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

Chapter · March 2020

DOI: 10.1007/978-3-030-43024-5_15

2 authors, including:

Freek van den Berg
University of Twente
11 PUBLICATIONS **76** CITATIONS

SEE PROFILE

A Domain Specific Language & Toolset 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, 5037 AB Tilburg, The Netherlands
B.R.H.M.Haverkort@tilburguniversity.edu

Abstract. Cyber-physical systems (CPSs) are widespread in various domains, e.g., aerospace, civil infrastructure, transportation, automotive, healthcare and agriculture. Many CPSs are safety-critical, which means their malfunctioning may cause serious property damage or even injure or kill people. One of the key challenges in the design of a safety-critical CPS is requirements analysis (RA). Current RA approaches range from informal, human-centered ones that are hard to automate, to formal, technical ones that 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 RA of safety-critical CPSs. 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, after which the aDSL toolset invites stakeholders to negotiate and merge their 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

1 Introduction

A Cyber-Physical System (CPS) is a real-time feedback system that is controlled and monitored by computer-based algorithms. It integrates [1] embedded systems, human users, networks [2], and concurrent physical systems [3, 4]. Designing and constructing a CPS requires expertise from many disciplines, including

signal processing [2], control engineering [2], computer engineering, software engineering and programming, electronics, and mechanical engineering [2]. CPSs are common in safety-critical. domains, e.g., aerospace, civil infrastructure, automotive, energy and agriculture. A Safety-Critical CPS (SC-CPS, [5]) is defined as a CPS that is designed to and needs to be fail-safe for safety purposes [6]. Malfunction of a SC-CPS may lead to: (i) death or serious injury to people; (ii) loss or damage to equipment or property; or, (iii) environmental harm. Hence, software 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 (RE) process.

RE is mainly concerned with defining the requirements [7] 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 RE activities. Literature proposes a 3D-model [8] 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 RE, i.e., requirement analysis (RA). Current RA processes range from informal, human-centered to formal, technical approaches. On one hand, informal approaches, often driven by Unified Modelling Langage (UML, [9]), tend to include natural language-based documents, e.g., use cases [10], user stories, and process specifications. Using natural language provides a great freedom of expression, but is hard to formalize when automating parts of the RA process. On the other hand, formal approaches aim to deliver artefacts that are unambiguous and ready for automatic processing. However, the 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. The advantage of this is that the RA approach can be used for a broad set of domains. However, due to the 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 [5]. A Domain Specific Language (DSL) enables such a dedicated approach. A DSL is a computer language specialized to a particular application domain. 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[3], and Xtext, an open-source software framework for developing DSLs[4]. For instance, iDSL [11–13], on which `aDSL` is inspired, has been constructed using Xtext and provides a DSL 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. The `aDSL` language 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 are evaluated using a case study on agro-machines [14] for validation.

This paper's remainder is organized as follows. Section 2 contains the problem statement and case study. Section 3 introduces a generic algorithm for RA of SC-CPS. Section 4 presents a formal model for RA of CPS. Section 5 provides an implementation for the generic algorithm of Section 3. In Section 6, the algorithm of Section 5 is validated. Section 7 concludes the paper.

2 Case study & Problem Statement

We use a case study inspired by previous work [14] in this paper. The original 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 the same. However, we introduce the following three additions:

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.
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 Section 5).
3. a measure for RA has been added to `aDSL` to configure an experiment. 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.

[3] JetBrains MPS `http://www.jetbrains.com`
[4] Xtext - Language Engineering Made East `http://eclipse.org`

