Extended Abstract: Early Validation of High-level System Requirements with Event Calculus and Answer Set Programming

Ondřej Vašíček[1], Joaquin Arias[2], Jan Fiedor[1,3], Gopal Gupta[4], Brendan Hall[5], Křena Bohuslav[1], Brian Larson[6], Sarat Ch. Varanasi[7] and Tomáš Vojnar[1,8]

[1]*Brno University of Technology, Brno, Czechia*
[2]*Universidad Rey Juan Carlos, Madrid, Spain*
[3]*Honeywell International s.r.o., Brno, Czechia*
[4]*The University of Texas at Dallas, Texas, USA*
[5]*Ardent Innovation Labs, USA*
[6]*Multitude Corporation, St Paul, Minnesota, USA*
[7]*GE Aerospace Research, USA*
[8]*Masaryk University, Brno, Czechia*

Abstract

This is an extended abstract of [1] *O. Vašícek, J. Arias, J. Fiedor, G. Gupta, B. Hall, B. Křena, B. Larson, S. C. Varanasi, T. Vojnar, Early Validation of High-level System Requirements with Event Calculus and Answer Set Programming*, which is a paper accepted at ICLP'24.

Keywords

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

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. However, there is a lack of suitable automated tools and techniques for this purpose. A crucial need here is that of a small semantic gap between the requirements and the formalism used to model them for the purposes of validation. As described by [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 EC, the work [5] has demonstrated the versatility of EC for modelling and reasoning about CPSs.

In this work, we develop a model of the core operation of the PCA pump [6]—a real safety-critical device that automatically delivers pain relief drugs into the blood stream of a patient—in order to assess, demonstrate, and improve the practical capabilities (and current limitations) of the EC+s(CASP) approach. The model operates in a way similar to an early prototype of the system and, thus, can be used to reason about its behaviour. However, due to the nature of EC, the behaviour of the model is very close to what the requirements themselves describe.

Modelling the PCA Pump Requirements in EC under s(CASP) The transformation of the requirements specification of the PCA pump into an EC representation implemented in s(CASP) was done manually. The requirements specification and all the source codes of its s(CASP) representation can be found at https://github.com/ovasicek/pca-pump-ec-artifacts/. The below is a brief, illustrative overview of s(CASP) code for the delivery of a patient bolus (one of the features of the pump), which

4th Workshop on Goal-directed Execution of Answer Set Programs (GDE'24), October 12, 2024
ivasicek@fit.vut.cz (O. Vašíček); joaquin.arias@urjc.es (J. Arias); ifiedor@fit.vut.cz (J. Fiedor); gupta@utdallas.edu (G. Gupta); bren@ardentinnovationlabs.com (B. Hall); krena@fit.vutbr.cz (K. Bohuslav); brl@multitude.net (B. Larson); SaratChandra.Varanasi@ge.com (S. Ch. Varanasi); vojnar@fi.muni.cz (T. Vojnar)
0000-0002-4944-2198 (O. Vašíček); 0000-0003-4148-311X (J. Arias); 0000-0002-2746-8792 (T. Vojnar)

is an extra dose of drug delivered upon the patient's request. We define events that start and end the delivery of the bolus which is represented by a state fluent (lines 1-4). The total amount of drug delivered to the patient and how the amount increases during the bolus is represented by a continuous fluent and a trajectory (lines 6-10). And finally, the bolus stops automatically once the right amount of drug (so called VTBI) is delivered which is represented by an event triggered based on the drug delivered during the ongoing bolus (lines 12-15). A new continuous fluent and trajectory are used to represent the volume of drug delivered by a bolus counting from zero instead of computing the difference of total drug delivered at the start and at the end of a bolus (lines 17-20).

```
1  fluent(patient_bolus_delivery_enabled).
2  event(patient_bolus_delivery_started).   event(patient_bolus_delivery_stopped).
3  initiates(patient_bolus_delivery_started, patient_bolus_delivery_enabled, T).
4  terminates(patient_bolus_delivery_stopped, patient_bolus_delivery_enabled, T).
5
6  fluent(total_drug_delivered(X)).
7  trajectory(patient_bolus_delivery_enabled, T1, total_drug_delivered(Total), T2) :-
8    basal_and_patient_bolus_flow_rate(FlowRate),
9    holdsAt(total_drug_delivered(StartTotal), T1),
10   Total #= StartTotal + ((T2 - T1) * FlowRate).
11
12 event(patient_bolus_completed).
13 happens(patient_bolus_completed, T2) :-   initiallyP(vtbi(VTBI)),
14   holdsAt(patient_bolus_drug_delivered(VTBI), T2).
15 happens(patient_bolus_delivery_stopped, T) :- happens(patient_bolus_completed, T).
16
17 fluent(patient_bolus_drug_delivered(X)).
18 trajectory(patient_bolus_delivery_enabled,T1, patient_bolus_drug_delivered(X),T2):-
19   patient_bolus_only_flow_rate(FlowRate),
20   X #= (T2 - T1) * FlowRate.
```

