Early Validation of High-level Requirements on Cyber-Physical Systems

Ondřej Vašíček[1]

[1]Brno University of Technology, Brno, Czechia

Abstract

My work is our research group's first step towards tackling the problem of automated, early validation of high-level system requirements with a focus on safety-critical, cyber-physical systems (CPS), such as airplanes. Such systems need to guarantee safe operation despite their ever-increasing complexity and development costs. Ensuring that high-level requirements are error-free is crucial to avoid their propagation into the rest of the development process and to avoid the high cost of fixing them in the later phases of development. Our approach is to transform the requirements into Event Calculus and to reason about them using ASP solvers. We have already successfully applied this approach to an open-source medical device (published at ICLP'24) and will be presenting another paper at ICLP'25 on non-termination issues in goal-directed Event Calculus reasoning.

Keywords

Requirements Validation, Event Calculus, Answer Set Programming, s(CASP)

1. Introduction and Motivation

Early validation of specifications describing requirements placed on cyber-physical systems (CPSs) under development is essential to avoid costly errors in later stages of the development, especially when the systems undergo certification. According to a study conducted within the AVSI SAVI project [1] around 70 % of errors in large systems are introduced during the specification of system requirements, yet over 50 % of those errors are only discovered during the integration testing phase much later in the development process. Unfortunately, the cost to fix an error in the integration testing phase is around 16-times higher than in the initial requirements specification phase.

To the best of our knowledge there are currently no ready-to-use solutions for automated, early validation of truly high-level requirements. Requirements specifications are still most often in textual form which traditionally suffers from ambiguity and is not easily machine understandable. The most common way of validating such requirements is expert review, which can lead to errors being overlooked due to the human factor, due to the size and complexity of requirements specifications, and due to implicit assumptions not being included explicitly in the specification. There is a need for better ways to express clear and unambiguous requirements, to relate them to system model elements, and to be able to automatically transform both the requirements and the models into suitable formalisms which could then be used by methods for validation and verification.

A crucial need, when trying to transform a requirements specification into a suitable formalism, is that the semantic gap between the requirements and the formalism used to model them for the purposes of validation is as small as possible. A larger semantic gap makes it more difficult to transform the requirements into a model, and, most importantly, any validation on such a model drifts away from validating the requirements themselves and closer to validating that particular model—influenced by design and implementation decisions. As described in [2], Event Calculus (EC) is a formalism suitable for commonsense reasoning. The semantic gap between a requirements specification and its EC encoding is near-zero because its semantics follows how a human would think of the requirements. Using Answer Set Programming (ASP) [3] and the s(CASP) [4] system for goal-directed reasoning in

Joint Proceedings of the Workshops and Doctoral Consortium of the 41st International Conference on Logic Programming, September 9–13, 2025, Rende, Italy

ivasicek@fit.vut.cz (O. Vašíček)

0000-0002-4944-2198 (O. Vašíček)

EC, the work [5] has demonstrated the versatility of EC for modelling and reasoning about CPSs while providing explainable results. My last paper [6] expanded on [5] further enhancing its capabilities, resulting in a practical application on the PCA pump [7] a real safety-critical medical device.

2. Background

The Event Calculus (EC) [2] EC is a formalism for reasoning about events and change, of which there are several axiomatizations. There are three basic concepts in EC: *events*, *fluents*, and *time points*: (i) an event is an action or incident that may occur in the world, e.g., the dropping of a glass by a person is an event, (ii) a fluent is a time-varying property of the world, such as the altitude of a glass, (iii) a time point is an instant of time. Events may happen at a time point; fluents have a truth value at any time point, and these truth values are subject to change upon an occurrence of an event. In addition, fluents may have quantities associated with them as parameters, which change discretely via events or continuously over time via trajectories. We chose EC as a formalism suitable for representing requirements specifications due the the low semantic gap between EC and the requirements.

