AIAA SciTech Forum
9 - 13 January 2017, Grapevine, Texas
AIAA Information Systems-AIAA Infotech @ Aerospace

Certifiable Runtime Assurance of Distributed Real-Time Systems *

Sagar Chaki† and Dionisio de Niz*
Carnegie Mellon Software Engineering Institute, Pittsburgh, PA 15213, USA

Distributed real-time embedded systems operating in uncertain and contested environments are of great relevance to the aerospace community. Such systems rely on components with unpredictable behavior to provide mission-critical capability. However, these components pose a challenge to assuring system-level safety and security. Runtime assurance (RA) has been used successfully to balance such *capability with confidence* in various non-DRTS domains. This paper presents a project we are pursuing with the aim of developing a provably correct approach for RA of DRTS. We discuss the core technical thrust areas, present two challenge problems we are using to guide our research, and conclude with an experimental testbed and initial results.

I. Introduction

Distributed real-time embedded systems operating in uncertain and contested environments (e.g., multi-UAS missions) are of great relevance to the aerospace community.[1] Typically, such systems rely on components with unpredictable behavior (and potentially from untrusted sources, such as COTS) to provide mission-critical capability (e.g., use machine learning for effective decision making under uncertainty). Yet, by their very non-deterministic nature, these components pose a challenge to assuring system-level safety and security. Runtime assurance (RA)[2] has been used successfully to balance such *capability with confidence* in various non-DRTS domains. Following this lead, our work aims at developing a provably correct approach for RA of DRTS.

The key idea behind RA is to monitor system behavior, and take preemptive action to prevent the system from entering an unsafe or insecure state. This idea has emerged in several domains. For example, the Simplex[3] architecture switches from an efficient untrusted controller to a trusted inefficient one to prevent instability. Similarly, runtime verification[4–7] generates monitors from formal specifications and uses them to detect errors by observing a system's execution traces. Mixed-criticality real-time schedulers, such as ZSRM,[8] monitor the execution time of threads and implement protection schemes that ensure graceful degradation in the presence of overloads. Finally, Schneider's *security automata*,[9] and their generalization to *edit automata* by Ligatti et al.,[10] are used to monitor and modify a system's interaction with its environment to enforce specific security policies.

However, the applicability of RA to DRTS operating in uncertain and adversarial environments, and with unpredictable components, is limited by several open challenges: (i) how to specify policies that involve both timing and functionality, e.g., event α should never happen within time t of event β; (ii) how to assure (functional and timing) correctness of the monitors themselves – *Quis custodiet ipsos custodes?*;[11] (iii) in

*This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN AS-IS BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT. [Distribution Statement A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution. DM-0004287

†Principal Researcher, Carnegie Mellon Software Engineering Institute, 4500 Fifth Ave., Pittsburgh, PA 15213, USA

practice, we envision that a component will need to respect multiple policies; interaction between monitors is non-trivial; determining whether (and how) to use a set of monitors together is an open problem; (iv) ultimately, we are interested in ensuring system-level policies; thus, we must verify that a set of component-level policies imply a system-level one; (v) this verification should be both *compositional* and *incremental* so that it applies to systems that have many *evolving* components; and (vi) given unpredictable components and an attacker, the RA must be implemented in a provably secure, yet effective and performant, manner.

We propose to address these challenges by: (i) using timed automata[12] to express component-level contracts that combine functional and timing correctness; (ii) developing a procedure to formally verify, at the source code level, that a monitor correctly implements a contract; (iii) developing a procedure to check compatibility between monitors, and determining how they should be applied on a component; (iv) developing assume-guarantee style verification techniques to prove that the composition of several component-level contracts leads to a specific system-level property; and (v) developing a system architecture where RA can be deployed in a secure manner on a system with unpredictable components, and an adversary with well-defined capabilities.

The rest of this paper is organized as follows. Section II surveys related work. Section III elaborates on the various thrust areas of our research. Section IV presents two challenge problems we are using to guide our research. Section V describes our experimental platform and preliminary experimental results, and Section VI concludes the paper.