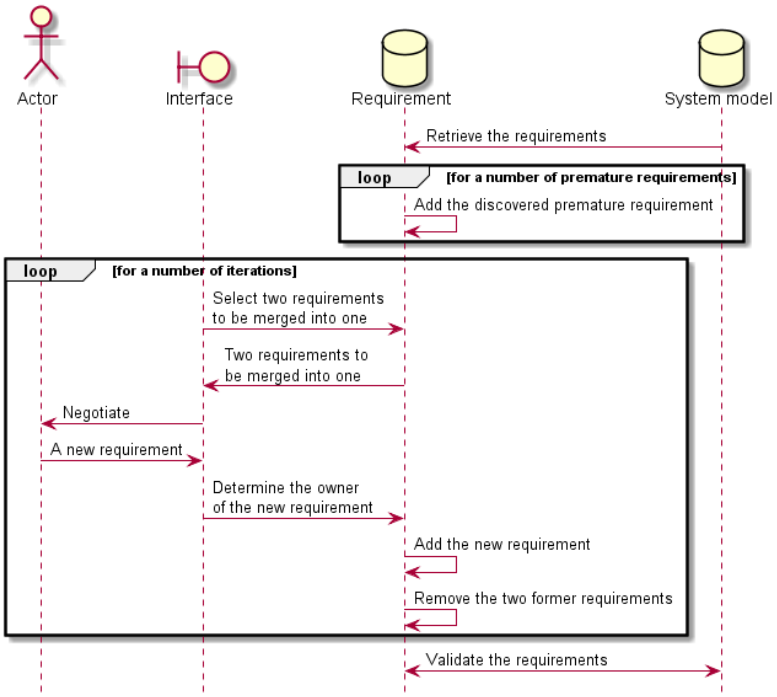

Fig. 1: The sequence diagram of the RA pseudo-algorithm.

3 An generic algorithm for RA of SC-CPS

This section introduces an algorithm for RA of CPS. The UML sequence diagram[5] of Figure 1 displays the subsequent steps and iterations of the algorithm, will be implemented in Section 5. It can be observed that Figure 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 (of Section 1, RE step (i)). This model will be implemented in Section 4.
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 (of Section 1, RE step (ii)).
3. Requirements analysis (of Section 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

[5] Figure 1 and 2 have been constructed using PlantUML: `http://plantuml.com`

owner and both original requirements are removed. By using a formal language, we consider requirements specification (of Section 1, RE step (iv)) to be taken care of automatically.
4. Requirement validation (of Section 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, as follows.

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 $r \in R$ has exactly one owner $o \in O$. Hence, $Ow : R \to O$ is defined, where Ow 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 $Rt : T \times R$ is a requirement $r \in R$ 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: $\{(0, pr) \mid pr \in Pr\}$
8. A transaction $Tr : T \times (Rt)^2 \to Rt$ at time $t \in T$ involves two timed, input requirements (t, r_1), $(t, r_2) \in Rt$ and yields a new timed requirement $(t + 1, r') \in Rt$.
9. The owner of a requirement resulting from a transaction with two timed requirements (t, r_1) and (t, r_2) is either $Ow(r_1)$ or $Ow(r_2)$; put formally: $Tr = (t, r_1, r_2, r')$ implies that either $Ow(r') = Ow(r_1)$ or $Ow(r') = Ow(r_2)$.

4 A formal model for RA of Cyber-Physical Systems

In this section, we present the `aDSL` model of a CPS. It extends previous work [14] with: (i) timed requirements; (ii) a measure for requirements analysis; and, (iii) experiments to conveniently define many measures. At its highest level of hierarchy, an `aDSL` instance comprises five so-called sections[6], as follows.

"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 (as explained later). At a lower level, a system comprises one or more so-called SystemOrParts, i.e., a **system**, **part**, **systemID** or **partID**.

[6] 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 } )
```

(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])
```