Validating Consistency of Use/Exception Cases and the Requirements The first validation method that we propose is a way to check the consistency between the behaviour defined by the requirements specification and the use cases (UC) and exception cases (ExC) based on which the requirements were created (or, in general, checking consistency of the behaviour against any scenarios defined at a different level of the specification) by, essentially, simulating the UC/ExC. This is done by transforming the UC/ExC into an EC narrative and forming a query based on the UC/ExC and its post-conditions. If running the query on the narrative using s(CASP) fails, then we have found an inconsistency. Using this technique we were able to identify a number of such inconsistencies in the PCA pump specification. A simplified example of such a narrative and query for *UC2: Patient-Requested Bolus* is shown below. The use case specifies the delivery of a patient requested bolus which is requested during basal delivery (the baseline delivery mode), is not denied by the pump, and the pump goes back to basal delivery after the bolus finishes. Lines 1-2 specify the input events which occur in the narrative, and lines 3-9 define the query to be checked. Full implementation of UC2 can be found in uc2.pl.

```
1 happens(start_button_pressed,     60).
2 happens(patient_bolus_requested, 120).
3 ?- holdsIn(basal_delivery_enabled,                          60, 120),
4    happens(patient_bolus_delivery_started,                     120),
5    not_happens(patient_bolus_denied,                           120),
6    initiallyP(vtbi(VTBI)),  happens(patient_bolus_completed,    T2),
7    holdsAt(patient_bolus_drug_delivered(VTBI),                  T2),
8    happens(basal_delivery_started,                              T2),
9    holdsAfter(basal_delivery_enabled,                           T2).
```

Table 1
Results of validating consistency of use/exception cases and the requirements

	Use Case Name	Variant	Result	Time (s)
UC2	Patient-Requested Bolus	no variants	OK	3.17
UC3a	Clinician-Requested Bolus	not suspended	OK	2.84
UC3b	Clinician-Requested Bolus	suspended and resumed	OK	37.13
ExC7a-f	Over-Flow Rate Alarm	defined in ExC step 1	FAIL	1.53-4.62
ExC13a	Maximum Safe Dose	during basal delivery	FAIL	25.62
ExC13b-c	Maximum Safe Dose	during each bolus	OK	31.61-53.45
ExC21	Reservoir Empty	no variants	OK	40.99

Table 2
Results of validating the requirements wrt. the possibility of an overdose

	Use Case Name	Variant	Model	Result	Time (m)
ExC13b	Maximum Safe Dose	patient bolus	original	OK	14.11
ExC13c	Maximum Safe Dose	clinician bolus	original	OK	20.85
ExC13a	Maximum Safe Dose	during basal	original	overdose	15.29
			fixed	OK	3.34
UC2	Patient-Requested Bolus	abduction	original	overdose	38.01
			fixed	OK	48.11

Validating the Requirements wrt. General Properties The second validation method that we propose is a way to check whether the requirements specification satisfies general properties, such as that the system should not allow an overdose of the patient or that the system should respond to an event within a given time limit. This is done by representing a general property as an s(CASP) query and checking that query on suitable narratives. In this way, we were able to detect that the patient can be overdosed by the PCA pump in certain specific narratives, which is a safety property violation caused by a missing requirement. We were further able to leverage the abductive reasoning capabilities of s(CASP) in order to generalize the narrative on which the property is being checked. In our case of checking the possibility of an overdose, we were able to abduce the parameters of an overdose (what volume of drug is allowed over what time period) and, subsequently, detect the possibility of an overdose in the "sunny day" narrative of UC2 in which the overdose does not directly occur otherwise.

