Designing cyber-physical systems with aDSL: A domain-specific language and tool support

Van Den Berg, F., Garousi, V., Tekinerdogan, B., & Haverkort, B. R. (2018). Designing cyber-physical systems with aDSL: A domain-specific language and tool support. In *2018 13th System of Systems Engineering Conference, SoSE 2018* (pp. 225-232). Article 8428770 Institute of Electrical and Electronics Engineers Inc.. https://doi.org/10.1109/SYSOSE.2018.8428770

Published in:
2018 13th System of Systems Engineering Conference, SoSE 2018

Document Version:
Peer reviewed version

Queen's University Belfast - Research Portal:
Link to publication record in Queen's University Belfast Research Portal

Publisher rights
Copyright 2019 IEEE. This work is made available online in accordance with the publisher’s policies. Please refer to any applicable terms of use of the publisher.

General rights
Copyright for the publications made accessible via the Queen's University Belfast Research Portal is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The Research Portal is Queen's institutional repository that provides access to Queen's research output. Every effort has been made to ensure that content in the Research Portal does not infringe any person's rights, or applicable UK laws. If you discover content in the Research Portal that you believe breaches copyright or violates any law, please contact openaccess@qub.ac.uk.

Open Access
This research has been made openly available by Queen's academics and its Open Research team. We would love to hear how access to this research benefits you. – Share your feedback with us: http://go.qub.ac.uk/oa-feedback

Download date:08. Nov. 2024

This is a pre-print of the paper that has been published by IEEE: https://doi.org/10.1109/SYSOSE.2018.8428770

Designing Cyber-Physical Systems with aDSL: a Domain-Specific Language and Tool Support

Freek van den Berg
University of Twente
Enschede, The Netherlands
f.g.b.vandenberg@utwente.nl

Vahid Garousi
and Bedir Tekinerdogan
Wageningen University
Wageningen, The Netherlands
{vahid.garousi, bedir.tekinerdogan}@wur.nl

Boudewijn R. Haverkort
University of Twente
Enschede, The Netherlands
b.r.h.m.Haverkort@utwente.nl

***Abstract*—A Cyber-Physical System (CPS) comprises the integration of computation, software, networking, and physical processes. Consequently, CPS models extend traditional embedded system models with an increased support for hybrid and heterogeneous models, networking, time synchronization, and especially interoperability. To assist engineers in designing CPSs, we have developed aDSL, a Domain-Specific Language (DSL) that comes with fully-automated tool support and is tailored to interoperability of CPS. The aDSL tool support includes: (i) interactive model description with input validation; (ii) the computation of possible operation modes of subsystems and parts; and, (iii) checking the adherence to requirements for various design alternatives and finding the Pareto optimal designs given these requirements. Moreover, aDSL generates intuitive visualizations throughout the toolchain which help design engineers to better understand the implications of design decisions and communicate them to stakeholders. aDSL has been applied to an agricultural tractor-trailer system case study in which aDSL quickly evaluated 48 designs and rendered all the visualizations of the results.**

I. Introduction

An embedded system is a computer system that has a dedicated function within a larger system [1], [2]. Many of the commonly used devices nowadays are embedded systems [3]. They range from simple digital watches to complex systems, such as medical machines [4], [5]. An embedded system is often used to perform safety critical tasks, which makes its malfunctioning prone to serious injury and fatalities, such as with medical systems [4], [5].

Multiple approaches exist that include safety and performance evaluation of an embedded system as an integral part of the design process. For instance, a Domain Specific Language (DSL) for collision prevention of medical systems [6], [7] that automatically transforms into executable code for functionality [7], Satisfiability Modulo Theories (SMT) to tackle safety issues [8], [9], and Parallel Object-Oriented Specification Language (POOSL) to simulate performance [10].

For another instance, a DSL is used to model service-oriented systems [11]–[13]. After this, the model is automatically transformed into timed automata (TA) to compute absolute latency bounds [11], [14] and probabilistic timed automata (PTA) to compute latency distributions [15], [16] for each service. Moreover, visualizations are automatically derived using Graphviz [17] and GNUplot [18].

Both these approaches take the impact of each design decision immediately into account, which dramatically reduces unexpected issues that are hard and costly to fix, especially the ones detected late. Besides this, both approaches yield a DSL that leads to: (i) solutions being expressed in the idiom and abstraction level of the problem owner and relevant stakeholders; (ii) validation at the domain level; (iii) new insights to the system developers; and, (iv) transformations into new languages for evaluation.

In this paper, we propose a similar approach for cyber-physical systems (CPSs). However, A CPS comprises the integration of computation, software, networking, and physical processes [19], [20]. This calls for a System-of-Systems (SoSs, [21])) approach in which many embedded systems and networks monitor and control the physical processes. Consequently, a CPS model extends an embedded system model with increased support for hybrid and heterogeneous models [22], networking, time synchronization [20], and interoperability [23]. Furthermore, well-designed abstractions and architectures are crucial for composability [24], [25] and the OS/programming language [26] split needs to be removed for increased flexibility and efficiency at the sensor and actuator level. The current approaches for embedded systems are not adequate for evaluating the safety and performance of a CPS.

In this paper, an approach is introduced which encompasses a DSL named aDSL, which in particular addresses the interoperability of CPSs. aDSL comes with fully automated tool support. The language supports: (i) a system comprising subsystems and parts; (ii) operation spaces of subsystems and parts; (iii) a design space that enables the investigation of many design alternatives; and, (iv) requirements which can be used to represent safety and performance properties. Additionally, tool support includes: (i) interactive input with input validation; (ii) the computation of possible operation modes of subsystems and parts; (iii) checking the adherence to requirements for different designs and finding optimal designs; and, (iv) visualizations that enable the system designer to understand and communicate the results.

This paper is further organized as follows. Section II provides literature on interoperability and CPS. Section III

Freek van den Berg was supported by a 4TU.NIRICT Research Community Building project, Safe & Secure Machine Cooperation in Agri & Food.

presents the gaps to be bridged. Section IV introduces a formal model for the interoperability of CPSs. Section V conveys tool support that specifies the semantics of the formal model. Section VI concludes the paper. Finally, the appendix contains the grammar of the model.