The s(CASP) System [4] s(CASP) extends the expressiveness of ASP [3] systems, based on the stable model semantics [8], by including predicates, constraints among non-ground variables, uninterpreted functions, and, most importantly, a top-down, query-driven execution strategy. These features make it possible to return answers with non-ground variables (possibly including constraints among them) and to compute partial models by returning only the fragment of a stable model that is necessary to support the answer to a given query. Answers to all queries can also include the full proof tree, making them fully explainable. Like other ASP implementations and unlike Prolog, s(CASP) handles non-stratified negation and returns the corresponding (partial) stable models. Further, s(CASP) implements abductive reasoning via even loops [4], where it automatically searches for suitable values of the predicates in the corresponding even loop in order to satisfy the main query. We chose s(CASP) as the solver for reasoning about EC models especially due to its grounding-free nature, which allows us to reason in continuous time and about continuous change of fluents.

3. Related Work

As already mentioned, we found EC suitable for reasoning about requirements specifications due to its low semantic gap against them. In comparison, the semantics of automata-based approaches, which are often used in the literature to model CPSs, such as timed automata [9] or hybrid automata [10], require one to "design" explicit states and transitions, and may lead to decomposition of the system into sub-systems each with their own automaton. Current industrial model-based engineering approaches, such as those based, e.g., on Matlab Simulink models and tools like HiLiTE [11], are only suitable for validation of low-level requirements. This is due to the low-level nature of the models they use, especially when automated generation of code from the models is required. Much research has been done on using temporal logics (e.g., LTL, CTL, CTL* [12]) and real time temporal logics (e.g., MTL [13]) to represent system properties. However, we have not considered temporal logics as a target for transforming high-level system specifications since the semantics of the temporal logics that we are aware of are further away from natural language which makes the transformation more difficult to perform and to understand in comparison with EC.

Apart from our EC-based approach [5, 6], there are other ones which aim to target automated validation of high-level requirements put on CPSs. The work [14] is based on ontologies and uses theorem proving, which traditionally requires significant manual work. The work [15] is based on transforming CPS specifications from templated-English into process algebras extended with real-time aspects, however, no continuous variables (apart from time) are dealt with and no experimental results are presented, which makes it difficult to judge the scalability of this approach.

Finally, there are other ASP solvers than the s(CASP) system that we are currently using, such as the grounding-based Clingo [16]. However, in our past experiments, Clingo has proven to be unsuitable for reasoning about fluents with large or continuous value domains due to the explosion in the grounding and a need to discretize the time. This causes the solver to quickly run out of memory even on models with very limited continuous value domains, while the discretized time steps introduce issues when trying to step on exact values during periods of continuous change. An advantage of Clingo is that it does not suffer from non-termination issues, which make things much more complicated in s(CASP). We are currently in the process of investigating the practical capabilities of hybrid versions of Clingo, such as Clingo[LP] [17], for our use case, which should be able to deal with the grounding problem.

4. Research Goals and Current State

The overarching objective of my research is to improve the development process of safety-critical systems with a focus on validation and verification. The main goal is to propose analyses for high-level requirements in order to detect more errors as early as possible in the development process. This goal consists of three sub-goals. In my research thus far, I have already partially completed the first two sub-goals and have achieved some results in the third one, which is still the focus of my current efforts.

1. A suitable formalism needs to be selected which will be able to adequately represent requirements specifications for the purpose of performing analyses on the requirements. In my research, I have selected Event Calculus as a suitable formalism based on already published results [5, 18] and on my own experiments. Experimenting with the practical capabilities of EC is an ongoing effort entailed with sub-goal 3.
2. Then, a way of transforming the requirements specifications into the selected formalism needs to be defined. For EC, limited manual transformations have already been shown in [5, 18]. In my research, I have since used further manual transformations for the time being. Once I sufficiently explore and advance the modelling and reasoning capabilities of EC, my focus will shift towards proposing a general and at least semi-automated way to transform requirements into EC.
3. Further research should focus on efficient analysis methods for validating the formalized requirements specifications. The aim is to make the reasoning more scalable, to make it capable of supporting more constructs from the requirements specifications, and to use it for new kinds of analyses. Since my selected formalism is currently EC, my recent research is focusing on the capabilities of (abductive) commonsense reasoning using ASP solvers [4, 16], which have already been shown to be promising by other researchers in the past [5, 18].

