



From Requirements to Executable Rules: An Ensemble of Domain-Specific Languages for Programming Cyber-Physical Systems in Warehouse Logistics

Malte Mauritz^(✉)  and Moritz Roidl

TU Dortmund University, 44227 Dortmund, Germany
{malte.mauritz,moritz.roidl}@tu-dortmund.de

Abstract. The fourth industrial revolution is driven by Software-enabled automation. To fully realize the potential of this digital transformation in a way that is beneficial to society, automation needs to become programmable by domain experts—the vision being a Software-assisted increase in productivity instead of replacing workers with Software. While domain experts, e.g., workers in production, typically have extensive experience with processes and workflows involving cyber-physical systems, e.g., production machines, they have little to no knowledge of programming and formal logic. In this paper, we present a framework for expressing executable rules in the context of a cyber-physical system at the conceptual level, akin to human reasoning, in almost natural sentences (e.g., *if a person is within 1 m of the machine then the light will turn red*). These requirements are automatically transformed by our framework into formal logic and can be executed and evaluated by a rule engine without additional input by domain experts. The framework is designed in a modular way that enables domain engineering, i.e., the development of new languages for individual application domains, with minimal effort. Only domain-specific entities and predicates (e.g., *is within*) need to be defined and implemented for a new domain. We demonstrate our framework in a logistics scenario on a shop floor that requires human-machine collaboration.

Keywords: Domain-specific languages · Logistics · Language programming · Language model transformation · Runtime monitoring

1 Introduction

Digitization by software automation is growing significantly in many organizations. In many companies with no particular background in Software (e.g.,

The Research has been executed within the centre of excellence Logistics and IT which is funded by the Ministry of Culture and Science of the State of North Rhine-Westphalia and the Fraunhofer Society for the Advancement of Applied Research.

© Springer Nature Switzerland AG 2021

T. Margaria and B. Steffen (Eds.): ISO_{LA} 2021, LNCS 13036, pp. 162–177, 2021.

https://doi.org/10.1007/978-3-030-89159-6_11

in the industrial and manufacturing sector), Software crept into processes and products—first slowly but then with an ever-increasing pace and scope, culminating in the mantra that “every company needs to become a software company”. This mantra comes with an inherently increasing demand for skilled software-engineers. Universities and colleges are unable to satisfy this demand adequately.

In many instances, Software is merely a means of automation, and we have to put the power of controlling automation into the hands of domain experts and the general public to bring digital transformation to its full potential in a way that is beneficial to society. Moreover, rules in automated systems need to be understandable at the conceptual level of human reasoning to be amenable, e.g., to ethical and legal considerations. Putting it bluntly, we need frameworks that enable “programming the real world”, e.g., specifying the behavior of objects on a shop floor using observable events, conditions, and actions.

Classical systems engineering approaches fall short as in these approaches requirements on system behavior are decomposed to subsystems and refined to the level of signals, making the originally expressed intention hard to reconstruct (cf. [1,2] for examples). New or changed requirements need to be decomposed and translated from scratch, requiring expensive manual effort, involving domain experts and system developers.

We propose an alternative approach that uses domain-specific languages to bridge the gap between the conceptual level of human reasoning about the world and concrete signals and conditions in complex logistical systems enabling domain experts to write requirements like “*if a person carrying a box is within 1 m of the designated storage location then the light will turn green*”. Such requirements have a well-defined meaning in a system through domain-specific abstractions, computing the value of predicates like “*within 1 m of*” in a concrete cyber-physical system. System-wide relations and emergent behavior within cyber-physical systems can be evaluated in the form of natural predicates at runtime by preserving the meaning of requirements in domain-specific abstractions.

In this paper, we report on the results of an exploratory design effort, in which we have developed the generic architecture, languages, and abstractions of a framework that enables domain experts to program the behavior of cyber-physical systems (CPS) in the logistics domain. We present the development of a framework with domain-specific languages that allow to express executable rules about moving things in the real world at an abstract level, akin to human reasoning, e.g., *the light should turn green if a person carrying a box is within 1 m of the designated storage location*.

The resulting framework relies on three individual connected languages: (1) a language for the definition of requirements, (2) a language defining domain-specific objects, functions, and predicates as well as their interpretations in terms of concrete system signals and data, and (3) an extended first-order logic using arbitrary predicates for monitoring and controlling CPS at runtime.

We demonstrate our approach in a case study from the logistics domain: a scenario on a shop floor that requires human-machine collaboration.

Outline. The paper is organized as follows: Sect. 2 introduces logistics as our problem domain and related work. Section 3 presents our case study with our logistics research lab. Section 4 outlines our framework for the independent specification of domain and requirements. Section 5 describe the monitoring of the requirements at runtime and reports on results from our case study. Section 6 discusses the development of our framework and the results from our case study. Section 7 concludes the paper.

