Real-Time Visualization of Stream-Based Monitoring Data

Jan Baumeister[1(✉)], Bernd Finkbeiner[1], Stefan Gumhold[2], and Malte Schledjewski[1]

[1] CISPA Helmholtz Center for Information Security, 66123 Saarbrücken, Germany
{jan.baumeister,finkbeiner,malte.schledjewski}@cispa.de
[2] Technische Universität Dresden, 01069 Dresden, Germany
stefan.gumhold@tu-dresden.de

Abstract. Stream-based runtime monitors are used in safety-critical applications such as Unmanned Aerial Systems (UAS) to compute comprehensive statistics and logical assessments of system health that provide the human operator with critical information in hand-over situations. In such applications, a visual display of the monitoring data can be much more helpful than the textual alerts provided by a more traditional user interface. This visualization requires extensive real-time data processing, which includes the synchronization of data from different streams, filtering and aggregation, and priorization and management of user attention. We present a visualization approach for the RTLola monitoring framework. Our approach is based on the principle that the necessary data processing is the responsibility of the monitor itself, rather than the responsibility of some external visualization tool. We show how the various aspects of the data transformation can be described as RTLola stream equations and linked to the visualization component through a bidirectional synchronous interface. In our experience, this approach leads to highly informative visualizations as well as to understandable and easily maintainable monitoring code.

Keywords: Runtime verification · Stream-based monitoring · Data visualization

1 Introduction

Over the past decades, the scope of runtime verification has grown from an essentially boolean check, indicating whether or not a program execution satisfies a given formal specification, towards the real-time computation of more and

This work was partially supported by the German Research Foundation (DFG) as part of the Collaborative Research Center Foundations of Perspicuous Software Systems (TRR 248, 389792660).

T. Dang and V. Stolz (Eds.): RV 2022, LNCS 13498, pp. 325–335, 2022.
https://doi.org/10.1007/978-3-031-17196-3_21

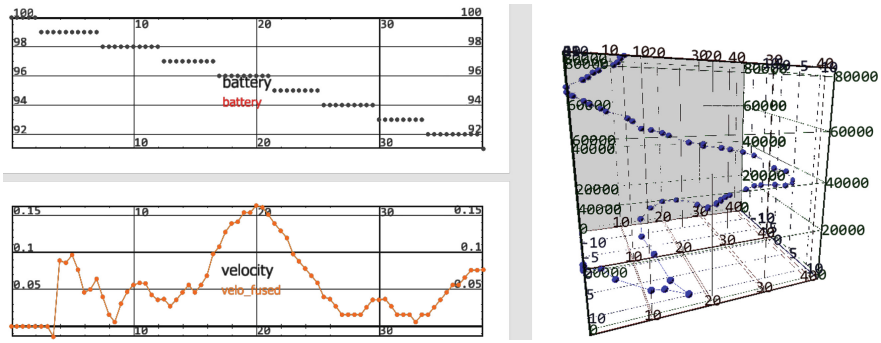

Fig. 1. Screenshot of a visualization displaying the battery status, velocity, and GPS coordinates of a UAS.

more expressive statistical data. A typical example are Unmanned Aerial Systems (UAS), where the monitor continuously collects and aggregates inputs from sensors and on-board components to provide the human operator with critical information in hand-over situations [1,3,26].

Traditionally, the interaction between runtime verification tools and their users has largely been based on textual interfaces, such as "alert" messages that are issued in case of a violation of the specification. In applications like UAS, however, such a simple user interface is often no longer sufficient. In addition to understanding that the monitor has detected a problem, the human operator must quickly grasp the situation and decide on potentially time-critical corrective action. A well-designed *visual* presentation of the available data is therefore of critical importance for the safe operation of the system.

Generating useful visualizations is far from trivial. First and foremost, the visualization must ensure that important or dangerous information is clearly visible to the user; because of the abundance of available data, data must be prioritized, and less important data must be hidden in favor of more important data. Similarly, the frequency of data points must be adjusted to provide meaningful information avoiding overlaps and adjusting for discrepancies in the availability of the raw data. All these computations have to be adjusted in response to actions by the user, who may look at multidimensional data from different angles or zoom into data areas of particular interest.

