A Tool for Requirements Analysis of Safety-Critical Cyber-Physical Systems

Freek van den Berg[1(✉)] and Boudewijn R. Haverkort[2(✉)]

[1] Eindhoven University of Technology, Groene Loper 3, 5612 AE Eindhoven, The Netherlands
f.g.b.v.d.berg@tue.nl

[2] Tilburg University, Warandelaan 2, 5037AB Tilburg, The Netherlands
B.R.H.M.Haverkort@tilburguniversity.edu

Abstract. One of the key challenges in the design of a Safety-Critical Cyber-Physical Systems is Requirements Analysis. Current Requirements Analysis approaches range from informal, human-centered ones that are hard to automate, to formal approaches that often lack freedom of expression. Furthermore, most approaches are general-purpose and do not focus on a particular domain, which makes identifying the specific requirements of a given domain less trivial.

To overcome these challenges, this paper presents `aDSL`, a Domain-Specific Language and toolset for Requirement Analysis of Safety-Critical Cyber-Physical Systems. The approach comprises a mixture of informal and formal elements to enable both automation and freedom of expression; a number of stakeholders introduce and negotiate about their requirements. The `aDSL` language is used to precisely, concisely and unambiguously describe all such requirements. We have validated `aDSL`, using simulation techniques and actors that represent the stakeholders, on a case in the agro-machines domain. The proposed approach allows the discovery of requirements in a semi-automatic way.

Keywords: Safety-Critical Cyber-Physical System · Requirement analysis · System designer · Domain-Specific Language · Stakeholders · Negotiation

1 Introduction

A Cyber-Physical System (CPS) is a real-time feedback system that is controlled and monitored by computer-based algorithms. A CPS integrates [1] embedded systems, human users, networks [2], and concurrent physical systems [1,3]. Designing and constructing a CPS requires expertise from many disciplines, including signal processing [2], control engineering [2], computer engineering,

The work in this paper was initiated using a 4TU.NIRICT grant involving a cooperation between University of Twente and Wageningen University & Research.

H. Hermanns (Ed.): MMB 2020, LNCS 12040, pp. 242–258, 2020.
https://doi.org/10.1007/978-3-030-43024-5_15

software engineering and programming, electronics, and mechanical engineering [2]. CPSs are common in safety-critical domains, such as aerospace, civil infrastructure, automotive, energy and agriculture. A Safety-Critical CPS (SC-CPS, [4]) is defined as a CPS that is designed to be fail-safe for safety purposes [5]. Malfunction of a SC-CPS may lead to: death or serious injury to people, loss or damage to equipment or property or environmental harm. Hence, system engineering for SC-CPSs is very challenging because there is no room for error. One of the key activities in the design of these SC-CPSs is the Requirements Engineering process.

Requirements Engineering is mainly concerned with defining the requirements [6] and typically comprises the following five subsequent phases: (i) **system modeling**: generate a system model to test the requirements on; (ii) **requirements elicitation**: research and discover the premature system requirements; (iii) **requirements analysis**: make the requirements clear, complete, consistent and unambiguous; (iv) **requirements specification**: document requirements in a formal artifact; and (v) **requirements validation**: check whether the documented requirements meet the needs of the stakeholders. Requirements evolve and mature as they go through the different Requirements Engineering activities. Pohl proposes a 3D-model [7] for the maturity of a requirement: (i) representation: from informal to formal, (ii) specification: from opaque to complete; and, (iii) from personal to common view; requirements tend to be informal, opaque and person during the elicitation phase, and formal, complete and common during the specification phase.

In this paper, we will primarily focus on the second phase of Requirements Engineering, that is, requirement analysis (RA). Current RA processes range from informal, human-centered to formal approaches. Informal approaches, often driven by Unified Modelling Langage (UML, [8]), tend to include natural language-based documents, e.g., use cases [9], user stories, and process specifications. The use of natural language provides a great freedom of expression, but is hard to formalize when parts of the RA process need to be automated. On the other hand, formal approaches aim to deliver artefacts that are unambiguous and ready for automatic processing. However, their freedom of expression is restricted to their grammar, making it hard for the stakeholders to freely express what they mean. Moreover, the grammar of the language is likely to differ from the language they are accustomed to.

Although RA has been an important activity in current system engineering life-cycle processes for quite some time, current approaches are usually general-purpose and do not focus on a particular domain. Due to this general purpose nature, identifying the specific requirements of the given domain is more challenging. Particularly, SC-CPSs require a dedicated approach that is easy to use and tailored to its domain [4]. A Domain Specific Language (DSL), i.e., a computer language specialized to a particular application domain, enables such a dedicated approach. Hence, a DSL tends to be less comprehensive, much more expressive in the domain, and exhibits less redundancy than general-purpose languages. Therefore, solutions can be expressed and validated at the level of

abstraction of the problem domain. On the downside, designing, implementing, maintaining and learning a DSL involves costs. DSLs are supported by tools such as JetBrains MPS, which is a DSL tool[1], and Xtext, an open-source software framework for developing DSLs[2]. For instance, iDSL [10–12], on which `aDSL` is inspired, has been constructed using Xtext and provides a DSL and toolset for performance evaluation of service-oriented systems.

To meet the aforementioned challenges, this paper proposes `aDSL`, a DSL and toolset for RA of SC-CPS. The `aDSL` approach is a mixture of informal and formal approaches to obtain the best of both worlds, i.e., the freedom of expression of informal methods and the ability to automate of formal methods. `aDSL` enables the system designer and other stakeholders to precisely describe the requirements of a SC-CPS, after which the `aDSL` toolset takes care of the RA task in a semi-automated manner. The `aDSL` language and toolset have been evaluated using a case study on agro-machines [13] for validation.