Table 1: The aDSL instance of the case study

(d) 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} )
```

(e) Design space

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

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 01 is at most $O2$, 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:

$$O1 \leq O2 \rightarrow (((d:v) \in O1 \rightarrow \nexists x (d:x) \in O2) \vee ((d:x) \in O1 \rightarrow (d:x) \in O2))) \tag{1}$$

Hence, only common dimensions are considered for comparison. Note that previous work [14] 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 [14]. They transform into premature requirements as part of the implementation to be introduced in Section 5.

"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 $n \in \mathbb{N}^+$ dimensions that each have a number of values. The n^{ary} Cartesian product of these dimensions form the set of experiment instances. we us a so-called espace construct with twelve parameters in order to vary all parameters of the experiment.

Case study. Table 1d 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.

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

$$ac_1, ac_2, ac_3.ac_4, rs_1, rs_2, os_1, os_2 \in \{0, 1, 2\}\} \quad \text{\#the 10 parameters are 0, 1 or 2.} \tag{2}$$

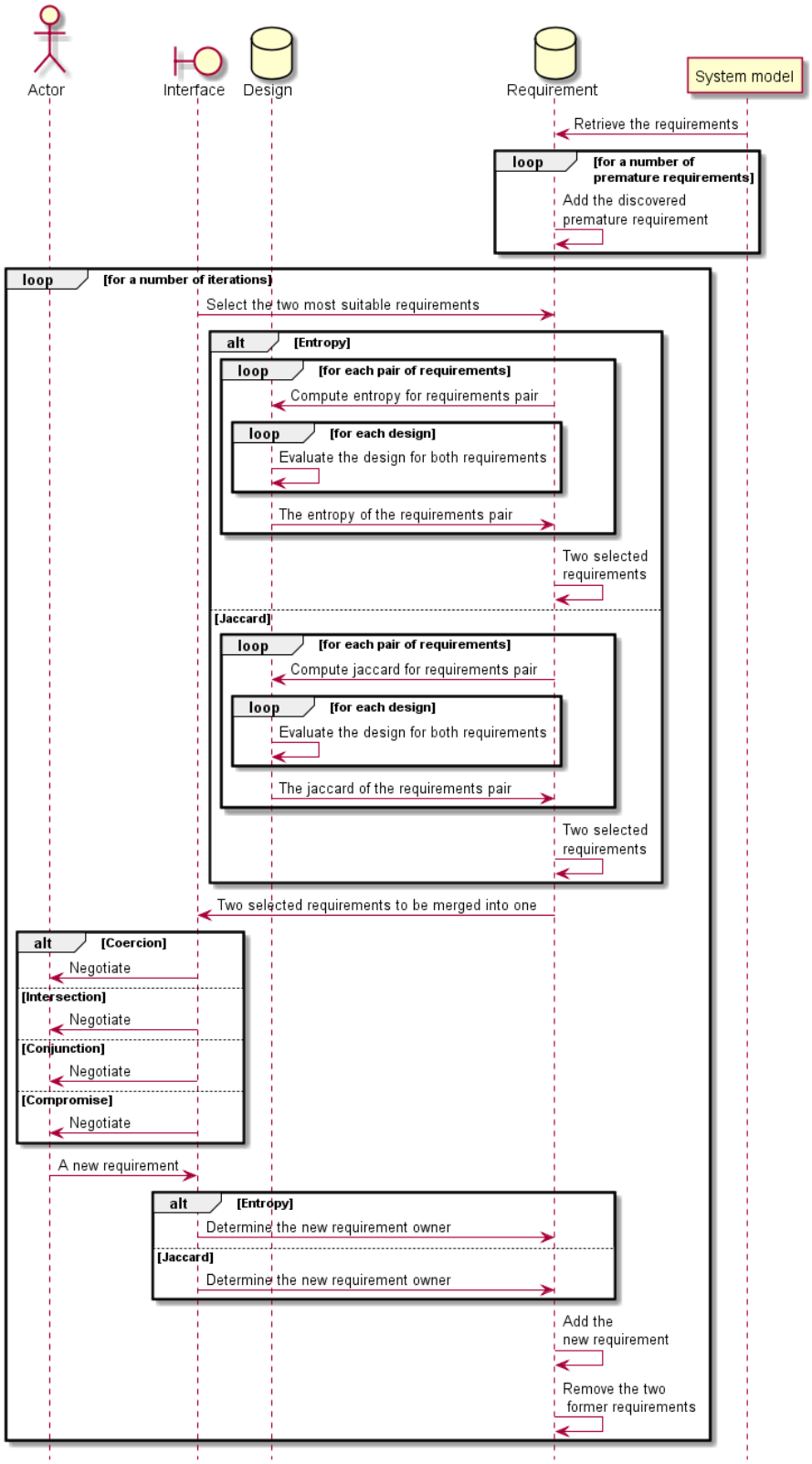

Fig. 2: The sequence diagram of the RA implemented-algorithm.

"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 1e 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.

5 An implemented algorithm for RA of SC-CPS

In Section 3, a generic algorithm was presented for RA of CPSs (as depicted in Figure 1). In this section, we implement this algorithm to make it executable[7]. To this end, the following individual parts are implemented as graphically depicted in Figure 2: (i) capturing the premature requirements (cf. Section 5.1); (ii) evolution of the requirements (cf. Section 5.2); and, validation of the requirements (cf. Section **??**). For precision, step (ii) is executed for 10 simulation runs as defined in Section Measure (see Table 1d).

5.1 Capturing the premature requirements

Capturing the requirements corresponds to system elicitation (see Section 1, RE step (ii)). In practice, this is a non-trivial, labor-intensive step which involves interviews, questionnaires, user observations and workshops, etc. Therefore, we simulate it using a reverse engineering step instead, as follows. We split the formal requirements (cf. Table 1c) of the system model of Section 4 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 Figure 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 $pr \in R$ 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:

$$R = \{(o'_{min}, o'_{max}) | o'_{min} = o_{min} \backslash e_i, e_i \in o_{min}, o'_{max} = o_{max} \backslash e_j, e_j \in o_{max}\} \quad (3)$$

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

Case study. For illustration, 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,

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

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})

third column) operation spaces. Hence, formal requirement RA2 yields $4 \times 4 = 16$ premature requirements, where Ri,j is the requirement with the i^{th} and j^{th} element omitted and $i, j \in \{1, 2, 3, 4\}$ (cf. Table 2a and 2b, first column). For illustration, the first (R1,1) and last premature requirement (R4,4) are:

- **Requirement** R1,1 **minimum OperationSpace** (fuelConsumption [4 7] price { medium } gears [4 24]) **maximum OperationSpace** (fuelConsumption [0 15] driverSkills {easy} pollution [4 8])
- **Requirement** R4,4 **minimum OperationSpace** (speed [7 12] fuelConsumption [4 7] price { gears } **maximum OperationSpace** (speed [0 35] fuelConsumption [0 15] driverSkills {easy})

5.2 Evolution of the requirements

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

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, viz., Entropy and Jaccard. Each iteration, Entropy is selected with probability $\frac{Rs_1}{Rs_1+Rs_2}$ and Jaccard with $\frac{Rs_2}{Rs_1+Rs_2}$ (cf. Equation 2 and Table 1d). 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; put formally:

$$k_{i,j} = \sum_{n=1}^{|D|} \mathbb{I}\{ \ M(d_n, R_i) = i, M(d_n, R_j) = j\} \tag{4}$$

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

$$E(r_1, r_2) = -\sum_{i=0}^{1}\sum_{j=0}^{1} l_{i,j} \cdot \log(l_{i,j}) \tag{5}$$

   where we assume that $0 \cdot \log(0) = 0$, since $\lim_{n\to 0}(n) \cdot log(n) = 0$.
2. Jaccard [16]: originally used for the similarity of two sets. A higher outcome corresponds to a higher similarity. It is defined as follows.

$$J(r_1, r_2) = \frac{k_{1,1}}{k_{1,1} + k_{0,1} + k_{1,0}} \tag{6}$$

Stakeholders negotiate about the requirements they own. Two stakeholders iteratively 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, i.e., human stakeholders. Here, to be able to conduct many experiments (see Section 6 for the results), we automatically simulate human behavior 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. Equation 4 and Table 1d), 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:

$$R_{new}^{min} = \{(d, x \cup y) \mid (d, x) \in R_1^{min}, (d, y) \in R_2^{min}\}, \tag{7}$$

$$R_{new}^{max} = \{(d, x \cap y) \mid (d, x) \in R_1^{max}, (d, y) \in R_2^{max}\}. \tag{8}$$

   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:

$$R_{new}^{min} = \{(d, x \cap y) \mid (d, x) \in R_1^{min}, (d, y) \in R_2^{min}\}, \tag{9}$$

$$R_{new}^{max} = \{(d, x \cup y) \mid (d, x) \in R_1^{max}, (d, y) \in R_2^{max}\}. \tag{10}$$

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[8] is used; put formally:

$$((d, x) \in R_1^{min} \wedge rnd_{bool} == \top) \rightarrow (d, x) \in R_{new}^{min} \tag{11}$$

$$((d, y) \in R_2^{min} \wedge rnd_{bool} == \bot) \rightarrow (d, y) \in R_{new}^{min} \tag{12}$$

   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:

$$R_{new} = \begin{cases} R_1 & \text{if } rnd_{bool} == \top, \\ R_2 & \text{if } rnd_{bool} == \bot, \end{cases} \tag{13}$$

The operation spaces of R_1 and R_2 in Equation 7-10 are assumed to have the same dimensions. This is not always 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 Equation 7 and 10. Also, $(d, x \cap y)$ is replaced by $(d, \emptyset)$ when dimension d is not present in one of the operation spaces in Equation 8 and 9.

