Using Runtime Monitoring to Enhance Offline Analysis

Sebastian Schirmer
Institute of Flight Systems, dept. Unmanned Aircraft
German Aerospace Center (DLR)
Braunschweig, Germany
sebastian.schirmer@dlr.de

Sebastian Benders
Institute of Flight Systems, dept. Unmanned Aircraft
German Aerospace Center (DLR)
Braunschweig, Germany
sebastian.benders@dlr.de

***Abstract*—Offline log file analysis of unmanned aircraft systems is a challenging experts task. Due to more automation, the amount and complexity of logged data increases. Experts need support, for instance by automatically generating additional statistical information or by correlating data. Runtime Monitoring is a formal method for analyzing system executions. It allows users to express temporal properties in a formal language which then can be used for the generation of a corresponding monitor. In this paper, we propose to integrate Runtime Monitoring into our offline analysis process. We show how the results of the monitor can be used to enhance the log file analysis and, therefore, support the expert. Specifically, we use the stream-based specification language LOLA and apply it to the log files of an unmanned cargo aircraft.**

***Index Terms*—Aerospace, Runtime Monitoring, LOLA, Offline Analysis, Log File Analysis, Unmanned Aircraft Systems**

I. INTRODUCTION

Safety is one of the biggest concerns in aviation. During development and operation, the state of the aircraft needs to be captured and degradation of the system has to be detected to take suitable counter-measures, e.g. to replace damaged or worn parts. Therefore, relevant data is stored and analyzed subsequently. Offline analysis is a challenging task and even more complicated due to the higher degree of automation induced by unmanned aircraft. The amount and complexity of relevant data has increased. One established way for analysis is to generate plots from the log files for visual inspection by an expert. There, the correlation of data is a challenging, tedious, and repeating task. In short, it is error-prone.

In this paper, we propose to integrate Runtime Monitoring (RM) into our offline analysis process. We further show how to use the monitor results to enhance the offline analysis supporting the expert. RM allows to express temporal properties in a formal and descriptive way which can be automatically translated into a corresponding correct monitor. Another benefit is the separation of the analysis code and the specified log file properties which improves the understanding and the maintenance. The work is motivated by the in-house DLR (German Aerospace Center) project ALAADy (Automated Low-Altitude Air Delivery). This prototype aircraft is based on a manned ultralight gyrocopter which was converted to an unmanned cargo aircraft with a payload of 200 kg. For this purpose avionics such as flight control computers, actuators [1], and sensors were installed [2]. During operation of the unmanned aircraft the subsystem which is responsible for controlling the core avionics of the aircraft, e.g. the actuators and the engine control, generates more than 370 logged system states and sensor signals in 16 log files [2]. Additionally, flight state sensors as well as the automatic flight control system are generating log files. A pure manual inspection of the log files is inefficient. The proposed approach automates simple checks and supports the analysis by adding statistical information to the visualization pointing towards potential errors identified by the monitors. Therefore, the approach reduces the workload of the operating crew and delivers quick evaluation results right in the field during flight-testing.

II. RELATED WORK

Work in the field of RM mostly focuses on the online use-case, i.e. while operating the system. Among others, RM has been used for unmanned aircraft systems [3], smart homes [4], cars [5], and ground rovers [6]. In [7], RM was used in an offline fashion to support testing of spacecraft flight software for the NASA 2011 Mars mission MSL (Mars Science Labratory). Instead of a stream-based specification language, the rule-based specification language of the tool LOGSCOPE was applied. Here, we focus more on the integration of RM into the current offline analysis workflow for a single log file, i.e. how to enhance the analysis by RM. In [8], the problem of synchronizing different log files is considered. There, the system is distributed and concurrent and, therefore, the locally observed, time stamped, and logged data might not be in order.

One established way for data analysis is to use libraries like the python data analysis library PANDAS [9]. It is open-source, easy-to-use, and expressive. Data queries can be easily stated, e.g. `data[data.a>data.b]` returns the data where `a>b`. However, since it is based on python, it is more imperative and it does not natively allow temporal queries, whose implementations are prone to errors.

III. APPROACH

In order to support the offline log file analysis, we propose to integrate RM into the analysis workflow. RM is a formal method for analyzing system executions. The correct behavior and statistical measurements of the systems are declared in

a *specification* file. Based on the specification, a *monitor* is generated which checks whether the specification is fulfilled during system execution. Due to this profiling of the system, the confidence in the system is improved. The events and signal data are either received due to system instrumentation during operation or read from log files afterwards. The former is referred to as *online monitoring* and the latter is referred to as *offline monitoring*. In both cases, the monitor outputs a *verdict* as a result which represents the adherence of the execution to the specification. Online and offline monitoring are depicted in Figure 1. As specification language, we chose LOLA which is presented in Section III-A

Currently, log files are inspected manually to find unexpected system behavior. The major analysis is done by reviewing generated signal graphs. Based on these plots, an expert has to decide whether the system did operate nominally or not. The approach, described in Section III-B, allows experts to explicitly state assumptions on the system behavior in a formal and unambiguous way and to automatically incorporate the analysis results into the normal workflow. We show examples of the enhanced offline analysis using RM in Section III-C.

A. *Formal Specification Language* LOLA

LOLA is a stream-based formal specification language for system online and offline monitoring. Originally, LOLA was designed for synchronous systems including circuits and embedded systems [10] and, lately, extended for network monitoring [11]. More recently, RTLOLA [12] was introduced, which extends previous versions by sliding windows over real-time intervals with aggregation functions. The corresponding LOLA tool [13] is actively developed at Saarland University. Here, we remain within the more basic LOLA fragment described in [14]. In a previous work, LOLA was used in an online monitoring setting [15] which indicates its applicability. There, an observation was that the development of specifications for the online usage can be significantly supported by testing them offline based on column-oriented log files. As a side-effect, the understanding of the system was greatly improved which motivated this approach.

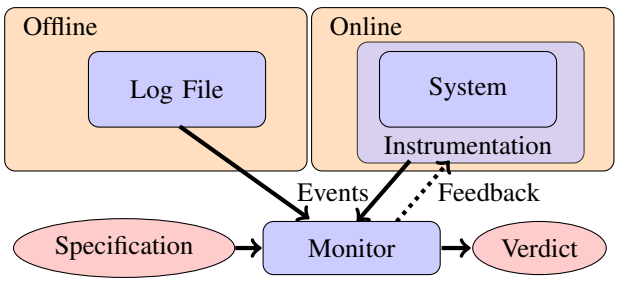

Fig. 1: *Based on a specification, a monitor is generated which checks the adherence to a specified system behavior. In offline monitoring, the system execution has finished and the log file is completely available. In online monitoring, the execution is still ongoing and therefore the sequence of events is continuously extended.*

```
const <type> <name> := <value> //constants
...
input <type> <name> //input streams
...
output <type> <name> := <expression> //output streams
...
```

Listing 1: *Basic* LOLA *specification structure*

A LOLA specification consists of a set of independent input streams and a set of dependent output streams. The basic structure of a LOLA specification is depicted in Listing 1. Output streams can be used to represent an error, diagnosis reports, or quantitative statistics. Input streams represent signals of the system under scrutiny. All streams are evaluated at the pace of a synchronous clock. For asynchronous systems, synchronization techniques like sample and hold can be applied. Note that since LOLA basically describes a system of equations, the order of input and output declarations is irrelevant for the underlying evaluation algorithm which improves the usage and the maintainability. Values of output streams are defined by their respective stream expression. A stream expression can refer to previous, present, or possible future values of streams (both, input and output streams). Further, the streams are typed (e.g binary, integer, double) and incrementally computable statistics can be specified. The so called dependency graph [10] is used to capture the dependencies between the streams, to identify whether a given specification is efficiently monitorable. Intuitively, this syntactic analysis states that the specification does not contain unbounded future stream accesses, i.e. the time until the output stream is evaluated is bounded. This allows to make statements about the memory consumption, e.g. only constant memory is required. The key features of LOLA, which lead to its usage, compared to other temporal formal languages (e.g. linear-time temporal logic), are: its descriptive nature, its structurability, and the resemblance of expressions with writing actual programs instead of abstract logical formulas. Further, especially temporal properties expressed in LOLA are easier to understand and to maintain compared to handwritten code. Additionally, when operating several aircraft, such a generic solution is highly desirable. Listing 2 shows a simple LOLA specification to compute the average velocity. The example uses one input stream `vel` for the current velocity of a vehicle and two output streams. The output `vel_sum` is an auxiliary stream which incrementally computes the sum over the seen velocities by accessing the previous value of `vel_sum` and the current velocity. The offset operator *s[x,y]* handles the access to previous (i.e. $x < 0$), present (i.e. $x = 0$), or future (i.e. $x > 0$) stream values of the stream s. The

```
input  double vel
output double vel_sum := vel_sum[-1, 0.0] + vel
output double vel_avg := vel_sum / double(position+1)
```

Listing 2: *Computation of the average velocity.*

```
ace_basic -> actuator_left, actuator_nose_wheel,
             actuator_right, actuator_rudder
ace_left_specific -> actuator_left
```

Listing 3: *Specification-to-logfile mapping. The* LOLA *specification* `ace_basic` *is applied to all four actuators whereas* `ace_left_specific` *is only used for the left actuator.*

default value y is used for accesses past the end or before the beginning of a stream. In Listing 2, `-1` indicates the access to the previous value and `0.0` is used as a default value at the first position. Finally, using the keyword `position`, representing the current step of the evaluation (starting from zero), the average velocity is computed in `vel_avg`.

B. Enhancing Offline Analysis

The presented approach supports the analysis of log files, see Figure 2. During system tests, log files are generated which need to be reviewed. To support the manual inspection of each log file, we want to capture common erroneous and desired behaviors within a formal LOLA specification. Based on these specifications monitors will be generated which analyze the files. Specifications can be named and mapped to specific log files. An example mapping is shown in Listing 3.

This is especially interesting for recurring specifications which can be declared in separate specification files and reused, e.g. timing properties. An example LOLA specification concerning timing is shown in Listing 4. Typically, time is captured via timestamps in log files. Interesting timing properties are the average, minimal, and maximal time jumps (`t_jmp`). Maximal and minimal values indicate frozen and rapid system states, respectively. Also, time jumps larger than the given threshold `t_safe` can be detected and reported. Note that a LOLA specification can be easily extended by an user due to the irrelevance of the stream declaration ordering. For instance, by adding an additional output stream, the number of violations can be counted. The properties are

```
input double time_s, time_us
output double time := time_s + time_us / 1000000.0
output double t_jmp := time - time[-1,0.0]
//User notification if t_jmp > t_safe
const double t_safe := 0.05
trigger t_jmp > t_safe with "VIOLATION: time jump!"
//Statistical information
output double t_sum := t_sum[-1, 0.0] + t_jmp
output double t_avg := t_sum / (double(position+1))
output double t_max := max(t_jmp,t_max[-1,double_min])
output double t_min := min(t_jmp,t_min[-1,double_max])
```

Listing 4: *Given the current timestamp (*`time_s,time_us`*), the elapsed time* `t_jmp` *is computed. Based on the comparison with the constant* `t_safe` *a trigger notification is raised representing a too large time jump. Further,* `t_sum` *aggregates* `t_jmp` *which can be used to compute the average time jumps* `t_avg`*. The output streams* `t_max` *and* `t_min` *compute the maximal and the minimal time jump, respectively.*

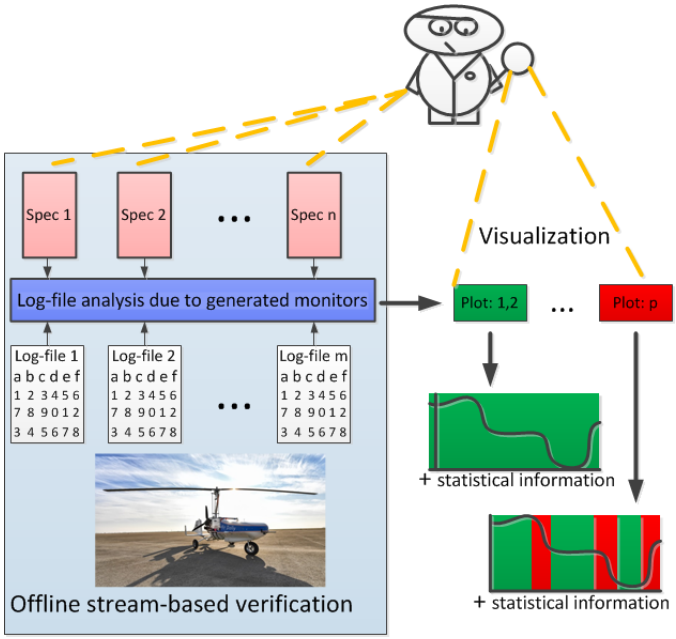

Fig. 2: *Illustration of the proposed offline analysis approach using RM. Meta-information is inferred to enhance log files. The extended log file is visualized and the meta-information is used to point out erroneous regions to the user.*

then automatically checked on the mapped log files by the generated monitors. After the analysis due to the monitors, the verdicts, e.g. the mentioned maximal values or raised notification, can be used to enhance the used visualization. Colors indicate whether the values for the respective plots were involved in a notified violation, i.e. a violation of an explicitly stated assumption. In Figure 2, the first two plots are healthy but some notifications where raised for plot p. In both cases, the users preferred visualization tool-chain can be enhanced by statistical information. For the erroneous case, the monitor result can guide the expert's attention towards interesting regions.

Considering the visualization of plot p shown in Figure 2, assuming the actuator position over time is depicted and timing notifications (see Listing 4) were evaluated, an expert can see that each time a time jump occurred, the position of the actuator changed.

C. Example Properties

Actually, the presented identification of time jumps, i.e. temporal system loss, was motivated by a real problem. Experts rely on different types of visualization to identify errors. Detecting time jumps using line plots is error-prone compared to using scatter plots. Also, humans tend to extend their tolerance limit over time, e.g. due to optimism, self-serving bias, or anchoring [16]. Hard thresholds explicitly stated in a specification do not only allow to reduce the reliance on gut-feeling but also allow to identify anomalies across different log files. Using this approach, we cannot only automatically check whether jumps exist in the current log files, we can also

make new assumptions, e.g. decrease `t_safe`, and re-evaluate the specification on the current and the historical log files. In the following, we show other simple but helpful properties of `ace_basic` and `ace_left_specific` (see Listing 3):

- Each actuator implements a state machine. State changes are logged as *device status*. In the end, the sequence of device states represent a valid execution.
- Bounds on the *demanded actuator position* differ for each actuator and should never be exceeded to avoid defects.
- The mechanical loads of the actuators and therefore their *consumption of current* differ.
- Statistical information on the average, maximal, and minimal values for *current consumption, timing, velocity,* and *voltage* are useful to get more insights into the system.
- The *relation between consumption of current and position demands* can be analyzed to differentiate between internal and external errors.
- Estimation of the *time difference* between the *position demand* and the time when the *target position* was reached.

Note that these are just examples for a single actuator log file. Figure 3 shows a visualization example utilizing the monitoring results. The orange and the green line indicate the specified upper and lower bound on the consumption of current, respectively. The red regions show where trigger notifications were raised by a monitor. Here, red regions indicate an increased current consumption despite no actuator position demand was commanded previously. In fact, considering the first red region, the increased current consumption is related to the initialization procedure of the actuator. With the specification violations in mind, i.e. the second red region around 400 s and the upper bound violation around 850 s, the expert can now examine the log files in detail to determine the root cause.

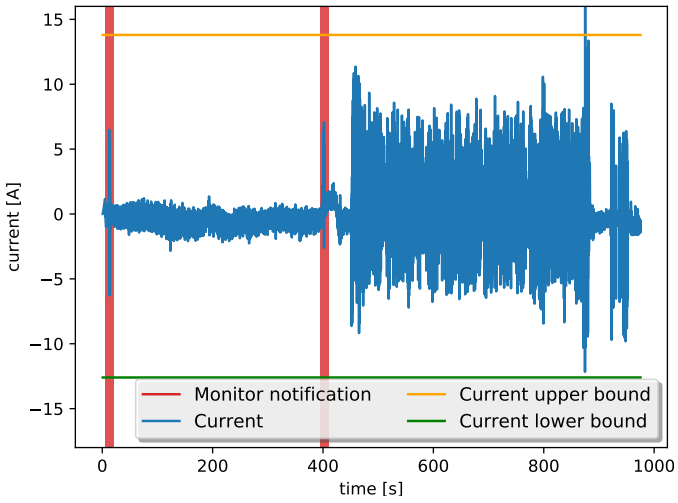

Fig. 3: *Current consumption of an actuator with specified lower and upper bounds. Red regions indicate a notification due to a comparison between current consumption and actuator position demand.*