2 Related Work

Logistics is the science of moving things in the real world in a reasonable and efficient manner. Logistics applications permeate throughout the human inhabited world and exist in a plethora of different shapes and forms. Logistics systems scale from simple packaging stations to planetary-wide interdependent supply-chain networks. Although there are almost as many types of logistics systems as there are industries, they all share the same basic primitives of movement.

In order to sustain the pace of digital transformation, these logistics systems need to become programmable by domain experts. Moreover, rules in automated systems need to be understandable at the abstract level of human reasoning in order to be amenable to ethical and legal considerations.

Logistics automation is an engineering discipline with a disposition towards algorithms that use numeric methods for optimizing highly context-specific parameters. As such, it is a very different style of reasoning compared to what the shop floor workers are doing. Typical programming languages of field-level automated systems (i.e. Programmable Logic Controllers (PLCs)) are not usable by domain experts as they are rooted in control engineering. High-level requirements have to be related to the domain of these systems for execution in the real world and monitoring and controlling CPS at runtime. Entities, i.e., objects and predicates, which are used in high-level requirements have to be defined over the available signals and system data.

For this reason, *No Code* or *Low Code* platforms, e.g. DIME [3], Mendix¹, Creatio², Pega³, among others, have become more and more popular in industry [4]. These platforms aim to facilitate the programming of mobile, Internet of Things (IoT), and are applications by non-technical employees based on domain-specific languages (DSL) [5] and models [6]. However, most solutions predominantly target business process automation and user applications but are not suited for the programming of technical systems, e.g., robots.

In Academia, various languages and tools have been developed to address the definition of requirement and the automated processing. ASSERT [7] provides the constrained natural language SADL for formalizing domain ontologies and the requirements language SRL to express requirements as conditions over controlled system variables. FRET [8,9] provides the language FRETISH for

¹ <https://www.mendix.com/>.

² <https://www.creatio.com>.

³ <https://www.pega.com/>.

specifying the requirements and transforming them into temporal logic formulas. However, these tools still utilize mathematical conditions over system variables in their requirements.

Academia has developed several controlled natural languages, e.g. Processable English (PENG) [10], Controlled English to Logic Translation (CELT) [11], Computer Processable Language (CPL) [12], which are close to natural languages but allow for knowledge representation and reasoning about the textual content. A good overview and classification of controlled languages are given in [13]. Attempto Controlled English (ACE) [14,15] provides a large subset of the English language to specify requirements which can be transformed unambiguously into a first-order logic [10]. However, ACE aims at knowledge representation and reasoning and does not provide a mechanism for the interpretation of text entities on system data for runtime monitoring and evaluation.

Other solutions for the requirements engineering in natural language aim at transforming natural language into machine-processable formats using fuzzy matching domain-based parsing techniques [16] or natural language processing (NLP) [17]. The problem with these tools is that their interpretation of the requirements within a real-world domain is undefined.

As the authors of [18], we see the importance for integrating multiple models with specialized focus. We explicitly see the importance for the definition of domain entities and predicates over data of the CPS independent of the specification of requirements. We, therefore, have developed a framework which allows the specification of domain entities and predicates based on system data and their usage in requirements and first-order logic for the runtime monitoring and control.

The following Sect. 3 describe our use case. An overview of our framework with the engineering of the domain and the specification of monitorable requirements is given in Sect. 4.

3 Case Study: Warehouse Logistics

Our case study for the application of our framework has been implemented in a special purpose logistics research lab. This lab is designed as a highly flexible testbed environment for CPS and features a central experimentation area that is free of permanently installed equipment at ground-level. This central area is 22 m long and 15 m wide. It is surrounded by 40 infrared cameras that are part of a Motion Capturing System (MoCap). Additionally, eight laser projectors are installed on the ceiling that can project colored vector graphics on the floor. The graphic shapes can be changed with a very high frame rate and low latency.

Figure 1 shows the framework of the lab. Physical objects are tracked by the MoCap system via uniquely identifiable collections of retro-reflective markers. The generated data streams are sent to the simulation environment. The simulation environment mirrors the perceived objects into a virtual representation of the physical space. The simulation controls the laser projection system and displays virtual objects via a mapping to vector graphics directly in the experimentation area.