II. Background and related work

A. CPS

A CPS is a feedback system that is adaptive, intelligent, real-time, and distributed; it is controlled or monitored by computer-based algorithms. A CPS integrates [19], [20] embedded systems, human users, networks [27], and physical systems that are inherently concurrent [28], [29]. Hence, a CPS can be depicted as SoS.

CPSs are often driven by: (i) wireless communication devices; (ii) abundant internet bandwidth; (iii) increased energy availability; (iv) low-cost, low-power, high-capacity and small form computing devices; and, (v) small, low-cost and increased capability sensors [27].

Specifying, modeling and analyzing a CPS requires a hybrid and heterogeneous model that incorporates networking, interoperability, performance and dependability. On top of that, expertise from many disciplines is needed, such as signal processing [27], design, process science, control engineering [27], computer engineering, electronics, telecommunications engineering, systems engineering, mechanical engineering [27] and cybernetics. CPSs are widespread [29], [30]; they can be found, among others, in the field of aerospace, civil infrastructure, chemical processes, transportation, automotive, healthcare, energy, manufacturing, and agriculture.

A CPS shares commonalities with the following three concepts. First, CPS and Internet of Things share similar architectures, but the former provides more combination and coordination between physical and computational elements on top of this. Second, a CPS puts emphasis on the link between the computational and physical elements, whereas an embedded system tends to focus on the computational elements [19]. Third, Industry 4.0 includes CPSs but moreover incorporates aspects of cloud and cognitive computing [31], [32].

B. Interoperability

Interoperability is concerned with systems and parts working together. Interoperability can be found in software [23], telecommunications [33], organization, and public safety [34]. Interoperability may involve exchanges between: (i) a range of products; (ii) similar products from different vendors; and, (iii) post and future revision of the same product.

Interoperability either relies on an open standard or is brought about in a post-facto way. An open standard is achieved via an implementation, representation, release and foundation, e.g., the International System for Agricultural Science and Technology [35]. In the latter case, one vendor sets the standard and tends to make it hard for competitors to adopt the standard. Nevertheless, post-facto interoperability can be useful to integrate components that are written in incompatible languages and use distinct protocols [36].

There are different measures for the degree of interoperability of which we mention three. First, the interoperability ladder [37] distinguishes technical, syntactic, semantic and conceptual interoperability. Similarly, the ATHENA integration framework [38] addresses the levels data, service, processes and enterprise. Finally, LCIM extended [39] covers conceptual interoperability with levels none, technical, syntactic, semantic, pragmatic, dynamic, and conceptual.

C. CPS and interoperability

CPS and interoperability are closely related concepts; the heterogeneous, SoSs structure of a CPS calls for mechanisms that allow its (sub)systems and parts to operate together. Hence, a mix of technologies and tools is needed. We mention a number of approaches that address the interoperability of CPSs, as follows.

In various approaches, a service-oriented architecture [32], [40]–[43] is used to: (i) enable networking and communication via open standards; (ii) provide flexibility; and, (iii) construct a SoSs architecture.

Other approaches emphasize modeling a CPS: In [44], three abstraction levels are suggested, viz., logical representation, virtual interoperability representation, and a CPS system. iCyPhy [45] advocates a model-based approach to generate system modeling, design and analysis tools and methodologies. Finally, Modelica [46] is a language that is tailored to modeling and simulating complex CPSs.

III. Gap analysis

In Section I, we have addressed two DSL-based approaches [7], [11] for evaluating the safety and performance of embedded systems, which are not adequate for CPSs. Section II discussed existing approaches that address the safety and performance of CPSs, which take advantage of service-oriented architectures and model-based approaches, respectively. However, they lack some advantages DSL-approaches provide, e.g., reasoning at the level of the problem and transformations into multiple languages for evaluation. In the following, we address three categories of gaps to bridge that enable a DSL-approach for CPSs, viz., modeling a CPS, modeling requirements of a CPS, and Design Space Exploration (DSE, [47]–[49]).

A. Modeling a CPS

A system designer models a CPS. This call for a language:

$\mathcal{G}1.1$ that is tailored to modeling CPSs,

$\mathcal{G}1.2$ that is formal and expressive enough to support unambiguous definition and reasoning,

$\mathcal{G}1.3$ that is concise and as simple as possible to enable effective communication with stakeholders,

$\mathcal{G}1.4$ that is easy to understand by the system designer, i.e., it uses the system designer's domain, concepts,

$\mathcal{G}1.5$ that can address interoperability; connecting systems happens on the basis of on shared operation modes.

B. Modeling requirements of a CPS

In this paper, we focus on the requirements that are related to interoperability (see $\mathcal{G}1.5$), which, however, can indirectly be concerned with safety and performance. A requirement:

$\mathcal{G}2.1$ constraints the allowed operation modes,
$\mathcal{G}2.2$ belongs to a distinct category, which corresponds to an owner to the requirement.

C. Design space exploration

DSE concerns finding an optimal design among many possible designs. This calls for an approach that:

$\mathcal{G}3.1$ automatically evaluates and compares many designs,
$\mathcal{G}3.2$ takes full advantage of the system designer's domain expertise by presenting its results comprehensively.

IV. Formal model for CPS interoperability

Next, we attempt to formally bridge the gaps of Section III. For this purpose, we have constructed a DSL named aDSL using Eclipse for DSLs [50]. This section presents the model. The appendix conveys the corresponding grammar (see Table II), whereas Table I contains an illustrative case study.

At its highest level of hierarchy, an aDSL instance comprises four concepts (see Table IIa), as follows. First, an aDSL instance comprises one **top-level system**, i.e, the CPS under study, and zero or more subsystems. Moreover, a system has at least one **subsystem** or **part** that constitute that system.

Second, an aDSL instance encompasses zero or more parts. On top of that, a system and a part have an **operation space** (to be discussed later).

Third, one or more **requirements** limit the operation space.

Fourth, a **design space** consists of a number of dimensions which in turn comprise a number of values. Next, we discuss the aforementioned concepts in more detail.

