Improving Usability and Trust in Real-Time Verification of a Large-Scale Complex Safety-Critical System*

Brian Kempa, Chris Johannsen, Kristin Yvonne Rozier
Iowa State University, Ames, Iowa, USA; email: {bckempa,cgjohann,kyrozier}@iastate.edu

Abstract

Large-scale complex safety-critical systems are inherently difficult to both verify in real-time and transparently validate. The iterative specification development process is challenging when the performance and reliability demands of target systems (e.g., flight software) require strict behavior of verification tools which often trade off usability for performance and conformance. Providing both strict behavioral guarantees and efficiency of this iterative process allows specification authors and engineers to more quickly deploy their systems and have more confidence in their verification efforts.

Our on-going work addresses this challenge by providing validation transparency for specification authors during system development while maintaining necessary performance during deployment by extending R2U2, a real-time verification tool specifically designed for resource-constrained systems. We also strengthen the trust in R2U2 by providing a robust suite of tests to show adherence to the strict requirements of safety-critical flight software. These tasks are efforts toward transitioning R2U2 from a research-grade tool to a flight-software-grade tool suitable for use by real-time safety-critical systems and thereby answer the calls for expanded developmental-to-operational verification by, e.g., the Vehicle System Management (VSM) team of the NASA Lunar Gateway.

Keywords: Real-Time and Safety-Critical Systems, Runtime Verification, Developmental Contract Verification, Assume-Guarantee Contracts.

1 Introduction

Complex autonomous real-time systems such as robots, rovers, satellites, and unmanned aerial systems (UAS) must operate reliably for extended periods without human intervention. Runtime verification is a family of techniques that enable such systems to check themselves during operation by identifying and correcting problems as they occur. The legacy approach to runtime verification in software is to use custom ad-hoc algorithms that are difficult to implement, susceptible to errors, and extremely difficult to verify [1]. Large-scale complex safety-critical systems require both real-time verification during system operation but also transparent requirement validation during system development that can carry through to runtime.

After an extensive survey of all currently-available verification tools, the NASA Lunar Gateway Project selected the R2U2 runtime verification engine for use in developing and monitoring autonomous spacecraft software, starting with the Vehicle System Manager (VSM) [1]. The choice was based primarily on R2U2's unobtrusive, flight-certifiable architecture, proven capacity for real-time runtime verification on-board safety-critical systems, and an open-source, extensible C codebase that integrates into the NASA core Flight System/core Flight Executive (cFS/cFE) [2] environment [3]. A hardware version of R2U2 that implements the same algorithms as the C version previously embedded in the space left over on the FPGA controlling NASA's Robonaut2's knee to provide real-time fault disambiguation [4]. The three implementations of R2U2 (hardware/FPGA, C, and C++) have verified many previous safety-critical systems; see [5] for a tool overview and summary of previous case studies. R2U2's underlying specification-monitoring algorithms were originally created specifically to fulfill NASA's needs for a Responsive, Reliable, Unobtrusive Unit (hence the name R2U2) [6], and optimized (with accompanying proofs of correctness) for the Robonaut2 study [4].

While design-by-contract systems like SPARK have provided formal verification in this domain [7], VSM focused on standalone monitors for their verification efforts because they sought runtime visibility of system status instead of design time prescription of component correctness, therefore their selected tool needed to be independent from the flight software implementation [1]. The VSM team has an established verification workflow that includes extensive requirement elicitation in the form of Assume-Guarantee Contracts (AGCs), and the design-time verification technique of model checking to verify AGCs against state-machine models of various sub-systems. However, specification (of models and their requirements) is the biggest bottleneck to verification of autonomy [8]. Developing a system model of required fidelity to fully leverage model checking involves significant effort and

*This work was supported in part by NASA Cooperative Agreement Grant #80NSSC21M0121 and NSF CAREER Award CNS-1552934.

expertise, so only some AGCs are suitable for model checking and others instead undergo a lighter-weight validation analysis with a modified form of runtime verification via R2U2 during development phase instead [9]. The end-goal is to validate AGCs by simulating the possible system executions that satisfy them during system development time, then verify them over simulated system runs (developmental verification) and eventual mission-time execution (operational verification); see [10] for a description of the distinction between developmental runtime verification and simulation.

We describe ongoing work extending the open-source, publicly-available runtime verification engine R2U2, to enable its use for public purposes that are relevant to NASA, including enabling system designers to transparently capture their desired requirements, and making verification results accessible to users. We aim to enhance R2U2 to make it more accessible to software developers and to make R2U2 output tie transparently to the input AGCs. Since the goal is to measurably increase R2U2's usability, user documentation/example uses, and both input and output interfacing, we are evolving R2U2 with regular feedback from a representative NASA mission, in this case, the Lunar Gateway Vehicle System Manager (VSM) team, as an outside check that these goals are being accomplished.