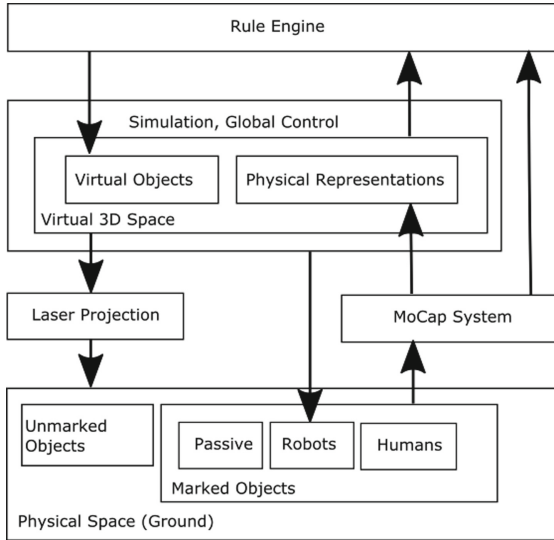


Fig. 1. Framework of the research lab

For the purpose of the application example, Message Queuing Telemetry Transport (MQTT) has been used as interface to allow the rule engine to generate virtual objects for laser projection to the floor. The incoming data stream of object positions could be either received by a subscription to the MoCap topic on the MQTT server or from the simulation environment. The latter option was used while developing the monitoring framework, as it allowed for faster testing of newly written code.

Our concrete scenario is the manual transports of assembly pieces between workstations, as it can be found in warehouses and plants. Assembly pieces are manually carried in a container between the workstations. The scenario consists of a worker carrying a container *KLT_2* for small assembly parts between two workstations *AS_1* and *AS_3* (cf. Figure 5). Positions of container and workstations are continuously tracked by a Motion Capturing System (MoCap) (cf. *position tracking* in Fig. 2). The tracking data is sent to a broker using the MQTT protocol [19] from where it is distributed to our rule engine.

The numerical data streams containing the objects' positions are transformed for the rule engine into abstract interpretations constituting logical constants, properties, and predicates (cf. *position abstraction* in Fig. 2). The abstract interpretations are used by the rule engine to reason about the relative positioning of container *KLT_2* and workstations *AS_1* and *AS_3*.

Based on the evaluation of formulas by the rule engine, actions within these formulas are executed to visualize relative positions of container *KLT_2* and workstations *AS_1* and *AS_3*. The rule engine emits the active actions within

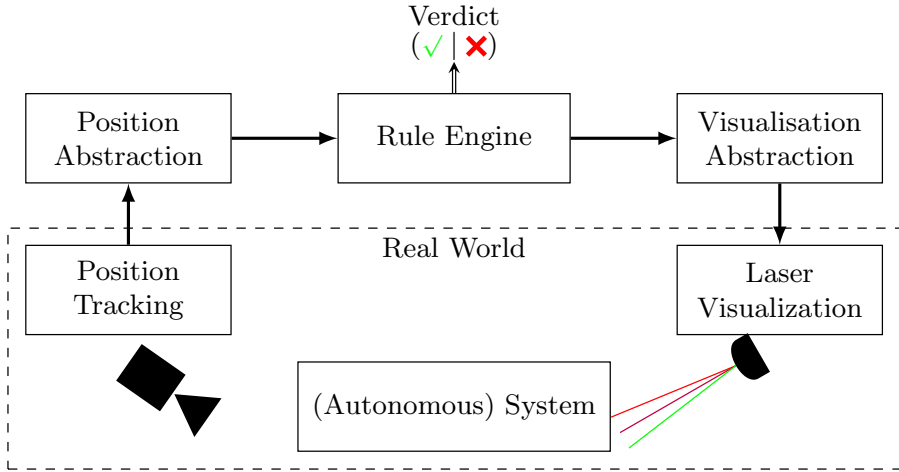


Fig. 2. Data flow at runtime

these formulas. These abstract actions are transformed into concrete visualization objects for the laser projectors of the testbed (cf. *visualization abstraction* in Fig. 2).

The following Sect. 4 gives an overview about our framework and presents the specification of domain and requirements in our case study.

4 Separation of Domain and Requirements

The behavior of logistics systems can become very complex, as these systems consist of many components with reactive behavior. The specification of emerging behavior in requirements solely based on internal signals and data can be difficult, if not impossible. Our vision is to bridge the gap between natural requirements and monitors or rules on the actual system through code generation and execution.

We want to enable domain experts to specify requirements for logistics systems akin to human reasoning, while directly monitoring and verifying these requirements at runtime. Domain experts do not need any knowledge about programming, language engineering, and formal methods. Otherwise, domain experts would be more concerned with technical or formal details, e.g., the syntax of first-order logic, than specifying the expected system behavior as requirements.

We hold domain experts away from the technical details of the domain and the logistics system by separating the requirements specification in into two main tasks - *domain engineering* and *requirements specification*:

1. The domain engineering defines the objects, properties, and predicates within the application domain and interprets these for the data of the CPS, and

- the requirements specification define the requirements about the CPS using these domain objects, properties, and predicates.