II. Related Work

The idea of using a monitor to ensure that a system does not get into an undesirables state was proposed in the context of control systems by Seto et al.[3] They called this approach "Simplex". The key idea was that if we have a complex and more capable, but untrusted, controller C_{comp}, and a simple but trusted controller C_{simp}, then we can use C_{comp} as long as it does not take the system into a state from which it can no longer be recovered, i.e., moved back to a safe state by C_{simp}. This condition is monitored, and as soon as the system is in danger of moving to a non-recoverable state, the monitor switches to C_{simp}. The Simplex idea has been demonstrated on a number of real-life systems, such as an inverted pendulum. More recently, Bak et al.[13] have developed a more sophisticated version of Simplex that combines offline analysis with hybrid reachability at runtime to further push the envelop of recoverability.

The idea of runtime monitoring has also been used in other contexts, such as formal verification.[4,5] The key idea here is that instead of verifying the system exhaustively in a static manner, which has scalability issues due to *statespace explosion*, we check for violations of safety properties at runtime. This is more tractable since we are only analyzing the states that are actually reachable during a concrete execution.

Schneider proposed "security automata"[9] as a formalism to express security properties whose violations can be detected at runtime. Originally, security automata were passive, i.e., they only monitored the system for safety violations. More recently, Ligatti at al.[10] have generalized this idea to "edit automata" that can not only monitor system inputs and outputs, but also modify them as needed.

In the domain of real-time scheduling, monitors have also been used widely, particularly to enforce CPU usage budgets by threads. For example, the ZSRM[8] mixed-criticality scheduler handles threads with different priorities and criticalities. It allocates CPU cycles to these threads in a way that respects priorities (during nominal execution) and criticalities (during overload execution). To this end, it uses alarms to intercept thread execution and take appropriate preemptive and budget enforcement steps.

Compared to all these works, our main contributions are in monitoring both logical and timing properties, producing verified monitor implementations, analyzing interactions between monitors in a compositional manner, and developing a secure deployment of monitors that is resilient against realistic attackers.

III. Thrust Areas

Our research will proceed in six inter-related thrust areas: (i) specifying contracts; (ii) verifying monitors; (iii) monitor compatibility and composition; (iv) verifying system-level policies; (v) system evolution; and (vi) secure deployment. We now discuss these thrust areas in more detail.

III.A. Specifying Contracts

The specification of component-level contracts is done via timed automata.[12] Timed automata are finite automata augmented with clocks. Transitions are guarded by conditions on clock values, and reset clocks as part of their associated action. Timed automata are able to express a rich class of timed behavior, and hence are an appropriate formalism for capturing component contracts that involve both functionality and timeliness.

Timing properties are particularly important for aerospace systems where both the correctness of a value computed as well as when such value is available are necessary to ensure the proper interaction with the physical environment. From the contract perspective and a component internal point of view, the contract needs to guarantee that that this component will receive enough CPU cycles to ensure that it will finish its computations on time. This means that given a contract for a component C_1 defined as a time automaton T_1^s that describes the minimal availability of CPU time provided to the component as intervals of time when the CPU is available (a.k.a. resource supply-bound function – sbf[14]) we should be able to apply a verification function $S(T_1^s, C_1)$ that verifies that all threads of component C_1 meet their timing requirements. From the inter-component point of view a contract needs to define a time automaton T_1^d that defines the maximum consumption of CPU as intervals of times when the CPU is used (a.k.a. resource demand-bound function – dbf[15]). T_1^d indicates the CPU consumed by C_1 (the threads inside C_1) and hence not available to other components. In the ideal case T_1^s will be equal to T_1^d (meaning that all the CPU time provided to the component is used – no waste) but that depends on the thread scheduler used inside the component, the scheduler used between components, the monitors used, and the allocation of CPU time to each component. In the general case the verification function can be expressed as $S(T_1^s, T_1^d)$ expressing that the resource supply (sbf) should be able to satisfy the resource demand (dbf). For instance in ARINC 653[16] sets of timeslots within a major frame are assigned to each component (activating them in their assigned slots) and within each slot fixed-priority scheduling is used.

