Monitoring CPS at Runtime – A Case Study in the UAV Domain

Michael Vierhauser, Jane Cleland-Huang, Sean Bayley
University of Notre Dame
South Bend, IN, USA
{mvierhau, janeclelandhuang, sbayley}@nd.edu

Thomas Krismayer, Rick Rabiser, Paul Grünbacher
CDL MEVSS, Johannes Kepler University
Linz, Austria
{thomas.krismayer, rick.rabiser, paul.gruenbacher}@jku.at

***Abstract*—Unmanned aerial vehicles (UAVs) are becoming increasingly pervasive in everyday life, supporting diverse use cases such as aerial photography, delivery of goods, or disaster reconnaissance and management. UAVs are cyber-physical systems (CPS): they integrate computation (embedded software and control systems) with physical components (the UAVs flying in the physical world). UAVs in particular and CPS in general require monitoring capabilities to detect and possibly mitigate erroneous and safety-critical behavior at runtime. Existing monitoring approaches mostly do not adequately address UAV CPS characteristics such as the high number of dynamically instantiated components, the tight int elements, and the massive amounts of data that need to be processed. In this paper we report results of a case study on monitoring in UAVs. We discuss CPS-specific monitoring challenges and present a prototype we implemented by extending ReMinds, a framework for software monitoring so far mainly used in the domain of metallurgical plants. Additionally, we demonstrate the applicability and scalability of our approach by monitoring a real control and management system for UAVs in simulations with up to 30 drones flying in an urban area.**

***Index Terms*—UAVs, runtime monitoring, cyber-physical systems.**

I. Introduction

An unmanned aerial system or unmanned aerial vehicle (UAV), commonly known as a drone, is an aircraft without a human pilot aboard [10]. Due to recent and ongoing advances, commercial UAV applications, such as aerial photography, package delivery, or disaster management, have become increasingly pervasive in everyday life. UAVs are cyber-physical systems (CPS): they integrate computational logic – on-board flight controllers and a ground control station (GCS) monitoring and controlling the behavior of a single or multiple UAVs – with physical processes, i.e., drones performing missions in the physical world [13]. Furthermore, there are feedback loops between the software, the hardware, and the physical world. For example, the software has to react to its current context, in order to avoid colliding with people or other obstacles.

UAVs in particular, and CPS in general exhibit safety-critical concerns requiring a thorough safety analysis [7] or formal verification [17, 33]. Runtime monitoring has proven viable [15, 20] for detecting and avoiding erroneous behavior. Diverse runtime monitoring approaches with different foci have been proposed [22], including requirements monitoring to check if systems behave according to their requirements [28], runtime verification to detect and possibly react to observed behavior [9], and performance monitoring to collect information about the consumption of computing resources by the monitored system [27].

In our previous work we developed ReMinds, a requirements monitoring framework for Systems of Systems (SoS) and successfully applied it in the domain of automation software for metallurgical plants [32]. Industrial automation in a metallurgical plant requires a CPS controlling the metallurgical processes, with feedback loops between the software, hardware, and the physical world. The automation software is also an SoS, exhibiting characteristics such as decentralized control; support for multiple platforms; inherently volatile and conflicting requirements; continuous evolution and deployment; as well as heterogeneous, inconsistent, and changing elements [21]. ReMinds was developed to address these characteristics and is capable of handling different types of event-based constraints [31]. In our earlier studies we focused on monitoring the automation software, not interacting with the plants' hardware such as cranes, transportation devices, or robots.

In this paper we describe how we applied ReMinds to a UAV control system designed for UAV use in an urban area. Specifically, we used the Dronology research incubator [5] as our study system. Dronology provides a full project environment for managing, monitoring, and coordinating the flights of multiple UAVs. The system provides features to assign missions and to simultaneously control multiple diverse UAVs. It can interact with real hardware (the flying physical UAVs) as well as a fully-fledged, high-fidelity Software-in-the-Loop (SITL) simulator that enables experimentation with virtual UAVs. Both physical and virtual UAVs are controlled by a dedicated GCS that handles commands and messages received from and sent to the UAV. Dronology can handle multiple GCS simultaneously and allows virtual and physical UAVs to interact in the same airspace.

Although UAVs and metallurgical plants are both CPS, there are obvious differences between them. Most notably, in the UAV context, multiple drones operate primarily independently (even though they may cooperate to fulfill a joint mission and to avoid collisions), while in a plant the produced material moves through metallurgical machinery. These and other domain-specific challenges motivated our

research question as to whether ReMinds could be adapted and extended to operate in an entirely different CPS space. Our paper makes the following contributions: (i) we analyze UAV CPS characteristics and derive *monitoring challenges specific for UAV CPS*; (ii) *we extend our* ReMinds *monitoring framework* for use with a UAV CPS; and (iii) we *evaluate the applicability and scalability* of our approach by *monitoring multiple simulated UAVs* and checking constraints at runtime.