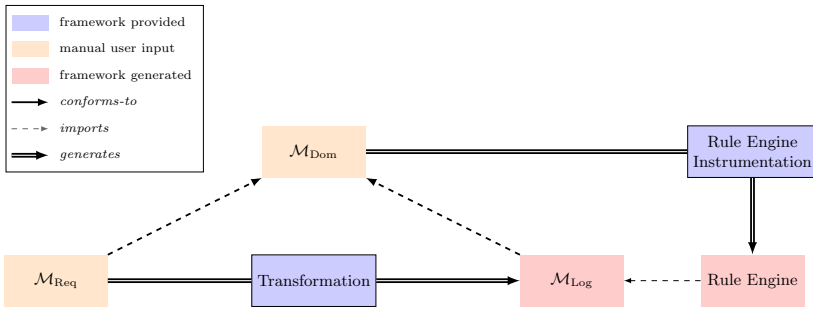


Fig. 3. Overview over the language framework

The separation in domain engineering and requirements specification (cf. Figure 4) is also represented by the models \mathcal{M} in framework. Our framework incorporates three different domain-specific languages for specifying requirements akin to the reasoning of domain experts and executing these requirements at runtime to monitor and control these systems (cf. Figure 3):

- Domain \mathcal{M}_{Dom} describes the domain and its specific terminology which are used in the requirements \mathcal{M}_{Req} , the first-order logic \mathcal{M}_{Log} , and the rule engine.
- Requirements \mathcal{M}_{Req} enables specifying requirements akin to human reasoning.
- Extended first-order logic \mathcal{M}_{Log} provides the unambiguous syntax and semantics for monitoring and executing requirements at runtime.

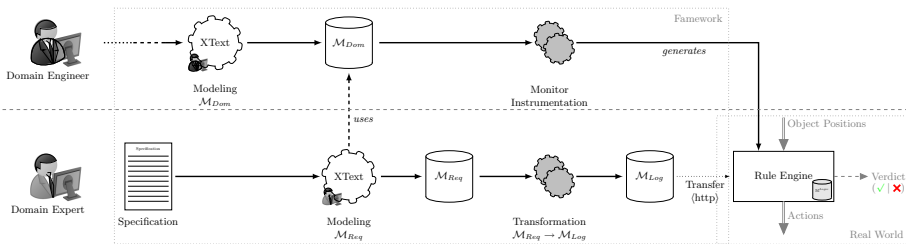


Fig. 4. Development workflow

As shown in Fig. 4, each task - domain engineering and requirements specification- requires a different skill sets. We, therefore, distinguish the roles *domain engineer* and *domain expert*:

- The *domain engineer* possesses sufficient knowledge about the application domain as well as programming and language engineering with domain-specific languages for comprehensively representing the application domain. From this domain specification, the rule engine is instrumented to interpret the entities and predicates in requirements for data from the monitored CPS.
- The *domain expert* has deep knowledge about the investigated CPS to exhaustively specify requirements about the system and its behavior akin to their natural reasoning. The natural requirements are automatically transformed into first-order logic for evaluation by the rule engine at runtime.

The domain engineering for warehouse logistics is described in Sect. 4.1, while the specification of requirements and their monitoring for the manual transport of goods in our logistics research lab are presented in Sect. 4.2.

4.1 Engineering the Warehouse Logistics Domain

The development in our use case commences with the definition of the domain \mathcal{M}_{Dom} for Warehouse Logistics. The domain \mathcal{M}_{Dom} is the central component in our solution and ensures that requirements \mathcal{M}_{Req} and formulas \mathcal{M}_{Log} reason about the identical application domain.

As shown in Fig. 3, respective interpretations within the domain \mathcal{M}_{Dom} are imported into requirements \mathcal{M}_{Req} and formulas \mathcal{M}_{Log} . The data interpretation of objects and predicates are used by the rule engine at runtime to evaluate domain objects and predicates for the current state of the CPS and process a verdict for requirements \mathcal{M}_{Req} and in formulas \mathcal{M}_{Log} at runtime e (cf. our case study in Sect. 5).

Domain \mathcal{M}_{Dom} defines the domain's objects and predicates as they are used in our case study within our logistics research lab and associates these entities to the data in our logistics research lab. The domain \mathcal{M}_{Dom} defines the item container *KLT_2*, workstations *AS_1* and *AS_3*, and predicates about the geometric relationship between objects, i.e., *besides*. An excerpt of the domain is shown in Listing 1.1.