III.B. Verifying Monitors

Given a monitor M at source code level, and contract expressed as a timed automaton T, our goal is to check that M enforces T, i.e., every behavior allowed by M is a legal behavior according to T. Formally, this boils down to verifying that T *simulates* M, i.e., $M \preceq T$. Since M is available as source code, we will check simulation by manually constructing a formula R that relates states of M and T, and using auto-active verification tools (such as Frama-C) to prove that R is a simulation relation. We simplify the verification by assuming that M is implemented as a while loop Loop that repeatedly waits for an action from the component being controlled (e.g., a function being called) and responds by passing it on, or replacing it with zero or more other actions (i.e., function calls.) Thus, using R, we only need to show that the body of Loop simulates the transition relation of T. Typically, this will mean that R is inductive over the execution of one iteration of Loop, and one step of T. From the timing point of view time monitors M^t will restrict the amount of CPU given to the component ensuring that the supply function (T^s) is respected, i.e., $M^t \preceq T^s$. Note that given that this is only an approximation, it is possible for the enforcer to provide more CPU time than is required leading to waste. The design of the appropriate contracts and enforcers is key for an efficient implementation. In this case, we will also verify the implementation of the monitors M^t.

III.C. Monitor Compatibility and Composition

Consider two monitors for a component C: M_1 counts the number of times C performs an action α, while M_2 suppresses α whenever it is attempted within time t of action β. Clearly, the order in which M_1 and M_2 are applied affects the final count obtained by M_1. A similar issue arises if M_2 inserts α instead of suppressing it. We address this challenge in two phases. In the first phase, an algorithm takes a set of monitors $M_1, \ldots, M_k$ and computes an ordering such that whenever M_i requires an action as input that is suppressed (or inserted) by M_j, then M_i appears before M_j in the ordering. In the second phase, this is generalized by distinguishing between actions produced by components and those produced by monitors. In this scenario, monitor M_1 could be interested either in α produced by C (in which case it should be applied before M_2) or by M_2 (in which case it should be applied after M_2). In all situations, we detect (and warn) when no feasible monitor orderings exist (e.g., due to circular dependencies). Timing monitors introduce an interesting problem given that they can pause the execution of a component when it tries to execute longer

than specified in the contract. As a result, a value expected by a functional monitor may be suddenly absent. Our composition scheme also handles this situation.

III.D. Verifying System-Level Policies

So far, we have considered component-level contracts and monitors only. In practice, systems will consist of a number of concurrently executing and interacting components. We see concurrency in DRTS at two-levels: (i) intra-node where multiple threads execute on the same processor, and interact via shared memory with sequential consistency; (ii) inter-node where threads executing on separate processors or physically separated nodes interact via shared memory with weaker notions of consistency (e.g., total store order) or via message passing. In both cases, the problem is to verify that a set of low-level monitors implement a global policy. This is challenging because monitors can interact in complex ways, and since they execute in parallel, verifying them exhaustively in a brute-force way leads to statespace explosion. To this end, we use assume-guarantee[17] reasoning verify a collection of monitors compositionally. We create assume-guarantee style proof rules for monitors that enforce timing constraints. In addition, we create proof rules that handle both shared memory and message passing based communication between monitors. From the timing perspective, integrating the components together implies ensuring that the *sbf* of all components can be satisfied by the system scheduler.

III.E. System Evolution

Systems are continuously evolving. This evolution typically happens in an incremental fashion by replacing, adding, and/or removing components. These components can be completely new or reused from previous projects. As a result, contracts and monitors should enable this evolution. From the timing perspective this requires the *sbf* of the contract of a component to be flexible enough to allow variations in the periodicity and execution times of the threads (among other timing parameters). The challenge here is to describe T^s (and their corresponding M_T) flexible enough to ensure that $S(T^s, T^{d'})$ holds for widely-varying $T^{d'}$s (modification of the original *dbf* contract T^d). Current standards that allow the definition of contracts are rather brittle. For instance, ARINC 653 defines time partitions that repeat every major frame. Typically, the period of each task must be a multiple of the major frame. Unfortunately, this means that even a small reduction in the period of a task can force a recalculation of major frame and the corresponding partitions, i.e., triggering a system-wide change. This has been a challenge for incremental verification. For instance, the FAA allows limiting the re-certification of a modified system to only certifying the modified components if it can be proven that their external behavior (contract) remains the same and does not interfere with the other components. Brittle contracts offer very little opportunities to preserve previous certifications. Furthermore, the composition of timing and logical contracts and the corresponding monitors has not been analyzed and remain an open challenge as discussed before.