Outline. This paper's remainder is organized as follows. Section 2 introduces a generic algorithm for Requirements Analysis. Section 3 formalizes this approach using an illustrative case study. Section 4 present an implementation. In Sect. 5, the algorithm of Sect. 4 is validated. Section 6 concludes the paper.

2 A Generic Algorithm for Requirement Analysis

This section introduces an algorithm for RA, which we will apply to a CPS. The UML sequence diagram[3] of Fig. 1 displays the subsequent steps and iterations of the algorithm, will be implemented in Sect. 4. It can be observed that Fig. 1 logically combines all RE activities as presented in the introduction, as follows.

1. The input of the generic algorithm is a system model with a formal system description and requirements (cf. Sect. 1, RE step (i)). This model will be implemented in Sect. 3.
2. Requirements elicitation is simulated using two subsequent steps, viz., retrieving the initial requirements from the system model, followed by transforming them into many premature requirements (cf. Sect. 1, RE step (ii)).
3. Requirements analysis (cf. Sect. 1, RE step (iii)) comprises some iterations. In each iteration, two requirements are selected to be merged into one requirement. Typically, these requirements are similar. To this end, the two actors that represent the stakeholders that own these requirements negotiate about these requirements, yielding a new requirement that summarizes the two original requirements. One of the actors becomes the new requirement owner and both original requirements are removed. By the fact that we use a formal language, we consider requirements specification (cf. Sect. 1, RE step (iv)) to be taken care of automatically.

[1] JetBrains MPS https://www.jetbrains.com/.
[2] Xtext - Language Engineering Made Easy https://eclipse.org/.
[3] Figures 1 and 2 have been constructed using PlantUML: https://plantuml.com/.

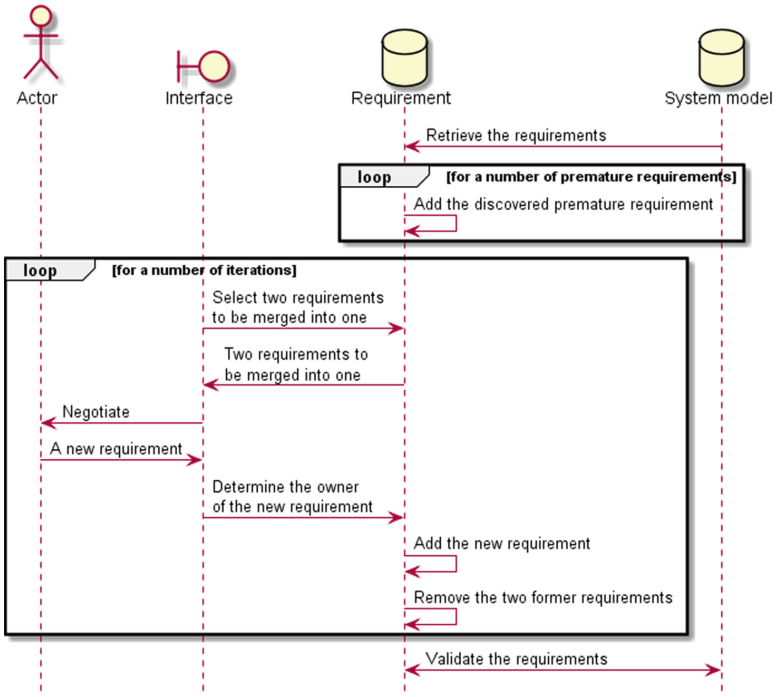

Fig. 1. The sequence diagram of the RA pseudo-algorithm.

4. Requirement validation (cf. Sect. 1, RE step (v)) ensures that the requirements meet stakeholders's needs and that the algorithm as presented in this paper is valid. As a consequence, the requirements generated by the algorithm and the initial requirements need to be similar.

We formalize the algorithm to reduce ambiguity and complexity, using the following notation.

1. Time $T : \{0, 1, 2, \ldots, m\}$ is discrete, where m is the number of iterations.
2. System model SM is constant for all time units $t \in T$. Hence, time is only used and relevant for the time it takes to run the algorithm.
3. System model SM encompasses n formal requirements $FR = \{r_1, r_2, \ldots, r_n\}$
4. A requirement $r \in FR$ has exactly one owner $o \in O$. Hence, $Owner : RF \rightarrow O$ is defined, where $Owner$ is the function that maps requirements to owners.
5. During elicitation, formal requirements $r_1, r_2, \ldots, r_n \in FR$ transform into premature requirements $r'_1, r'_2, \ldots, r'_o \in PR$, with o premature requirements.
6. A timed requirement $Rt : T \times FR$ describes a requirement $r \in FR$ achieved at $t \in T$.
7. All premature requirements are initiated at $t = 0$. Therefore, timed requirements $(0, r'_1), (0, r'_2), \ldots, (0, r'_o)$ exist; put formally: $Tir = \{(0, pr) \,|\, pr \in PR\}$, with PR the set of premature requirements.

8. A transaction $Trans : (T \times FR)^2 \rightarrow (T \times FR)$ at time $t \in T$ involves two timed, input requirements (t, r_1), $(t, r_2) \in (T \times FR)$ and yields a new timed requirement $(t + 1, r') \in (T \times FR)$.
9. The owner O of a requirement resulting from a transaction involving two timed requirements (t, r_1), $(t, r_2) \in (T \times FR)$ is either the owner of requirement r_1 or r_2; put formally: $Trans((t, r_1, t), (r_2)) = (t + 1, r') \rightarrow (Owner(r') = Owner(r_1) \bigvee Owner(r') = Owner(r_2))$.

3 A Formal Model for Requirement Analysis of CPSs

In this section, we present the model of `aDSL`, which is the domain-specific language and toolset for CPSs we have constructed. For this purpose, we built on an illustrating case study that has been introduced in previous work [13]. This case study concerns a CPS, which is a tractor that is connected to one out of several trailers. Moreover, a tractor can have different engines and transmissions, yielding many design alternatives. We have left this case study as intact as possible; the CPS, its subsystems, its parts and the design alternatives are as in [13]. To add support for Requirements Analysis to `aDSL`, we introduce the following three additions to the language:

1. `aDSL` has been extended with a **notion of time** to allow requirements to evolve over time. Initially, only premature requirements are defined, which evolve into mature requirements via a mechanism in which stakeholders that own an requirement *negotiate*. This mechanism enables the system designer and stakeholders to discover the requirements in a systematic, yet creative manner. In the `aDSL` language, the notion of time can been observed in the RA measure (cf. Table 1e), viz., the measure contains a parameter iterations that specifies the time the algorithm runs.
2. The formal requirements of the `aDSL` instance transform into **premature requirements** via reverse engineering, which are more informal, opaque and personal (to be implemented in Sect. 4). In the `aDSL` language, the conversion from formal requirements into premature ones is not visible because the toolset uses a fixed strategy.
3. A measure for RA has been added to `aDSL` to **configure experiments**. An experiment involves a number of iterations in which stakeholders with certain behavior, represented by automated actors, negotiate about requirements and the way these requirements are selected. A so-called experiment space can then be used to conveniently define many different instances of this measure. The RA measure and corresponding experiment space of the case study are introduced Table 1e

At its highest level of hierarchy, an `aDSL` instance comprises five so-called sections[4], as follows.

[4] The underlying `aDSL` grammar can be downloaded from https://www.utwente.nl/en/eemcs/adsl/appendices/appendix-online.pdf.

Table 1. The aDSL instance of the case study

(a) System

```
Section system
  Top-level System TractorTrailorCombination
    AbstractSystem Tractor tractor
    DesAlt(trailor)
    { chiselPlow AbstractPart TrailorChiselPlow tCPlow }
    { trailortiller AbstractPart TrailorTiller tTiller }
    { chaserbin AbstractPart ChaserBin chaserBin }
    { notrailor Part NoTrailor OperationSpace (load [0 0]
                        activity {none} ) }
System tractor
    AbstractSystem Transmission trans   AbstractSystem Fuel fuel

System trans
    DesAlt ( transmission )
    { unsynchronized AtomicSystem transUnsynchronized
        OperationSpace ( driverSkills { advanced moderate easy }
            continousOperation { no } gears [ 1 24 ] ) }
    { doubleClutch AtomicSystem transDoubleClutch OperationSpace
        ( driverSkills { moderate easy } gears [ 1 24 ] ) }
    { CVT AtomicSystem transCVT OperationSpace ( driverSkills
              { easy } efficiency { frictionLoss } gears [ 1 1000 ] ) }
System fuel
    DesAlt ( engineFuel )
    { steam AtomicSystem fuelSteam
        OperationSpace ( pollution [5 10] speed [0 30] ) }
    { diesel AtomicSystem fuelDiesel OperationSpace ( pollution [4 8]
fuelConsumption [3 5] speed [0 40] price {medium high} ) }
    { gasoline AtomicSystem fuelGasoline OperationSpace
        ( pollution [2 5] fuelConsumption [4 10] speed [0 50]
        price { medium high } ) }
    { electric AtomicSystem fuelElectric OperationSpace ( pollution[0 4]
        fuelConsumption[8 12] speed[0 55] price{high})}
```

(b) Part

```
Section part
    Part tCPlow OperationSpace
      ( speed [0 25] agility { low veryLow } activity { plow } )
    Part tTiller OperationSpace
      ( speed [0 25] agility { low veryLow } activity { till } )
    Part chaserBin OperationSpace
      ( speed [0 15] agility { veryLow } activity { harvest } )
```

Table 1. (continued)

(c) Requirement

```
Section requirement
 Requirement RA1 minimum OperationSpace (speed [5 15]
       fuelConsumption [8 10] price medium gears [1 30])
   maximum OperationSpace (speed [0 45] fuelConsumption [0 20] price
       { low medium high } driverSkills { easy } pollution [0 6])
 Requirement RA2 minimum OperationSpace (speed [7 12]
       fuelConsumption [4 7] price medium gears [4 24])
   maximum OperationSpace (speed [0 35] fuelConsumption [0 15]
       driverSkills easy pollution [4 8])
```

(d) Design space

```
Section design space
  DesignSpace ( transmission {unsynchronized doubleClutch CVT}
      trailor {chiselPlow trailortiller chaserbin notrailor}
      engineFuel { steam diesel gasoline electric } )
```

(e) Measure

```
Section measure
  Measurement RAnalysis simulation has 12 iterations 10 runs, actors
    have distribution espace (actorIntersection) intersection
    espace (actorConjunction) conjunction espace (actorCompromise)
    compromise espace (actorCoerce) coerce, requirements are selected
    using espace (selectReq) entropy (2-espace(selectReq)) jaccard,
    requirement owner is selected using espace (selectOwner) entropy
    (2-espace (selectOwner)) jaccard
  ExperimentSpace ( actorIntersection {0 1 2} actorConjunction
    {0 1 2} actorCompromise {0 1 2} actorCoerce {0 1 2}
    selectReq {0 1 2} selectOwner {0 1 2} )
```

Section system comprises one **top-level system**, i.e., the CPS under study, and its subsystems. The CPS and subsystems have a name and operation space (to be explained later). At a lower level, a system comprises one or more so-called SystemOrParts, i.e., either a **system**, **part**, **systemID** or **partID**.

Case Study. Table 1a contains the `aDSL` system instance. The top-level system "TractorTrailorCombination" comprises a "Tractor" (of type tractor) and a selection out of four possible trailors, which is expressed using the Design Alternative (DesAlt) construct in `aDSL`. In turn, a "Tractor" consists of a "Transmission", a DesAlt construct with three options, and a "Fuel" kind, a DesAlt construct with four options.

Section part encompasses zero or more parts. On top of that, a system and a part have an **operation space**, which represents a number of operational modes.

An operation space consist of zero or more dimensions. A dimension is either a bounded integer range or a finite set of elements.

Case Study. Table 1b shows parts "tCPlow", "tTiller" and "chaserBin", and their operation spaces. E.g., "chaserBin" has an operation space with dimensions "speed" (range: $[0:15]$), "agility" (value: veryLow) and "activity" (value: harvest).

Section requirement contains **requirements** that limit the valid operation spaces of a CPS. For this purpose, a requirement is defined using a minimum and maximum operation space. An ordering on operation spaces is then used to determine if the operation spaces of each system and part of a CPS meet a requirement, as follows. Let $O1$ and $O2$ be operation spaces, represented as sets of dimension and value pairs. Then $O1$ is at most $O2$ (denoted with the usual "$\leq$" operator) when $O1$ comprises dimension d, either $O2$ does not comprise dimension d or dimension d of $O2$ contains at least all values of dimension d of $O1$; put formally:

$$O1 \leq O2 \rightarrow \begin{cases} (d:v) \in O1 \rightarrow \nexists x (d:x) \in O2 \\ (d:x) \in O1 \rightarrow (d:x) \in O2 \end{cases}$$

Hence, only common dimensions are considered for comparison. Note that previous work [13] discusses this ordering of operation spaces more extensively.

Case Study. Table 1c conveys two requirements, which are a combination of the requirements of previous work [13]. They transform into premature requirements as part of the implementation to be introduced in Sect. 4.

Section measure comprises measure "RAnalysis" to enable the experiments for requirements analysis. An experiment is initialized using twelve parameters, e.g., to define the actor behavior and the way requirements are selected, as carefully explained next in the case study. To conveniently define many instances of this measure, `aDSL` has also been equipped with a so-called experiment space. An experiment space comprises $K \in \mathbb{N}^+$ dimensions that each have a number of values. The K^{ary} Cartesian product of these dimensions form the set of experiment instances. We use a so-called espace construct with twelve parameters in order to vary all parameters of the experiment.

Case Study. Table 1e contains Measure "RAnalysis". It is initialized with a constant number of 12 iterations and 10 simulation runs in this case. The remaining ten parameters contain espace constructs that refer to ExperimentSpace dimensions. We define experiment space $\hat{E}$, which comprises the set of experiments E that have different parameters, as follows:

$$
\begin{aligned}
\hat{E} = \{E(12, 10, & \quad \#\text{the number of iterations (12) and runs (10).} \\
ac_1, ac_2, ac_3, ac_4, & \quad \#\text{ the relative probabilities of actor} \\
& \quad \text{intersection}(ac_1), \text{conjunction } (ac_2), \\
& \quad \text{compromise } (ac_3)\text{and coercion } (ac_4). \\
rs_1, rs_2, & \quad \#\text{ the relative probabilities of using Entropy } (rs_1) \\
& \quad \text{and Jaccard } (rs_2) \text{ for requirement selection.} \\
os_1, os_2) \mid & \quad \#\text{the relative probabilities of using Entropy } (os_1) \\
& \quad \text{and Jaccard}(os_2) \text{ for requirement owner selection.} \\
ac_1 + ac_2 + ac_3 + ac_4 > 0 & \quad \#\text{ at least one relative actor probability is not zero.} \\
rs_1 + rs_2 = 2 & \quad \#\text{ the req. sel. rel. probs. are (2,0), (1,1) or (0,2).} \\
os_1 + os_2 = 2 & \quad \#\text{ the owner sel. rel. probs. are (2,0), (1,1) or (0,2).} \\
ac_1, ac_2, ac_3.ac_4, rs_1, rs_2, os_1, os_2 \in \{0, 1, 2\}\} & \quad \#\text{the 10 parameters are 0, 1 or 2.}
\end{aligned} \tag{1}
$$

Section design space contains a **design space** with n dimensions. The n^{ary} Cartesian product of the n dimensions is then set of design alternatives. On top of that, DesAlt refers to a dimension, and maps each dimension value to a SystemOrPart.

Case Study. Table 1d contains three dimensions, viz., "transmission", "trailor and "engineFuel". Combined, they yield $3 \times 4 \times 4 = 48$ design alternatives. The `aDSL` system (see Table 1a) contains three DesAlts, which are so-called variation points. That is, for each design alternative they are evaluated differently.

4 An Implemented Algorithm for RA of SC-CPS

In Sect. 2, a generic algorithm was presented for RA of CPSs (as depicted in Fig. 1). In this section, we implement this algorithm to make it executable[5]. To this end, the following individual parts are implemented as graphically depicted in Fig. 2: (i) capturing the premature requirements (cf. Sect. 4.1); (ii) evolution of the requirements (cf. Sect. 4.2); and, (iii) validation of the requirements (cf. Sect. 4.3). For precision, step (ii) is executed for 10 simulation runs as defined in Section Measure (see Table 1e).

4.1 Capturing the Premature Requirements

Capturing the requirements corresponds to system elicitation (cf. Sect. 1, RE step (ii)). In practice, this is a non-trivial, labor-intensive step which involves interviews, questionnaires, user observations, workshops, etc. Therefore, we simulate it using a reverse engineering step instead, as follows. We split the formal

[5] The work in this paper has been implemented using Xtext for DSLs and Xtend.

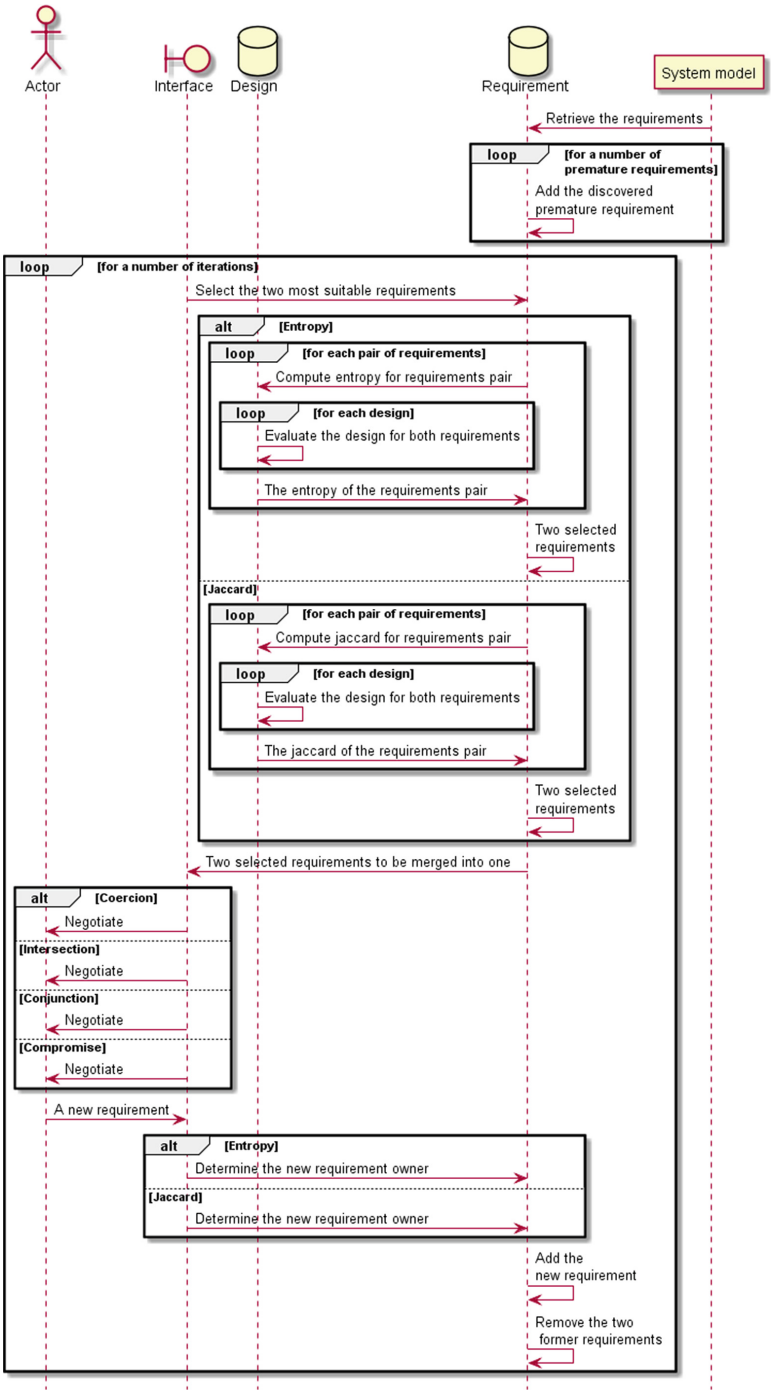

Fig. 2. The sequence diagram of the RA implemented-algorithm.

requirements (cf. Table 1c) of the system model into many smaller premature requirements. These premature requirements should then lead to the formal requirements again after executing the algorithm, which validates the approach (see Fig. 2). Premature requirements are generated, as follows.

Let $(o_{min}, o_{max}) = r$ represent formal requirement r with minimum operation space o_{min} and maximum operation space o_{max}. Then each premature requirement $pr \in PR$ derived from r has a minimum operation space o'_{min} and maximum operation space o'_{max}, where o'_{min} is o_{min} reduced by one element and o'_{max} is o_{max} reduced by one element; put formally:

$$R = \{(o'_{min}, o'_{max}) | o'_{min} = o_{min} \backslash e_i, e_i \in o_{min}, o'_{max} = o_{max} \backslash e_j, e_j \in o_{max}\} \quad (2)$$

where R is the set of generated requirements for requirement $(o_{min}, o_{max}) = r$

Table 2. The generation of premature requirements from requirement RA2

(a) Minimum operation spaces

i	e_i	$o_{min} \backslash e_i$
1	speed [7 12]	(fuelConsumption [4 7] price { medium } gears [4 24])
2	fuelConsumption [4 7]	(speed [7 12] price { medium } gears [4 24])
3	price { medium }	(speed [7 12] fuelConsumption [4 7] gears [4 24])
4	gears [4 24]	(speed [7 12] fuelConsumption [4 7] price { gears }

(b) Maximum operation spaces

j	e_j	$o_{max} \backslash e_i$
1	speed [0 35]	(fuelConsumption [0 15] driverSkills {easy} pollution [4 8])
2	fuelConsumption [0 15]	(speed [0 35] driverSkills {easy} pollution [4 8])
3	driverSkills {easy}	(speed [0 35] fuelConsumption [0 15] pollution [4 8])
4	pollution [4 8]	(speed [0 35] fuelConsumption [0 15] driverSkills {easy})

Case Study. We generate the premature requirements for formal requirement RA2 (see Table 1c), as follows. First, we compute the elements of operation spaces o'_{min} and o'_{max} (as shown in Table 2), which are the elements of o_{min} and o_{max} with one element omitted, respectively. Second, the operation spaces of the generated premature requirements are the Cartesian product of the resulting minimum (cf. Table 2a, third column) and maximum (cf. Table 2b, third column) operation spaces. Hence, formal requirement RA2 yields $4 \times 4 = 16$ premature requirements, where $R_{i,j}$ is the requirement with the i^{th} and j^{th} element omitted with $i, j \in \{1, 2, 3, 4\}$ (cf. Table 2a and b, first column). For illustration, the first ($R_{1,1}$) and last premature requirement ($R_{4,4}$) are:

- **Requirement** $R_{1,1}$ **minimum OperationSpace** (fuelConsumption [4 7] price { medium } gears [4 24]) **maximum OperationSpace** (fuelConsumption [0 15] driverSkills {easy} pollution [4 8])
- **Requirement** $R_{4,4}$ **minimum OperationSpace** (speed [7 12] fuelConsumption [4 7] price { gears } **maximum OperationSpace** (speed [0 35] fuelConsumption [0 15] driverSkills {easy})

4.2 Evolution of the Requirements

The evolution of the requirements takes place in iterations of the following three steps (as in Figs. 1 and 2): (i) selecting requirements to evolve; (ii) stakeholders negotiating about requirements resulting in new requirements; and, (iii) select an owner for the requirement.

(i) Selecting two requirements to evolve occurs on the basis of their similarity, viz., when two requirements are similar, one of them is likely to be obsolete. `aDSL` provides two similarity measures to determine which two requirements are most similar, viz., Entropy and Jaccard. Each iteration, the measure Entropy is selected with probability $\frac{Rs_1}{Rs_1+Rs_2}$ and measure Jaccard with $\frac{Rs_2}{Rs_1+Rs_2}$ (cf. Eq. 2 and Table 1e). Let $R_1, R_2, \ldots, R_n$ be n requirements from which we would like to select the two most similar requirements. To this end, we evaluate all requirements for all n designs $D_1, D_2, \ldots, D_n$. Consequently, $M(d, r) = \top$ when design d meets requirement r, and $M(d, r) = \bot$ when it does not.

Next, we focus on pairs of requirements R_i and R_j. Hence, a design can be evaluated in four ways, viz., $M(d, R_i) = \bot$ or $\top$, and simultaneously $M(d, R_j) = \bot$ or $\top$. We then count all these four options individually, as follows:

$$k_{i,j} = \sum_{n=1}^{|D|} \mathbb{I}\{ \mid M(d_n, R_i) = i, M(d_n, R_j) = j \mid \} \tag{3}$$

where $k_{i,j}$ is the number of designs that yield i for R_1 and j for R_2, $i, j \in \{\bot, \top\}$, $\mathbb{I}(\top) = 1$ and $\mathbb{I}(\bot) = 0$, $|D|$ the number of designs, d_n the n^{th} design, and R_1, R_2 the compared requirements. Also, $l_{i,j} = k_{i,j} \,/\, |D|$ is a relative version of $k_{i,j}$. Now, we compute Entropy and Jaccard for selecting requirements, as follows.

1. Entropy [14]: the expected amount of information. A higher outcome corresponds to less similarity. It is defined as:

$$E(r_1, r_2) = -\sum_{i=0}^{1}\sum_{j=0}^{1} l_{i,j} \cdot \log(l_{i,j}) \tag{4}$$

   where we assume that $0 \cdot \log(0) = 0$, since $\lim_{n\to 0}(n \cdot log(n)) = 0$.
2. Jaccard [15]: originally used for the similarity of two sets. A higher outcome corresponds to a higher similarity. It is defined as:

$$J(r_1, r_2) = \frac{k_{1,1}}{k_{1,1} + k_{0,1} + k_{1,0}} \tag{5}$$

(ii) Stakeholders Negotiate About the Requirements They Own. Two stakeholders negotiate about the requirement they each own, i.e., R_1 and R_2, and come up with one new requirement, i.e., R_{new} that replaces the two previous

requirements. In reality, this transformation is performed by two humans, actors or stakeholders. To be able to conduct many experiments (see Sect. 5 for the results), we simulate human actors using a combination of four actor implementations. In each iteration, they are selected with probabilities $\frac{Ac_n}{Ac_1+Ac_2+Ac_3+Ac_4}$ (cf. Eq. 3 and Table 1e), where $n \in \{1,2,3,4\}$ is the option number. The implementations are defined as follows.

1. **Intersection**: requirement R_{new} is satisfied iff the operation spaces of both requirements R_1 and R_2 are satisfied; put formally:

$$\begin{aligned} R_{new}^{min} &= \{(d, x \cup y) \mid (d,x) \in R_1^{min}, (d,y) \in R_2^{min}\}, \\ R_{new}^{max} &= \{(d, x \cap y) \mid (d,x) \in R_1^{max}, (d,y) \in R_2^{max}\}. \end{aligned} \tag{6}$$

   where d is an operation space dimension, and x and y operation space values.
2. **Conjunction**: requirement R_{new} is satisfied iff the operation spaces of at least one requirement are satisfied; put formally:

$$\begin{aligned} R_{new}^{min} &= \{(d, x \cap y) \mid (d,x) \in R_1^{min}, (d,y) \in R_2^{min}\}, \\ R_{new}^{max} &= \{(d, x \cup y) \mid (d,x) \in R_1^{max}, (d,y) \in R_2^{max}\}. \end{aligned} \tag{7}$$

3. **Compromise**: the operation spaces of requirement R_{new} contain dimensions from either R_1 or R_2. When a dimension appears in only one operation space of R_1 or R_2, it is automatically added to the operation space of R_{new}. When a dimension appears in both operation spaces, a coinflip[6] is used; put formally:

$$\begin{aligned} ((d,x) \in R_1^{min} \wedge rnd_{bool} == \top) &\rightarrow (d,x) \in R_{new}^{min} \\ ((d,y) \in R_2^{min} \wedge rnd_{bool} == \bot) &\rightarrow (d,y) \in R_{new}^{min} \end{aligned} \tag{8}$$

   where rnd_{bool} is a random boolean generator that draws from $\{\top, \bot\}$. R_{new}^{max} is determined analogously. For conciseness, we have omitted the equations in which a dimension appears in one of the operation spaces of R_1 or R_2.
4. **Coercion**: one actor overrules the other, e.g., by having more power in a business. Hence, either R_1 or R_2 gets selected via a coinflip; put formally:

$$R_{new} = \begin{cases} R_1 & \text{if } rnd_{bool} == \top, \\ R_2 & \text{if } rnd_{bool} == \bot. \end{cases} \tag{9}$$

The operation spaces of R_1 and R_2 in Eqs. (6) and (7) are assumed to have the same dimensions, which might not always be the case. To compensate for this, $(d, x \cup y)$ is replaced by either (d,x) (or (d,y)) when dimension d is missing in one of the operation spaces in (6) and (7). Also, $(d, x \cap y)$ is replaced by $(d, \emptyset)$ when dimension d is not present in one of the operation spaces in (6) and (7).

[6] All random numbers used have been uniquely generated for this paper using https://www.random.org/.

(iii) Selecting an Owner for the New Requirement. The owner of the new requirement R_{new} is the owner of one of the former requirements (R_1 or R_2), viz., the one that is most similar to the new requirement. Entropy and Jaccard (as introduced in Eqs. 4 and 5) are used here for each iteration, with probabilities $\frac{Os_1}{Os_1+Os_2}$ and $\frac{Os_2}{Os_1+Os_2}$ (cf. (1) and Table 1e), respectively. The owner of R_{new} is R_1 when the distance between R_1 and R_{new} is smaller than the distance between R_2 and R_{new}, and R_2 otherwise.

4.3 Validation of the Requirements

We compare the generated requirements with the orginal formal system requirements (see Table 1c). For this purpose, we compute the relative number of satisfied requirements for each design, as follows:

$$RS_s(d) = \sum_{i=1}^{|R_s|} \mathbb{I}\left(M(d, r_i)\right) \tag{10}$$

where $RS_s(d) \in [0:1]$ is the relative amount of satisfied requirements for design d, $|R_s|$ the number of requirements in collection s, r_i the i^{th} requirement, $\mathbb{I}(\top) = 1$ and $\mathbb{I}(\bot) = 0$. Following this, RS is used to compute the relative number of satisfied designs for the initial system requirements (RS_{init}) and the generated requirements (RS_{gen}). We compare them as follows.

$$V(RS_{init}, RS_{gen}) = \sqrt{\sum_{i=1}^{|D|}(RS_{init}(d_i) \;-\; RS_{gen}(d_i))^2}, \tag{11}$$

where $V \in [0:1]$ is the so-called *validity* degree (lower is better), RS_{init} and RS_{gen} are the relative number of satisfied requirements (of (10)), and d_i is the i^{th} design.

Finally, we define two ways to compare the outcomes of experiments. First, we compare outcomes of experiments on the best average value, as follows.

$$V_{average} = \frac{\sum_{i=1}^{n} V(RS_{init}, RS_{gen})_i}{n} \tag{12}$$

where n is the number of experiments and $V(RS_{init}, RS_{gen})_i$ the outcome for the i^th experiment. Second, we compare outcomes of experiments on the best incidental value as:

$$V_{max} = \min_{i \in \{1..n\}} V(RS_{init}, RS_{gen})_i \tag{13}$$

where n is the number of experiments and $V(RS_{init}, RS_{gen})_i$ is the outcome for the i^{th} experiment.

5 Experimental Results

We validate our approach by executing the case study of Table 1 in `aDSL`; `aDSL` executes the algorithm of Fig. 2 for each experiment of (2). Each experiment is simulated 10 times for more reliable results and contains 12 iterations in which stakeholders negotiate. At the end of each experiment, a validity degree is computed according to (11) to compare experiments. Table 3 shows the experiments that score best on average and incidentally, as follows.

Experiments that score **best on average** (see Table 3a and (12)) yield the lowest average score for V on 10 simulation runs. The six highest ranked experiments rely completely on compromise (ac_3) as the actor negotiation strategy,

Table 3. Experiments that score best

(a) on average.

Rank	**Experiment**								**Average ± 95%CI**
	ac_1	ac_2	ac_3	ac_4	rs_1	rs_2	os_1	os_2	
1	0	0	1	0	1	1	1	1	0.371 ± 0.076
2	0	0	1	0	0	2	2	0	0.374 ± 0.021
3	0	0	1	0	2	0	2	0	0.374 ± 0.021
4	0	0	1	0	1	1	2	0	0.376 ± 0.022
5	0	0	1	0	2	0	1	1	0.396 ± 0.062
6	0	0	1	0	0	2	1	1	0.408 ± 0.104
7	1	0	2	0	2	0	2	0	0.411 ± 0.068
8	1	0	2	0	0	2	2	0	0.416 ± 0.09
9	1	0	1	0	0	2	2	0	0.421 ± 0.055
10	1	0	2	0	1	1	2	0	0.423 ± 0.067

(b) incidentally.

Rank	**Experiment**								**Minimum run**
	ac_1	ac_2	ac_3	ac_4	rs_1	rs_2	os_1	os_2	
1	0	0	2	1	2	0	1	1	0.346
2	2	2	1	0	0	2	1	1	0.349
3	0	0	1	0	1	1	1	1	0.354
4	0	0	1	0	0	2	1	1	0.354
5	0	0	2	1	1	1	1	1	0.354
6	2	0	1	0	0	2	1	1	0.354
7	2	0	2	1	0	2	1	1	0.354
8	1	0	1	1	0	2	1	1	0.354
9	2	0	1	0	2	0	1	1	0.356
10	1	0	2	2	0	2	1	1	0.356

Legend: The relative probability of actor intersection is ac_1, conjunction ac_2, compromise ac_3 and coercion ac_4. The relative probability of requirement selection with Entropy is rs_1 and with Jaccard rs_2. The relative probability of requirement owner selection with Entropy is os_1 and with Jaccard os_2.

whereas the remaining four are a combination of this compromise strategy with the intersection (ac_1) strategy. Presumably, the conjunction (ac_2) strategy generates many invalid requirements by permitting all combinations of the operations spaces of the two old requirements. Moreover, the coercion (ac_4) strategy appears aggressive and is very random by selecting one requirement and neglecting the other requirement completely. Furthermore, we deduce that selecting either the Entropy or Jaccard measure for selecting two requirements or assigning an owner to a new requirement appears to be not significant; experiments with similar actor negotiation strategies score high.

Experiments that score **best incidentally** (see Table 3b and (13)) yield a low score for V on at least one of the 10 simulation runs. Hence, they sometimes perform well but do not need to do this structurally. Again, experiments with an intersection (ac_1, 6 experiments in the list) and compromise (ac_3, all experiments in the list) actor negotiation strategy score high. On top of this, the coercion (ac_4, 5 experiments in the list) is noteworthy. Namely, making random decisions, e.g., as with the coercion strategy, in order to score well on one experiment.

Summarized, `aDSL` has shown to be able to automatically compute the validation scores for all experiments. This enables us to define an actor negotiation strategy and see how it performs. Four negotiation strategies have been tested of which "intersection" and "compromise" are most promising. Besides this, the "coercion" strategy scores well, albeit only incidentally. Using either the Entropy or Jaccard measure does not seems to make much difference.

6 Conclusion

Many cyber-physical systems are safety-critical, which implies that their malfunctioning may cause serious property damage or even injure people. Requirement analysis is one of the key activities in the design of SC-CPSs. Current RA approaches range from informal, hard to automate ones to formal ones that lack freedom of expression; furthermore, most approaches are general-purpose, while in case of a SC-CPS, an approach that is easy to use and tailored to the application domain at hand is called for.

In this paper, we have presented `aDSL`, a DSL and toolset for requirements analysis of safety-critical cyber-physical systems. The approach is a mixture of informal, human-centered and formal approaches to enable both automation and freedom of expression. It comprises an algorithm that automatically requests stakeholders to negotiate about similar requirements and replaces them with new ones, viz., when two requirements are similar, one might be obsolete.

In practice, our approach would require real humans for testing. Instead, we have defined actors that automatically simulate the negotiation behavior of the stakeholders, enabling large-scale and high-speed testing. Consequently, we have been able to conduct many experiments which slightly differ in actor behavior. As somehow expected, the resulting requirements are of better quality when the actors compromise opposed to them displaying coercive behavior. Humans actors can keep these findings in mind, while engaging in real negotiations. Besides this, `aDSL` could be used to develop and test new actor strategies.

References

1. Lee, E.A.: Cyber-physical systems-are computing foundations adequate. In: Position Paper for NSF Workshop On Cyber-Physical Systems: Research Motivation, Techniques and Roadmap, vol. 2 (2006)
2. Rajkumar, R.R., Lee, I., Sha, L., Stankovic, J.: Cyber-physical systems: the next computing revolution. In: Proceedings of the 47th Design Automation Conference, pp. 731–736. ACM (2010)
3. Khaitan, S.K., McCalley, J.D.: Design techniques and applications of cyberphysical systems: a survey. IEEE Syst. J. **9**(2), 350–365 (2015)
4. Banerjee, A., Venkatasubramanian, K.K., Mukherjee, T., Gupta, S.K.: Ensuring safety, security, and sustainability of mission-critical cyber-physical systems. Proc. IEEE **100**(1), 283–299 (2012)
5. Bowen, J.: The ethics of safety-critical systems. Commun. ACM **43**(4), 91–97 (2000)
6. Kotonya, G., Sommerville, I.: Requirements Engineering: Processes and Techniques. Wiley Publishing, Hoboken (1998)
7. Pohl, K.: The three dimensions of requirements engineering. In: Rolland, C., Bodart, F., Cauvet, C. (eds.) CAiSE 1993. LNCS, vol. 685, pp. 275–292. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-56777-1_15
8. Fowler, M.: UML Distilled: A Brief Guide to the Standard Object Modeling Language. Addison-Wesley Professional, Boston (2004)
9. Adolph, S., Cockburn, A., Bramble, P.: Patterns for Effective Use Cases. Addison-Wesley Longman Publishing Co., Inc., Boston (2002)
10. van den Berg, F., Remke, A., Haverkort, B.R.: A domain specific language for performance evaluation of medical imaging systems. In: 5th Workshop on Medical Cyber-Physical Systems, Service OpenAccess Series in Informatics, vol. 36, pp. 80–93. Schloss Dagstuhl (2014)
11. van den Berg, F., Haverkort, B.R., Hooman, J.: iDSL: automated performance evaluation of service-oriented systems. In: Katoen, J.-P., Langerak, R., Rensink, A. (eds.) ModelEd, TestEd, TrustEd. LNCS, vol. 10500, pp. 214–236. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68270-9_11
12. van den Berg, F., Remke, A., Haverkort, B.R.: iDSL: automated performance prediction and analysis of medical imaging systems. In: Beltrán, M., Knottenbelt, W., Bradley, J. (eds.) EPEW 2015. LNCS, vol. 9272, pp. 227–242. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23267-6_15
13. van den Berg, F., Garousi, V., Tekinerdogan, B., Haverkort, B.R.: Designing cyber-physical systems with aDSL: a domain-specific language and tool support. In: 13th System of Systems Engineering Conference. IEEE (2018)
14. Gray, R.: Entropy and Information Theory. Springer, Heidelberg (2011). https://doi.org/10.1007/978-1-4419-7970-4
15. Niwattanakul, S., Singthongchai, J., Naenudorn, E., Wanapu, S.: Using of Jaccard coefficient for keywords similarity. In: Proceedings of the International MultiConference of Engineers and Computer Scientists, vol. 1, no. 6 (2013)