Results of Experimental Evaluation We have simulated all relevant UCs and ExCs from the PCA pump specification and checked the possibility of an overdoes of the patient. Some of the cases appear in multiple variants of the narrative and we aggregate the measurements of variants of the same case that led to the same result to save space. Implementations of all experiments can be found in the narratives_and_queries folder. Table 1 shows selected representative results of simulating UCs and ExCs. All UCs were simulated successfully, but quite a few ExCs failed. This has led to the discovery of a number of issues in the specification, such as inconsistencies in alarm responses or defined constants. Table 2 shows results and execution times of querying overdose on variants of ExC13 (implemented in overdose-ec13b.pl and similar) and of using abduction on UC2. Execution of the overdose queries takes much longer than the simulation queries from Table 1 due to the higher complexity of the overdose query. However, the abductive queries are the slowest ones due to the higher complexity of abduction in general but also due to the limitations of its current implementation in s(CASP).

Techniques Used to Empower s(CASP) Reasoning We further present a number of challenges encountered during the translation of the requirements to EC encoded in s(CASP) and during the subsequent evaluation, based on deductive as well as abductive reasoning, which was often too costly

or non-terminating. We have applied and, in multiple cases, also newly developed various techniques that helped us resolve many of these challenges. These include extensions of the axiomatization of the EC (in a similar way as [7]) and special ways of translating certain parts of the specifications in order to avoid non-termination, in particular when dealing with certain trajectories. Further, we present an original approach to abductive reasoning in s(CASP) with incrementally refined abduced values in order to assure consistency of the abduced values whenever abduction on the same value is used multiple times in the reasoning tree. Next, we proposed a mechanism for caching evaluations (failure-tabling and tabling of ground sub-goal success) of selected predicates (in a similar way as mode-directed tabling [8, 9]) that was added into s(CASP) as a prototype leading to a significant increase in performance. We also describe a way of separating the reasoning about the trigger and the effect of certain complexity-inducing triggered events into multiple reasoning runs where each run produces new facts to be used in the subsequent ones, which reduces their performance impact.

Conclusion Our work demonstrated that EC can be used 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.

Our future work involves improving s(CASP) in order to make it more efficient and to reduce the number of non-termination issues, and making our approach more general and practically usable, in particular targeting the transformation phase by introducing a more structured specification language such as [10].

References

[1] O. Vašícek, J. Arias, J. Fiedor, G. Gupta, B. Hall, B. Křena, B. Larson, S. C. Varanasi, T. Vojnar, Early Validation of High-level System Requirements with Event Calculus and Answer Set Programming, in: proc. of ICLP, 2024. doi:10.48550/arXiv.2408.09909.

[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, 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) 337–354. doi:10.1017/S1471068418000285.

[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: proc. of PADL'22, volume 13165 of *LNCS*, Springer, 2022, pp. 181–190.

[6] 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.

[7] M. Shanahan, An Abductive Event Calculus Planner, The Journal of Logic Programming 44 (2000).

[8] H.-F. Guo, G. Gupta, Simplifying Dynamic Programming via Mode-directed Tabling, Software: Practice and Experience 38 (2008) 75–94. doi:10.1002/spe.824.

[9] J. Arias, M. Carro, Incremental Evaluation of Lattice-Based Aggregates in Logic Programming Using Modular TCLP, in: proc. of PADL'19, volume 11372 of *LNCS*, Springer, 2019, pp. 98–114. doi:10.1007/978-3-030-05998-9_7.

[10] B. Hall, J. Fiedor, Y. Jeppu, Model Integrated Decomposition and Assisted Specification (MIDAS), INCOSE International Symposium 30 (2020). doi:10.1002/j.2334-5837.2020.00757.x.