III.F. Secure Deployment

We envision an architecture where each monitor is implemented as a module that is logically and temporally isolated from its target component. This means that the monitor will perform correctly under arbitrary actions and CPU usage by the component. This takes care of the requirement that the monitors are unpredictable (and potentially untrustworthy). We explore implementing each monitor as a HypApp of the XMHF hypervisor.[18] XMHF has been already verified to provide memory integrity (i.e., logical isolation) between the untrusted guest OS (containing all components) and its HypApps. Moreover, each HypApp is implemented as a function that will be invoked (by the hardware) whenever a specific low-level intercept (e.g., accessing a device) occurs. This maps naturally to our monitors as event handlers. However, additional research is needed to determine a procedure by which high-level contracts can be mapped to low-level intercepts, and a collection of verified monitors can be developed and deployed as XMHF HypApps.

IV. Challenge Problems

To guide our work, we are using two challenge problems, inspired by real-world scenarios.

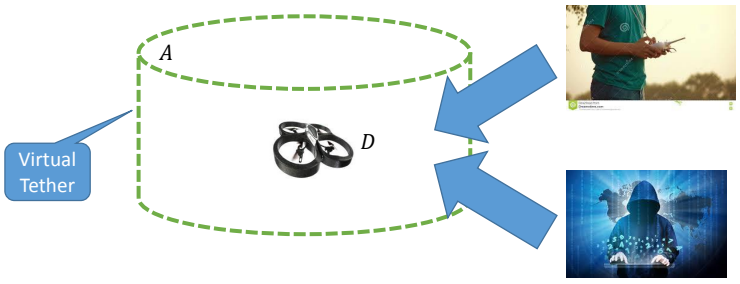

Figure 1. The Virtual Tether Challenge Problem.

IV.A. Virtual Tether

In our initial experiments with commercially available drones, we noticed that they would occasionally move in ways that were not intended by the human operator. For example, in indoor settings, this often manifested as the drone suddenly moving up and hitting the ceiling. To solve this problem, we considered tethering the drone physically by a wire to a heavy weight on the ground. Our first challenge problem is to achieve this goal via runtime assurance. Specifically, given a drone D, and a well-defined 3-dimensional geographical area A, we want to develop an enforcer that will ensure that D does not move out of A. Our threat model, as shown in Figure 1, includes both incorrect remote guidance by an human operator, as well as malicious intrusions by cyber-attackers.

Obviously, this technique can also be used to solve the complementary problem of ensuring that the drone never enters a forbidden area F. This problem is sometimes referred to as *geo-fencing*[a]. Indeed, there have been proposals [b] to use geo-fencing to regulate drone movement without stifling innovation.

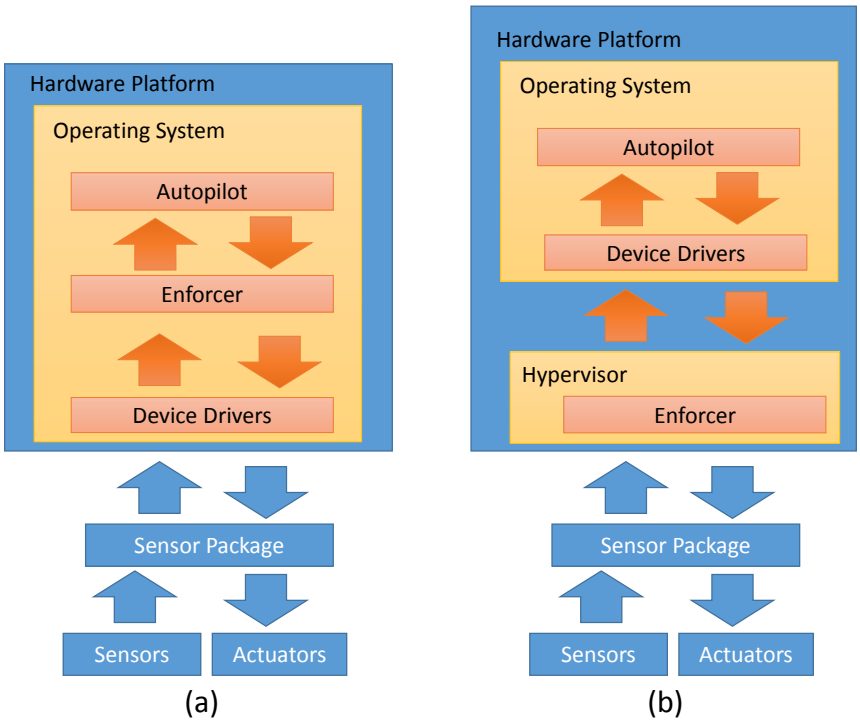

Figure 2. Runtime architecture for the two stages of virtual tether.