In this paper, we report on our recent effort in extending the RTLola monitoring framework with real-time visualization capabilities. RTLola [4,11] is a stream-based monitoring framework for cyber-physical systems and networks. RTLola processes, evaluates, and aggregates streams of input data, such as sensor readings, and provides a real-time analysis in the form of comprehensive statistics and logical assessments of the system's health. An RTLola monitor is generated from a formal description given in the RTLola specification language. The specifications consist of stream equations that translate input streams into

output streams. RTLola specifications are statically analyzed to determine the required memory and are then either directly executed by the RTLola interpreter, or compiled onto an FPGA.

The fundamental insight of our approach is that the data processing needed to generate the visualization should be the responsibility of the monitor itself, rather than that of some external visualization tool. The monitor has access to all information and is therefore in the best position to determine the relevancy of individual data points. Because of the expressive power of the monitoring language, the monitor also has the computational means to interpolate and aggregate the raw data as required. Finally, keeping all data manipulations in one place reduces redundancy and avoids errors and misinterpretations.

We organize the RTLola specification of the data processing for the visualization into three functional areas: **1. Data Synchronization:** This part of the specification guarantees *synchronous* data updates for the visualization. This is important because the visualization combines data from different streams into a single entry in the plot: for example, a point indicating the position of a drone might be annotated with the speed of the drone. If different attributes have different timing, for example because of the variations in the frequency of the sensors, the monitor interpolates the missing data. **2. Filtering and Aggregation:** This part of the specification avoids overlapping markers in the visualization, which are caused if readings arrive at a high rate. The monitor smoothes the input signal and adjusts the rate according to the current visualization. **3. Prioritization and Attention Management:** This part of the specification determines the criticality of the available information and ensures that the human operator does not miss important information.

The monitor and the visualization component are connected via a synchronous interface. The responsibility of the visualization component is to create the graphical display and to react to user requests. Since this user interaction may affect the visibility of plots or change the scaling, a backchannel provides this information to the monitor in the form of additional input streams. Fig. 1 shows a screenshot of our prototype implementation, which is based on the monitoring framework RTLola and the visualization framework cgv [16]. The monitor interacts with configurable 2D and 3D plots that support time-series plots, scatter plots, trajectory plots and multi-variate visualization through a flexible mapping of data attributes to the visual attributes color, opacity and size. We have applied our approach to the real-time visualization of UAS and other cyber-physical systems, based on existing RTLola case studies; our experience suggests that adding the visualization specification inside the monitor leads to highly informative visualizations as well as to understandable and easily maintainable monitoring code.

1.1 Related Work

This paper connects two traditionally separate areas of research: runtime monitoring and visualization. Somewhat surprisingly, visualization has not played a major role in monitoring research before. Despite a wide range of monitoring

approaches, from formal logic [9,12,17,25] to stream-based specification languages [6,7,10,15], most tools have in common that they rely on textual, rather than visual, methods for data presentation. This paper shows that stream-based monitoring languages like RTLola are very well suited to carry out the needed data processing for useful visualizations. Our focus on RTLola is motivated by recent work on RTLola-based monitoring for UAS [2,3] and other cyber-physical systems [4,5,11]. However, the approach of the paper is clearly transferrable to other monitoring tools for CPS [22–24].

In the area of visualization, research on streaming visualization is also still in an early stage. A notable result are streaming processing models for data [28] and techniques for kernel density estimation in aggregated views over 2D maps [19]. There has been a systematic discussion of the suitability and problems of traditional visual analysis techniques [8,18,27]. Additionally, visualization frameworks [14] and visualization techniques that allow the user to better cope with changes over time have been developed, such as zoomable navigation [21], paged views [13], and transformation-based smooth transitions [20]. These approaches differ substantially from the approach taken in this paper, in that these are independent visualization tools, which prepare the data for the visualization independently of the monitor. By contrast, our setup tightly integrates the monitor with the visualization, keeping all data manipulations in one place.

2 RTLola

RTLola is a stream-based monitoring framework for cyber-physical systems and networks. An RTLola monitor is generated from a formal specification description given in the RTLola specification language. The specification consists of stream equations that describe the transformation of incoming data streams into output streams, and a set of trigger conditions that result in notifications to the user. The RTLola framework includes automatic static analysis methods that ensure the predictability of the monitor with respect to memory consumption and other relevant properties. We illustrate the RTLola specification language with a small example; for more details, we refer the reader to [4,11].

```
input gps: (Float64, Float64), charge: Float64, time: Float64
output charge_time @charge := time.hold(or: 0.0)
output filtered_gps filter gps != (0.0,0.0) := gps
trigger δ(charge) / δ(charge_time) > 2.0
trigger filtered_gps.0 > 6.0 ∧ filtered_gps.1 > 6.0
```