This report previews ongoing work expanding R2U2's usability and trust as a runtime verification engine optimized for a minimal resource footprint running on-board safety-critical systems. Figure 1 displays the features discussed here bounded by the dashed box. The contributions of this paper are (1) extensions to the input and output formats of R2U2 to improve usability when validating and verifying complex specifications and (2) increasing trust in the underlying runtime monitor of R2U2 while maturing research software for flight. Section 2 explores extensions to R2U2's input and output syntax to facilitate specification authorship and validation. Our approach to preparing research software for flight is highlighted in Section 3. Finally, Section 4 summarizes work in progress and next steps.

2 Usability

R2U2 utilizes a minimal input/output syntax to meet real-time deadlines in resource-constrained embedded systems. This low-level syntax makes authoring specifications, validating specifications against requirements, and interpreting verification results more difficult for system engineers. As systems scale in complexity (and often criticality) the mental overhead quickly becomes untenable; however, intuitive naming and reporting schemes can improve specification transparency, reducing iteration time during specification and system development. We extend R2U2's specification syntax with three enhancements to facilitate human validation: Assume-Guarantee Contracts (AGCs), set aggregation operations, and more readable syntax. These ergonomic improvements impact efficiency in authoring and validating specifications, as shown in the "develop" path in Figure 1.

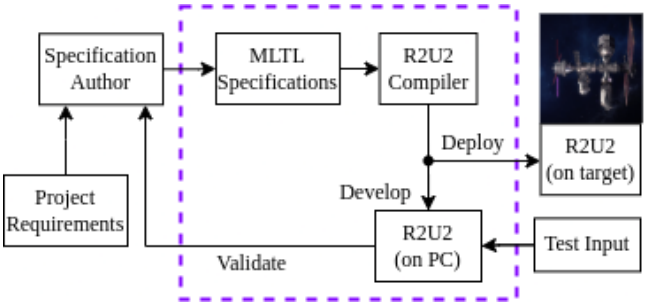

Figure 1: Workflow of the design, validation, and deployment of R2U2 on board the Lunar Gateway where the components in the dashed box are those that the contributions of this paper refer to. A specification author translates the set of AGC project requirements to a set of Mission-time Linear Temporal Logic (MLTL) specifications, compiles these specifications into an R2U2 configuration, then tests and validates the configuration locally. Once the author validates a configuration, engineers then deploy the configuration onto the target platform. (Photo of Lunar Gateway, `https://flic.kr/p/2mHPaLg`, by NASA/Alberto Bertolin CC BY-NC-ND 2.0 / Cropped)

2.1 Assume-Guarantee Contracts

VSM uses AGCs to capture requirements [1]. AGCs are a simple yet powerful requirement framework, but encoding them using logical implication $(assumption \rightarrow guarantee)$ conflates inactive and verified contracts because R2U2's verdicts are Boolean.

The "Old Syntax" in Figure 2 demonstrates a workaround encoding the three conditions as separate formulas in a "one-hot" pattern such that one and only one is true at each timestep. In contrast, we implmented an AGC operator "⇒" in R2U2, which efficiently computes the trinary status and outputs a single clear verdict. The new syntax is easier to write, read, and validate.

2.2 Set Aggregation

We also implement set-aggregation operations as first suggested in [11], permitting succinct requirements over sets of expressions. The set-aggregation operators are syntactic sugar and eliminate long repetitive structures that may hide typos. They accept a set of expressions and evaluate a set property such as "exactly one of these" and are simpler for system engineers to write, interpret, and validate.

For example, a requirement may state that exactly one task shall be active simultaneously. The "Old Syntax" in Figure 2 demonstrates writing this as a disjunction of conjunctions that grow exponentially with the number of tasks, while the "New Syntax" captures this succinctly with a set-aggregation operation.

