CISTER
Research Centre in
Real-Time & Embedded
Computing Systems

Technical Report

Capturing and Validating Requirements for Real-Time and Hybrid Systems

Reydel Olano

CISTER-TR-241204

2024

Capturing and Validating Requirements for Real-Time and Hybrid Systems

Reydel Olano

CISTER Research Centre

Polytechnic Institute of Porto (ISEP P.Porto)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail: arrie@isep.ipp.pt

https://www.cister-labs.pt

Abstract

Specifying requirements is essential for aligning project goals, guiding development, ensuring quality, managing risks, optimizing resource use, and achieving stakeholder satisfaction. The goal of this studyis focused on a comprehensive investigation of the logics, formalisms, and specification languages commonly used to describe behavioural properties, including tool support. Next, we select a set of use cases from existing industrial partners, such as vehicle manoeuvrer computation in the Route25 project and motor controllers for railway systems in the VALU3S project. The final step involves specifying the collected requirements using informal and formal specifications, like Fretish, HPL, and dedicated temporal logic. We aim to capture various requirements, addressing discrete computations and continuous evolution, to support formal verification and validation processes. This analysis will clarify the target properties and ensure comprehensive coverage of desired and forbidden system behaviours.

© 2024 CISTER Research Center
www.cister-labs.pt

Capturing and Validating Requirements for Real-Time and Hybrid Systems

Reydel Arrieta Olano

CISTER & Faculty of Engineering of the University of Porto, Portugal
up202101511@up.pt

Abstract. Specifying requirements is essential for aligning project goals, guiding development, ensuring quality, managing risks, optimizing resource use, and achieving stakeholder satisfaction. The goal of this study is focused on a comprehensive investigation of the logics, formalisms, and specification languages commonly used to describe behavioral properties, including tool support. Next, we select a set of use cases from existing industrial partners, such as vehicle manoeuvrer computation in the Route25 project and motor controllers for railway systems in the VALU3S project. The final step involves specifying the collected requirements using informal and formal specifications, like Fretish, HPL, and dedicated temporal logic. We aim to capture various requirements, addressing discrete computations and continuous evolution, to support formal verification and validation processes. This analysis will clarify the target properties and ensure comprehensive coverage of desired and forbidden system behaviors.

Keywords: Specifications · Requirements · Validations

1 Introduction

In real-time and hybrid systems, the ability to describe behavioural properties is crucial for ensuring their correct and reliable operation. These systems, used in a wide range of industrial and technological applications, require specification tools and languages that provide adequate precision and robustness, especially for the creation of runtime monitors.

Requirement specifications [7] for hybrid systems, which combine discrete and continuous components, involve a detailed description of both types of behaviours and their interactions. These specifications are crucial for accurately capturing the dynamics of systems that exhibit both digital and analogue characteristics. They provide a clear framework for designing, developing, and verifying hybrid systems by detailing the expected performance, safety, and reliability criteria. The process includes using specialized specification languages and formalisms to express complex requirements, ensuring comprehensive coverage and facilitating formal verification and validation to meet rigorous industry standards. Natural language descriptions and formal modelling languages each present unique advantages to system designers. Ghosh et al. [3] claim that natural language can effectively initiate stakeholder discussions during the early design stages with its informal nature. However, it may also confuse, lack of automation, and errors. In contrast, formal specifications provide rigour that eliminates ambiguity, supports consistency checking, and enables automatic test case generation. Nevertheless, mastering formal notations demands considerable training and a high level of mathematical sophistication. Refining formal specifications [14] enables the incremental development of a complex specification. This process starts with an abstract model and progressively adds more concrete details to the formal model, step by step.

Various logics, formalisms, and specification languages describe the behavioural properties of real-time and hybrid systems for runtime monitors. Torfah [15] discusses stream-based runtime monitoring, often using real-time stream specification languages like RTLola. Francalanza et al. [2] underscore the importance of expressive specification logic, such as a modal μ -calculus variant, for runtime monitoring. Havelund and Reger [5] highlights the distinctions between specification languages for runtime verification and temporal logics used in model checking, noting the rise of various runtime verification specification languages. Klaedtke [6] introduces POLIMON, a monitoring tool that checks temporal properties over out-of-order streams at runtime using the real-time logic MTL or its extension with the freeze quantifier. These studies collectively emphasize the diversity of logics and languages used in the field of runtime monitoring.

2 Formal Requirements for Runtime Verification

We started by comparing different logics commonly used for runtime verification. We list below a selection of these logics, and highlight some properties of each of the logics in Table 1. Namely, whether the logics use events, states or both to model and reason about the behaviour of systems in their temporal evolution; whether they can be implemented to analyse systems working in real time; examples of tools that can be used to implement each of the logics; and an example of their declaration highlighting some of their core constructs.