In our last year's paper [6], we have practically applied the above approach. The work used EC (from sub-goal 1) and a manual transformation (from sub-goal 2) in order to explore and improve the reasoning capabilities of ASP solvers, especially s(CASP), for the purposes of requirements validation (towards sub-goal 3). The latest result of our research takes a step back from applying the whole approach to instead focus on dealing with general problems that can arise in reasoning about Event Calculus. A brief overview of both of the above papers (emphasis on the latest one) is given the next section.

5. Current Results

5.1. Application on the PCA Pump [ICLP'24]

In my first paper in this area [6], I have applied the EC+s(CASP) approach on the specification of a real safety-critical system in order to both assess and demonstrate its practical capabilities (and current limitations) and to guide further work on improving it. We have developed a model of the core operation of the PCA pump [7]—a real safety-critical device that automatically delivers pain relief drugs into the blood stream of a patient. We have then proposed two validation methods using automated reasoning powered by s(CASP). The first one checks the consistency between the requirements specification and

the use cases and exception cases based on which the requirements were created. The the second one checks whether the requirements specification satisfies general properties, such as that the system should never overdose the patient, where we were able to leverage abductive reasoning. The work demonstrated that EC is able to model the requirements specification of a non-trivial, real-life cyber-physical system in s(CASP) and the reasoning involved can lead to discovering issues in the requirements while producing valuable evidence towards their validation. Indeed, the work resulted in the discovery of a number of issues in the PCA pump specification, which we have discussed and confirmed with the authors of the specification. The work also opened up a number of new research avenues, one of which we have addressed in our latest paper, discussed in the next paragraph.

5.2. Addressing Non-termination in s(CASP) [ICLP'25]

Non-termination (like in Prolog resolution) is one of the main limitations of s(CASP) that we have encountered while working on [6], where we have spent significant effort on avoiding it in a case-by-case manner. Our latest paper [19] explores the causes of non-termination when reasoning about EC using s(CASP) in continuous (or dense) time and proposes solutions. Based on our experiments so far, we believe that the main (if not only) source of non-termination when reasoning about Event Calculus are triggered events. Occurrences of such events are implied by the rules of the system instead of being given only as facts in a narrative. This means that there can be cyclic dependencies between the triggered events which can lead to an infinite number of event occurrences when reasoning in continuous time. Exploring an infinite number of events then manifests as non-termination in s(CASP) resolution.

We have identified a number of general, common EC modeling patterns that lead to non-termination and classified them as *Zeno-like behaviors*, named for their similarity to the well known Zeno's paradoxes. A common definition of *Zeno behavior* (or Zeno runs, Zeno executions,...) is that an infinite number of events (or state transitions) occurs in finite time, which is not possible in a real system and is a product of abstraction. This is a well-known problem in timed/hybrid systems and automata [20, 21, 22, 23, 24, 20, 25, 26]. However, as far as we know, a systematic exploration of Zeno behaviors in EC has not been conducted yet. Our definition of *Zeno-like behavior* is that the reasoning *explores* an infinite number of events in a finite time interval, but an infinite number of events do not *actually occur*, making it only similar to Zeno behavior. Using eleven examples from the literature, we have identified four general types of Zeno-like behaviors in common Event Calculus modeling tasks (or modeling patterns) and proposed one or more solutions for each of them. Below we very briefly summarize the types of Zeno-like behaviors.

Preventing repeated triggering of events: A very common concept in EC is ensuring that an event is only triggered once while its trigger condition holds. This is demonstrated on the bank account example in [2]. However, we show that this example fundamentally has no solution in continuous time due to the nature of the EC axioms. The problem is that a trigger condition of an event holds in a time interval with a non-inclusive lower bound. If the event is only to be triggered once, then the earliest occurrence prevents subsequent occurrences, however, we are unable to find the earliest occurrence because it is impossible to express a timepoint which is infinitely close to the non-inclusive lower bound. Essentially, the example calls for an infinitely fast (but not instantaneous) response, which is not realistic. The solution is either to remodel the example to use an instantaneous response or to define some response delay.