To solve this problem, we assume the drone has access to localization information, i.e., knowledge of where it is situated in a well-defined 3-dimensional coordinate system. In practice, this can be achieved

[a] https://en.wikipedia.org/wiki/Geo-fence
[b] http://alcalde.texasexes.org/2015/03/todd-humphreys-dont-overregulate-drones

via GPS, or a more localized mechanism, depending on the situation. We plan to solve the problem in two stages:

- In the first stage, we will trust not only the hardware, but also all the software components other than the ones that may be providing movement commands to the drone. For example, in a human-operated scenario, we trust the auto-pilot but not the human operator. If the drone moves autonomously, then we do not trust the software components producing guidance information, such as waypoint. Figure 2(a) shows a possible runtime architecture for this stage of the problem. From a security perspective, this has a very weak attacker model, since we have are assuming a large trusted computing base, including the autopilot and the entire operating system. Given the large number of software vulnerabilities that are discovered and exploited routinely, a solution at this stage is only a stepping stone toward a secure deployment.
- In the second stage, we will reduce our trusted computing base to only the hardware. Figure 2(b) shows a possible runtime architecture for this stage of the problem. Since we no longer trust the operating system, we must implement the enforcer in a layer that is outside the control of the OS. We plan to explore virtualization as a means to achieve this goal.

This challenge problem will enable us to make progress on several of our research thrusts, such as specifying contracts, verifying monitors, and secure deployment. However, it has limited scope to explore monitor compatibility and composition, since it is restricted to a single agent. Our second challenge problem, described next, is designed to address this issue.

IV.B. Autonomous Intersection

There has been considerable recent interest in autonomous vehicles on roads and highways. One area that has received particular attention is the development of techniques to ensure that autonomous vehicles can cross road intersections safely (i.e., without collisions). A number of different approaches have been proposed, such as intersection protocols,[19]but none have been verified at the implementation level. We believe that runtime assurance can be used to address this challenge. The key idea is to develop an enforcer that steps in at the right moment to avoid a collision, while allowing the vehicle to operate normally otherwise. An important challenge here is the interaction between multiple enforcers executing on different vehicles that get into a potential collision situation. Once again, we may be able to assume good localization and communication among the vehicles. Nevertheless, verifying that the enforcers correctly collaborate to avoid collisions under all possible situations is an open problem, and directly relevant to our research thrusts.

V. Experimental Platform and Validation

In order to validate our results, we have set up an experimental testbed comprising of an indoor localization system, and a set of trackable minidrones. We also have preliminary results to indicate that the platform works as expected, which we describe in this section.

V.A. Indoor Localization

Since both our challenge problems require accurate localization information, we have set up an indoor localization system using the commercial Optitrack system[c]. Our current setup, shown in Figure 3(a), includes 6 motion capture cameras setup (roughly) in a circle with a diameter of 18 feet. This area is sufficient for our initial experiments, and can be increased as needed by adding more cameras.

Figure 3(b) shows a camera from up close. The cameras can track small spherical markers (that come with the system) which are coated with special reflective surface. The tracking information from each camera is sent via ethernet cables to a central computer (shown in Figure 3(c)). A software (also supplied by Optitrack) collects this tracking information and combines it to produce the precise location of the ball in a 3-dimensional coordinate system setup by the cameras. Thus, for each marker, we get a value of X, Y and Z. The software also allows three or more markers with fixed relative positions (e.g., if the markers are fixed to the surface of a solid object) to be identified and combined into a *rigid body*. For each such rigid body,

[c] `http://optitrack.com`

(a)

(b)

(c)

(d)

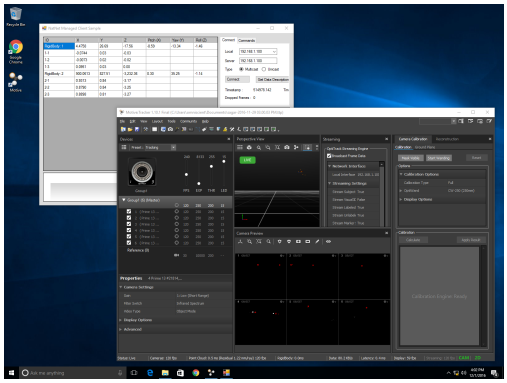

Figure 3. Optitrack Localization System.

the software generates both the location (X, Y and Z) of its center of mass, and it orientation in terms of *roll*, *pitch* and *yaw*.

The location of each marker, and the location of orientation of each rigid body is continuously multicast in real-time over the local network at a rate of 120Hz. The format of the packets used to represent this data is known, and there is open source (C++) client software that can receive the data over the network, and extract the location and orientation of each marker and rigid body by unpacking it. Figure 3(d) shows a screenshot of our central computer with two windows. The one on top is the Optitrack software that constructs the location and orientation information for each marker and rigid body, and broadcasts them. The one on the bottom is the client software receiving this data and unpacking it to extract and display the localization information.

V.B. Mini-Drones

We are also using commercially available minidrones manufactured by Parrot as mobile agents that can be used to test our localization infrastructure, as well as to deploy runtime assurance solutions for our first challenge problem. Our choice of minidrones was motivated by two factors:

1. They have an appropriate size, and power, for safe indoor experimentation, and in particular to be tracked via our localization system.

(a) (b)

Figure 4. Minidrones and experimental setup to validate localization.

2. They are relatively inexpensive that we can purchase several within our budget. In addition, they are robust against minor crashes, which are inadvertent in experimental situations.

3. There is publicly available open-source software which can be used to control them programmatically[d] using standard Bluetooth hardware, and a Linux-based laptop.

Figure 4(a) shows two minidrones with Optitrack markers attached. Figure 4(b) shows the laptop and gamepad controller we are using to control the minidrones as part of our preliminary experiments, which we describe next.

V.C. Platform Validation

Our preliminary experiments are aimed mainly at testing the efficacy of our testbed. We implemented a simple enforcer to create a virtual tether for a minidrone. The enforcer executes periodically on the laptop. During each execution, it receives user commands via the gamepad controller, and localization and orientation data from Optitrack. Using the localization information, the enforcer first determines whether the minidrone is within the safe area A defining the virtual tether. If this is the case, the enforcer passes the user command as is to the minidrone. Otherwise, it uses a very simple algorithm to determine a new command aimed at bringing the minidrone back to within A. In our initial experiments we observed that, when the enforcer is enabled, the minidrone is brought back to the safe area A as soon as it strays outside. When the enforcer is disabled, the human operator is easily able to navigate the minidrone to a point outside A and keep it there. We would like to emphasize again that these results only indicate that our tested is working as expected. While this is encouraging, we still need to address the core technical thrust areas presented earlier in the paper.

VI. Future Work and Conclusion

Distributed real-time (DRTS) embedded systems operating in uncertain and contested environments are of great relevance to the aerospace community. Such systems rely on components with unpredictable behavior to provide mission-critical capability. These components pose a challenge to assuring system-level safety and security. This paper presents an approach for developing a provably correct approach for Runtime Assurance of DRTS. We discuss the core technical thrust areas, present two challenge problems we are using to guide our research, and conclude with an experimental setup and initial results. We believe this work will lead to the advancement in practical development of high-assurance software for highly adaptive and autonomous aerospace systems.

References

[1]Pike, L., Wegmann, N., Niller, S., and Goodloe, A., "Copilot: monitoring embedded systems," *Innovations in Systems and Software Engineering (ISSE)*, Vol. 9, No. 4, December 2013, pp. 235–255.

[d]`https://github.com/voodootikigod/node-rolling-spider`

[2]Clark, M., Koutsoukos, X., Kumar, R., Lee, I., Pappas, G., Pike, L., Porter, J., and Sokolsky, O., "A Study on Run Time Assurance for Complex Cyber Physical Systems," Technical report, Air Force Research Laboratory, Wright-Patterson Air Force Base, OH, April 2013, http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA585474.

[3]Seto, D., Krogh, B., Sha, L., and Chutinan, A., "The simplex architecture for safe online control system upgrades," *Proceedings of the American Control Conference*, 1998.

[4]Kim, M., Viswanathan, M., Ben-Abdallah, H., Kannan, S., Lee, I., and Sokolsky, O., "Formally specified monitoring of temporal properties," *Proceedings of the 11th Euromicro Conference on Real-Time Systems (ECRTS '99)*, IEEE Computer Society, York, England, UK, June 1999, pp. 114–122.