2.3 Syntax Readability

Internally, R2U2 addresses all values by their index positions in internal vectors. Accordingly, interpreting the output of the old syntax seen in Figure 2 requires knowing the formula number, and mentally mapping that back though the atomic

Old Syntax

```
a0 && ((a1 && !a2 && !a3) || // AGC:
       (!a1 && a2 && !a3) || // TRUE
       (!a1 && !a2 && a3));
!a0; // AGC: INACTIVE
a0 && !((a1 && !a2 && !a3) || // AGC:
        (!a1 && a2 && !a3) || // FALSE
        (!a1 && !a2 && a3));

a0 = bool(s0) == 1;
a1 = bool(s1) == 1;
a2 = bool(s2) == 1;
a3 = bool(s3) == 1;
```

New Syntax

```
RVALID: resRactive => resRvalid;

taskAactive = bool(Aactive) == 1;
taskBactive = bool(Bactive) == 1;
taskCactive = bool(Cactive) == 1;
resRactive = bool(Ractive) == 1;
resRvalid =
  exactly-one-of(active_tasks) == 1;

active_tasks = {taskAactive,
                taskBactive,
                taskCactive};
```

Input

Time	s0 resRactive	s1 taskAactive	s2 taskBactive	s3 taskCactive
0	T	T	F	F
1	T	F	T	F
2	F	F	F	F
3	T	T	T	F

Old Output	**New Output**
0:0 T	RVALID:0 TRUE
1:0 F	
2:0 F	
0:1 T	RVALID:1 TRUE
1:1 F	
2:1 F	
0:2 F	RVALID:2 INACTIVE
1:2 T	
2:2 F	
0:3 F	RVALID:3 FALSE
1:3 F	
2:3 T	

Figure 2: An example usage of R2U2 with the old and new syntaxes. The specification shown captures the system behavior that when a shared resource R is active, exactly one task is using that resource. There is no native support for AGCs and variable names in the old syntax so the specification must be written without these features i.e., each case of the AGC must be explicitly written out and each variable uses a generic name. The new syntax adds these features and as such is more human-readable and easier to validate.

number to the input signal number, increasing the complexity of writing, reading and validating specifications.

Our new syntax and tooling support human-readable labels for formulas, variables, and subexpressions. Named subexpressions allow specifications to resemble the requirements they monitor more closely, while formula names carry through to the output stream, both easing validation. Because the VSM team selected R2U2 for its real-time performance and bounded resource guarantees under flight software restrictions [1], these ergonomic improvements cannot impact the deployed monitor performance. Most of these features only affect the formula compiler, but human-readable output like formula names requires auxiliary information and runtime actions. While development and deployment workflows utilize the same specification files, R2U2 now stores auxiliary data like formula names separately. Deployment monitors do not compile the auxiliary output hooks or read the auxiliary data files, leaving them strictly more performant than development builds, under equivalent conditions.

Additionally, we added an option to dynamically map input signals by the name used in the specification. This added input robustness decouples specification authorship from engineering decisions until target integration, i. e., changing structure definitions no longer requires specification modification.

3 Trust

Academic research software ("gradware") is developed under different motivations than projects targeting third-party use, and unpublishable custodial tasks (e.g., documentation, testing) are often are not attended to beyond what is required for peer acceptance. Software deployed in critical applications, however, must meet a higher bar than standard software best practices. As we convert R2U2 from a research tool to a flight-certified component, we establish trust in R2U2's output with a hierarchical testing campaign, automated analysis-guided peer-review, and adherence to open-source best practices.

3.1 Testing

Our new R2U2 test suite design supports fast iteration as we react to VSM's needs and meet established flight software verification standards, bridging traditional and formal methods.

Unit Tests: Following NASA's standards for VSM flight software, unit tests verify individual functions (e. g., queue operations) and must exercise every line and branch. We parameterize tests over the Cartesian product of the input parameters, covering the input space without repeated code.

R2U2's 66 unit tests currently cover 98.1% of the 577 executable lines and 52.3% of the 2276 branches. The low branch coverage results from a standard C macro idiom for debug print statements that create a do-while structure that can never repeat, generating an unreachable jump instruction. Crucially, these spurious branches do not appear in deployment binaries.

Integration Tests: These black-box tests confirm implementation correctness by comparing the output of R2U2 with a slower but simpler Python oracle over a benchmark set with

2000+ combinations of formulas and input signals. We curate this collection to exercise all logical operators in varied compositions, including published and randomly-generated benchmark specifications. A core set of 50 acceptance tests that cover common cases and check for regressions of previous issues runs in under a minute on consumer hardware. Although the total space of formulas and inputs is infeasible to cover exhaustively, we "fuzz" for edge cases beyond the curated set with randomized inputs and formulas.

3.2 Automated Analysis and Review

GitLab provides version control; all changes automatically trigger the *Continuous Integration* (CI) server, which scans the source with linters and static analysis tools, builds a debug binary with maximum compiler warnings, and runs both test suites with the sanitizer runtimes linked to catch memory mistakes not detectable at compile time. We use CodeChecker[2] to aggregate analysis results from Clang Tidy, CLang Static Analyzer, Cppcheck, Infer, and cpplint. The CI report assists in finding potential defects during code reviews. CI does not measure performance since benchmarks are highly sensitive to environmental context (e. g., working directory, cache alignment, etc.) [12]. Instead, profile-guided optimization is performed on integration target hardware.