The specification declares three input streams: The first stream `gps` represents readings received by the GNSS (global navigation satellite system) module, the second stream `charge` shows the battery charge status, and the third one `time` contains the current time. Next, the specification declares the `charge_time` output stream, which filters the `time` stream to timestamps of newly received battery readings. For this, it binds the timing of the `charge_time` stream to the timing of `charge`, indicated by the `@charge` annotation. In RTLola, such a filter is called a static filter. As `time` might have a different timing, the value is

```
input charge: Float64, gps: (Float64, Float64)
input pixel_scale: (Float64, Float64), visible: Bool
output xLim: (Float64, Float64) @gps
:=(min(gps.0, xLim.0.offset(by:-1, or:gps.0)), max(gps.0, xLim.1.offset(by:-1, or:gps.0)))
output yLim: (Float64, Float64) @ gps := ...
output δx @gps∨charge
:= (gps.hold(or:gps_s).0 - marker.offset(by:-1, or:marker_s).0) /
     (xLim.hold(or:1.0).1-xLim.hold(or:1.0).0) * pixel_scale.hold(or:pixel_scale_s)
output δy @gps∨charge := ...
output δc @gps∨charge := charge.hold(or:charge_s) - marker.offset(by:-1, or:marker_s).2
output send @gps∨charge := sqrt(δx**2.0 + δy**2.0) > τ_gps ∨ δc > τ_charge
output marker: (Float64, Float64, Float64) @gps ∨ charge
  filter send ∧ visible.hold(or: false)
  := (gps.hold(or:gps_s).0, gps.hold(or:gps_s).1,charge.hold(or:charge_s))
```

Fig. 2. RTLola specification demonstrating the interplay.

accessed via a 0-order hold interpolation. The next stream `filtered_gps` uses a dynamic filtering to exclude noisy sensor readings. In our example, the GNSS sends $(0.0, 0.0)$ coordinates during initialization that the specification can discard. The last two lines contain triggers checking if there is an unusual drop in the battery status and if the coordinates do not exceed some thresholds.

In RTLola, the static and dynamic filters are combined into a *pacing type*, which defines the timing of each stream. This type is either inferred or explicitly annotated. RTLola's type checker verifies the timing of the streams and RTLola provides different operators to interpolate data if the timing constraints cannot be guaranteed in the stream expression.

3 Generating Visualization Data

We now describe in more detail the generation of visualization data with RTLola stream equations. For the communication from the monitor to the visualization component, the specification contains output streams that are mapped to plot coordinates and visual attributes such as size and color. It also contains an output stream per axis, setting its displayed range. For the backchannel, the specification has input streams that receive the data from the visualization reflecting the interaction of the visualization component with the user. For each 2D-plot, we include one input stream to transfer the current scale factors; for each 3D-plot we include two input streams, representing the projection matrix and window size. Additionally, the specification has an input stream for each plot indicating which plot is visible.

We structure the generation of the visualization data into three areas: *Data Synchronization and Interpolation*, *Filtering and Aggregation*, and *Prioritization*. For each area, we shortly describe the problem, then describe the mechanism of how RTLola solves the task and explain the solution in more detail with our running example shown in Fig. 2. We display the coordinates of a GNSS in a 2-dimensional plot, and the remaining battery charge is mapped onto the color

Fig. 3. Screenshot of the prototype with different monitors. On the left side, we use a monitor forwarding all data, whereas the right monitor filters the data using the specification in Fig. 2.

of the marker. The input streams `charge` and `gps` represent the sensor readings followed by the streams `pixel_scale` and `visible` implementing the backchannel. The output streams `xLim` and `yLim` compute the upper and lower bound per axis which is in our case the global minimum and maximum. Alternatively, our plot could represent with the following stream expression the data over a time-period σ:

```
output x_limits: (Float64, Float64) @1Hz
  := (gps.0.aggregate(over: σ, using: min), gps.0.aggregate(over: σ, using: max))
```

The next streams are helper streams to filter the data and the last output stream `marker` contains all information needed for a new marker in the plot.