Listing 1.1. Defintion of domain \mathcal{M}_{Dom}

```

1 def KLT_2 : Object {
2   Requirement: KLT_2
3   Logic: KLT_2
4   Data: positionObjects [name==KLT_2]
5 }

7 def AS_1 : Object {
8   Requirement: AS_1
9   Logic: AS_1
10  Data: positionObjects [name==AS_1]
11 }

13 def beside : Predicate {
```



```

14 Requirement: beside dist:Float of
15 Logic: beside(Con1:Constant,Con2:Constant,dist:Float)
16 Data: ABS(positionObjects[name==Con2]->position -
      positionObjects[name==Con1]) < dist
17 }

```

The objects *KTL_2* and *AS_1* both have the type *Object* and are identically expressed in the requirements \mathcal{M}_{Req} and first-order logic \mathcal{M}_{Log} (cf. keywords *Requirement* and *Logic* in Listing 1.1). The predicate *beside* is defined in \mathcal{M}_{Req} by the word *within* and an arbitrary distance *dist* (cf. keyword *Requirement* in Listing 1.1). The objects which are compared by the predicate *beside* are determined by the sentence in which the predicate is used. For the first-order logic \mathcal{M}_{Log} , the predicate is defined with three parameters; the first two parameters take constants *Con1* and *Con2* as inputs while the third parameter takes a float value *dist* for the distance (cf. keyword *Logic* in Listing 1.1).

At runtime, predicate *beside* is interpreted and evaluated for the incoming streams of position data from the MoCap System. The data from the MoCap system is provided to the rule engine via the MQTT broker as JSON data [20, 21]. The interpretation of the system data is defined by the data function (cf. keyword *Data* in Listing 1.1); predicate *beside* will be evaluated as satisfied if the absolute amount of the distance between the position of object *Con1* and the position of constant *Con2* is below the given distance *dist*. Objects *Con1* and *Con2* are filtered in the list of objects *positionObjects* in the MQTT data based on their names and their common attribute *position* is assessed for the position values. These data interpretations are used in the abstraction of the numerical position data to abstract representations in first-order logic (cf. *position abstraction* in Fig. 2).

For our demonstration, we defined five additional predicates for evaluating the relative position of container and workstations. All six predicates are shown in Table 1.

In addition to the six predicates about the relative position of container and workstations, actions *draw rectangle* and *draw circle* are defined in the domain \mathcal{M}_{Dom} (cf. Table 1). Action *draw circle* will result in a circle as laser visualization while action *draw rectangle* yields to the projection of a rectangle by the laser projectors. Both actions have parameters for **position**, **color**, **size**, **duration**, and **animation** of the projected **rectangle** resp. **circle**. Positions of the laser visualizations can be defined either as fixed positions in world coordinates or the visualizations can be attached to the positions of objects. Attachments to objects provide the advantages that the visualizations will change their positions according to the movement of these objects.

The following Sect. 4.2 describes how the requirements \mathcal{M}_{Req} are defined for our logistics case study using the definitions in domain \mathcal{M}_{Dom} .

4.2 Specification of Requirements

Requirements \mathcal{M}_{Req} are specified by domain experts using an almost natural but domain-specific language. Requirements \mathcal{M}_{Req} have the basic clause

Table 1. Overview of predicates in the domain \mathcal{M}_{Dom} .

Predicate	Requirements	First-Order Logic
inside	[Obj] is inside [Box]	inside(Obj,Box)
rest on	[Obj] is on [Box]	reston(Obj,Box)
rest under	[Obj] is under [Box]	restunder(Obj,Box)
above	[Obj] is above [Box]	above(Obj,Box)
below	[Obj] is below [Box]	below(Obj,Box)
beside	[Obj] is within X m of [Box]	besides(Obj,Box,X)
draw circle	draw circle with position x in color y ... circle(position, color, ...)	
draw rectangle	draw rectangle with position x in color y ... rectangle(position, color, ...)	

if $\langle \text{conditions} \rangle$ **then** $\langle \text{actions} \rangle$

where $\langle \text{conditions} \rangle$ and $\langle \text{actions} \rangle$ can be conjunctions of multiple conditions resp. actions, e.g., *if a package is available in one of the loading areas, then an idling robot has to move to the leading area and pick up the package*. Requirements \mathcal{M}_{Req} import the specific interpretations of domain-specific entities, i.e., item container *KTL_2*, workstations *AS_1* and *AS_3*, and predicates about the geometric relationship between objects, from domain \mathcal{M}_{Dom} (cf. Section 4.1).

Listing 1.2. Initial definition of requirements.

```

1 Req Req1: If KTL_2 is within 4m of AS_1 and KTL_2 resides
  within 4m to AS_3 then show a rectangle with scale 6.0
  at position {1.0,0.0,0.0} with magenta color.
2 Req Req2: If KTL_2 is within 3m of AS_1 then print a
  rectangle over AS_1 in yellow color with scale 2.0.
3 Req Req3: If KTL_2 is within 3m of AS_3 then print a
  rectangle over AS_1 in yellow color with scale 2.0.