The remainder of this paper is structured as follows: in Sections II and III we provide a brief introduction to the areas of UAV CPS and runtime monitoring and present CPS-specific monitoring challenges. We then describe our ReMinds framework and the extensions and adaptations that were necessary to use it for UAV CPS monitoring (Section IV). We evaluate the extended CPS-ReMinds by applying it to a control and management system for UAVs in Section V. We conclude the paper with a discussion (Section VI) and a summary and outlook on future work (Section VII).

II. Context and Related Work

CPS exist in various domains including industrial and smart production systems [6, 19], autonomously driving vehicles [3, 24], and Smart Grids [11, 18], to name a few. Hardware and software components need to be monitored at runtime to detect and react to undesired or even harmful behaviour [28]. In the case of UAVs this includes deviations from assigned flight routes (e.g., due to inaccurate sensor data), unsafe behavior (e.g., violation of minimum separation distances, or entering restricted areas), or implausible sensor data (e.g., due to damaged or faulty hardware). Furthermore, CPS often require features to adapt themselves to their environment at runtime. To this end, several approaches for monitoring CPS in general, and UAVs in particular, have been proposed ranging from approaches using formal techniques such as temporal logic to control theory:

In the domain of CPS, Lee *et al.* [14] have created a system architecture for industry 4.0-based manufacturing systems. Their "5C architecture" provides guidelines for developing and deploying a CPS for manufacturing applications. This includes condition-based monitoring and capabilities for prognostics and health management of individual machines and sensors. Yan *et al.* [26] proposed a spatio-temporal event model for cyber-physical systems, which unlike ReMinds does not support checking values and ranges of monitored data.

Doherty *et al.* [8] presented a task planning and execution monitoring framework for unmanned aircraft using mission plans. The framework relies on temporal action logic to specify the behavior of the system and for reasoning about actions and changes to constraints (e.g., safety constraints). In contrast to their approach, ReMinds supports different kinds of properties including temporal constraints on event sequences and data checks. Engineers can define constraints in a domain-specific language (DSL) [23]. Braberman *et al.* presented MORPH, a reference architecture for self-adaptation [4]. Using the MAPE-K model for monitoring, analysis, planning and execution, they describe three different layers for goal management, strategy management, and strategy enactment in the context of mission planning for UAVs. Their architecture, similar to the ReMinds infrastructure, incorporates probes for instrumenting the target system and a monitoring ("logging") infrastructure combined with a goal model. ReMinds, however, does not target the adaptation of system behavior but focuses on collecting events and data from different systems, checking constraints, and visualizing the monitoring results [12]. Similar work in the domain of self-adaptive systems, for example by Shevtsov *et al.* [25], used Control Theory. They proposed SimCA* an approach for self-adaptive systems used in the context of unmanned underwater systems. SimCA* can handle three different types of requirements – setpoint-, threshold- and optimized requirements – and can deal with changes of requirements at runtime.

Machin *et al.* [15] proposed a formal approach in the domain of autonomous systems for synthesizing monitored behavior rules. They use Computation Tree Logic to describe monitorable properties and a model-checker to validate safety strategies. However, none of these approaches combine the structured collection of safety-related assumptions and constraints (e.g., in the form of a safety-assurance case) with monitoring of these constraints at runtime. Barbosa *et al.* [2] presented Lotus@Runtime, an approach for monitoring execution traces of self-adaptive systems via Labeled Transition Systems. These approaches could complement our approach with capabilities for self-adaptation, e.g., to dynamically adapt the behavior of UAVs.

III. Challenges for Monitoring UAV CPS

In our recent work on requirements-based monitoring [29, 32] we have addressed challenges that arise from the specific characteristics of SoS. Some of these, such as *monitoring across different systems* or *monitoring of different technologies* [32] also hold true for UAV CPS. However, when analyzing and experimenting with the Dronology system we discovered additional characteristics that are crucial for monitoring UAVs.

C1–Tight Integration of HW and SW components: In CPS the actual hardware, in addition to the software, is of particular interest for monitoring. This means, for instance, that probes [16] need to be provided for both software and hardware components. In the case of Dronology and UAV hardware, this includes sensor data collected from the various on-board sensors or firmware-related properties from the UAV flight controllers. Furthermore, since hardware and software are tightly integrated, often constraints exist between those two levels. For example, an internal state change in the flight controller (e.g., from flying to landing) needs to be reflected in the Dronology control component and will trigger certain internal actions that can be observed and monitored.

For ReMinds, only software probes have been developed so far as pointed out above. ReMinds probes need to be able to send timestamped events, including their type and scope (representing the event's origin) and optionally any form of

event data over a network connection to a monitoring server. This should be possible for most hardware devices.