3.3 Release Best Practices

Though R2U2 is already open source, code availability is insufficient to ensure the project remains maintainable and accessible for developers of R2U2 and projects incorporating using it. Popular open-source libraries solve this problem with a series of best practices R2U2 is adopting: an open Git repository with full version history, public issue tracking, an established open license, and documentation targeting both users. These tasks are vital to transitioning any research-grade software to software suitable for flight. Beyond the existing in-line comments, we are preparing three documents: 1) a user's guide detailing the use of R2U2 (e.g., formula syntax, output format, target platform integration), 2) a developer's guide with architectural decisions, code style, and algorithm descriptions with proofs, and 3) an API reference autogenerated from the source using Doxygen.

4 Conclusion

NASA's VSM team is actively developing specifications for the Lunar Gateway using our tool chain. The new usability and trust features are crucial for the transition of R2U2 from a research-grade academic tool to one suitable for safety-critical flight-software systems. We continue to collaboratively evaluate user needs, modify the tool accordingly, and monitor the effectiveness of delivered solutions. We are looking forward to insightful experience reports and technical evaluations at the end of the project.

Additionally we are working on: 1) Adding an optimization pass to formula compilation that removes unnecessary instructions (e. g., double negations) and improves partial result reuse to improve performance and reduce resource requirements. 2) Building a visual configuration utility for tuning the static memory bound parameters that provides statistics on formula resource usage. This is also useful when designing new formula sets for a monitor with existing bounds.

[2] https://github.com/Ericsson/codechecker

References

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

[2] NASA, "core Flight System (cFS) Background and Overview." Online: https://cfs.gsfc.nasa.gov/cFS-OviewBGSlideDeck-ExportControl-Final.pdf, 2014.

[3] J. B. Dabney, P. Rajagopal, and J. M. Badger, "Using assume-guarantee contracts in autonomous spacecraft." Flight Software Workshop (FSW) Online: https://www.youtube.com/watch?v=zrtyiyNf674, February 2021.

[4] B. Kempa, P. Zhang, P. H. Jones, J. Zambreno, and K. Y. Rozier, "Embedding Online Runtime Verification for Fault Disambiguation on Robonaut2," in *FORMATS, Proc. 18th*, vol. 12288 of *LNCS*, (Vienna, Austria), pp. 196–214, Springer, September 2020.

[5] K. Y. Rozier and J. Schumann, "R2U2: Tool Overview," in *RV-CUBES*, vol. 3, (Seattle, WA, USA), pp. 138–156, Kalpa Publications, September 2017.

[6] T. Reinbacher, K. Y. Rozier, and J. Schumann, "Temporal-logic based runtime observer pairs for system health management of real-time systems," in *TACAS, Proc. 20th*, vol. 8413 of *LNCS*, pp. 357–372, Springer, April 2014.

[7] P. Neto, J. Tojal, J. Veríssimo, and S. M. de Sousa, "Towards a formally verified space mission software using spark.," *Ada User Journal*, vol. 40, no. 4, pp. 243 – 246, 2019.

[8] K. Y. Rozier, "Specification: The biggest bottleneck in formal methods and autonomy," in *VSTTE, Proc. 8th*, vol. 9971 of *LNCS*, (Toronto, ON, Canada), pp. 1–19, Springer-Verlag, July 2016.

[9] J. B. Dabney, P. Rajagopal, and J. M. Badger, "Using assume-guarantee contracts for developmental verification of autonomous spacecraft." Flight Software Workshop (FSW) Online: https://www.youtube.com/watch?v=HFnn6TzblPg, February 2022.

[10] K. Y. Rozier, "From simulation to runtime verification and back: Connecting single-run verification techniques," in *SpringSim*, (Tucson, AZ, USA), pp. 1–10, Society for Modeling & Simulation Int'l, April 2019.

[11] A. Hammer, M. Cauwels, B. Hertz, P. Jones, and K. Y. Rozier, "Integrating runtime verification into an automated uas traffic management system," *Innovations in Systems and Software Engineering: A NASA Journal*, July 2021.

[12] T. Mytkowicz, A. Diwan, M. Hauswirth, and P. F. Sweeney, "Producing wrong data without doing anything obviously wrong!," *ACM Sigplan Notices*, vol. 44, no. 3, pp. 265–276, 2009.