```

In our demonstration, we defined three requirements about the position of the container *KTL_2* in relation to the workstations *AS_1* and *AS_3* in the requirements \mathcal{M}_{Req} using the definition of the domain \mathcal{M}_{Dom} . The requirements are shown in Listing 1.2. Requirement *Req1* will result in the drawing of a magenta rectangle with 6m edges at the global position $\{1.0,0.0,0.0\}$ if the container *KTL_2* is within 4m of both workstations *AS_1* and *AS_3*. The second requirement *Req2* will result in a yellow rectangle drawn around the workstation *AS_1* with edge of 2m if the distance between container *KTL_2* and workstation *AS_1* is less than 3m. Requirement *Req3* is identical to requirement *Req2* with workstation *AS_1* replaced by workstation *AS_3*.

The following Sect. 5 describe how the requirements \mathcal{M}_{Req} are transformed into first-order formulas \mathcal{M}_{Log} and monitor in our shop floor scenario in our logistics research lab.

5 Runtime Monitoring and System Control

Requirements \mathcal{M}_{Req} are automatically transformed by our framework into formulas \mathcal{M}_{Log} in first-order logic for monitoring and controlling the CPS at runtime. Our framework uses classic first-order logic extended by operators for equality $==$ and the implication $|>$ between conditions and actions.

Similar to the requirements \mathcal{M}_{Req} , the specific interpretations of entities in domain \mathcal{M}_{Dom} , i.e., item container *KLT_2*, workstations *AS_1* and *AS_3*, and predicates about the geometric relationship between objects, are imported into the formulas \mathcal{M}_{Log} . The semantics of formulas \mathcal{M}_{Log} are given by the interpretation of these domain-specific entities for the data of CPS.

As shown in Listing 1.3, three formulas \mathcal{M}_{Log} are generated in first-order logic from the three requirements in Listing 1.2. The requirement *Req1* in Listing 1.2 is transformed to formula *Req1* in Listing 1.3.

Listing 1.3. Formulas \mathcal{M}_{Log} for the running example

```

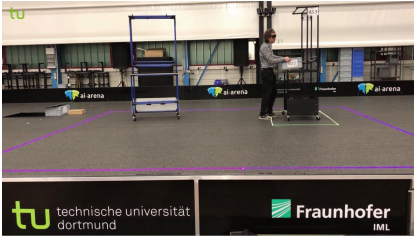
1 Req1: beside(KLT_2, AS_1, 4) AND beside (KLT_2, AS_3, 4) |>
    rectangle (Req1_1, {-1.0, 0.0, 0.0}, 1, magenta, 4, 6, 6, none)
;
2 Req2: beside(KLT_2, AS_1, 3) |> rectangle (AS_1, 1, yellow
    , 4, 2, 2, none);
3 Req3: beside(KLT_2, AS_3, 3) |> rectangle (AS_3, 1, yellow
    , 4, 2, 2, none);

```

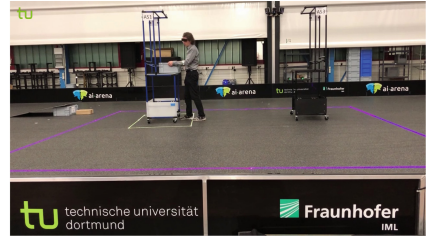
The container *KTL_2* and workstations *AS_1* and *AS_3* are equally used in requirements \mathcal{M}_{Req} as well as formulas \mathcal{M}_{Log} . Therefore, representations of these objects in requirements \mathcal{M}_{Req} are directly transformed into corresponding representations in formulas \mathcal{M}_{Log} using the definitions in domain \mathcal{M}_{Dom} . The predicate *within* in \mathcal{M}_{Req} maps to the predicate *beside* in formulas \mathcal{M}_{Log} . The container *KTL_2*, workstation *AS_1* resp. *AS_3*, and constraint *4m* are assigned to the parameters *Con1*, *Con1*, and *dist* respectively. The preceding verbs *is* and *resides* for the predicate *within* in requirements \mathcal{M}_{Req} are syntactic sugar and are not considered for the transformation of predicate *within* into the formulas \mathcal{M}_{Log} .

The action in the *action part* of requirements *Req1* in Listing 1.2 is transformed in the predicate *rectangle* in \mathcal{M}_{Log} with its parameters derived from the additional constraints in the *action part*, e.g., *at position* $\{-1.0, 0.0, 0.0\}$ and *in magenta color*. Requirements *Req2* and *Req3* are transformed into formulas similar to the transformation of requirement *Req1*. The semantics of these three formulas in Listing 1.3 is consistent with the semantics of their original requirements in Listing 1.2.

Results from runtime evaluation of the three formulas in Listing 1.3 for the manual transport of container *KLT_2* between the two workstations *AS_1* and *AS_3* are shown in Fig. 5. The rule engine for this demonstration has been instrumented for evaluating the three formulas in Listing 1.3 at runtime using the definition of the domain in Listing 1.1.



(a) Visualization for container KLT_2 at workstation AS_3 .



(b) Visualization for container KTL_2 at workstation AS_1 .