Linear Temporal Logic (LTL). LTL is used for specifying and verifying properties of reactive systems, which are systems that continuously interact with their environment and must respond to events over time [12]. In particular, LTL uses special operators $\Box$ (resp. $\diamond$) that express that the rest of the formula must always (resp. eventually) become true. The example in Table 1 captures mutual exclusions, i.e., variables Cs1 and Cs2 are never true at the same time.

Past Linear Temporal Logic (PLTL). Variation of LTL that expresses properties of the past states instead of the future states, as in the more traditional LTL [9]. This logic uses special operators $\blacksquare$ which expresses that the rest of the formula historically becomes true and the operator $\rightarrow$ which expresses that the true state of the variable M2 implies that the variable V(valve) at some point (i.e. through the operator $\blacklozenge$) was open. The example in Table 1 captures time response, i.e., if M2 measured results are within regular flow rate, then V should have been opened at some past time.

Metric Temporal Logic (MTL). It is a version of LTL that specifies complex temporal properties with specific timing constraints in control systems. The complexity of satisfiability and model-checking problems for different MTL fragments ranges from polynomial space to undecidable. Despite this, MTL is well-suited for real-time monitoring requirements, as it allows for the specification of time limits within which certain properties must hold [1]. The example in Table 1 describes a conditional and response time property, i.e., when the traffic light turns on green will become red after 5-time units

Signal Temporal Logic (STL). Allow the specification of temporal properties of real-valued signals (unlike MTL which focuses on discrete events occurring at specific points in time). It has the advantage of naturally admitting quantitative semantics which, in addition to the binary answer to the question of satisfaction, provides a real number indicating the quality of the satisfaction or violation [11]. The example in Table 1 captures a time response requirement, i.e. whenever A vehicle is close to B vehicle, i.e., within the range of 2m, the A vehicle should come to a stop ($| A | < 0.1$) for a short period (2s).

Fretish. It is a restricted structured natural language for writing unambiguous requirements [4] such as safety, response, progress and prevention requirements. It is currently used to generate formal logics, including PLTL, and code for runtime monitors e.g., using CoPilot.

Tessla. A specification language based on stream run-time verification, designed for monitoring a specific class of real-time signals [8] unlike LTL, MTL, and STL which focus on specifying and verifying temporal properties of discrete and continuous systems, with varying approaches to time handling and granularity [10].

High-Level Property Specification Language (HPL). It is a minimalistic specification language tailored for behavioural properties of message-based systems [13]. HPL was developed to be an integral part of the HAROS framework. HAROS can define and extract architectural models of ROS systems (the ROS Computation Graph). Such models are, inherently, mostly concerned with the structure of the analysed system.

From the information obtained from the Table 1 it can be concluded that all logics use states to reason about the behaviour of the systems use cases except for Tessla which can do it through states or events and HPL which does it through events. It is possible to notice that some of the logics work with a specific tool where even the name of the tool coincides with the name of the temporal logic, such as FRET and Tessla, while the rest can be implemented in different tools but one of the most popular ones was selected for its representation. Concerning the ability to represent requirements for systems working in real time it is observed that except for LTL and PLTL all logics can do so. The examples of each logic show how the syntax varies from one to another, this feature implicitly implies that one logic can be more explicit than another in reflecting the specifics of the requirements of a use case.

Table 1. Temporal logics for runtime verifications

Logic	State	Event	Tool support	Real-time	Example
LTL	Yes	No	R2U2	No	$\Box\neg($ in_Cs1 $\wedge$ in_Cs2$)$
PLTL	Yes	No	CoPilot	No	$\blacksquare($M2_flow$\rightarrow \blacklozenge$ V_open$)$
MTL	Yes	No	R2U2	Yes	$\Box(\text{green} \rightarrow (\neg\text{red } U_{[5]} \text{ red}))$
STL	Yes	No	StoRM	Yes	$\Box(\lvert A-B\rvert < 2) \rightarrow \Box_{[0,2]} (\lvert A\rvert < 0.1)$
Fretish	Yes	No	Fret	No	*“in roll hold mode RollAutopilot shall always satisfy autopilot engaged && no other lateral mode”*
Tessla	Yes	Yes	Tessla	Yes	`in e: Events<Unit>` `in s: Signal<Int>` `define comp:=eventCount(e) > s` `define allow:=within(-1,1,filter(e,comp))` `define ok := implies(s > 5, allow)` `out ok`
HPL	No	Yes	Haros and HPL-RV	Yes	globally: no /laser {dist<0 or dist>100} within 0.1s

3 Work Plan

This section outlines the strategy to fulfil the hypothesis. The first step comparing logics, formalisms, and specification languages commonly used for describing behavioral properties of real-time and hybrid systems, with a focus on runtime monitors and tool support.