3.1 Data Synchronization and Interpolation

For drawing a marker, the visualization needs to know all its visual attributes which might be based on sensors with different frequencies. We use RTLola's type system to guarantee that the monitor sends synchronized updates per plot. For this task, we use the concept of pacing types as introduced in Sect. 2. We define a pacing for each plot and annotate the streams for this plot with the desired pacing. Then, we use RTLola's type checker to verify that the data is available. In our example, we want to create a marker whenever at least one sensor sends an update and therefore use the disjunction of the two input streams as the pacing type. This pacing type `@gps`$\vee$`charge` is annotated to the stream `marker`. Similarly, we want to update the axis limits, represented by the streams `xLim` and `yLim`, whenever we get a new GPS sensor reading.

We cannot directly access the current value of each stream because they may have different timings. Instead, we specify how missing data is interpolated. RTLola offers different approaches for this task, e.g., by using data aggregations, zero-order hold operations, or even different forms of data interpolations. In our example, we use a zero-order hold on the missing data.

3.2 Data Filtering and Aggregation

This section shows how a monitor prepares data to provide more understandable updates to the user. Figure 3 shows two plots from the same execution. On the

Fig. 4. On the left side, one marker with irrelevant information covers a critical marker, whereas the monitor on the right encodes a form of priority.

left side, the monitor forwards all data to the visualization resulting in overlapping markers. These overlapping markers overload the plot with unrequired information and even overlap some information, as in our example, the color illustrating the battery status. On the right side, the monitor tailors the data for visualization and transfers the prepared data, so we do not have this problem.

The monitor on the right uses RTLola's dynamic filtering mechanism to prepare the data. With this filtering approach, the monitor can dynamically adapt the throughput to the visualization. Dependent on the scenario, different filters are helpful: For example, a filter forwards the GPS data dependent on the current velocity or increasing the sample rate if the system violates a property could be easily expressed in RTLola. In our running example, we forward the markers only if the difference between the new marker and the previous marker exceeds a threshold and if the plot is visible. For this, the output stream `marker` has two such filters connected by a disjunction. Both filters use the information provided by the visualization to describe the current state of the plot – visibility and scaling. With the second disjunct, we ensure that a marker is transmitted only if the plot is visible. The other filter is encoded by the stream `send`. In this stream, we decide whether the difference to the previously transmitted marker is sufficient to warrant a new marker. For this, we first compute the distance between the pixel coordinates of the candidate marker and the last marker based on the current bounds and scaling, and compare it with a defined threshold that prevents overlapping. We also check whether the difference in the charging level warrants a new marker. A similar approach also applies to 3D plots. Instead of the `pixel_scale`, we use the projecting matrix that encodes besides the scaling, the viewing rotation, and perspective.

Depending on the scenario and the size of the plot, it can be useful to aggregate values (such as by computing the average, minimum, and maximum of the data since the last marker) instead of dropping values. In RTLola, this can easily be done using the corresponding aggregation functions.

3.3 Attention Management

In Sect. 3.2, we have already discussed how the monitor can filter data points if they do not contain relevant information. Often, however, this is not sufficient, and we need to *prioritize* information: Fig. 4 shows two plots containing a critical state that the operator should recognize, illustrated by the red marker. This

marker is partially covered on the left by plotting a new marker that does not contain this information anymore. The operator can easily miss this information, so the monitor on the right prioritizes them and thus does not send the candidate marker to the visualization tool.

Again, we use RTLola's dynamic filtering mechanism to prevent the coverage of higher prioritized information. We also introduce new streams encoding the priority of information and checking the coverage of markers. In our running example, we need to change the stream expression of `send` and add the following streams to the specification:

```
output critical: Bool @gps∨charge := ...
output marker_lc @gps∨charge
:= if send ∧ critical then marker else marker_lc.offset(by:-1, or:marker_s)
output δxc @gps∨charge
:= (gps.hold(or:gps_s).0 - marker_lc.offset(by:-1, or:marker_s).0) /
     (xLim.hold(or:1.0).1-xLim.hold(or:1.0).0) * pixel_scale.hold(or:pixel_scale_s)
output δyc @gps∨charge := ...
output send @gps ∨ charge
:= (sqrt(δx**2.0 + δy**2.0) > τ_gps ∨ δc > τ_charge) ∧ ((sqrt(δxc**2.0 + δyc**2.0) > τ_c ∨
     critical)
```

The stream `critical` encodes the priority of a marker and the next stream `marker_lc` stores the values of the last critical marker. The changed stream `send` now also determines if a potential new marker would overlap the last critical marker by computing the distance between the markers with the help of δ`xc` and δ`yc` and then checking if this distance is sufficient.

Preventing covering markers with less relevant information is only one example of how we can encode the priority of information. Another example occurs when the specification aggregates data points to make the plots more readable: With aggregations, we lose information about the system. In general, this behavior is intended because the human supervisor cannot process all information from every sensor. In critical situations, however, the operator usually is focused on the part that misbehaves. In these situations, the monitor can switch to transferring each data point instead of aggregating them, or it might reduce the required difference for new markers, so the supervisor gets all the required information. Such a property can be expressed in RTLola by adapting the timing of a stream or by using different aggregation functions.

4 Conclusions

In this paper, we have introduced a principled approach to the real-time visualization of stream-based monitoring data. The key contributions are the novel design principle, which shifts the responsibility for the data preparation from the visualization component to the monitor; the organization of the approach into three major functional areas; and the solution of the visualization challenges with the mechanisms of a stream-based monitoring language.

Our practical experience with the approach of the paper has been very positive. We have used the approach to visualize stream-based monitoring data from recent case studies that use RTLola for UAS [2,3] and other cyber-physical systems [4,5,11]. The visual tools provided by cgv have proven very useful for

the type of data produced by our monitors. For example, we have visualized the failure of the GPS module in a drone, which was recognized by the system health check in the existing monitor specification, by adding a halo to the markers of the estimated position and by increasing the marker frequency. While clearly more research is needed in order to determine the best visualizations, our experience already indicates that this type of visualization is very helpful in quickly understanding complicated situations.

We hope that this paper will inspire other developers of runtime verification tools to invest in real-time visualization as well. We believe that our "monitoring-oriented" visualization approach provides a significant step towards meaningful visualizations that exploit the wealth of information available within the monitor. In future work, it might even be possible to integrate explicit visualization operators into monitoring languages like RTLola, and thus largely automate the visualization process presented in this paper.

References

1. Adolf, F., Faymonville, P., Finkbeiner, B., Schirmer, S., Torens, C.: Stream runtime monitoring on UAS. In: Lahiri, S.K., Reger, G. (eds.) RV 2017. LNCS, vol. 10548, pp. 33–49. Springer (2017). https://doi.org/10.1007/978-3-319-67531-2_3
2. Adolf, F.M., Faymonville, P., Finkbeiner, B., Schirmer, S., Torens, C.: Stream runtime monitoring on UAS. In: Lahiri, S., Reger, G. (eds.) Runtime Verification, pp. 33–49. Springer International Publishing, Cham (2017). https://doi.org/10.1007/978-3-319-67531-2_3
3. Baumeister, J., Finkbeiner, B., Schirmer, S., Schwenger, M., Torens, C.: RTLola cleared for take-off: monitoring autonomous aircraft. In: Lahiri, S.K., Wang, C. (eds.) CAV 2020. LNCS, vol. 12225, pp. 28–39. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-53291-8_3
4. Baumeister, J., Finkbeiner, B., Schwenger, M., Torfah, H.: FPGA stream-monitoring of real-time properties. ACM Trans. Embedded Comput. Syst. **18**(5s), 1–24 (2019). https://doi.org/10.1145/3358220
5. Biewer, S., Finkbeiner, B., Hermanns, H., Köhl, M.A., Schnitzer, Y., Schwenger, M.: RTLola on board: testing real driving emissions on your phone. In: TACAS 2021. LNCS, vol. 12652, pp. 365–372. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-72013-1_20
6. Convent, L., Hungerecker, S., Leucker, M., Scheffel, T., Schmitz, M., Thoma, D.: TeSSLa: temporal stream-based specification language. In: Massoni, T., Mousavi, M.R. (eds.) SBMF 2018. LNCS, vol. 11254, pp. 144–162. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03044-5_10
7. D'Angelo, B., et al.: Lola: Runtime monitoring of synchronous systems. In: 12th International Symposium on Temporal Representation and Reasoning (TIME2005), pp. 166-174. IEEE Computer Society Press (2005)
8. Dasgupta, A., Arendt, D.L., Franklin, L.R., Wong, P.C., Cook, K.A.: Human factors in streaming data analysis: challenges and opportunities for informtion visualization. Comput. Graph. Forum **37**(1), 254–272 (2018). https://doi.org/10.1111/cgf.13264
9. Donzé, A., Ferrère, T., Maler, O.: Efficient robust monitoring for STL. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 264–279. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_19