Fig. 5. Demonstration of logistics scenario.

In Fig. 5a, the container KLT_2 is at the workstation AS_3 . The formulas $Req1$ and $Req3$ are satisfied in this situation while requirement $Req2$ fails due to the large distance to workstation AS_1 . As result, a large magenta rectangle and a small yellow rectangle around workstation AS_3 are drawn according to the satisfied formulas $Req1$ and $Req3$ (cf. Listing 1.2).

After carrying the container KLT_2 over to the workstation AS_1 , the yellow rectangle around workstation AS_3 disappears while a new yellow rectangle appears around workstation AS_1 . This behavior is consistent with formulas $Req2$ and $Req3$ (cf. Listing 1.2) because the predicate *beside* in formula $Req3$ is now invalid while it is now true for formula $Req2$. Formula $Req1$ is satisfied in both situations. Therefore, the large magenta rectangle is drawn throughout the complete transition of container KLT_2 from workstation AS_3 to workstation AS_1 .

Our lessons from the development and application of our framework are presented in the following section.

6 Discussion

The work presented in this paper is highly explorative. We have aimed for a first technical prove-of-concept.

While the current implementation of our framework is a prototype, we can draw some initial conclusions from its development, the conducted application.

Remark 1: *Separation of languages seems to support the distribution and reuse in system development.*

Our case study has provided first indications that the separation of domain engineering and requirements specification has a positive impact in the development of logistical CPS. Requirements \mathcal{M}_{Req} and domain \mathcal{M}_{Dom} seem to be reusable in another context independent of each other. Requirements can be used for CPS in other domains by just redefining the objects and predicates in requirements for the new domain. The development of new systems, may reuse

an existing domain \mathcal{M}_{Dom} in its new requirements \mathcal{M}_{Req} of the new system. There is no need to adapt the objects and predicates in the domain \mathcal{M}_{Dom} to the new system.

Requirements and domain seem to be easily developed independent of each other, providing organizations with the opportunity to efficiently employ domain engineers within the development of these systems. As critical resource due to their technical knowledge, domain engineers may concentrate on the definition of domain \mathcal{M}_{Dom} while domain expert with limited technical knowledge can define the relevant requirements akin to their natural reasoning. This approach enables organization to introduce more non-technical domain experts to the development of these technical systems.

Remark 2: *Designing a domain-specific language for natural requirements, akin to human reasoning, is expensive.*

The specification of requirements has to feel as natural as possible for domain experts—akin to their usual reasoning of these logistical systems—in order for domain experts to adapt languages and tools into their work. Otherwise, domain expert are more occupied with the expression of requirements in a specific language grammar than comprehensively representing the systems in their requirements.

English and other natural language provide various clauses that allow for different sentence structures with identical semantic meaning in requirements. Replicating this grammatical flexibility in domain-specific languages is difficult and expensive. The majority of clauses from the natural base language have to be defined in the requirements language to suggest grammatical flexibility. Furthermore, all these clauses have to be mapped to expressions in the first-order logic for the evaluation at runtime. However, clauses often contain similar sentence structure which can cause problems for parser generators, i.e., ANTLR [22]. Clauses with similar sentence structure hinder the parser to associate corresponding parts of requirements deterministically to a single clause for building an abstract syntax tree. As a result, the grammar of the requirements language has to be carefully specified for these parser generators to work correctly. There exist some controlled natural languages, e.g., Attempto Controlled English (ACE) [14, 15], Processable English (PENG) [10], and Computer Processable Language (CPL) [12] among others, which provide the grammatical flexibility of English, but these languages miss the mapping of objects and predicates to real system data.

Remark 3: *Temporal Constraints are an essential part of today's technical requirements in logistics.*

Within our case study, we have recognized that many technical requirements in logistics are time-dependent, but our requirements and logic are currently not able to express such temporal expressions. We neglected temporal constraint in our framework because we focused on the architectural combination of the three models, i.e., requirements, domain, and logic, to enable the specification

of monitorable requirements at runtime. It is essential that we address temporal expression in our requirements and extend the logic by temporal operators as they are used in, e.g., metric first-order temporal logic [23], in future work.

Remark 4: *System data has to be provided in sufficient quantity and quality.*

An important part for monitoring requirements at runtime is the data provided by the CPS. This system data defines the scope of domain and requirements, which can be evaluated by the rule engine at runtime. The CPS has to provide data in the appropriate form, quantity, and quality to sufficiently represent the real application domain in domain \mathcal{M}_{Dom} . Objects and predicates without any processable data function in domain \mathcal{M}_{Dom} could be potentially used in requirements \mathcal{M}_{Req} to form syntactic correct requirements, but these requirements cannot be interpreted by the rule engine at runtime. There exist two possibilities for evaluating an object or predicate for the data from the CPS, if (1) the provided data contains a corresponding data item or (2) the object and predicate can be calculated from the available data items. For example, the rule engine will evaluate the velocity of a robot, only if the provided data from the robot does include a data item about the robot's velocity or the remaining data items allow to calculate the velocity.

7 Conclusion

We have presented a framework for the development of domain-specific languages that express executable rules about CPS in logistics at the conceptual level, akin to the reasoning of domain experts. The framework defines three individual languages: (1) the domain language allows to specify the application domain with its entities, and predicates over these entities, (2) the requirements language provides the grammatical flexibility to define requirements akin to human reasoning, (3) the first-order logic provides clear syntax and semantics for monitoring and controlling CPS at runtime. We have demonstrated our framework in a logistics scenario with human-machine collaboration. The case study shows that the control and monitoring of CPS in logistics is viable and beneficial under the separation of requirements and domain specification. The separate definition of domain entities, i.e., objects, their properties, and predicates, over the system data is essential for specifying requirements akin to human reasoning. However, the development of our framework has also shown that the design of an almost natural language for the requirements specification is very costly and difficult.

One point to consider for the future is the consideration of existing and proven controlled natural languages, e.g. ACE or PENG, for the specification of requirements. Additional future improvements for our framework to consider: (1) temporal expressions in our requirements language and formal logic, (2) different languages than standard English, and (3) the integration of our framework in a requirements engineering workflow. We also envisage a potential of our work in other domain than logistics and want to harden the expressive power within our framework through application in more diverse domains.

References

1. Brat, G., Bushnell, D., Davies, M., Giannakopoulou, D., Howar, F., Kahsai, T.: Verifying the safety of a flight-critical system. In: Björner, N., de Boer, F. (eds.) FM 2015. LNCS, vol. 9109, pp. 308–324. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19249-9_20
2. Geist, J., Rozier, K.Y., Schumann, J.: Runtime observer pairs and bayesian network reasoners on-board FPGAs: flight-certifiable system health management for embedded systems. In: Bonakdarpour, B., Smolka, S.A. (eds.) RV 2014. LNCS, vol. 8734, pp. 215–230. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11164-3_18
3. Boßelmann, S., et al.: DIME: a programming-less modeling environment for web applications. In: Margaria, T., Steffen, B. (eds.) ISO/IEC 26262:2018. LNCS, vol. 9953, pp. 809–832. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47169-3_60
4. Vincent, P., Iijima, K., Driver, M., Wong, J., Natis, Y.: Magic Quadrant for Enterprise Low-Code Application Platform. Gartner report (2020)
5. Fowler, M.: Domain-Specific Languages. Pearson Education (2010)
6. Zweihoff, P., Steffen, B.: A generative approach for user-centered, collaborative, domain-specific modeling environments. arXiv preprint [arXiv:2104.09948](https://arxiv.org/abs/2104.09948) (2021)
7. Crapo, A., Moitra, A., McMillan, C., Russell, D.: Requirements capture and analysis in ASSERT™. In: 2017 IEEE 25th International Requirements Engineering Conference (RE), pp. 283–291 (2017)
8. Mavridou, A., et al.: The ten lockheed martin cyber-physical challenges: formalized, analyzed, and explained. In: 2020 IEEE 28th International Requirements Engineering Conference (RE), pp. 300–310 (2020)
9. Giannakopoulou, D., Pressburger, T., Mavridou, A., Schumann, J.: Generation of formal requirements from structured natural language. In: Madhavji, N., Pasquale, L., Ferrari, A., Gnesi, S. (eds.) REFSQ 2020. LNCS, vol. 12045, pp. 19–35. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-44429-7_2
10. Schwitter, R.: Controlled natural languages for knowledge representation. In: *Coling Posters*, vol. 2010, pp. 1113–1121 (2010)
11. Pease, A., Li, J.: Controlled English to logic translation. In: Poli, R., Healy, M., Kameas, A. (eds.) *Theory and Applications of Ontology: Computer Applications*. Springer, Dordrecht (2010). https://doi.org/10.1007/978-90-481-8847-5_11
12. Clark, P., Harrison, P., Jenkins, T., Thompson, J.A., Wojcik, R.H., et al.: Acquiring and using world knowledge using a restricted subset of English. In: *Flairs conference*, pp. 506–511 (2005)
13. Kuhn, T.: A survey and classification of controlled natural languages. *Comput. Linguist.* **40**(1), 121–170 (2014). <https://www.aclweb.org/anthology/J14-1005>
14. Fuchs, N.E., Kaljurand, K.: Attempto controlled English meets the challenges of knowledge representation, reasoning, interoperability and user interfaces (2006)
15. Fuchs, N.E., Kaljurand, K., Kuhn, T.: Attempto controlled English for knowledge representation. In: Baroglio, C., Bonatti, P.A., Małuszyński, J., