Swarm Mentality: Toward Automatic Swarm State Awareness with Runtime Verification*

Brian Kempa and **Nick B. Cramer** and **Jeremy D. Frank**
Intelligent Systems Division
NASA Ames Research Center, Moffett Field, CA 94035
Brian.C.Kempa@nasa.gov

Abstract

Cyber-Physical Systems (CPSs) already exhibit impressive performance in all areas of human life, and swarms of CPSs promise to increase their capabilities even further. However, to effectively utilize CPS swarms their complexity of operation has to scale sub-linearly with the number of swarm members. Presenting the swarm to an operator as a single entity almost eliminates the additional per-member overhead entirely. To operate a swarm as one entity, and/or to increase the swarm's autonomy, the operator and the swarm members need to reason and communicate at the same level of abstraction, i.e. the swarm needs a sense of "self." Therefore, we require the ability to specify whole swarm properties yet monitor them at the member level.

We examine one architecture for achieving this awareness by: 1) Defining a taxonomy for comparing techniques that synthesize this belief-state 2) Propose use of the Runtime Verification formal method to fill this role 3) Present preliminary designs for extending and embedding such a system in the Distributed Spacecraft Autonomy architecture to generate per-member monitors from swarm level specification

1 Introduction

Cyber-Physical Systems (CPSs) permeate our lives in ways both obvious and invisible. Systems we use daily like aircraft, satellites, and biomedical devices are clear examples of complex CPSs but continual decline in the cost of embedded systems, coupled with the ease of internet connectivity, has driven the transformation of basic consumer goods into CPSs even as the next wave of CPS capabilities in autonomous cars, drone swarms, and robotics enter market. While there is no single agreed upon definition, some common features of CPSs are: 1. consist of communicating controllers passing observations over a network (201 2014) 2. nearly inseparable physical (continuous) and logical (discrete) dynamics i. e., hybrid dynamics (Platzer 2018), and 3. tight coupling preventing effective use of soiled subsystem assumptions used in traditional engineering practice (He, Dong, and Fu 2018).

Hierarchical layers are a common abstraction to compartmentalize the complexity in these tightly coupled modules (Badger, Strawser, and Claunch 2019). For example: a robotic actuator might contain a suite of sensors and controllers, exposing only it's best position estimate to a higher level motion planner which commands desired attitudes in turn. This abstraction pattern generates a hierarchy which generally seeks to minimizes the assume-guarantee contracts between components. Sitting atop these stratified abstractions, decision making for a CPS can often remain a "simple" *sense-plan-act* (SPA) loop. This recalls the subsumption architecture of robotics and hierarchical control of industrial controls systems (Choi et al. 2013).

However, the contracts between the planning layer and rest of the system may not be sufficient to keep belief-sate between the sense and plan stages within the planner's tractable regime. This is particularly common when scaling CPSs to swarm deployments where all the uncertainty issues endemic to distributed systems pollute the belief-state. To keep planning tractable, systems may resort to application-specific, ad-hoc components sitting between the sense and plan steps to compensate for the additional uncertainty. While the communications and planning systems are well defined fields of study, techniques used to bridge them come from many disciplines impeding direct comparison.

To design, operate, and trust CPSs that operate with significant high-level uncertainty, we must analyze the methods of reducing the size of the belief-state that is fed to the planner (a process we call orientation) and structure their development. When the resulting belief-state represents a whole swarm, it forms a situational awareness, common operational picture, or sense-of-self for the swarm to plan against. Our goal is to automatically and formally generate this state, in a manner lightweight enough to run in critical embedded domains like spacecraft swarms.

Contributions and Organization First, we define a taxonomy for comparing orientation techniques in Section 2. In Section 3, use of runtime verification as an orientation technique is proposed. Then, preliminary work on extending a runtime verification tool for orientaion of a swarm is overviewed in Section 4 and applied to the Distributed Space Autonomy mission in Section 5. Finally, Section 6 concludes with a discussion of open questions and future work.

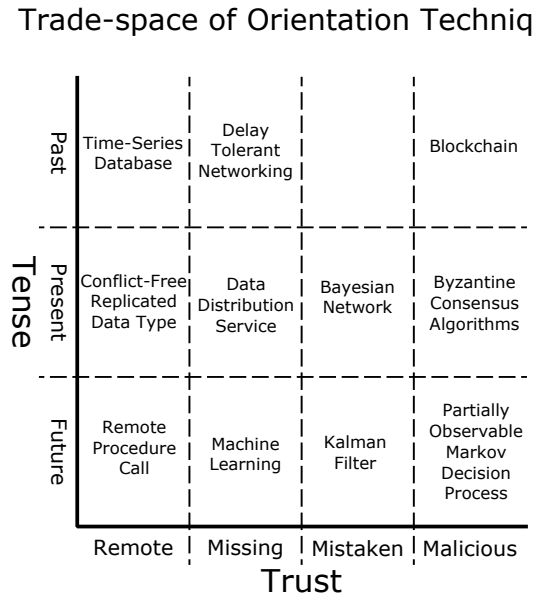


Figure 1: A taxonomy for comparing Orientation techniques. The horizontal axis indicates trust received observations from full trust on the left to Byzantine fault assumptions on the right. The vertical axis indicates the tense of the recovered state, from complete historical trace recovery at the top, though best-effort/last-known methods in the center to predictive methods at the bottom. Techniques are not limited to the sample shown and may cross boundaries.

2 Orienting to Orientation

OODA Decision Model

The *Observe-Orient-Decide-Act* (OODA) decision model explicitly names the section between the collection of state information and planning in recognition of need to synthesize a belief-state upon which to base decision making (Grant 2005). The "orientation" step (called "contextualize" or "analyze" in similar models) resolves uncertainty and reduces the dimensionality of the system state fed though to the planner by leveraging domain or mission specific assumptions. Two reviews of requirements for space system autonomy, noted an improvement in "awareness" of the system (i. e., orientation) as a key requirement and that monitoring is "a prerequisite for awareness, it constitutes a subset of awareness" (Jonsson, Morris, and Pedersen 2007; Vassev and Hinchey 2013).

Compared to the communications and planning components, the orientation module lacks a clean boundary and tends to bleed into the other domains. This flexibility in definition makes many techniques from varied fields of study viable approaches. The required capabilities can be derived by quantifying the gap between the guarantees of the communications system and and assumptions of the planner.

The Trust and Tense taxonomy

Direct comparison of the orientation techniques is complicated by the very flexibility that gives them power. Figure 1 organizes orientation techniques by their assumptions of observation validity (trust) and the time span of the recovered states (tense).

Trust The horizontal trust axis parameterizes the assumed validity of received information, encoding the strength of the communication system's guarantees. The most trusting approaches assume information is merely remote and can be eventually received with some latency. As communication guarantees weaken, the orientation and planning systems must contend with incomplete, inaccurate, or even maliciously crafted messages (i. e., Byzantine faults).

Tense The vertical tense axis describes the length of the reconstructed data trace, encoding the amount of historical information required by the planner. At one extreme, the system will eventually resolve the full historical trace, while at the other it predicts a future state without memory of the previous states. In the middle are last-known/best-known/state-estimation beliefs about the remote parameters.

Discussion

In general, most past tense techniques come from databases, consensus protocols dominate the present tense, and the future tense is the domain of state estimation techniques.

Lower trust approaches raise the separate but related question of how much uncertainty to expose to the planner. If the goal of orientation is to present the simplest belief-state to the planner, then approaches that wait for certainty (like a blockchain) or keep uncertainty internally and report their best known value (like a Kalman filter) can be used. When using uncertainty aware planning, reporting a confidence level with the suspected value (like with Bayesian inference) or the whole set of potential states can be used.

Tense has a relationship to the CAP theorem from distributed systems theory. Brewer's *Consistency, Availability, Partition-tolerance* theorem describes that a distributed system can only guarantee two out of the three eponymous properties (Fox and Brewer 1999). Since partition tolerance is a requirement for swarms assuming imperfect communication, these methods can be viewed as a trade-off between availability and completeness. Overall, the future methods are more available since they can provide a projected value for anytime (accuracy not withstanding), while past tense methods favor consistency at the cost of speed.

The amount of development effort and prior knowledge generally increases from the remote-past toward malicious-future. From an information theory perspective, it is reasonable that methods actively compensating for unknown information require more structure like system models, parameter tuning or learning periods.

3 Runtime Verification Orientation

Formal methods have shown promising results in CPS even when traditional engineering trust methods break down (Hartnett 2016; University 2011) and are gaining acceptance by standards for establishing trust (Jacklin 2012).

In response to the inflexibility (and often intractability) of traditional formal methods like model checking and automated theorem proving, *lightweight formal methods* were

developed to trade away guarantees of existing formal techniques to gain advantages in specific use cases (Leucker and Schallhart 2009). *runtime verification* (RV) is a lightweight formal method that trades off complete reasoning over all potential system executions for reasoning over a singular system trace(Havelund and Rosu 2001). Most work in this field is applied to systems postmortem (offline RV) (Falcone et al. 2018), however some RV monitors run in real-time (online RV) with the required time and space known a priori, making them well suited as an orientation technique for CPSs (Desai, Dreossi, and Seshia 2017).

RV systems can digest large streams of data into Boolean verdicts in real-time and this transform is described by a formal language. We can use this efficiently reduce the difficulty of planning with formal trade-offs and validation of the "lost" information. This can be especially advantageous for temporal properties which can be converted into Boolean compliance verdicts by an RV monitor, removing the temporal dependency from the planner altogether.

We propose leveraging RV as an orientation technique to simplify the state space prior to planning by deriving the specification from the project requirements used to develop the planner. Since off-the-shelf RV monitors can be used, this reduces mission-specific code, simplifying validation and verification of the decision loop.

Prior Work

Adoption of RV has remained slow, despite promising results in both academia and industry (Rozier 2019; Desai, Dreossi, and Seshia 2017; Srivastava and Schumann 2013; Torens, Dauer, and Adolf 2018). Existing work has demonstrated runtime monitoring protocol conformance in a CPS (Fadil and Koning 2005) and use of RV in a distributed system with a hierarchy (El-Hokayem and Falcone 2021). Several teams (Dixon et al. 2012; Konur, Dixon, and Fisher 2012; Winfield et al. 2005b; Rouff et al. 2004; Winfield et al. 2005a) have focused on specification of emergent behavior, verifying conformance at design time.

Proposed Workflow and Project Process

Creating an RV monitor that tracks status requires encoding the definition of "good" (or bad) in a formal specification.

Even without a formal verification campaign, traditionally engineered CPSs have a source of (semi-)formal specification – the project requirements. This reduces the specification design from system modeling to translation, lowering the previous experience required and easing the "specification bottleneck" (Rozier 2016).

Since the project requirements encode the same conditions used to develop the control logic, they naturally describe properties of interest to the planner. By having requirements guide specification, RV monitors can produce minimal system states of values needed by the planner.

Writing specification can be difficult, and as a corollary to Conway's law (MacCormack, Baldwin, and Rusnak 2012) it is only made more difficult when the process is at odds with the organizational flow of the project. However, deriving specification directly from requirements both matches the existing project structure of traditional engineering efforts, and allows the specifications to be developed ahead of the implementation. The requirements-first flow then enables low fidelity testing to be completed and specifications iterated early in the design cycle.

The resulting verdicts are relatively easy to debug (and can assist with debugging the rest of the system), since the intermediate value have a semantic mapping to the requirements - e. g., system decided to go into safe-mode, looking at the health and safety verdicts the maximum discharge rate property was not upheld.

Discussion

RV orientation would be categorized as a remote-present technique in the trust and tense taxonomy of Figure 1.

Strengths 1. Utilizes existing formalism in project, reducing specification effort 2. Naturally expresses assume-guarantee contracts and temporal properties 3. Enables planning with temporal properties without maintaining historical state 4. Produces concise sates that trace to requirements, aiding explainability 5. Model-free method allowing early development and testing 6. Lighter-weight than pure state estimation, with formalized trade-offs

Weaknesses 1. Writing and validating formal specifications is a specialized skill 2. Not robust to missing information 3. Being model-free can beget opaque system assumptions if requirement changes are not closely tracked

Overall, this approach is a strong candidate when most uncertainty can be resolved with a locally greedy approach - i. e., assuming known information is true.

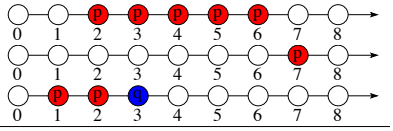
4 RV Orientation from Swarm Specification

One of the identified goals of NASA's *Distributed Spacecraft Autonomy* (DSA) project is "Advance command and control methodologies for controlling a swarm of spacecraft as a single entity" (Cellucci, Cramer, and Frank 2020). Presenting the swarm to an operator as a single entity almost eliminates the additional per-member overhead entirely. To support this, the operator must be able to reason about the state of the full swarm at once and each member must make autonomous decisions based on the state of the whole group.

Just as with the specifications in the simple RV orientation module, swarm member decisions are made out of mission rules sets collected in project requirements. Again, by using these as templates for specifications we leverage existing formalism, ensure traceability, and bolster explainability.

To reason about the swarm as a whole, each agent necessarily requires information about other agents. While this shift in perspective matters to the monitor, we do not want implementation details impacting the specifications. We propose a process of automatically decomposing specifications describing the whole swarm into distributed monitors. To aid in this conversion, we also extend the specification language to natively express two common patterns in swarm specification: N-out-of-M properties and indirect evidence.

Table 1: Mission-time Linear Temporal Logic[2]

Symbol	Operator	Timeline
$\Box_{[2,6]}p$	ALWAYS$_{[2,6]}$	
$\Diamond_{[0,7]}p$	EVENTUALLY$_{[0,7]}$	
$p\mathcal{U}_{[1,5]}q$	UNTIL$_{[1,5]}$	

Use of RV as an orientation technique alone would be a remote-present technique in Figure 1, but by adding indirect evidence we can move this toward the region dominated by state estimation methods, without the heavy overhead or system modeling required by those techniques.

Prior Work Some great languages exist for specification of distributed systems *Vienna Development Method* (VDM) (Bjorner and others 1978) and *Distributed Ontology, Model, and Specification Language*[1] (DOL) however they both are tightly coupled to a system model. In (Briola, Mascardi, and Ancona 2015), distributed runtime verification is implemented over the JADE multiagent system platform. Specifications were derived from protocols and represented as *attribute global types* (AGTs). AGTs are great for protocol spec but difficult to read and write. Interesting thoughts on the duality of distributed monitoring and distributed computing are raised in (Audrito et al. 2019). The three-value logic was used to attack the inherently asynchronous nature of distributed RV in (Scheffel and Schmitz 2014). Lessons learned from an agent designed for multi-satellite mission with a similar message passing architecture are found in (Schetter, Campbell, and Surka 2003) Case studies on the required level of feedback and control for a human in the loop with a swarm are prevalent in defense literature such as (Beal 2012; Sauter and Bixler 2019) Similar issues are studied in other fields, and in examining the cross between Health Management and Internet-of-Things, (Karamitsios, Orphanoudakis, and Dagiuklas 2016) ended up with some similar designs and lessons learned even though the application was focused on network stack performance.

Goals 1. Derive temporal logic swarm-level specifications from requirements, not implementations 2. Automatically generate per-member monitors from swam specification 3. Create RV orientation module that does not require mission specific code i. e., uses native cFS features and is fully configurable from specification 4. Raise the level of abstraction in the constructed belief-state to match the operator's view of the swarm as an entity We are developing the *Swarm Sense of Self* (S3) application to address these goals.

Specification Language

Temporal Logic is a common specification language for RV monitors (Falcone et al. 2018) Temporal Logic specifications describe desired (or undesired) timelines of executions, much like mission specifications, reducing the semantic gap between the source requirements and the monitored specification. This simplifies specification writing and reduces the required RV knowledge to validate the specifications against project requirements. Previous distributed space systems verification repeatedly identified temporal properties as an important expressive shibboleth (Das, Wu, and Truszkowski 2001; Winfield et al. 2005a; Araguz, Bou-Balust, and Alarcón 2018; Badger, Strawser, and Claunch 2019).

For our implementation we have selected *Mission-time Linear Temporal Logic* (MLTL), a Metric Temporal Logic fragment for finite discrete traces introduced in (Reinbacher, Rozier, and Schumann 2014). MLTL contains the standard Boolean propositional operations (And, Or, Not, and Implies) along with start and end bounds on the future-time temporal operators as shown in Table 1.

Some projects prefer past-time temporal logic (Havelund and Roşu 2002; Roşu, Chen, and Ball 2008), presumably because all needed information already exists at the time of reasoning. Our monitor supports both; however, we've elected to use future-time because: 1. future time matches the tense our operations requirements are expressed in making them easier to write, and 2. future time make missing information very explicit. In an imperfect communications environment just because the event has passed doesn't mean you have knowledge of it.

We extend MLTL to be written at the swarm level and automatically decompose graph to per-agent monitors. To assist in writing meaningful swarm-level specifications we have identified two initial language features: 1. N-out-of-M Properties, and 2. Indirect Evidence

N-out-of-M properties are Boolean signals like in standard R2U2, but instead of looking for one value to meet a condition it waits for a threshold of swarm members' local values to meet that condition - e. g., "battery charge above 80% for at least 75% of the swarm."

Indirect evidence allows swarm members to "out-vote" a local value from their outside perspective. For example, if one swarm member reports they are actively illuminating a target, but a quorum of other members do not agree the self-reported result can be considered faulty.

Monitor Core

Our implementation is built atop the *Realizable Responsive Unobtrusive Unit*[3] (R2U2) runtime monitor (Reinbacher, Rozier, and Schumann 2014) which has specifically engineered to conform to flight software requirements and has flight heritage on the SwiftUAS drone platform (Schumann et al. 2015), previous integration with NASA's Robonaut2 platform (Kempa et al. 2020) and is currently being considered for use by the Lunar Gateway space station's Vehicle Management System (Dabney, Badger, and Rajagopal 2021). R2U2's architecture is flight software friendly as it does not allocate memory, has a separate atomic checker module for evaluating Boolean proposition from system signals, and can be reconfigured to monitor different specifications without recompiling the binary.

[1] `https://www.omg.org/dol/`

[2] (Reinbacher, Rozier, and Schumann 2014) with permission

[3] `http://r2u2.temporallogic.org`

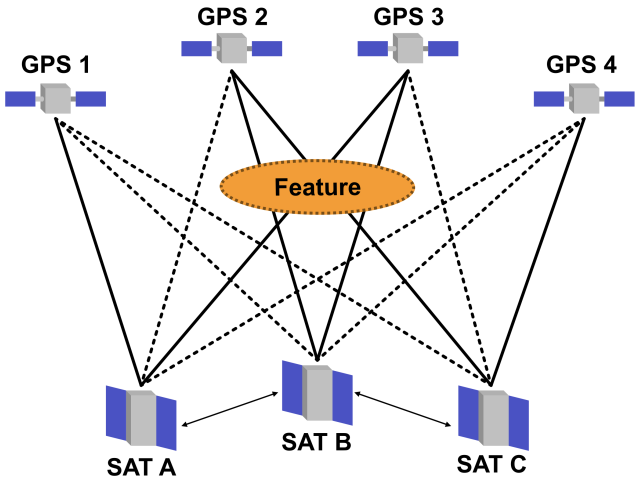


Figure 2: Simplified DSA mission architecture. Agents coordinate via radio link with their neighbors and monitor (solid line) a subset of visible (dotted line) GPS signals.

We hope to use R2U2's Bayesian reasoning capabilities (Reinbacher, Rozier, and Schumann 2014) to implement indirect evidence, and have investigated using formula rewriting for N-out-of-M properties, but both are currently implemented as input filters in R2U2's atomic checker module - an efficient if less interesting method.

Swarm Specifications & Member Monitors

To be viable, our toolchain must be capable of automatically generating per-member monitors from the swarm level specification. R2U2's use of an observer tree to represent the formula evaluation drastically simplifies development - we only need to make two modifications to the R2U2 toolchain. First, an extended ANTLR[4] parser generator grammar file is provided with the additional syntax to construct an *Abstract Syntax Tree* (AST) of the specification observer at the swarm level, just like in a standard R2U2 formula compiler (Kempa et al. 2020). Then, before the R2U2 configuration is recursively generated from the tree, the member monitor visitor walks the tree and tweaks the necessary parameters for it's host monitor. If there are n swarm members, the same tree is walked and potentially modified n times, emitting a configuration at the end of each walk. This process is demonstrated in Section 5. Since our implementation's only language extensions are both implemented in the atomic checker, the visitor only needs to visit the leaves of the AST and mark values as local or remote.

5 RV Swarm Orientation in Practice

In flight, the four DSA mission spacecraft form a "string of pearls" topology with radio cross-links between adjacent members. Agents share state information over this link to select which GPS signals to monitor. Figure 2 is a simplified depiction of this configuration; in flight there will be 4 DSA spacecraft, each with up to seven visible GPS signals, and the visibility graph will not be fully connected. The goal is to maximize the number of monitored signals that pass through a ionospheric feature of interest, while maintaining coverage - monitored channels with each visible GPS transmitter to detect other potential features. We next demonstrate development of an S3 monitor for coverage, evaluate the behavior on simulated mission data, and examine results from the perspective of the swarm agents and operators.

[4] https://www.antlr.org

Example Agent Monitor from Swarm Property

Swarm Property and Specification The project requirement we are encoding represents a behavior of the whole swarm beyond the local knowledge of a singular agent and can be written as: *All visible GPS signals should be monitored by at least one agent* Full visibility is assumed at the swarm level, therefore we can test set membership without regard to data locality and express "GPS number x is visible" as $\mathrm{G}x \in \mathrm{GPS}_{\mathrm{vis}}$ and similarly with assignment. While this is a literal encoding of the property, we can partially mitigate implicit assumptions gaps between the spirit and letter of the law in this specification. For example, we leverage the temporal logic operator F[0,5] as a "debounce" to give some leeway for network transmission time. The specification below (with the expansion to all GPSs elided for space) encodes coverage and is read as "within the next 5 ticks, GPS visibility implies GPS assignment."

$$F[0,5]\,(\mathrm{G0} \in \mathrm{GPS}_{\mathrm{vis}} \rightarrow \mathrm{G0} \in \mathrm{GPS}_{\mathrm{assign}}) \wedge \ldots)$$

Agent Specification To construct an agent monitor, the swarm specification is converted to an AST and traversed in post-order, applying rewriting rules and marking remote data. In this example, set membership checks are rewritten as disjunctions of atomic propositions in the form $\mathrm{GPS}_{x\_\mathrm{vis/asn}\_n}$ (read as: "GPS x visible/assigned by agent n") and atomics where n is not the agent number are remote.

$$\begin{aligned} F_{[0,5]}[&(\mathrm{GPS}_{1\_\mathrm{vis}\_0} \vee \mathrm{GPS}_{1\_\mathrm{vis}\_1} \vee \mathrm{GPS}_{1\_\mathrm{vis}\_2} \vee \mathrm{GPS}_{1\_\mathrm{vis}\_3}) \\ &\rightarrow(\mathrm{GPS}_{1\_\mathrm{asn}\_0} \vee \mathrm{GPS}_{1\_\mathrm{asn}\_1} \vee \mathrm{GPS}_{1\_\mathrm{asn}\_2} \vee \mathrm{GPS}_{1\_\mathrm{asn}\_3})]\wedge \\ &\vdots \quad \text{(subexpressions for GPS } 2-31 \text{ ellided)} \\ &[(\mathrm{GPS}_{32\_\mathrm{vis}\_0} \vee \mathrm{GPS}_{32\_\mathrm{vis}\_1} \vee \mathrm{GPS}_{32\_\mathrm{vis}\_2} \vee \mathrm{GPS}_{32\_\mathrm{vis}\_3}) \\ &\rightarrow(\mathrm{GPS}_{32\_\mathrm{asn}\_0} \vee \mathrm{GPS}_{32\_\mathrm{asn}\_1} \vee \mathrm{GPS}_{32\_\mathrm{asn}\_2} \vee \mathrm{GPS}_{32\_\mathrm{asn}\_3})] \end{aligned}$$

Agent Monitor The remote atomics are retrieved from the cross-link communication system and use the last known value by default. Integration of indirect evidence is not necessary for this specification. Use of time-to-live in the messaging layer can ensure stale data is not reused in the even of a partition, though the relative motion of the swarm will quickly lead to violations when using old data without this extra feature.

The derived agent expression consists of four atomic loads and three disjunctions for each set existence checks, there are two such checks and an additional three temporal operations are used by the equivalence relation R2U2 uses

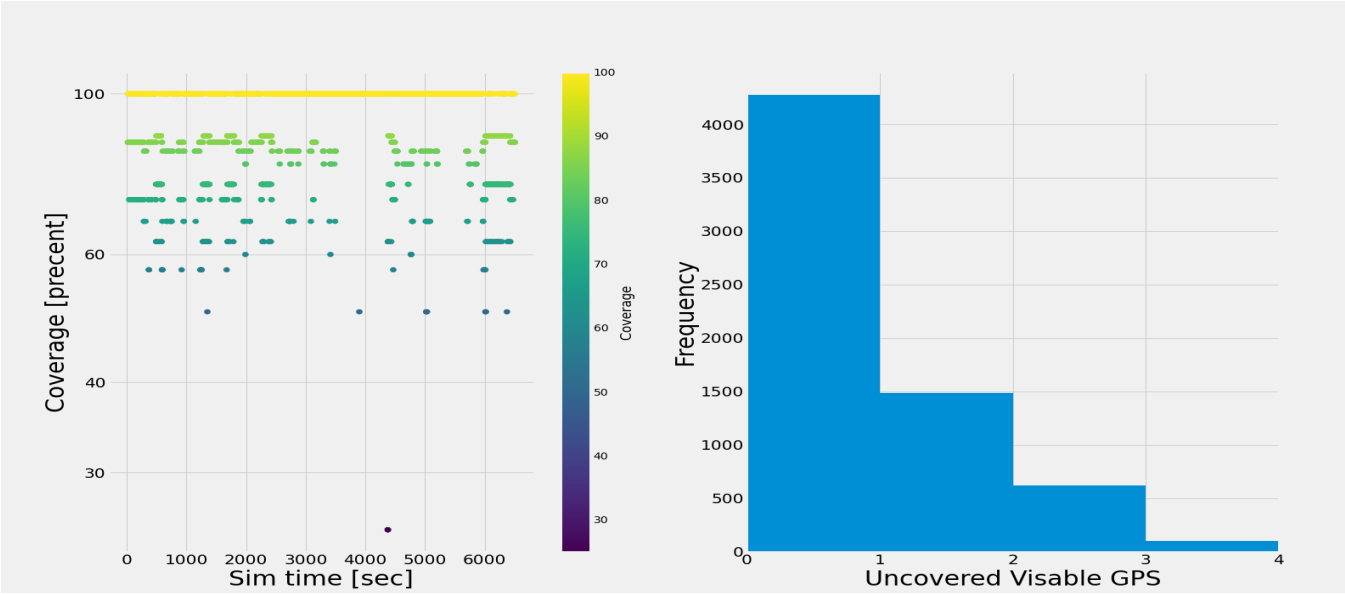


Figure 3: Coverage percentage over time (left) and frequency of missed signals (right) without strict coverage enforcement.

for implies for each of the 32 channels. Finally, these 32 per-GPS checks are conjuncted by 31 more operations and the temporal debounce is applied for a total of 576 R2U2 temporal logic operations - sufficiently small for evaluation under a real-time deadline (Kempa et al. 2020) and R2U2 optimizations like native set operations (Cauwels et al. 2020) further improve the speed in practice.

Scenario Results

Figure 3 summarizes the result of the coverage check on a simulated DSA activity using real GPS data from the ESA Swarm mission (Cellucci, Cramer, and Frank 2020). The agent was set to balance coverage with maximal observation of potential features rather than strictly enforce the coverage condition. The swarm averaged 93.31% coverage and met coverage specification 81.99% of the simulation.

Agent Perspective Rigorous detection of a specification violation is the first step toward fault disambiguation and automatic mitigation. The specific failing subexpression coupled with other system variables constitute a signature that can be matched with a specific fault or mitigation trigger. These rules can be expressed as specifications themselves and encoded in the S3 monitor.

Operator Perspective For operations staff evaluating swarm performance, specification violations automatically highlight regions of interest in the telemetry and decompose into the natural follow-on questions. The temporal debounce filters out acceptable transients, like the short coverage loss due to a new satellite entering view after assignments completed at time 3901, reducing alarm fatigue and better directing operator attention. A violation may be acceptable and caused by an interesting feature, in this example which has now been timestamped by the S3 monitor. Otherwise it may be the results of missing or inaccurate data such as during the initial start-up or after a network partition.

6 Conclusion

As CPSs increase in complexity, their hierarchical abstractions can no longer shield higher level of decision making from intractable uncertainty. While the value of formal methods in generating controllers or evaluating behavior of these systems is acknowledged, we believe there is value in applying these approaches at runtime as well.

Using a model that maintains clean separability between sensors, beliefs, and logic, we consider many disparate techniques for constructing belief-states. We proposed using runtime verification to construct a lightweight, real-time orientation module for synthesizing belief-states. Our work explores methods to lower the barrier to entry with formal methods, such as utilizing existing formalism in the project. As a bonus, this semantic connection improves the explainability of the belief-sate. We previewed extensions to shift RV techniques toward handling more uncertainty without sacrificing model-free specification, shifting RV toward a trust level of missing or mistaken data. Finally, preliminary work on matching abstraction levels between swarm operators and members to simplify operations and development for the humans while providing the swarm with the necessary "sense-of-self" to act as one.

Future Work

With the initial version of the S3 component in place, we want to begin deciding and acting on this new belief-state with investigations on effective ways to offload complexity from the planning module. This is the original goal of the project - to see how RV could best assist planning and scheduling tools.

A future optimization would be to identify branches of the AST that can be shared over the network instead of only treating the leaves as potential remotes. This has interesting implications for speed, loss, and robustness. In some use-cases it might be better to repeatedly resend a single composite value over a poor connection than trying to get a full set of the component values though, and vice verse. The visitor pattern is generic enough at accommodate these optimization targets.

To continue the development of the S3 prototype, we will attempt to encode specifications from real missions and published case studies to check that the language is expressive enough and the converter handles real-world use cases. Additional features like confidence thresholds and indirection will be evaluated for usefulness and complexity before being considered for inclusion.

Work on a specification language extension to allow addressing values in different traces is awaiting peer review. This should make writing from the swarm and member perspectives seamless. Building off that capability, we hope to allow alternative methods of incorporating indirect evidence as well as support reasoning over traces with missing data like in (Taleb, Khoury, and Hallé 2021).

References

[201 2014] 2014. *Applied cyber-physical systems Sang C. Suh ...[et. al.], editors; foreword by B. Thurasingham.* New York: Springer Science.

[Araguz, Bou-Balust, and Alarcón 2018] Araguz, C.; Bou-Balust, E.; and Alarcón, E. 2018. Applying autonomy to distributed satellite systems: Trends, challenges, and future prospects. *Systems Engineering* 21(5):401–416.

[Audrito et al. 2019] Audrito, G.; Damiani, F.; Stolz, V.; and Viroli, M. 2019. On distributed runtime verification by aggregate computing. volume 302, 47–61.

[Badger, Strawser, and Claunch 2019] Badger, J. M.; Strawser, P.; and Claunch, C. 2019. A distributed hierarchical framework for autonomous spacecraft control. In *2019 IEEE Aerospace Conference*, 1–8. IEEE.

[Beal 2012] Beal, J. 2012. A tactical command approach to human control of vehicle swarms. In *2012 AAAI Fall Symposium Series*.

[Bjorner and others 1978] Bjorner, D., et al. 1978. The vienna development method: The meta-language.

[Briola, Mascardi, and Ancona 2015] Briola, D.; Mascardi, V.; and Ancona, D. 2015. Distributed runtime verification of jade multiagent systems. In *Intelligent Distributed Computing VIII*. Springer. 81–91.

[Cauwels et al. 2020] Cauwels, M.; Hammer, A.; Hertz, B.; Jones, P. H.; and Rozier, K. Y. 2020. Integrating runtime verification into an automated uas traffic management system. In *European Conference on Software Architecture*, 340–357. Springer.

[Cellucci, Cramer, and Frank 2020] Cellucci, D.; Cramer, N. B.; and Frank, J. D. 2020. Distributed spacecraft autonomy. In *ASCEND 2020*. 4232.

[Choi et al. 2013] Choi, J.-S.; McCarthy, T.; Yadav, M.; Kim, M.; Talcott, C.; and Gressier-Soudan, E. 2013. Application patterns for cyber-physical systems. In *2013 IEEE 1st International Conference on Cyber-Physical Systems, Networks, and Applications (CPSNA)*, 52–59. IEEE.

[Dabney, Badger, and Rajagopal 2021] Dabney, J. B.; Badger, J. M.; and Rajagopal, P. 2021. Adding a verification view for an autonomous real-time system architecture. In *AIAA Scitech 2021 Forum*, 0566.

[Das, Wu, and Truszkowski 2001] Das, S.; Wu, C.; and Truszkowski, W. 2001. Distributed intelligent planning and scheduling for enhanced spacecraft autonomy. In *Proceedings of the 2001 AAAI Spring Symposium Series, Palo Alto, CA*.

[Desai, Dreossi, and Seshia 2017] Desai, A.; Dreossi, T.; and Seshia, S. A. 2017. Combining Model Checking and Runtime Verification for Safe Robotics. In Lahiri, S., and Reger, G., eds., *Runtime Verification*, Lecture Notes in Computer Science, 172–189. Springer International Publishing.

[Dixon et al. 2012] Dixon, C.; Winfield, A. F.; Fisher, M.; and Zeng, C. 2012. Towards temporal verification of swarm robotic systems. *Robotics and Autonomous Systems* 60(11):1429–1441.

[El-Hokayem and Falcone 2021] El-Hokayem, A., and Falcone, Y. 2021. Bringing runtime verification home: a case study on the hierarchical monitoring of smart homes using decentralized specifications. *International Journal on Software Tools for Technology Transfer* 1–23.

[Fadil and Koning 2005] Fadil, H., and Koning, J.-L. 2005. A formal approach to model multiagent interactions using the b formal method. In Ramos, F. F.; Larios Rosillo, V.; and Unger, H., eds., *Advanced Distributed Systems*, 516–528. Berlin, Heidelberg: Springer Berlin Heidelberg.

[Falcone et al. 2018] Falcone, Y.; Krstić, S.; Reger, G.; and Traytel, D. 2018. A Taxonomy for Classifying Runtime Verification Tools. In Colombo, C., and Leucker, M., eds., *Runtime Verification*, Lecture Notes in Computer Science, 241–262. Springer International Publishing.

[Fox and Brewer 1999] Fox, A., and Brewer, E. A. 1999. Harvest, yield, and scalable tolerant systems. In *Proceedings of the Seventh Workshop on Hot Topics in Operating Systems*, 174–178. IEEE.

[Grant 2005] Grant, T. 2005. Unifying planning and control using an ooda-based architecture. In *Proceedings of Annual Conference of the South African Institute of Computer Scientists and Information Technologists*, volume 20, 111–122.

[Hartnett 2016] Hartnett, K. 2016. Computer scientists close in on perfect, hack-proof code.

[Havelund and Rosu 2001] Havelund, K., and Rosu, G. 2001. Preface: Volume 55, issue 2. *Electronic Notes in Theoretical Computer Science* 55(2):287 – 288. RV'2001, Runtime Verification (in connection with CAV '01).

[Havelund and Roşu 2002] Havelund, K., and Roşu, G. 2002. Synthesizing monitors for safety properties. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 342–356. Springer.

[He, Dong, and Fu 2018] He, X.; Dong, Z.; and Fu, Y. 2018. A Systematic Approach for Developing Cyber Physical Systems. 456–505.

[Jacklin 2012] Jacklin, S. 2012. Certification of safety-critical software under do-178c and do-278a. In *Infotech@ Aerospace 2012*. 2473.

[Jonsson, Morris, and Pedersen 2007] Jonsson, A.; Morris, R. A.; and Pedersen, L. 2007. Autonomy in space: Current capabilities and future challenge. *AI Magazine* 28(4):27.

[Karamitsios, Orphanoudakis, and Dagiuklas 2016] Karamitsios, K.; Orphanoudakis, T.; and Dagiuklas, T. 2016. Evaluation of iot-based distributed health management systems. In *Proceedings of the 20th Pan-Hellenic Conference on Informatics*, PCI '16. New York, NY, USA: Association for Computing Machinery.

[Kempa et al. 2020] Kempa, B.; Zhang, P.; Jones, P. H.; Zambreno, J.; and Rozier, K. Y. 2020. Embedding online runtime verification for fault disambiguation on robonaut2. In *International Conference on Formal Modeling and Analysis of Timed Systems*, 196–214. Springer.

[Konur, Dixon, and Fisher 2012] Konur, S.; Dixon, C.; and Fisher, M. 2012. Analysing robot swarm behaviour via probabilistic model checking. *Robotics and Autonomous Systems* 60(2):199–213.

[Leucker and Schallhart 2009] Leucker, M., and Schallhart, C. 2009. A brief account of runtime verification. *The Journal of Logic and Algebraic Programming* 78(5):293–303.

[MacCormack, Baldwin, and Rusnak 2012] MacCormack, A.; Baldwin, C.; and Rusnak, J. 2012. Exploring the duality between product and organizational architectures: A test of the "mirroring" hypothesis. *Research Policy* 41(8):1309–1324.

[Platzer 2018] Platzer, A. 2018. *Logical foundations of cyber-physical systems*. Cham, Switzerland: Springer.

[Reinbacher, Rozier, and Schumann 2014] Reinbacher, T.; Rozier, K. Y.; and Schumann, J. 2014. Temporal-logic based runtime observer pairs for system health management of real-time systems. In *Proceedings of the 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 8413 of *Lecture Notes in Computer Science (LNCS)*, 357–372. Springer-Verlag.

[Roşu, Chen, and Ball 2008] Roşu, G.; Chen, F.; and Ball, T. 2008. Synthesizing monitors for safety properties: This time with calls and returns. In *International Workshop on Runtime Verification*, 51–68. Springer.

[Rouff et al. 2004] Rouff, C.; Vanderbilt, A.; Hinchey, M.; Truszkowski, W.; and Rash, J. 2004. Verification of emergent behaviors in swarm-based systems. In *Proceedings. 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems, 2004.*, 443–448. IEEE.

[Rozier 2016] Rozier, K. Y. 2016. Specification: The biggest bottleneck in formal methods and autonomy. In *Proceedings of 8th Working Conference on Verified Software: Theories, Tools, and Experiments (VSTTE 2016)*, volume 9971 of *LNCS*, 1–19. Toronto, ON, Canada: Springer-Verlag.

[Rozier 2019] Rozier, K. Y. 2019. From simulation to runtime verification and back: Connecting single-run verification techniques. In *Proceedings of the Spring Simulation Conference (SpringSim)*, TBD. Tucson, AZ, USA: Society for Modeling & Simulation International.

[Sauter and Bixler 2019] Sauter, J. A., and Bixler, K. 2019. Design of unmanned swarm tactics for an urban mission. In *Unmanned Systems Technology XXI*, volume 11021, 110210K. International Society for Optics and Photonics.

[Scheffel and Schmitz 2014] Scheffel, T., and Schmitz, M. 2014. Three-valued asynchronous distributed runtime verification. In *2014 Twelfth ACM/IEEE Conference on Formal Methods and Models for Codesign (MEMOCODE)*, 52–61.

[Schetter, Campbell, and Surka 2003] Schetter, T.; Campbell, M.; and Surka, D. 2003. Multiple agent-based autonomy for satellite constellations. *Artificial Intelligence* 145(1-2):147–180.

[Schumann et al. 2015] Schumann, J.; Rozier, K. Y.; Reinbacher, T.; Mengshoel, O. J.; Mbaya, T.; and Ippolito, C. 2015. Towards real-time, on-board, hardware-supported sensor and software health management for unmanned aerial systems. *International Journal of Prognostics and Health Management (IJPHM)* 6(1):1–27.

[Srivastava and Schumann 2013] Srivastava, A. N., and Schumann, J. 2013. Software health management: a necessity for safety critical systems. *Innovations in Systems and Software Engineering* 9(4):219–233.

[Taleb, Khoury, and Hallé 2021] Taleb, R.; Khoury, R.; and Hallé, S. 2021. Runtime verification under access restrictions. In *Proceedings of the 9th International Conference on Formal Methods in Software Engineering*.

[Torens, Dauer, and Adolf 2018] Torens, C.; Dauer, J. C.; and Adolf, F. 2018. Towards Autonomy and Safety for Unmanned Aircraft Systems. In Durak, U.; Becker, J.; Hartmann, S.; and Voros, N. S., eds., *Advances in Aeronautical Informatics: Technologies Towards Flight 4.0*. Cham: Springer International Publishing. 105–120.

[University 2011] University, C. M. 2011. Keeping bugs out of software for self-driving cars: Analysis verifies safety of distributed car control system.

[Vassev and Hinchey 2013] Vassev, E., and Hinchey, M. 2013. On the autonomy requirements for space missions. In *16th IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC 2013)*, 1–10. IEEE.

[Winfield et al. 2005a] Winfield, A. F.; Sa, J.; Fernández-Gago, M.-C.; Dixon, C.; and Fisher, M. 2005a. On formal specification of emergent behaviours in swarm robotic systems. *International Journal of Advanced Robotic Systems* 2(4):39.

[Winfield et al. 2005b] Winfield, A. F.; Sa, J.; Gago, M.; and Fisher, M. 2005b. Using temporal logic to specify emergent behaviours in swarm robotic systems. *Proceedings of Towards Autonomous Robotic Systems (TAROS)*.