A. A formal model for a system

As mentioned, an aDSL instance comprises one top-level system, i.e., the actual CPS under study, followed by zero or more subsystems. A system has a name, an operation space (to be explained in Section IV-C), and one or more so-called SystemOrParts. A SystemOrPart is either a recursively defined system, i.e., System, or a part, i.e., Part (to be explained in Section IV-B), or a reference to a system, i.e., AbstractSystem, or a reference to a part i.e., AbstractPart, or a number of design alternatives, i.e., DesAlt (to be explained in Section IV-E). The corresponding grammar can be found in Table IIb.

Case study: Section "system" (in Table Ia) conveys that the Top-level system *TractorTrailorCombination* comprises an AbstractSystem *Tractor* that refers to *tractor* and a DesAlt *trailor*. In turn, System *tractor* encompasses an AbstractSystem *Transmission* referring to *trans* and an AbstractSystem *Fuel* referring to *fuel*. Finally, system *trans* comprises DesAlt *transmission*, whereas System *fuel* contains DesAlt *engineFuel*.

TABLE I: The code of the case study

(a) "System"

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

 System tractor OperationSpace () {
   AbstractSystem Transmission trans
   AbstractSystem Fuel fuel }

System trans OperationSpace() { DesAlt (transmission){
  unsynchronized Part transUnsynchronized
      OperationSpace ( driverSkills {advanced moderate
      easy} continousOperation { no } gears [ 1 24 ] )
  doubleClutch Part transDoubleClutch
      OperationSpace ( driverSkills { moderate easy }
      gears[ 1 2 4])
  CVT Part transCVT
      OperationSpace ( driverSkills {easy}
      efficiency { frictionLoss } gears [ 1 1000 ] )}}

System fuel OperationSpace () { DesAlt (engineFuel){
  steam Part fuelSteam
   OperationSpace (pollution [5 10] speed [0 30] )
  diesel Part fuelDiesel
   OperationSpace (pollution [4 8] fuelConsumption [3 5]
    speed [0 40] price { medium high } )
  gasoline Part fuelGasoline
   OperationSpace(pollution [2 5] fuelConsumption [4 10]
    speed [0 50] price { medium high })
  electric Part fuelElectric
   OperationSpace(pollution [0 4] fuelConsumption [8 12]
    speed [0 55] price { high })}}}
```

(b) "Part"

```
Section part
 Part TrailorChiselPlow
   OperationSpace ( speed [0 25] agility { low veryLow }
   activity { plow } )
 Part TrailorTiller
   OperationSpace ( speed [0 25] agility { low veryLow }
   activity { till } )
 Part ChaserBin
   OperationSpace ( speed [0 15] agility { veryLow }
   activity { harvest } )
```

(c) "Requirements"