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 Equation 5 and 6) are used here for each iteration, with probabilities $\frac{Os_1}{Os_1+Os_2}$ and $\frac{Os_2}{Os_1+Os_2}$ (cf. Equation 1 and Table 1d), 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.

5.3 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.

$$RS_s(d) = \sum_{i=1}^{|R_s|} I\left(M(d, r_i)\right) \tag{14}$$

where $RS_s \in [0:1]$ 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 $\mathbb{I}(\bot) = 0$. After 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.

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

$$V(RS_{init}, RS_{gen}) = \sqrt{\sum_{i=1}^{|D|}(RS_{init}(d_i) \ - \ RS_{gen}(d_i))^2} \tag{15}$$

where $V \in [0:1]$ is validity degree (lower is better), RS_{init} and RS_{gen} the relative number of satisfied requirements (of Equation 14), and d_i the i^{th} design.

6 Validation via experiment results

We validate the approach by executing the case study of Table 1 in `aDSL`; `aDSL` executes the algorithm of Figure 2 for each experiment of Equation 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 used Equation 15 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) 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, 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 seems aggressive and depends on chance much 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) yield a low score for V on at least one of the simulation runs. Again, experiments with an intersection (ac_1, 6 experiments) and compromise (ac_3, all experiments) actor negotiation strategy score high. Contrarily, the coercion (ac_4, 5 experiments) is noteworthy. Namely, in order to score well on one experiment, making random decisions, e.g., as with the coercion strategy, is statistically bound to yield good results.

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 incidentally. Using either the Entropy or Jaccard measure does not seems to make much difference.

7 Conclusion

Many CPSs are safety-critical, which means their malfunctioning may cause serious property damage or even injure people. RA is one of the key activities in the design of a SC-CPS. Current RA approaches range from informal, hard to automate ones to formal ones that lack freedom of expression. Also, RA approaches are generally general-purpose, while in case of a SC-CPS an approach that is easy to use and tailored to the SC-CPS domain is called for.

Table 3: Experiments that score best
(a) on average.

Rank	**Experiment**										**Average ± 95%CI**
	it	sim	ac_1	ac_2	ac_3	ac_4	rs_1	rs_2	os_1	os_2	
1	12	10	0	0	1	0	1	1	1	1	0.371 ± 0.076
2	12	10	0	0	1	0	0	2	2	0	0.374 ± 0.021
3	12	10	0	0	1	0	2	0	2	0	0.374 ± 0.021
4	12	10	0	0	1	0	1	1	2	0	0.376 ± 0.022
5	12	10	0	0	1	0	2	0	1	1	0.396 ± 0.062
6	12	10	0	0	1	0	0	2	1	1	0.408 ± 0.104
7	12	10	1	0	2	0	2	0	2	0	0.411 ± 0.068
8	12	10	1	0	2	0	0	2	2	0	0.416 ± 0.09
9	12	10	1	0	1	0	0	2	2	0	0.421 ± 0.055
10	12	10	1	0	2	0	1	1	2	0	0.423 ± 0.067

(b) incidentally.

Rank	**Experiment**										**Minimum run**
	it	sim	ac_1	ac_2	ac_3	ac_4	rs_1	rs_2	os_1	os_2	
1	12	10	0	0	2	1	2	0	1	1	0.346
2	12	10	2	2	1	0	0	2	1	1	0.349
3	12	10	0	0	1	0	1	1	1	1	0.354
4	12	10	0	0	1	0	0	2	1	1	0.354
5	12	10	0	0	2	1	1	1	1	1	0.354
6	12	10	2	0	1	0	0	2	1	1	0.354
7	12	10	2	0	2	1	0	2	1	1	0.354
8	12	10	1	0	1	1	0	2	1	1	0.354
9	12	10	2	0	1	0	2	0	1	1	0.356
10	12	10	1	0	2	2	0	2	1	1	0.356