C2–Dynamic Instantiation of Multiple Instances: CPS with tightly integrated hardware exhibit a high degree of dynamism compared to many software systems, as hardware components may need to be frequently added or removed at runtime. For instance, in UAVs sensors or devices may be added during operation. For example, in Dronology, physical or virtual UAVs can be activated at any time while the system is running. This dynamism also affects the monitoring infrastructure and poses additional requirements. When multiple instances of a device exist, the same constraints may need to be checked for each of the currently operating devices. This might be easy for data-related constraints, for example when checking sensor data, but bears additional complexity when checking constraints on the sequences of events, possibly happening concurrently in different UAVs. Multiplicity also needs to be taken into account in the user interface of a monitoring tool, when informing users about violations of constraints that occurred on specific instances. Also, while certain constraints might be applicable to all kinds of UAVs regardless of their configuration – e.g., the number of motors – others might only target UAVs equipped with certain types of sensors or UAVs executing certain tasks. Furthermore, cross-device constraints may exist that require data to be collected from multiple components within the CPS. For example, when checking the minimum safety distance between UAVs operating in the airspace, and controlled by Dronology, location data needs to be collected and analyzed from several different UAVs, potentially operated by multiple different ground control stations.

While ReMinds originally was not intended to support multiple dynamic component instances, its scope model [29] does provide support for multiple instances and dealing with cross-scope constraints. Due to its systems-of-systems focus, ReMinds also already provides support for defining and checking constraints across multiple scopes (i.e., systems or components).

C3–Data-Driven and Event-Driven Monitoring: A key challenge in the CPS context is that both data and events need to be processed. In the case of UAVs, data comprises sensor values indicating attributes such as location, direction, and speed of the drones. At the same time events such as the start of a mission or reaching of a waypoint indicate important state changes that need to be continuously monitored. Furthermore, hybrid constraints are needed for checking both event and data properties in a single constraint. For example, a typical constraint for a UAV is to check that after a mission has been started, a waypoint is reached within a certain time span and that the drone has achieved a certain speed and height. ReMinds addresses this challenge as hybrid constraints can be defined in its DSL [31].

C4–Data Aggregation and Filtering: The focus on data-driven constraints corresponds with an increasing amount of data transmission that needs to be processed at runtime. This calls for additional mechanisms for locally filtering events and data to reduce bandwidth and for processing raw data before subsequent constraint checking. Solutions include simple filters, reducing the amount of data sent to the monitoring infrastructure by the probes, as well as more complex functions for aggregating events and event data. To this end ReMinds provides support for dynamically activating and deactivating probes and constraints at runtime, thus enabling dynamic adaptation and control of the amount of data to be sent. It further provides initial support for processing data by facilitating the development of custom *processors* that can subscribe to certain events and perform filtering and aggregation operations.

To the best of our knowledge no framework or runtime monitoring infrastructure exists that fully addresses all four challenges. While challenges C1 and C3 are to a large extent already addressed by ReMinds, the other two challenges required extensions. In the following we thus report on how we extended and applied ReMinds to provide monitoring support for the Dronology CPS.

IV. ReMinds for Monitoring CPS

The ReMinds framework consists of four layers, each responsible for a certain part of the monitoring process, and a fifth cross-cutting layer providing capabilities for managing variability [32, 30]. The Probing & Instrumentation as well as the Views layers of the framework allow the development of arbitrary system-specific probes and monitoring tools for different domains and technologies, whereas the Aggregation & Distribution and Processing & Analysis layers are independent of the actual systems to be monitored. An EventBroker serves as a central aggregation and distribution point between probes, sending events and data monitored from different systems on the one hand, and applications consuming and processing events and data on the other hand.

The *Requirements Monitoring Model* (RMM) [29] provides the foundation for the ReMinds framework. The model captures three dimensions of SoS monitoring: The *Monitoring Scopes* define the areas of interest to be monitored and hierarchically represent the SoS architecture and, to a certain extent, the organizational structure of the SoS. *SoS Requirements* describe functionalities, properties, or behavior of the SoS to be checked at runtime. Modelers define constraints that formalize SoS requirements to be checked at runtime based on events and event data. *SoS Events* are collected by probes instrumenting different systems in the SoS. They conform to event models providing a uniform representation and common scheme of events and event data. Links are established between the three dimensions, i.e., between requirements and scopes, between constraints (formalized requirements) and events, and between probes (and their events) and scopes, to analyze and diagnose SoS behavior.

The RMM enables abstractions from different technologies and implementation languages (as common in different systems contributing to an SoS), as well as from different hardware and software components (cf. *Challenge C1*). The generic event representation enables generation of events from arbitrary sources, containing arbitrary data, e.g., internal states

Listing 1. Two constraints for Dronology system in the ReMinds DSL.

```
//checks battery level provided by the UAV and its
//respective ground control station
trigger = if event 'UAVStateMessage' occurs
from scope ('GCS.*')
condition = data('state','data/battery') >= 15.

//checks the internal sequence of events that must
//occur when a new flight plan is activated
trigger = if event 'FlightPlanActivated' occurs
condition = events
'GoToCommand' where data('item','uavid')==
trigger.data('item','uavid'),
'WaypointReached' where data('item','uavid')==
trigger.data('item','uavid'),
occur until event 'FlightPlanComplete'
```

of a software component or sensor data collected from a hardware device. This means, that as long as a connection can be established to the EventBroker, events and data can be collected from any hardware or software component.