Self-ending trajectories: Another common concept in EC are autoterminating trajectories, i.e., trajectories that are terminated by a triggered event whose trigger condition depends on the value defined by the trajectory itself. We show this problem on the example of a fading light (very similar to the falling object [2] or the filling vessel [27]). In s(CASP) reasoning a very simple non-terminating loop is created due to the effect of the trigger rule being considered as a way to prove its own trigger condition. The solution is to restrict the rule choice so that only the right rules are considered.

Circular trajectories: The axioms of EC are defined in such a way that in order to prove the effect of a trajectory at some time `T2` we must first prove its start time `T1`. However, when there is a circular dependency between trajectories the reasoning gets stuck trying the prove the start time of an infinite number of infinitely short trajectories while never checking what their effect will be or if they will actually occur. We demonstrate this problem using the example of a pulsing light (a circular version of the fading light, similar to a simplified bouncing ball [22]). The solution to this problem is to introduce duration information into the loop, which is only suitable for trajectories with constant (or context independent) duration, or to employ forward reasoning (via incremental reasoning) starting with the very first trajectory, which by definition has no preceding trajectory to consider as its starting trigger.

Self-ending trajectories with inequality: Typically a trajectory is terminated exactly when it reaches a given threshold, i.e, based on equality. However, sometimes inequality is called for, terminating the trajectory based on its value being above (or below) some threshold. Should such a trajectory start when its value is already above the threshold, then we face a manifestation of the repeated trigger problem, where a trajectory is defined to end infinitely quickly with an infinitely short duration. The solutions are similar, either we remodel to an instantaneous response to not start the trajectory at all (since its supposed to end infinitely quickly) or we introduce a delay in the form of a minimum duration.

The above problems can be solved using four types of general solutions: restricting rule choice, incremental reasoning, adding a delay or duration, and remodeling to an instantaneous response.

We have modeled three examples that feature true Zeno behavior. The example of two water tanks [28], which are being slowly drained and one at a time is being refilled by an input arm, combines all the prior Zeno-like behaviors as sub-problems and, in addition, introduces real Zeno behavior. We showed that all the prior solutions can be applied to address the sub-problems and that this results in the Zeno behavior being dealt with as well.

Finally, we have identified a unifying feature that is present in the reasoning tree of all the Zeno-like behavior problems that we have presented. A *Zeno-descending chain of triggered events* is a sequence of repeated occurrences of the same event at timepoints `T1 > T2 > T3 > ... > TN`. Where the variables representing the timepoints have the same value intervals and are constrained into a strictly descending sequence relative to each other without any given delay in between. Such a chain is not well-founded w.r.t. the temporal axis. We leveraged the existence of the Zeno-descending chain to implemented a detection technique, which looks for the chain during reasoning and terminates the execution if a chain is detected. This prevents the non-termination and informs the user of the problem. The technique is able to reliably and quickly detect all the presented Zeno-like behaviors. The true Zeno behaviors currently cannot be detected by this technique, because they do not feature a Zeno-descending chain but rather its more complex variation, in which there are delays defined between the events in the chain (i.e., the timepoint variables will not have the same value domains) but the delay gets shorter and shorter with each event.

6. Open Issues

There are still many open issues to be tackled in future work. A range of them consists of implementation tasks needed to improve the s(CASP) system, many of which are less interesting from the research perspective due to their engineering nature.

Performance scaling We observe an exponential increase in solving complexity when introducing more events into a narrative. We believe that the scaling can be greatly improved via a more efficient approach to reasoning. One source of inefficiency is the need to prove the entire history of the timeline from time zero up to the current time whenever we reason about any aspects of EC at a given timepoint. We believe that many predicates are being re-proven multiple times throughout the process of answering

a query due to the need to re-prove the entire history. We have already partially addressed this via a prototype cache, however, the scaling currently still remains exponential. A promising approach is incremental solving which reasons about smaller intervals of the narrative timeline at a time. It has shown a potential to significantly improve the performance scaling.