```
Section requirements
 Legal Requirement speedRange
   minimum OperationSpace ( speed [5 15] )
   maximum OperationSpace ( speed [0 45] )
 Business Requirement fuelAndPurchaseCosts
   maximum OperationSpace
      ( fuelConsumption [0 20] price {medium})
 Design Requirement operability
   maximum OperationSpace ( driverSkills easy )
```

(d) "Design space"

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

B. A formal model for a part

A part has a name and operation space (to be explained in Section IV-C) like a system, but lacks SystemOrParts. Hence, a part can be depicted as an elementary building block out of which systems are built. This is illustrated by the grammar in Table IIb.

Case study: Section "system" (of Table Ia) contains seven parts: *transUnsynchronized*, *transDoubleClutch*, *transCVT*,*fuelSteam*, *fuelDiesel*, *fuelGasoline*, and *fuelElectric*, which each have their own operation space. There are also three abstract parts, viz., *trailorChiselPlow*, *trailorTiller* and *chaserBin*, which are implemented in Section "part" (see Table Ib) as *TrailorChiselPlow*, *TrailorIiller* and *Chaserbin*, respectively.

C. A formal model for an operation space

An operation space represents a number of operational modes. For this purpose, we represent an operation space as a number of operation dimensions. A dimension contains either a finite number of elements, i.e., OperationDimensionValue, or is a real number range bounded by a minimum and maximum value, i.e., OperationDimensionInterval. A system and a part have an operating space to specify the operation modes they support. In contrast, a requirement uses operation spaces to specify a minimum and maximum bound on operation modes (to be explained in Section IV-D). The corresponding grammar can be found in Table IIc.

Additionally, we define an operation mode (not further addressed it in this paper) as a number of dimensions, each having exactly one value. Namely, an operation mode is an element of an operation space when the operation space contains the corresponding values for all the dimensions.

Case study: Operation spaces are widespread in the model (as can be seen in Table Ia, Ib and Ic). For instance, the operation space of part *TrailorChiselPlow* has three dimensions, viz., *speed* has range $[0 : 25]$, *agility* has values "low" and "veryLow", and activity has value "plow". For illustration, the operation mode (speed:16, agility:low, activity:plow) is part of this operation space.

D. A formal model for a requirement

A requirement restricts the allowed operation modes. This is represented by a minimum and maximum operation space. For this purpose, an ordering on operation spaces is needed (to be introduced in Section V-C). On top of that, a requirement has a category, e.g., legal, ethical and business. Table IId presents the grammar.

Case study: Section "requirements" (of Table Ic) has three requirements: *speedRange*, i.e,. a minimum and maximum speed, *fuelAndPurchaseCosts*, i.e., a maximum fuel consumption and price, and *operability*, i.e., demanding easy operation.

E. A formal model for a design space

A design space provides a shorthand representation of many designs that differ on one or a few aspects. To this end, a design space is presented as n dimensions that each in turn contain a number of values. The design space is then defined as the n-ary Cartesian product of the n dimensions. Each element of the design space is a so-called design, design instance or design alternative. The grammar can be found in Table IIe.

Case study: Section "design space" (of Table Id) contains three dimensions, as follows. A tractor and trailor are connected to interoperate. A tractor can have three *transmissions* and four *engine fuels*. Moreover, each of these tractor combinations can be connected to four *trailors*. We derive that the design space has $3 \star 4 \star 4 = 48$ designs.

F. A formal model for a design alternative

A design alternative, i.e., DesAlt, has a design dimension name and a SystemOrPart for each value of this dimension (as defined in Section IV-E). A design alternative introduces a so-called variation point, viz., during evaluation, one of the SystemOrParts is selected depending on the dimension value of the design at hand. The corresponding grammar is revealed in Table IIe.

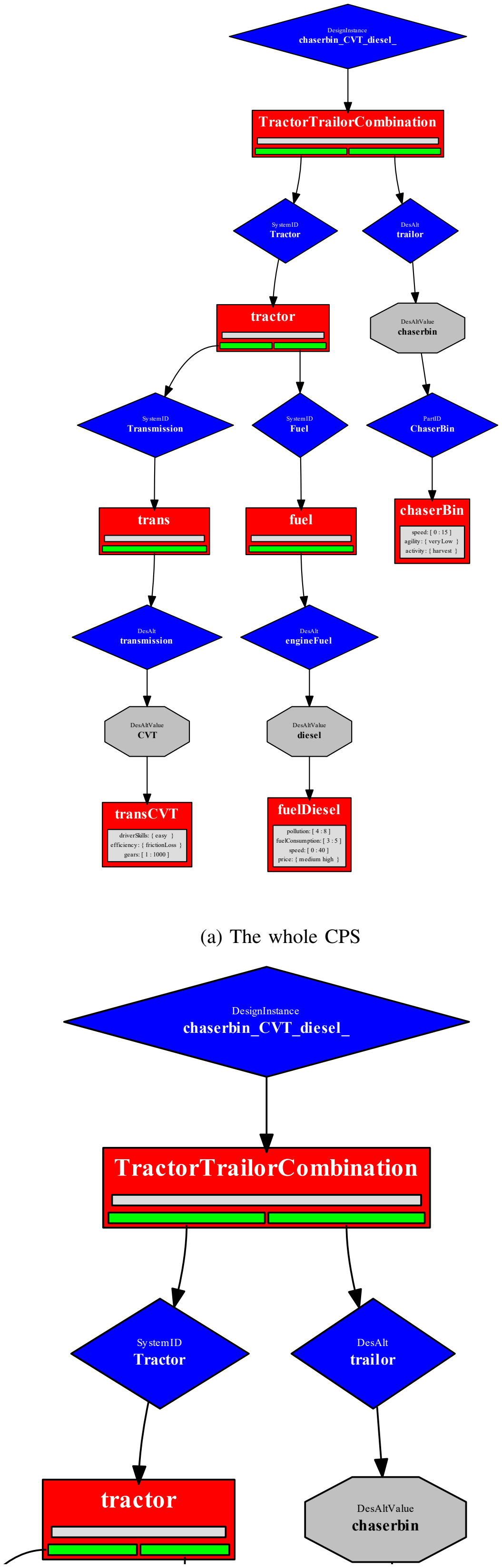
Case study: Section "system" (in Table Ia) conveys that the Top-level system *TractorTrailorCombination* comprises a DesAlt *trailor*. This DesAlt has four SystemOrParts, viz., *chiselPlow*, *trailortiller*, *chaserbin*, or *notrailor*. One of these SystemOrParts is selected during evaluation on the basis of the design at hand. For instance, when this DesAlt is evaluted for design $(chaserbin, CVT, steam)$, which has value *chaserbin* for dimension *trailor*, the result will be AbstractPart *chaserBin*.

V. Formal toolchain for CPS interoperability

Section IV introduced the aDSL model. Here, the aDSL semantics is provided by explaining how aDSL instances are evaluated and transformed into results, using three steps: (i) the system designer models a CPS (see Section V-A); (ii) aDSL evaluates the operation spaces of all subsystems (see Section V-B); and, (iii) aDSL evaluates the adherence to requirements for all subsystems and looks for the optimal designs (see Section V-C).

A. Modeling a CPS using an IDE

The system designer interactively creates aDSL models using the Eclipse Integrated Development Environment (IDE [51]); as the system designer types, the IDE shows feedback messages next to the code. To not overload the system designer with too much information, we define four levels of feedback messages: (i) Syntax: the standard Eclipse functionality, e.g., syntax checking and highlighting, as well as input completion; (ii) Concepts: making sure systems and parts have unique names and validating system and part references; (iii) Cycles: detecting cycles that may be introduced by system references; and, (iv) Guidance: displaying the designs that meet certain