IV. Conclusion

We have presented RM and the specification language Lola. Further, we motivated why and showed how RM can be used for offline log file analysis. Specifically, we illustrated how manual inspection due to visualization can be enhanced due to the outputs of the formally specified monitors. We plan to fully implement the integration of runtime monitors into our offline analysis tool-chain. Also, the reasoning across different log files is very helpful and will be central since future systems tend to be highly distributed. Currently, Lola is able to read column-oriented *.csv* and *.dat* files. Accepting other data formats, e.g. CDF, NetCDF, HDF, would improve the general applicability. In future, we will investigate how offline monitoring enhances online monitoring of future sessions and, vice versa, how online monitoring helps to provide targets to detailed offline monitoring of the same session. Further, the benefits of the proposed approach in regards of cost savings, usability, and quality improvements compared to a manual inspection will be investigated.

References

[1] A. Bierig, S. Lorenz, M. Rahm, P. Gallun, “Design considerations and test of the flight control actuators for a demonstrator for an unmanned freight transportation aircraft”, Recent Advances in Aerospace Actuation Systems and Components, 2018

[2] S. Benders, L. Goormann, S. Lorenz, J.C. Dauer, “Softwarearchitektur für einen unbemannten Luftfrachttransportdemonstrator”, Deutscher Luft- und Raumfahrtkongress, 2018

[3] T. Reinbacher, K.Y. Rozier, J. Schumann, “Temporal-Logic Based Runtime Observer Pairs for System Health Management of Real-Time Systems”, Tools and Algorithms for the Construction and Analysis of Systems (TACAS), 2014

[4] A. El-Hokayem, Y. Falcone, “Bringing Runtime Verification Home”, Runtime Verification - 18th International Conference (RV), 2018

[5] J.V. Deshmukh, A. Donzé, S. Gosh, X. Jin, G. Juniwal, S.A. Seshia, “Robust Online Monitoring of Signal Temporal Logic”, Runtime Verification - 15th International Conference (RV), 2015

[6] D. Phan, J. Yang, M. Clark, R. Grosu, J. Schierman, S. Smolka, S. Stoller, “A Component-Based Simplex Architecture for High-Assurance Cyber-Physical Systems”, 17th International conference on application of concurrency to system design (ACSD), 2017

[7] H. Barringer, K. Havelund, D. Rydeheard, A. Groce, “Rule Systems for Runtime Verification: A Short Tutorial”, Runtime Verification Lecture Notes in Computer Science, 2009

[8] D. Basin, M. Harvan, F. Klaedtke, E. Zalinescu, “Monitoring Data Usage in Distributed Systems”, IEEE Transactions on Software Engineering, 2013

[9] Python Data Analysis Library - pandas, https://pandas.pydata.org/, last visited: 05.01.2019

[10] B. D’Angelo, S. Sankaranarayanan, C. Sánchez, W. Robinson, B. Finkbeiner, H. B. Sipma, S. Mehrotra, Z. Manna, “Lola: Runtime Monitoring of Synchronous Systems”, 12th International Symposium on Temporal Representation and Reasoning (TIME’05), 2005

[11] P. Faymonville, B. Finkbeiner, S. Schirmer, H. Torfah, “A Stream-Based Specification Language for Network Monitoring”, Runtime Verification - 16th International Conference (RV), 2016

[12] P. Faymonville, B. Finkbeiner, M. Schwenger, H. Torfah, “Real-time Stream-based Monitoring”, Available on https://arxiv.org/abs/1711.03829

[13] Lola - Stream-based Runtime Monitoring, https://www.react.uni-saarland.de/tools/lola/, last visited: 07.01.2019

[14] S. Schirmer, “Runtime Monitoring with Lola”, Master’s Thesis, 2016

[15] F-M. Adolf, P. Faymonville, B. Finkbeiner, S. Schirmer, C. Torens, “Stream Runtime Monitoring on UAS”, Runtime Verification - 17th International Conference (RV), 2017

[16] R. Collins, B. Leathley, “The Psychology of Errors in the Engineering Process”, Safety and Reliability, 1995