The second step involves selecting use-cases from CISTER's industrial partners. A key candidate is an engine for computing vehicle maneuvers in the Route25 project. Alternatives include with alternatives including motor controllers for a railway company in the VALU3S project, or components for semi-autonomous vehicles by CEiiA.

The third step involves specifying requirements using informal (e.g., Fretish, HPL) and formal (e.g., temporal logic) specifications. These requirements should capture desired and forbidden system properties, considering both discrete and continuous behaviors. The goal is to compile diverse requirements, including aspects like failure probability and good-enough criteria, for formal verification and validation.

4 Conclusion

The comparative strategy implemented allowed us to know in more detail the characteristics of the temporal logic to allow a clear understanding of how the properties of our target should be taken into account. At present, the work is in the phase where it is necessary to define the use case that will be studied to select from the previously analysed temporal logic which one is the most suitable to represent. The idea is to present in the best possible way all the specifications of the requirements that guarantee the correct functioning of the selected use case.

5 Acknowledgments

This work was supported by national funds through the FCT/MCTES, Portuguese Foundation for Science and Technology, within the CISTER, ISEP/IPP Research Unit (UIDP/UIDB/04234/2020),

It was also supported by National Funds through FCT/MCTES and by project Ibex (ref. PTDC/CCI-COM/4280/2021) financed by national funds through FCT.

References

1. Chatterjee, K., Henzinger, T.A.: Formal modeling and analysis of timed systems, vol. 6246. Springer (2010)
2. Francalanza, A., Aceto, L., Achilleos, A., Attard, D.P., Cassar, I., Della Monica, D., Ingólfsdóttir, A.: A foundation for runtime monitoring. In: International Conference on Runtime Verification. pp. 8–29. Springer (2017)
3. Ghosh, S., Elenius, D., Li, W., Lincoln, P., Shankar, N., Steiner, W.: Arsenal: automatic requirements specification extraction from natural language. In: NASA Formal Methods: 8th International Symposium, NFM 2016, Minneapolis, MN, USA, June 7-9, 2016, Proceedings 8. pp. 41–46. Springer (2016)
4. Giannakopoulou, D., Mavridou, A., Rhein, J., Pressburger, T., Schumann, J., Shi, N.: Formal requirements elicitation with fret. In: International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ-2020). No. ARC-E-DAA-TN77785 (2020)
5. Havelund, K., Reger, G.: Runtime verification logics a language design perspective. Models, Algorithms, Logics and Tools: Essays Dedicated to Kim Guldstrand Larsen on the Occasion of His 60th Birthday pp. 310–338 (2017)
6. Klaedtke, F.: Polimon: Checking temporal properties over out-of-order streams at runtime. arXiv preprint arXiv:2404.15723 (2024)
7. Laplante, P.A., Kassab, M.: Requirements engineering for software and systems. Auerbach Publications (2022)
8. Leucker, M., Sánchez, C., Scheffel, T., Schmitz, M., Schramm, A.: Tessla: runtime verification of non-synchronized real-time streams. In: Proceedings of the 33rd Annual ACM Symposium on Applied Computing. pp. 1925–1933 (2018)
9. Mao, X., Li, X., Huang, Y., Shi, J., Zhang, Y.: Programmable logic controllers past linear temporal logic for monitoring applications in industrial control systems. IEEE Transactions on Industrial Informatics **18**(7), 4393–4405 (2021)
10. Nickovic, D.: Checking timed and hybrid properties: Theory and applications. (vérification de propriétés temporisées et hybrides: théorie et applications) (2008), https://api.semanticscholar.org/CorpusID:42021523
11. Raman, V., Donzé, A., Sadigh, D., Murray, R.M., Seshia, S.A.: Reactive synthesis from signal temporal logic specifications. In: Proceedings of the 18th international conference on hybrid systems: Computation and control. pp. 239–248 (2015)
12. Rozier, K.Y.: Linear temporal logic symbolic model checking. Computer Science Review **5**(2), 163–203 (2011)
13. Santos, A., Cunha, A., Macedo, N.: Schema-guided testing of message-oriented systems (2022)

14. Sayar, I., Souquières, J.: Formalization of requirements for correct systems. In: 2020 IEEE Workshop on Formal Requirements (FORMREQ). pp. 28–34. IEEE (2020)
15. Torfah, H.: Stream-based monitors for real-time properties. In: Runtime Verification: 19th International Conference, RV 2019, Porto, Portugal, October 8–11, 2019, Proceedings 19. pp. 91–110. Springer (2019)