(a) The whole CPS

(b) The top-level system and two subsystems

Fig. 1: The structure of CPS design (chaserbin,cvt,diesel), automatically generated with GraphViz [17].

requirements for each requirement. Given this, aDSL only shows messages of a certain level when there are no messages of lower levels to show. For instance, messages of type (iii) are only shown when there are neither messages of type (ii) nor messages of type (i) left to show.

After a valid model has been generated, aDSL automatically generates a visualization of the the model structure for each design. For illustration, Figure 1a visualizes the generated model of the use case of the previous section for design $(chaserbin, cvt, diesel)$. In Figure 1b, we zoom in on the top-level system and its two subsystems.

B. Evaluating the operation spaces of all subsystems

At this stage, the operation spaces of all subsystems are computed for each design, as follows. For each system, the operation spaces of all its underlying SystemOrParts and itself are merged into a new operation space, which is then assigned to the system. Function $\mathcal{M}^*$ is used for merging, which takes a number of operation spaces and returns one operation space with the intersection of all operation modes. It is defined as follows. Let O_1 and O_2 be operation spaces, represented as sets of dimension and values pairs. Then function $\mathcal{M}(O_1, O_2) = O_o$ takes two operation spaces and merges them into one operation space O_o, using the following three generation rules:

- If $(d : v) \in O_1$ and $\nexists x(d : x) \in O_2$, then $(d : v) \in O_o$
- If $(d : v) \in O_2$ and $\nexists x(d : x) \in O_1$, then $(d : v) \in O_o$
- If $(d : v) \in O_1$ and $(d : v') \in O_2$, then $(d : v \cap v') \in O_o$

where $(v_1 \cap v_2)$ is either the overlap of intervals for an interval dimension, or the shared values for a finite value dimension.

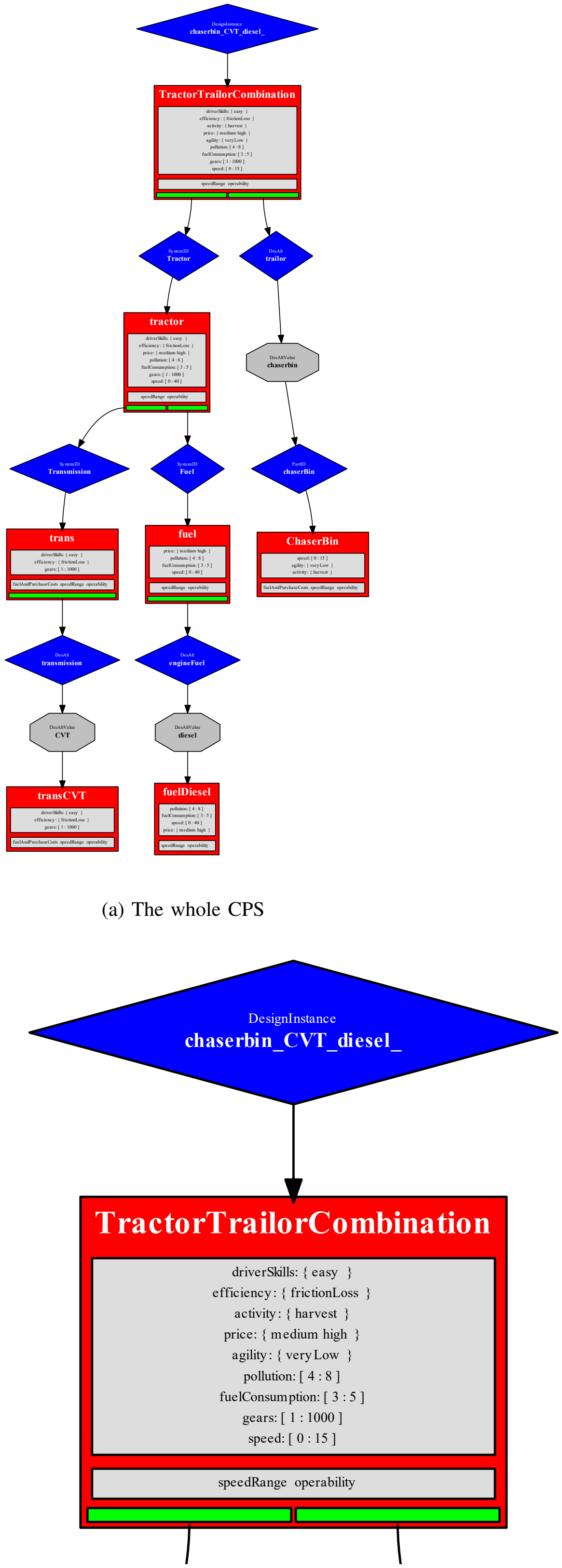
Next, function $\mathcal{M}^*$, which is used to merge many operation spaces, is defined by induction using base case $\mathcal{M}^*(O_1, O_2) = \mathcal{M}(O_1, O_2)$ and inductive step $\mathcal{M}^*(O_1, O_2, \cdots, O_n) = \mathcal{M}(O_1, \mathcal{M}^*(O_2, \cdots, O_n))$.

aDSL applies $\mathcal{M}^*$ in a bottom up way; the subsystems at the lowest level are evaluated first and the top-level system and subsystems inherit dimensions from all their children. Thus, the top-level system and subsystems at a higher level tend to end up with operation spaces that contain more dimensions.

Case study: Figure 2 extends Figure 1 with the evaluated operation modes and the outcomes for the requirements. Figure 2a contains a overall view of the structure and operation modes for design $(chaserbin, cvt, diesel)$. Figure 2b zooms in on the top-level system, which has an operation space with relatively many dimensions (due to inheritance from all subsystems). The design satisfies the requirements *speedRange* and *operability*, but dissatisfies *fuelAndPurchaseCosts* due to the high value for price.

C. Evaluating the adherence to the requirements

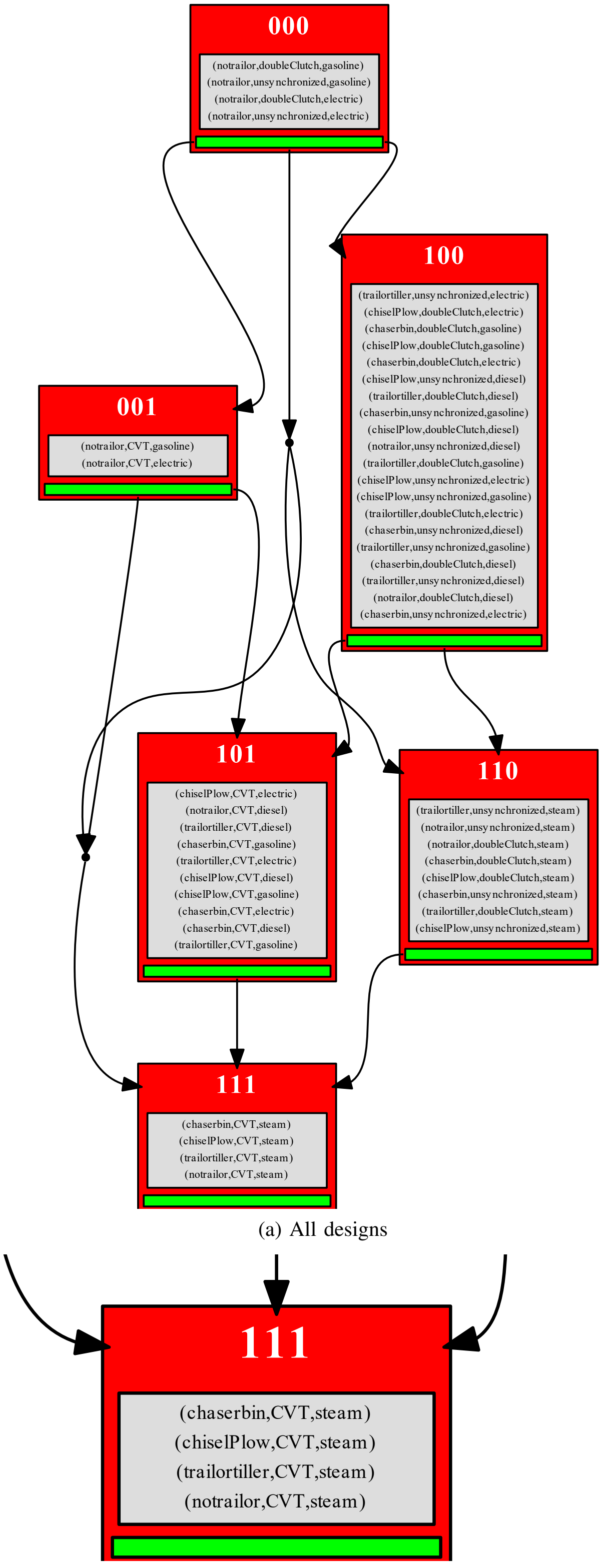
A requirement imposes a minima and/or maxima on a operation space. Thus, we define an ordering on operation spaces to determine if an operation space meets a requirement, as follows. Operation spaces can comprise different dimensions, hence, aDSL only considers the common dimensions for comparison. Let O_1 and O_2 be operation spaces, represented as



(a) The whole CPS

(b) The top-level system

Fig. 2: The evaluation of CPS design (chaserbin,cvt,diesel), automatically generated with GraphViz [17].


000
(notrailor,doubleClutch,gasoline)
(notrailor,unsynchronized,gasoline)
(notrailor,doubleClutch,electric)
(notrailor,unsynchronized,electric)
100
(trailortiller,unsynchronized,electric)
(chiselPlow,doubleClutch,electric)
(chaserbin,doubleClutch,gasoline)
(chiselPlow,doubleClutch,gasoline)
(chaserbin,doubleClutch,electric)
(chiselPlow,unsynchronized,diesel)
(trailortiller,doubleClutch,diesel)
(chaserbin,unsynchronized,gasoline)
(chiselPlow,doubleClutch,diesel)
(notrailor,unsynchronized,diesel)
(trailortiller,doubleClutch,gasoline)
(chiselPlow,unsynchronized,electric)
(chiselPlow,unsynchronized,gasoline)
(trailortiller,doubleClutch,electric)
(chaserbin,unsynchronized,diesel)
(trailortiller,unsynchronized,gasoline)
(chaserbin,doubleClutch,diesel)
(trailortiller,unsynchronized,diesel)
(notrailor,doubleClutch,diesel)
(chaserbin,unsynchronized,electric)
001
(notrailor,CVT,gasoline)
(notrailor,CVT,electric)
101
(chiselPlow,CVT,electric)
(notrailor,CVT,diesel)
(trailortiller,CVT,diesel)
(chaserbin,CVT,gasoline)
(trailortiller,CVT,electric)
(chiselPlow,CVT,diesel)
(chiselPlow,CVT,gasoline)
(chaserbin,CVT,electric)
(chaserbin,CVT,diesel)
(trailortiller,CVT,gasoline)
110
(trailortiller,unsynchronized,steam)
(notrailor,unsynchronized,steam)
(notraolor,doubleClutch,steam)
(chaserbin,doubleClutch,steam)
(chiselPlow,doubleClutch,steam)
(chaserbin,unsynchronized,steam)
(trailortiller,doubleClutch,steam)
(chiselPlow,unsynchronized,steam)
111
(chaserbin,CVT,steam)
(chiselPlow,CVT,steam)
(trailortiller,CVT,steam)
(notrailor,CVT,steam)
111
(chaserbin,CVT,steam)
(chiselPlow,CVT,steam)
(trailortiller,CVT,steam)
(notrailor,CVT,steam)

(a) All designs

(b) Pareto-optimal designs

Fig. 3: Designs ordered by the requirements they meet, automatically generated with GraphViz [17].

