly. Six of the seven use cases described in this paper were newly found and unexpected functional deviations in the system, one was a confirmation of a problem. Six of those were found in the field and would have lead to very high costs for Dräger or lowered the reputation if the root cause would have remained unknown or the analysis would have taken more time, because potentially service technicians would have had to switch multiple pieces of hardware or the software department would have tried different software solutions.

LoLa4D runs reliably even on multi-million-line logs, and is straightforward to apply to device log files once a specification exists. The principal drawback is the steep complexity of writing correct, maintainable Lola specifications; in practice, domain experts seeking analysis support (e.g. software developers, life-cycle engineers) do not author specs themselves. Instead, a small number of trained specialists translate hypotheses, formulated by domain experts based on case descriptions, into executable LoLa4D monitors. This division of labor works, but it limits scalability and slows knowledge transfer. Introducing curated *Lola specification patterns* (reusable, parameterized templates for common failure narratives) would lower the barrier for domain experts, reduce time-to-analysis, and make expert know-how more broadly accessible without sacrificing rigor.

5 Conclusion

The adoption of LoLa4D has shifted bug finding from reactive searching toward a disciplined, knowledge-driven practice. By combining rigorous pre-shipment testing, structured logging, and expressive temporal pattern matching, Dräger has built a robust capability to detect, explain, and prevent complex failures in the field. While not many bugs have been found at all in the field, thanks to Dräger's quality assurance measures during development, LoLa4D turned out to be beneficial for spotting rare bugs emerging in the field. Additionally LoLa4D helps to set the stage for expectable analysis challenges from Dräger's upcoming networked product offerings where log data from multiple sources need to be evaluated in a unified context. Dräger is working on setting up a comprehensive repository of bug patterns to be checked for automatically during continuous integration. Further future work includes the extension of LoLa4D towards handling several logfiles in parallel, allowing runtime verification across multiple logfiles.

References

1. D'Angelo, B., Sankaranarayanan, S., Sánchez, C., Robinson, W., Finkbeiner, B., Sipma, H.B., Mehrotra, S., Manna, Z.: LOLA: runtime monitoring of synchronous systems. In: 12th International Symposium on Temporal Representation and Reasoning (TIME 2005), 23-25 June 2005, Burlington, Vermont, USA. pp. 166–174. IEEE Computer Society (2005). https://doi.org/10.1109/TIME.2005.26, https://doi.org/10.1109/TIME.2005.26