Regarding the instantiation of components at runtime (cf. *Challenge C2*), ReMinds already provides basic functionality for dynamically creating scope instances. However, the simultaneous instantiation of a large number of individual component instances was not anticipated based on the requirements in the industrial automation systems ReMinds has been used before. Thus, modifications in both the ReMinds user interface and the scope model were necessary for coping with this additional complexity. Dynamic scope instances now are created automatically and attached to the respective parent scope. All constraints assigned to the parent scope are handed down to the instantiated scopes. This allows our ReMinds constraint checker [31] to instantiate and check constraints on individual, dynamically instantiated components. In order to cope with the large number of scope items in the user interface, we also added capabilities to expand and collapse scopes and aggregate information in the parent scope.

The ReMinds DSL is capable of handling diverse types of constraints. Thus, ReMinds is particularly useful for supporting both event and data-driven constraint checks (cf. *Challenge C3*). Furthermore, the DSL supports hybrid constraints combining checks regarding event sequences with data checks. Two examples of UAV constraints using our constraint DSL are described in Listing 1. The first one performs a simple data check on the "StateMessage" provided by the GCS of the UAV, whereas the second one checks a more complex (internal) event sequence for flight plan execution.

Additionally, regarding *Challenge C4*, ReMinds provides templates for *processors* that are instantiated for certain event types. This allows custom functions to be added and also provides support for creating new events (and data), which in turn can be checked by other constraints. This mechanism can, for example, be used to implement processors that create custom events for checking restricted no-fly areas or for calculating distances between UAVs controlled by Dronology for collision avoidance.

V. Case Study: The Dronology UAV System

To investigate the usefulness of the ReMinds framework in the context of a UAV CPS, we applied it to the Dronology system. This included the creation of a Dronology monitoring model (i.e., scopes for hardware and software components, constraints to be checked, and types of events); developing probes generating the events in the Dronology system; and implementing processors to aggregate events and data for complex constraints. Furthermore, to evaluate scalability, we collected performance data for the ReMinds framework and the constraint checker for three different scenarios using the SITL simulator. More precisely, we aimed to answer the following research questions:

RQ1: Is the ReMinds *approach applicable to a UAV CPS, i.e., can the essential constraints be defined and monitored?* Based on the architecture of the Dronology system, we selected four different internal components and the external GCS and created scopes and constraints to be checked at runtime for these elements.

RQ2: Does the ReMinds *framework and the constraint checking approach scale to the needs of UAV CPS?* To assess the scalabiliy in the context of UAV CPS, which involve a potentially high number of hardware instances, we conducted a series of simulations with an increasing number of UAVs, events, and constraint instances to be checked at runtime. Specifically, in three different simulation runs we measured the number of events collected, the number of constraint checks, as well as the time necessary to evaluate single constraint instances.

A. RQ1: Applicability to Dronology CPS

ReMinds was designed to instrument technologically heterogeneous systems using probes [32]. Collecting events and data from Dronology and the respective UAVs was thus pretty straightforward. We implemented a "Connector" that receives information from the Dronology internal monitoring component as JSON messages and transforms them into ReMinds' internal representation of events and data, the foundation for subsequent constraint checks. We thus could collect information from both the actual hardware (i.e., the UAVs connected via a GCS) as well as internal states and information provided via probes instrumenting the Dronology system itself (cf. *Challenge C1*). Internally, we employed a logging approach in Dronology for our prototype implementation. Logging statements are directly added to the source code to relay information to ReMinds. This could be easily replaced by other techniques, such as aspect-oriented probes we have used in the past to instrument and monitor systems without direct access to the source code [32].

Specifically, for monitoring the internal behavior of Dronology we decided to instrument four vital components of the system and represent them as scopes in our RMM (cf. Fig. 1): *UAV Flight Management* handling active flights (including safe take-off and landing), *UAV Route Planning* handling flight plans (i.e., series of waypoints) and assigning them to UAVs, *UAV Vehicle Control* internally managing the individual UAVs

TABLE I
CONSTRAINTS MONITORED IN THE DRONOLOGY SYSTEM.

Constr.	Type	Description	# Constraints checked			Median Eval. Time [ms]		
			S1	S2	S3	S1	S2	S3
C-01	data	Flight-Controller Mode: The internal mode of the UAV needs to be set to either "GUIDED" or "STABILIZE".	179,566	358,786	536,667	0.22	0.74	0.70
C-02	data	Speed-Limit: The UAVs ground speed must not exceed 36 km/h (10 m/s).	179,566	358,786	536,667	0.27	0.58	0.72
C-03	data	Altitude-Limit: The UAVs exceed a maximum altitude of 120 m (400 ft).	179,566	358,786	536,667	0.26	0.58	0.72
C-04	data	Battery-Level: The battery level of the UAV must not go below 10%.	179,566	358,786	536,667	0.27	0.57	0.72
C-05	data	GPS-Fix: The UAV must have at least a fix with 5 satellites at any time.	179,566	358,786	536,667	0.2	0.55	0.69
C-06	internal	Separation-Distance: The minimum distance between each UAV flying must be > 10 m at any time.	1,490,954	6,416,862	13,631,540	0.11	0.66	0.69
C-07	internal	Handshake-Location: The home location of the UAV sent on handshake with the GCS must correspond to the location received afterwards.	10	20	30	1.35	2.02	3.67
C-08	internal	GoTo-Waypoint: After a "Take-off" command is issued by the system a "Goto-Waypoint" command must be issued within 5 seconds.	10	20	30	13.84	1.12	1.17
C-09	internal	GoTo-Waypoint: After a "Goto-Waypoint" command is issued by the system a "Set-Groundspeed" command must be issued within 5 seconds.	2,784	5,248	8,414	3.47	1.27	1.22
C-10	internal	FlightPlan Sequence: When a new flight plan is assigned to a UAV, any number of "Goto-Waypoint" commands may be executed before a "FlightPlan-Complete" event is received.	56	110	141	7.63	0.84	0.95

and their internal states and performing respective transitions, and *UAV Safety Monitor* handling internal data such as location in the airspace. In total we created 12 different probe points collecting information from these four components.

We defined ten different constraints related to safe behavior of the UAV (e.g., maximum prescribed altitude, minimum separation distance) and internal functionality, such as correct execution of a flight plan. A full list of constraints can be found in Table I.

Internal constraints, for example, include proper transitions between different states (e.g., a drone may only take-off after safety checks have been performed), or data checks upon activation (e.g., when a new UAV is activated at a location, its position needs to be within a predefined area surrounded by a geofence). The simulated or physical UAVs are managed by their respective GCS, which dynamically instantiates UAV scopes when they register with the GCS. Constraints regarding UAV data – real or simulated – (cf. C-01 – C-05 in Table I) are associated with the GCS and are automatically instantiated for each new UAV scope (cf. Fig. 2).

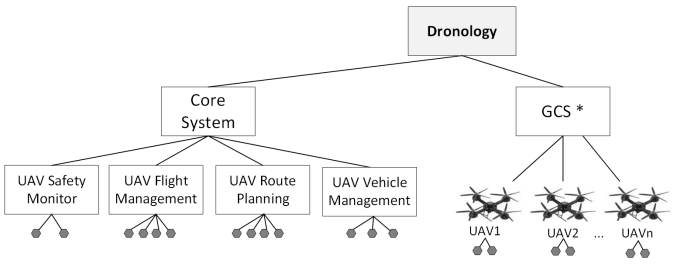

Fig. 1. Monitoring scopes of the requirements monitoring model for Dronology.

Some of the constraints used in our experiments are safety-critical, such as the proper mode when a UAV is activated (cf. C-01), or maintaining a minimum battery level (cf. C-04). Others provide important information regarding the overall system's behavior (e.g., C-08 or C-09), checking that a command sent to the UAV has been received within a certain amount of time. For such constraints we used REMINDS' capabilities for defining the severity for different constraints.

We also used REMINDS capabilities to implement processors that interface with REMINDS and pre-process raw events and data before checking constraints. For example, for calculating the distance between UAVs, a "Separation Distance" processor collected and aggregated location events sent by the UAV instances and periodically calculated the distance between them which in turn resulted in new "processed_distance" events that were used for checking the respective minimum separation distance constraint (cf. constraint C-06).

B. RQ2: Scalablity

For demonstrating the scalability of REMINDS in the context of a UAV CPS, we conducted a series of simulations with Dronology. We used our monitoring infrastructure and the ardupilot SITL simulator [1], a high-fidelity simulator – which

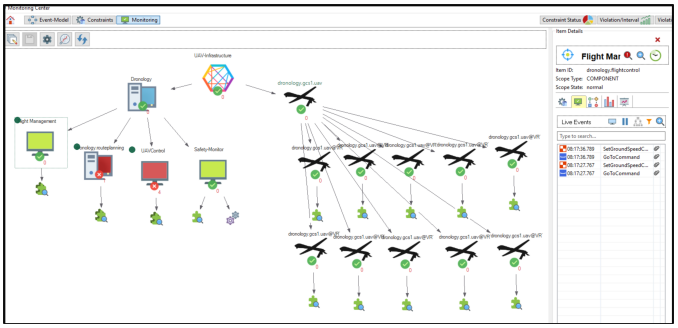

Fig. 2. ReMinds monitoring GUI showing Dronology internal scopes (left) and dynamically instantiated UAV scopes (right).

we connected to the Dronology environment – for conducting simulations with virtual UAVs. We performed three different simulation runs, with an increasing number of UAVs (from 10 to 30) resulting in a growing number of events collected and constraints checked. An overview of the three scenarios alongside the results can be found in Table II. For each of the scenarios we performed a 5-hour simulation run where we randomly assigned flight routes (i.e., a series of waypoints) to the UAVs.