[5]Havelund, K. and Rosu, G., "Monitoring Programs Using Rewriting," *Proceedings of the 16th International Conference on Automated Software Engineering (ASE '01)*, IEEE Computer Society, Coronado Island, San Diego, CA, USA, November 2001, pp. 135–143.

[6]Goodloe, A. and Pike, L., "Monitoring Distributed Real-Time Systems: A Survey and Future Directions," Technical report NASA/CR-2010-216724, NASA Langley Research Center, Langley, VA, USA, July 2010, https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20100027427.pdf.

[7]Jahanian, F., Rajkumar, R., and Raju, S. C. V., "Runtime Monitoring of Timing Constraints in Distributed Real-Time Systems," *Real-Time Systems (RTS)*, Vol. 7, No. 3, 1994, pp. 247–273.

[8]de Niz, D., Lakshmanan, K., and Rajkumar, R., "On the Scheduling of Mixed-Criticality Real-Time Task Sets," *Proceedings of the 30th Real-Time Systems Symposium (RTSS '09)*, IEEE Computer Society, Washington, DC, USA, December 2009, pp. 291–300.

[9]Schneider, F. B., "Enforceable security policies," *ACM Transactions on Information and System Security (TISSEC)*, Vol. 3, No. 1, February 2000, pp. 30–50.

[10]Ligatti, J., Bauer, L., and Walker, D., "Edit automata: enforcement mechanisms for run-time security policies," *International Journal of Information Security (IJIS)*, Vol. 4, No. 1-2, February 2005, pp. 2–16.

[11]Laurent, J., Goodloe, A., and Pike, L., "Assuring the Guardians," *Proceedings of the 15th International Conference on Runtime Verification (RV '15)*, edited by E. Bartocci and R. Majumdar, Vol. 9333 of *Lecture Notes in Computer Science*, Springer-Verlag, Vienna, Austria, September 2015, pp. 87–101.

[12]Alur, R. and Dill, D. L., "A Theory of Timed Automata," *Theoretical Computer Science (TCS)*, Vol. 126, No. 2, April 1994, pp. 183–235.

[13]Bak, S., Johnson, T. T., Caccamo, M., and Sha, L., "Real-Time Reachability for Verified Simplex Design," *Proceedings of the 35th Real-Time Systems Symposium (RTSS '14)*, IEEE Computer Society, Rome, Italy, December 2014, pp. 138–148.

[14]Shin, I. and Lee, I., "Periodic Resource Model for Compositional Real-Time Guarantees," *Proceedings of the 24th Real-Time Systems Symposium (RTSS '03)*, IEEE Computer Society, Cancun, Mexico, December 2003, pp. 2–13.

[15]Baruah, S. K., "A General Model for Recurring Real-Time Tasks," *Proceedings of the 19th Real-Time Systems Symposium (RTSS '98)*, IEEE Computer Society, Madrid, Spain, December 1998, pp. 114–122.

[16]ARINC653, "Aeronautical Radio, Inc. Avionics Application Software Standard Interface: ARINC Specification 653," June 2013.

[17]Pasareanu, C. S., Giannakopoulou, D., Bobaru, M. G., Cobleigh, J. M., and Barringer, H., "Learning to divide and conquer: applying the L* algorithm to automate assume-guarantee reasoning," *Formal Methods in System Design (FMSD)*, Vol. 32, No. 3, June 2008, pp. 175–205.

[18]Vasudevan, A., Chaki, S., Jia, L., McCune, J. M., Newsome, J., and Datta, A., "Design, Implementation and Verification of an eXtensible and Modular Hypervisor Framework," *Proceedings of the 34th IEEE Symposium on Security and Privacy (Oakland '13)*, IEEE Computer Society, San Francisco, CA, USA, May 2013, pp. 430–444.

[19]Azimi, S. R., Bhatia, G., Rajkumar, R., and Mudalige, P., "Reliable intersection protocols using vehicular networks," *Proceedings of the 4th International Conference on Cyber-Physical Systems (ICCPS '13)*, edited by C. Lu, P. R. Kumar, and R. Stoleru, Association for Computing Machinery, Philadelphia, PA, USA, April 2013, pp. 1–10.