Abductive reasoning We have so far only implemented a limited form of abductive reasoning in s(CASP) for Event Calculus. Abductive reasoning in continuous time introduces many challenges that need to be addressed and for which the stock implementation of abduction in s(CASP) is not ready. In particular, abducing event occurrences is currently not possible because the reasoning will try to abduce an infinite number of occurrences due to the use of continuous time.

Generality and automation Further open issues are related to making our approach more general and practically usable. These belong more into the Software Engineering Engineering research domain, rather than in the domain of Logic Programming. Currently, our transformation of requirements to EC is entirely manual, although we try to keep it as general as possible. In order for our approach to be practically usable in the industry, we need to propose at least a semi-automated transformation for general requirements of a wide enough class of systems. The main obstacle in the way of automation is the format of requirements specifications as they are largely written in unconstrained natural language. Natural language is inherently ambiguous and hard to reliably process by machines. We believe that introducing a more structured language, such as MIDAS by [29], for facilitating formulation of requirements should provide enough structure and context to the requirements in order to enable a more general and at least semi-automated transformation of the requirements into EC. Another option to tackle this issue might be the use of LLMs due to their recent success in processing natural language, however, the issue of hallucinations and lack of explainability makes their use difficult in the domain of safety-critical systems which have to go through certification.

Acknowledgments

For their guidance and contributions, I am grateful to my supervisors Tomáš Vojnar, co-supervisor Jan Fiedor, and other collaborators Joaquin Arias, Gopal Gupta, Brendan Hall, Bohuslav Křena, and Brian Larson.

Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

References

[1] D. Ward, AVSI: Moving SAVI to the Launch Pad, NDIA 2012 (2012).

[2] E. T. Mueller, Commonsense Reasoning: An Event Calculus Based Approach, Morgan Kaufmann, 2014. doi:10.1016/B978-0-12-801416-5.00002-4.

[3] V. Lifschitz, Answer Set Programming, Springer International Publishing, Cham, 2019. doi:10.1007/978-3-030-24658-7.

[4] J. Arias, M. Carro, E. Salazar, K. Marple, G. Gupta, Constraint Answer Set Programming without Grounding, TPLP 18 (2018).

[5] S. C. Varanasi, J. Arias, E. Salazar, F. Li, K. Basu, G. Gupta, Modeling and Verification of Real-Time Systems with the Event Calculus and s(CASP), in: Practical Aspects of Declarative Languages, Springer International Publishing, Cham, 2022, pp. 181–190.

[6] O. Vasicek, J. Arias, J. Fiedor, G. Gupta, B. Hall, B. Krena, B. Larson, S. C. Varanasi, T. Vojnar, Early Validation of High-Level System Requirements with Event Calculus and Answer Set Programming, TPLP 24 (2024) 844–862. doi:10.1017/S1471068424000280.