The goal of our evaluation was to investigate, whether the ReMinds infrastructure is capable of handling the events of the UAVs and the different kinds of constraints. We did not perform an extensive, long-running performance analysis of the monitoring infrastructure or the constraint checker, as this was part of our previous work [31, 32].

To minimize invalid data measurements due to the influence of the Java Compiler we included a warm-up time of 5 minutes before each run. We measured the number of events that occurred, the number of constraint checks performed, the (median) evaluation time for each constraint (in ms), and the memory consumption of the framework (in MB). The simulation environment and the monitoring infrastructure were set up on a standard Desktop machine with an Intel(R) Xeon CPU with 3.40GHz and 64GB RAM running Windows 10 64-Bit.

TABLE II
Simulation Scenarios with # Events and # Constraint Checks.

Scen.	# UAVs	Scope	# Events	# Const. Checks
S1	10	UAV Flight Mgmt.	5,570	2,784
		GCS	358,859	897,827
		UAV Safety Monitor	1,489,708	1,490,954
		UAV Route Planning	1,823	56
		UAV Vehicle Control	206	10
		(Total)	1,856,166	2,391,641
S2	20	UAV Flight Mgmt.	10,488	5,268
		GCS	715,768	1,793,930
		UAV Safety Monitor	6,400,708	6,416,862
		UAV Route Planning	3,609	110
		UAV Vehicle Control	387	20
		(Total)	7,130,949	8,216,190
S3	30	UAV Flight Mgmt.	16,856	8,444
		GCS	1,073,074	2,683,321
		UAV Safety Monitor	13,628,342	13,631,540
		UAV Route Planning	3,609	141
		UAV Vehicle Control	624	30
		(Total)	14,723,450	16,323,476

During the three five-hour simulation runs, approximately 1.9 (S1), 7.1 (S2) and 14.7 (S3) million events and related data were captured, and 2.4 (S1), 8.2 (S2), and 16.3 million constraints checks were performed resulting in an average of about 133 checks per second for S1, 457 checks per second for S2, respectively 907 checks per second for S3. A detailed overview of events collected for the five different scopes (cf. Section V-A), as well as checks performed for the different constraints can be found in Table II and Table I.

Similar to our previous work [31] we also measured the time required for performing a constraint check (i.e., triggered by an event received from Dronology). Our goal was to assess whether the type of the system, or the different types of constraints that were checked influence the evaluation time as they rely on checking different kinds of data with different frequencies and complexity. An overview of the different constraints and their evaluation times can be found in Fig. 3. The median evaluation times are low, ranging from 0.11 ms (C-06) to 13.89 ms (C-08), which is more than sufficient to provide instant feedback to the user, at runtime. In scenario 1, the evaluation times for C01-C06 are below 1ms, while for C07-C10 they range from 1 to 14ms. For scenarios 2 and 3, the evaluation times for all constraints are around 1ms, except for C-07. Overall, these evaluation times are acceptably very low. The higher evaluation times for some constraints can be explained by the fact that multiple event data elements such as location and waypoint information need to be processed. Also, the memory consumption of the Java Virtual machine remained unobtrusive with a maximum of 930 MB during the S3 five-hour evaluation run.

VI. Discussion and Threats to Validity

We had to extend ReMinds with additional capabilities for adding processors and providing better support for dynamic instantiation of scopes in order to address the specific challenges of monitoring the UAV CPS. Our evaluation demonstrates that the ReMinds framework and its constraint checking mechanism are capable of checking constraints on large numbers of events as needed in CPS. The monitoring model allowed us to model the Dronology CPS and to express domain-relevant constraints for both internal components and the external hardware components in the constraint DSL of ReMinds (RQ1). Additionally, the constraint checker and the framework itself were able to handle very large numbers of events (>14 million for S3) and constraint checks (> 16 million for S3) during the five-hour simulation runs (RQ2). We are therefore confident that the framework implementation is a suitable basis for (UAV) CPS applications.

As with any case study research, the results of our work might not generalize beyond the case of UAV control and

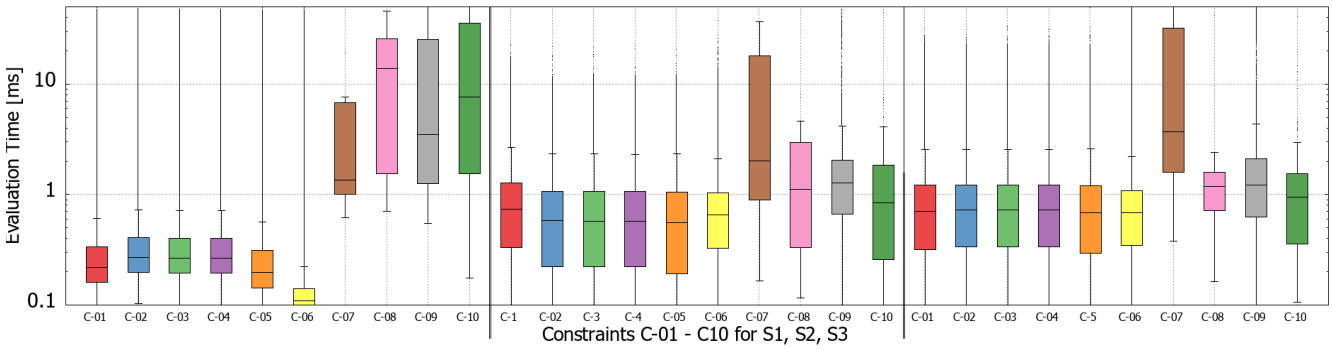

Fig. 3. Analysis of constraint evaluation times for all ten constraints grouped by the three scenarios

management that we considered. We presented an in-depth analysis of the selected case, which we believe is of value to the CPS community. Further, our study has confirmed that ReMinds is extensible and flexible to address monitoring challenges specific to CPS. We are thus confident that it can be adapted to other similar systems.

Regarding data measurement, as the scenarios are based on using simulated UAVs, the resulting events and data might be different using physical UAVs and hardware. However, by using the high-fidelity SITL-simulator, and based on our experiences with both virtual and physical UAVs, we argue that these resemble the actual behavior of physical UAVs to a large extent. In terms of simulation parameters, we increased the number of UAVs for each simulation run (from 10 for S1, to 30 for S3) to demonstrate scalability under realistic conditions. Another threat to validity is the duration of our evaluation runs. While five-hour runs may seem rather short for typically long running CPS systems, previous evaluations [29, 31] of the monitoring framework confirmed that ReMinds is capable of handling large amounts of events and data also over longer periods of time. Also, in the context of UAVs missions typically take less than five hours due to limitations such as battery capacity.

VII. Conclusions

In this paper we presented a case study in which the ReMinds requirements monitoring framework was used to monitor a CPS infrastructure for managing and controlling UAVs. Based on analyzing the case study system Dronology, we derived new challenges for monitoring CPS and then presented the extensions we developed for ReMinds to address these challenges. Our experimental results show that ReMinds, together with the extensions we implemented for the Dronology system, can monitor the CPS at runtime based on collecting events from different software and hardware components and can also effectively check constraints. In our future work we plan to add support for adapting the behavior of UAVs based on constraint evaluation results. We will also improve the visualization of monitoring results for different components of the CPS.

Acknowledgements

This work has been supported by the Christian Doppler Forschungsgesellschaft Austria, Primetals Technologies, the US National Science Foundation (Grant No. CCF-1741781) and the Austrian Science Fund (FWF) under Grant No. J3998-N31.

References

[1] ArduPilot. SITL Simulator. http://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html, 2018. [Online; accessed 04-June-2018].

[2] D. M. Barbosa, R. G. D. M. Lima, P. H. M. Maia, and E. Costa. Lotus@Runtime: A tool for runtime monitoring and verification of self-adaptive systems. In *Proc. of the 12th Int'l Symp. on Software Engineering for Adaptive and Self-Managing Systems*. IEEE, 2017.

[3] C. Berger and B. Rumpe. Autonomous driving – 5 years after the urban challenge: The anticipatory vehicle as a cyber-physical system. *CoRR*, abs/1409.0413, 2014.

[4] V. A. Braberman, N. D'Ippolito, J. Kramer, D. Sykes, and S. Uchitel. MORPH: a reference architecture for configuration and behaviour self-adaptation. In *Proc. of the 1st Int'l Workshop on Control Theory for Software Engineering*, pages 9–16, 2015.

[5] J. Cleland-Huang, M. Vierhauser, and S. Bayley. Dronology: An Incubator for Cyber-Physical Systems Research. In *Proc. of the 40th Int'l Conf. on Software Engineering: New Ideas and Emerging Results*, 2018.

[6] A. W. Colombo, S. Karnouskos, and T. Bangemann. Towards the next generation of industrial cyber-physical systems. In *Industrial cloud-based cyber-physical systems*, pages 1–22. Springer, 2014.

[7] E. Denney, G. Pai, and I. Whiteside. Modeling the safety architecture of UAS flight operations. In *Proc. of the 36th Int'l Conf. on Computer Safety, Reliability, and Security*, pages 162–178, 2017.

[8] P. Doherty, J. Kvarnstròm, and F. Heintz. A temporal logic-based planning and execution monitoring framework for unmanned aircraft systems. *Autonomous Agents and Multi-Agent Systems*, 19(3):332–377, 2009.

[9] C. Ghezzi, A. Mocci, and M. Sangiorgio. Runtime monitoring of component changes with Spy@Runtime. *34th Int'l Conf. on Software Engineering*, pages 1403–1406, 2012.

[10] ICAO. (International Civil Aviation Organization), Unmanned Aircraft Systems (UAS), Circular, CIR 328, AN/190. CIR 328, AN/190, Montreal, 2011.

[11] S. Karnouskos. Cyber-physical systems in the SmartGrid. In *9th IEEE Int'l Conf. on Industrial Informatics*, pages 20–23. IEEE, 2011.

[12] L. M. Kritzinger, T. Krismayer, M. Vierhauser, R. Rabiser, and P. Grünbacher. Visualization support for requirements monitoring in systems of systems. In *Proc. of the 32nd IEEE/ACM Int'l Conf. on Automated Software Engineering*, ASE 2017, pages 889–894, 2017.

[13] E. A. Lee. Cyber physical systems: Design challenges. In *11th IEEE Int'l Symp. on Object-oriented Real-time Distributed Computing*, pages 363–369. IEEE, 2008.

[14] J. Lee, B. Bagheri, and H.-A. Kao. A cyber-physical systems architecture for industry 4.0-based manufacturing systems. *Manufacturing Letters*, 3:18–23, 2015.

[15] M. Machin, F. Dufossé, J.-P. Blanquart, J. Guiochet, D. Powell, and H. Waeselynck. Specifying safety monitors for autonomous systems using model-checking. In *Proc. of the 33rd Int'l Conf. on Computer Safety*, pages 262–277, 2014.

[16] M. Mansouri-Samani and M. Sloman. Monitoring distributed systems. *IEEE Network*, 7(6):20–30, 2003.

[17] S. P. Miller, M. W. Whalen, and D. D. Cofer. Software model checking takes off. *Communications of the ACM*, 53(2):58–64, 2010.

[18] Y. Mo, T. H.-J. Kim, K. Brancik, D. Dickinson, H. Lee, A. Perrig, and B. Sinopoli. Cyber–physical security of a smart grid infrastructure. *Proc. of the IEEE*, 100(1):195–209, 2012.

[19] L. Monostori. Cyber-physical production systems: Roots, expectations and R&D challenges. *Procedia CIRP*, 17:9 – 13, 2014.

[20] P. Moosbrugger, K. Y. Rozier, and J. Schumann. R2u2: monitoring and diagnosis of security threats for unmanned aerial systems. *Formal Methods in System Design*, 51(1):31–61, Aug 2017.

[21] C. B. Nielsen, P. G. Larsen, J. Fitzgerald, J. Woodcock, and J. Peleska. Systems of systems engineering: Basic concepts, model-based techniques, and research directions. *ACM Computing Surveys*, 48(2):1–41, 2015.

[22] R. Rabiser, S. Guinea, M. Vierhauser, L. Baresi, and P. Grünbacher. A comparison framework for runtime monitoring approaches. *Journal of Systems and Software*, 125:309–321, 2017.

[23] R. Rabiser, M. Vierhauser, and P. Grünbacher. Assessing the usefulness of a requirements monitoring tool: a study involving industrial software engineers. In *Proc. of the 38th Int'l Conf. on Software Engineering*, pages 122–131, 2016.

[24] R. Salay, R. Queiroz, and K. Czarnecki. An analysis of ISO 26262: Using machine learning safely in automotive software. *arXiv preprint arXiv:1709.02435*, 2017.

[25] S. Shevtsov, D. Weyns, and M. Maggio. Handling new and changing requirements with guarantees in self-adaptive systems using SimCA. In *Proc. of the 12th Int'l Symp. on Software Engineering for Adaptive and Self-Managing Systems*, pages 12–23. IEEE, 2017.

[26] Y. Tan, M. C. Vuran, and S. Goddard. Spatio-temporal event model for cyber-physical systems. In *Proc. of the 29th IEEE Int'l Conf. on Distributed Computing Systems, Workshops*, pages 44–50. IEEE, 2009.

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

[28] M. Vierhauser, R. Rabiser, and P. Grünbacher. Requirements monitoring frameworks: A systematic review. *Information and Software Technology*, 80:89–109, 2016.

[29] M. Vierhauser, R. Rabiser, P. Grünbacher, and B. Aumayr. A requirements monitoring model for systems of systems. In *Proc. of the 23rd IEEE Int'l Requirements Engineering Conf.*, pages 96–105. IEEE, 2015.

[30] M. Vierhauser, R. Rabiser, P. Grünbacher, C. Danner, S. Wallner, and H. Zeisel. A flexible framework for runtime monitoring of system-of-systems architectures. In *Proc. of the IEEE/IFIP Conf. on Software Architecture*, pages 57–66. IEEE, 2014.

[31] M. Vierhauser, R. Rabiser, P. Grünbacher, and A. Egyed. Developing a DSL-based approach for event-based monitoring of systems of systems: Experiences and lessons learned. In *Proc. of the 30th IEEE/ACM Int'l Conf. on Automated Software Engineering*, pages 715–725. IEEE, 2015.

[32] M. Vierhauser, R. Rabiser, P. Grünbacher, K. Seyerlehner, S. Wallner, and H. Zeisel. ReMinds: A flexible runtime monitoring framework for systems of systems. *Journal of Systems and Software*, 112:123–136, 2016.

[33] M. W. Whalen, D. Cofer, and A. Gacek. Requirements and architectures for secure vehicles. *IEEE Software*, 33(4):22–25, 2016.