10. Faymonville, P., Finkbeiner, B., Schirmer, S., Torfah, H.: A stream-based specification language for network monitoring. In: Falcone, Y., Sánchez, C. (eds.) RV 2016. LNCS, vol. 10012, pp. 152–168. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46982-9_10
11. Faymonville, P., et al.: StreamLAB: stream-based monitoring of cyber-physical systems. In: Dillig, I., Tasiran, S. (eds.) CAV 2019. LNCS, vol. 11561, pp. 421–431. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-25540-4_24
12. Finkbeiner, B., Sipma, H.: Checking finite traces using alternating automata. Formal Methods Syst. Des. **24**(2), 101–127 (2004). https://doi.org/10.1023/B:FORM.0000017718.28096.48
13. Fischer, F., Keim, D.A.: NStreamAware: real-time visual analytics for data streams to enhance situational awareness. In: Proceedings of the Eleventh Workshop on Visualization for Cyber Security, pp. 65–72. VizSec 2014, Association for Computing Machinery, New York, NY, USA (2014). https://doi.org/10.1145/2671491.2671495
14. Fischer, F., Mansmann, F., Keim, D.A.: Real-time visual analytics for event data streams. In: Proceedings of the 27th Annual ACM Symposium on Applied Computing, pp. 801–806. SAC 2012, Association for Computing Machinery, New York, NY, USA (2012). https://doi.org/10.1145/2245276.2245432
15. Gorostiaga, F., Sánchez, C.: Striver: stream runtime verification for real-time event-streams. In: Colombo, C., Leucker, M. (eds.) RV 2018. LNCS, vol. 11237, pp. 282–298. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03769-7_16
16. Gumhold, S.: CGV. https://github.com/sgumhold/cgv (2022)
17. Havelund, K., Roşu, G.: Synthesizing monitors for safety properties. In: Katoen, J.-P., Stevens, P. (eds.) TACAS 2002. LNCS, vol. 2280, pp. 342–356. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-46002-0_24
18. Krstajic, M., Keim, D.A.: Visualization of streaming data: Observing change and context in information visualization techniques. In: 2013 IEEE International Conference on Big Data, Silicon Valley, CA, USA, pp. 41–47. IEEE (2013). https://doi.org/10.1109/BigData.2013.6691713
19. Lampe, O.D., Hauser, H.: Interactive visualization of streaming data with kernel density estimation. In: 2011 IEEE Pacific visualization symposium, pp. 171–178. IEEE (2011)
20. Li, C., Baciu, G., Han, Y.: StreamMap: smooth dynamic visualization of high-density streaming points. IEEE Trans. Visual. Comput. Graph. **24**(3), 1381–1393 (2018). https://doi.org/10.1109/TVCG.2017.2668409
21. Li, C., Baciu, G., Han, Y.: Interactive visualization of high density streaming points with heat-map. In: 2014 International Conference on Smart Computing, pp. 145–149 (2014). https://doi.org/10.1109/SMARTCOMP.2014.7043852
22. Luppen, Z., et al.: Elucidation and analysis of specification patterns in aerospace system telemetry. In: Deshmukh, J.V., Havelund, K., Perez, I. (eds.) NFM 2022. LNCS, vol. 13260. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-06773-0_28
23. Moosbrugger, P., Rozier, K.Y., Schumann, J.: R2U2: monitoring and diagnosis of security threats for unmanned aerial systems. Formal Methods Syst. Des. **51**(1), 31–61 (2017). https://doi.org/10.1007/s10703-017-0275-x
24. Pike, L., Goodloe, A., Morisset, R., Niller, S.: Copilot: a hard real-time runtime monitor. In: Barringer, H., et al. (eds.) RV 2010. LNCS, vol. 6418, pp. 345–359. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16612-9_26

25. Raskin, J.-F., Schobbens, P.-Y.: Real-time logics: fictitious clock as an abstraction of dense time. In: Brinksma, E. (ed.) TACAS 1997. LNCS, vol. 1217, pp. 165–182. Springer, Heidelberg (1997). https://doi.org/10.1007/BFb0035387
26. Schumann, J., Moosbrugger, P., Rozier, K.Y.: R2U2: monitoring and diagnosis of security threats for unmanned aerial systems. In: Bartocci, E., Majumdar, R. (eds.) RV 2015. LNCS, vol. 9333, pp. 233–249. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23820-3_15
27. Smestad, G.: Interactive visual analysis of streaming data, master's thesis, Universitetet i Bergen (UiB) (2014)
28. Szewczyk, W.: Streaming data. Wiley Interdisc. Rev. Comput. Stat. **3**(1), 22–29 (2011). Publisher: Wiley Online Library