sets of dimension and values pairs. Then $O_1 \leq O_2$ implies that $(d : v) \in O_1 \rightarrow (\nexists x(d : x) \in O_2) \vee ((d : x) \in O_2 \wedge v \subseteq x)$. That is, when O_1 comprises dimension d, either O_2 does not comprise dimension d or O_2 contains dimension d with at least as many values for this dimension as O_1 does. Note that relation $\leq$ is a preorder, viz., it is partial ordering without the antisymmetric property. For instance, if operation space O_1 and O_2 are different operation spaces, which each have values for a single but different dimension, then both $O_1 \leq O_2$ and $O_2 \leq O_1$ hold without O_1 and O_2 being equal to each other.

aDSL uses relation $\leq$ to determine which top-level systems and subsystems meet which requirements, for all designs. aDSL also visualizes the way designs meet the requirements as a partial ordering.

Case study: Figure 3a shows, for all 48 designs, a overall view of the adherence to requirements at one glance, viz., the ones and zeros in the titles of the rectangles encode meeting a requirement or not, respectively. For instance, designs that are categorized "110" meet the first and second requirement (in order of occurrence in Table Ic), but not the third one. Hence, designs that are categorized "111" (see also Figure 3b), i.e., $(chaserbin, CVT, stream)$, $(chiselPlow, CVT, stream)$,$(trailorTiller, CVT, steam)$, and $(notrailor, CVT, steam)$, meet all three requirements and are thus Pareto optimal. aDSL evaluated all 48 CPS designs and generated Figure 2 (for each design) and Figure 3 (once) in a matter of seconds.

VI. Conclusion and future work

DSL-approaches to model embedded systems for safety and performance evaluation exist for embedded systems, but cannot directly be applied to CPSs. Meanwhile, the current CPS-approaches lack the advantages DSLs deliver, e.g., reasoning at the level of the problem and transformations into multiple languages for evaluation. Therefore, we have constructed aDSL, a DSL for modeling the interoperability of CPSs with tool support.

In aDSL, requirements are modeled as interoperability limits. Moreover, aDSL makes fully-automated design space exploration possible via variation points in the model. In a case study, aDSL evaluated 48 CPS designs, of which four met all requirements, and rendered visualizations with results in a matter of seconds.

In future work, we would like to investigate how the requirements of an aDSL model are elicited and validated. We envisage an approach in which requirements traceability plays a major role and the tool support of aDSL is extended.

Furthermore, we plan to extend aDSL with advanced algorithms under-the-hood to automatically verify more properties of a CPS, e.g., investigating timing and concurrency properties, and performing fault tree analysis. Also, we would like to automatically generate executable code.

References

[1] I. Lee, J. Y. Leung, and S. H. Son, *Handbook of real-time and embedded systems*. CRC Press, 2007.

[2] T. Henzinger and J. Sifakis, "The Embedded Systems Design Challenge," in *Formal Methods*. Springer, 2006, vol. 4085, pp. 1–15.

[3] C. Ebert and C. Jones, "Embedded Software: Facts, Figures, and Future." *IEEE Computer*, vol. 42, no. 4, pp. 42–52, 2009.

[4] H. Alemzadeh, R. Iyer, Z. Kalbarczyk, and J. Raman, "Analysis of Safety-Critical Computer Failures in Medical Devices," *IEEE Security & Privacy*, vol. 11, no. 4, pp. 14–26, 2013.

[5] R. Redington and W. Berninger, "Medical imaging systems," *Physics Today*, vol. 34, no. 8, pp. 36–44, 2008.

[6] Interventional X-ray systems, http://www.usa.philips.com/healthcare/solutions/interventional-xray/ixr-systems.

[7] A. J. Mooij, J. Hooman, and R. Albers, "Gaining industrial confidence for the introduction of domain-specific languages," in *Computer Software and Applications Conference Workshops (COMPSACW), 2013 IEEE 37th Annual*. IEEE, 2013, pp. 662–667.

[8] S. Keshishzadeh, A. J. Mooij, and M. R. Mousavi, "Early fault detection in dsls using SMT solving and automated debugging," in *Software Engineering and Formal Methods - 11th International Conference, SEFM 2013, Madrid, Spain, September 25-27, 2013. Proceedings*. Springer, 2013, pp. 182–196.

[9] S. Keshishzadeh, A. Mooij, and M. Mousavi, "Early Fault Detection in DSLs Using SMT Solving and Automated Debugging." in *Software Engineering and Formal Methods*, ser. Lecture Notes in Computer Science, vol. 8137. Springer, 2013, pp. 182–196.

[10] F. van den Berg, A. Remke, A. Mooij, and B.R. Haverkort, "Performance Evaluation for Collision Prevention Based on a Domain Specific Language," in *Computer Performance Engineering*, ser. Lecture Notes in Computer Science. Springer, 2013, vol. 8168, pp. 276–287.

[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 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.

[13] 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, vol. 10500, pp. 214–236.

[14] J. Bogdoll, A. David, A. Hartmanns, and H. Hermanns, "MCTAU: Bridging the Gap between Modest and UPPAAL," in *Proceedings* 19^{th} *International SPIN Workshop on Model Checking of Software*, ser. Lecture Notes in Computer Science, vol. 7385. Springer, 2012, pp. 227–233.

[15] F. van den Berg, B.R. Haverkort, and J. Hooman, "Efficiently Computing Latency Distributions by Combined Performance Evaluation Techniques," in *Proceedings of the* 9^{th} *EAI International Conference on Performance Evaluation Methodologies and Tools*, ser. VALUETOOLS'15. ICST, 2015, pp. 158–163.

[16] F. van den Berg, J. Hooman, A. Hartmanns, B.R. Haverkort, and A. Remke, "Computing Response Time Distributions Using Iterative Probabilistic Model Checking." in *Computer Performance Engineering*, ser. Lecture Notes in Computer Science. Springer, 2015, vol. 9272, pp. 208–224.

[17] J. Ellson, E. Gansner, L. Koutsofios, S. North, and G. Woodhull, "Graphviz—open source graph drawing tools," in *Graph Drawing*, ser. Lecture Notes in Computer Science, vol. 2265. Springer, 2002, pp. 483–484.

[18] J. Racine, "GNUplot 4.0: a portable interactive plotting utility," *Journal of Applied Econometrics*, vol. 21, no. 1, pp. 133–141, 2006.

[19] 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.

[20] 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.

[21] J. Boardman and B. Sauser, "System of systems-the meaning of of," in *System of Systems Engineering, 2006 IEEE/SMC International Conference on*. IEEE, 2006, pp. 6–pp.

[22] T. A. Henzinger, "The theory of hybrid automata," in *Verification of Digital and Hybrid Systems*. Springer, 2000, pp. 265–292.

[23] P. Miller, "Interoperability: What is it and why should i want it?" *Ariadne*, no. 24, 2000.

[24] S. Graham, G. Baliga, and P. Kumar, “Abstractions, architecture, mechanisms, and a middleware for networked control,” *IEEE Transactions on Automatic Control*, vol. 54, no. 7, pp. 1490–1503, 2009.

[25] B. Balaji, A. Faruque, M. Abdullah, N. Dutt, R. Gupta, and Y. Agarwal, “Models, abstractions, and architectures: the missing links in cyber-physical systems,” in *Proceedings of the 52nd Annual Design Automation Conference*. ACM, 2015, p. 82.

[26] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, “System architecture directions for networked sensors,” *ACM SIGOPS operating systems review*, vol. 34, no. 5, pp. 93–104, 2000.

[27] 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.

[28] 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.

[29] S. K. Khaitan and J. D. McCalley, “Design techniques and applications of cyberphysical systems: A survey,” *IEEE Systems Journal*, vol. 9, no. 2, pp. 350–365, 2015.

[30] E. A. Lee, “Cyber physical systems: Design challenges,” in *Object oriented real-time distributed computing (isorc), 2008 11th ieee international symposium on*. IEEE, 2008, pp. 363–369.

[31] J. Lee, B. Bagheri, and H.-A. Kao, “A cyber-physical systems architecture for industry 4.0-based manufacturing systems,” *Manufacturing Letters*, vol. 3, pp. 18–23, 2015.

[32] R. Jardim-Goncalves, D. Romero, and A. Grilo, “Factories of the future: challenges and leading innovations in intelligent manufacturing,” 2017.

[33] F. Standard, “1037c,” *Department of Defense Dictionary of Military and Associated Terms in support of MIL-STD-188*, 1996.

[34] D. K. Allen, S. Karanasios, and A. Norman, “Information sharing and interoperability: the case of major incident management,” *European Journal of Information Systems*, vol. 23, no. 4, pp. 418–432, 2014.

[35] International System for Agricultural Science and Technology, ”agris.fao.org”.

[36] L. R. Power, “Post-facto integration technology: new discipline for an old practice,” in *Systems Integration, 1990. Systems Integration’90., Proceedings of the First International Conference on*. IEEE, 1990, pp. 4–13.

[37] J. Searle and J. Brennan, “General interoperability concepts,” Tech. Rep., 2006.

[38] A.-J. Berre, B. Elvesæter, N. Figay, C. Guglielmina, S. Johnsen, D. Karlsen, T. Knothe, and S. Lippe, “The athena interoperability framework.” in *IESA*. Springer, 2007, pp. 569–580.

[39] W. Wang, A. Tolk, and W. Wang, “The levels of conceptual interoperability model: applying systems engineering principles to m&s,” in *Proceedings of the 2009 Spring Simulation Multiconference*. Society for Computer Simulation International, 2009, p. 168.

[40] T. Lojka, M. Miškuf, and I. Zolotová, “Industrial iot gateway with machine learning for smart manufacturing,” in *IFIP International Conference on Advances in Production Management Systems*. Springer, 2016, pp. 759–766.

[41] N. Chen, X. Zhang, and C. Wang, “Integrated open geospatial web service enabled cyber-physical information infrastructure for precision agriculture monitoring,” *Computers and Electronics in Agriculture*, vol. 111, pp. 78–91, 2015.

[42] E. Tantik and R. Anderl, “Potentials of the asset administration shell of industrie 4.0 for service-oriented business models,” *Procedia CIRP*, vol. 64, pp. 363–368, 2017.

[43] W. Dai, W. Huang, and V. Vyatkin, “Knowledge-driven service orchestration engine for flexible information acquisition in industrial cyber-physical systems,” in *Industrial Electronics (ISIE), 2016 IEEE 25th International Symposium on*. IEEE, 2016, pp. 1055–1060.

[44] M. Spichkova, H. Schmidt, and I. Peake, “From abstract modelling to remote cyber-physical integration/interoperability testing,” *arXiv preprint arXiv:1403.1005*, 2014.

[45] A. Fisher, C. A. Jacobson, E. A. Lee, R. M. Murray, A. Sangiovanni-Vincentelli, and E. Scholte, “Industrial cyber-physical systems–icyphy,” in *Complex Systems Design & Management*. Springer, 2014, pp. 21–37.

[46] P. Fritzson, “Modelicaa cyber-physical modeling language and the openmodelica environment,” in *Wireless Communications and Mobile Computing Conference (IWCMC), 2011 7th International*. IEEE, 2011, pp. 1648–1653.

[47] F. van den Berg, “Automated Performance Evaluation of Service-Oriented Systems,” Ph.D. dissertation, University of Twente, 2017.

[48] S. Haveman and M. Bonnema, “Requirements for High Level Models Supporting Design Space Exploration in Model-based Systems Engineering.” in *Procedia Computer Science*, ser. Procedia Computer Science, vol. 16. Elsevier, 2013, pp. 293–302.

[49] T. de Gooijer, A. Jansen, H. Koziolek, and A. Koziolek, “An industrial case study of performance and cost design space exploration,” in *Proceedings of the* 3^{rd} *International Conference on Performance Engineering*, WOSP/SIPEW. ACM, 2012, pp. 205–216.

[50] R. Gronback, *Eclipse modeling project: a domain-specific language (DSL) toolkit*. Pearson Education, 2009.

[51] Eclipse IDE for Java and DSL Developers, http://www.eclipse.org/downloads/packages/eclipse-ide-java-and-dsl-developers/junosr2.

[52] Xtext - Language Engineering Made Easy!, https://eclipse.org/Xtext/.

APPENDIX

In this appendix, the grammar of aDSL (see Table II) is presented in the Xtext [52] format. At its highest level or hierarchy, it comprises a model that decomposes into four sections, viz., a system and part, an operation space, a requirement, and a design space section.

TABLE II: The grammar of aDSL

(a) Model