Legend: it is the number of iterations, sim the number of simulation runs. 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.

This paper proposes `aDSL`, a DSL and toolset for RA of SC-CPS. The approach is a mixture of informal, human-center 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 replace them with one new one, 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 anticipated, 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.

Summarized, we have created an RA approach based on a DSL for SC-CPS that is automated to a great extend but also leaves room for human creativity.

References

1. E. A. Lee, "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. R. R. Rajkumar, I. Lee, L. Sha, and J. Stankovic, "Cyber-physical systems: the next computing revolution," in *Proceedings of the 47th Design Automation Conference*. ACM, 2010, pp. 731–736.
3. E. A. Lee, "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.
4. S. K. Khaitan and J. D. McCalley, "Design techniques and applications of cyber-physical systems: A survey," *IEEE Systems Journal*, vol. 9, no. 2, pp. 350–365, 2015.
5. A. Banerjee, K. K. Venkatasubramanian, T. Mukherjee, and S. K. Gupta, "Ensuring safety, security, and sustainability of mission-critical cyber-physical systems," *Proceedings of the IEEE*, vol. 100, no. 1, pp. 283–299, 2012.
6. J. Bowen, "The ethics of safety-critical systems," *Communications of the ACM*, vol. 43, no. 4, pp. 91–97, 2000.
7. G. Kotonya and I. Sommerville, *Requirements engineering: processes and techniques*. Wiley Publishing, 1998.
8. K. Pohl, "The three dimensions of requirements engineering," in *International Conference on Advanced Information Systems Engineering*. Springer, 1993, pp. 275–292.
9. M. Fowler, *UML distilled: a brief guide to the standard object modeling language*. Addison-Wesley Professional, 2004.
10. S. Adolph, A. Cockburn, and P. Bramble, *Patterns for effective use cases*. Addison-Wesley Longman Publishing Co., Inc., 2002.
11. F. van den Berg, A. Remke, and B.R. Haverkort, "A Domain Specific Language for Performance Evaluation of Medical Imaging Systems," in *5th Workshop on Medical Cyber-Physical Systems*, ser. OpenAccess Series in Informatics, vol. 36. Schloss Dagstuhl, 2014, pp. 80–93.
12. F. van den Berg, B. R. Haverkort, and J. Hooman, "iDSL: Automated Performance Evaluation of Service-Oriented Systems," in *ModelEd, TestEd, TrustEd*, ser. Lecture Notes in Computer Science. Springer, 2017, vol. 10500, pp. 214–236.
13. F. van den Berg and A. Remke and B.R Haverkort, "iDSL: Automated Performance Prediction and Analysis of Medical Imaging Systems," in *Computer Performance Engineering*. Springer, 2015, vol. 9272, pp. 227–242.
14. F. van den Berg, V. Garousi, B. Tekinerdogan, and B. R. Haverkort., "Designing Cyber-Physical Systems with aDSL: a Domain-Specific Language and Tool Support," in *13th System of Systems Engineering Conference*. IEEE, 2018.
15. R. Gray, *Entropy and information theory*. Springer Science & Business Media, 2011.
16. S. Niwattanakul, J. Singthongchai, E. Naenudorn, and S. Wanapu, "Using of Jaccard coefficient for keywords similarity," in *Proceedings of the International MultiConference of Engineers and Computer Scientists*, vol. 1, no. 6, 2013.