[7] J. Hatcliff, B. Larson, T. Carpenter, P. Jones, Y. Zhang, J. Jorgens, The Open PCA Pump Project: An Exemplar Open Source Medical Device as a Community Resource, SIGBED Rev. 16 (2019) 8–13. doi:10.1145/3357495.3357496.
[8] M. Gelfond, V. Lifschitz, The Stable Model Semantics for Logic Programming, in: Proc. of 5th International Conference on Logic Programming, 1989, pp. 1070–1080. doi:10.2307/2275201.
[9] K. G. Larsen, F. Lorber, B. Nielsen, 20 Years of UPPAAL Enabled Industrial Model-Based Validation and Beyond, in: Proc. of ISoLA'18, LNCS 11247, Springer, 2018.
[10] G. Frehse, An Introduction to Hybrid Automata, Numerical Simulation and Reachability Analysis, Springer, 2015.
[11] D. Bhatt, G. Madl, D. Oglesby, K. Schloegel, Towards Scalable Verification of Commercial Avionics Software, in: Proc. of Infotech@Aerospace'10, AIAA, 2010. doi:https://doi.org/10.2514/6.2010-3452.
[12] E. M. Clarke, T. A. Henzinger, H. Veith, R. Bloem (Eds.), Handbook of Model Checking, Springer, 2018. doi:10.1007/978-3-319-10575-8.
[13] S. Konur, A survey on temporal logics for specifying and verifying real-time systems, Frontiers of Computer Science 7 (2013) 370–403. doi:10.1007/S11704-013-2195-2.
[14] A. Crapo, A. Moitra, C. McMillan, D. Russell, Requirements Capture and Analysis in ASSERT(TM), in: Proc. of RE'17, 2017. doi:10.1109/RE.2017.54.
[15] M. Arnaud, B. Bannour, A. Lapitre, G. Giraud, Investigating process algebra models to represent structured requirements for time-sensitive CPS, in: SEKE'21, 2021. doi:10.18293/SEKE2021-147.
[16] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, Multi-shot ASP solving with clingo, Theory and Practice of Logic Programming 19 (2019) 27–82. doi:10.1017/S1471068418000054.
[17] T. Janhunen, R. Kaminski, M. Ostrowski, S. Schellhorn, P. Wanko, T. Schaub, Clingo goes linear constraints over reals and integers, Theory and Practice of Logic Programming 17 (2017) 872–888. doi:10.1017/S1471068417000242.
[18] B. Hall, S. C. Varanasi, J. Fiedor, J. Arias, K. Basu, F. Li, D. Bhatt, K. Driscoll, E. Salazar, G. Gupta, Knowledge-Assisted Reasoning of Model-Augmented System Requirements with Event Calculus and Goal-Directed Answer Set Programming, in: HCVS 2021, EPTCS 344, 2021.
[19] O. Vasicek, J. Arias, J. Fiedor, G. Gupta, B. Hall, B. Krena, B. Larson, T. Vojnar, On Zeno-like Behaviors in the Event Calculus with Goal-directed Answer Set Programming, in: Proc. of ICLP'25, 2025.
[20] J. Zhang, K. H. Johansson, et. al., Zeno hybrid systems, IJRNC 11 (2001) 435–451. doi:https://doi.org/10.1002/rnc.592.
[21] T. A. Henzinger, The theory of hybrid automata, in: LICS, IEEE CS, 1996, pp. 278–292. doi:10.1109/LICS.1996.561342.
[22] K. H. Johansson, M. Egerstedt, J. Lygeros, S. Sastry, On the regularization of zeno hybrid automata, Systems & control letters 38 (1999) 141–150.
[23] R. Gómez, H. Bowman, Efficient detection of zeno runs in timed automata, in: FORMATS, volume 4763 of *LNCS*, Springer, 2007, pp. 195–210. doi:10.1007/978-3-540-75454-1_15.
[24] J. Rinast, S. Schupp, Static detection of zeno runs in uppaal networks based on synchronization matrices and two data-variable heuristics, in: FORMATS, Springer, Berlin, Heidelberg, 2012, pp. 220–235.
[25] A. G. Lamperski, A. D. Ames, Lyapunov theory for zeno stability, IEEE TAC 58 (2013) 100–112. doi:10.1109/TAC.2012.2208292.
[26] F. Herbreteau, B. Srivathsan, Coarse abstractions make zeno behaviours difficult to detect, Logical Methods in Computer Science Volume 9, Issue 1 (2013). doi:10.2168/LMCS-9(1:6)2013.
[27] M. Shanahan, Solving the frame problem, MIT Press, 1997.
[28] R. Alur, T. A. Henzinger, Modularity for timed and hybrid systems, in: CONCUR, volume 1243 of *LNCS*, Springer, 1997, pp. 74–88. doi:10.1007/3-540-63141-0_6.
[29] B. Hall, J. Fiedor, Y. Jeppu, Model Integrated Decomposition and Assisted Specification (MIDAS), in: INCOSE 2020, volume 30, 2020.