```
Model:
   ’Section system’ ’Top-level’ cps=System s=System*
   ’Section Part’ p=Part*
   ’Section requirement’ requirements=Requirement*
   ’Section design space’ ds=DesignSpace;
```

(b) System and part

```
System:
  ’System’ name=ID
  oSpace+=OperationSpace
  ’{’ sop=SystemOrPart+ ’}’;
SystemOrPart: aSystem | SystemID | Part | PartID |
  DesAlternative;
PartID: ’AbstractPart’ name=ID referencePartID=ID;
SystemID: ’AbstractSystem’ name=ID referenceSystemID=ID;
Part: ’Part’ name=ID oSpace=OperationSpace;
```

(c) Operation space

```
OperationSpace:
   ’OperationSpace’ ’(’ o+=OperationDimension* ’)’;
OperationDimension: OperationDimensionValue |
   OperationDimensionInterval;
OperationDimensionValue: oModeDName=ID ’{’vals=ID+’}’;
OperationDimensionInterval:oModeDName=ID ’[’r+=Range’]’;
Range: begin=INT end=INT;
```

(d) Requirement

```
Requirement: rk=RequirementKind ’Requirement’ name=ID
  (’minimum’o=OperationSpace)?
  (’maximum’o=OperationSpace)?
RequirementKind: ’Legal’ | ’Ethical’ | ’Societal’ |
  ’Business’ | ’Functional’ | ’Performance’ | ’Design’ |
  ’Environmental’ | ’Physical’ | ’Resource’ | ’Other’;
```

(e) Design space

```
DesignSpace: ’DesignSpace’’(’ d+=DesignSpaceDimension+’)’;
DesignSpaceDimension: dsDimName=ID ’{’vals=ID+’}’;
DesAlternative: ’DesAlt’ ’(’ dsDimensionName=ID ’)’
  ’{’ dvsp=DesValueSystemOrPartPair+ ’}’;
DesValueSystemOrPartPair: ’{’dsVal=ID s=SystemOrPart’}’;
```

View publication stats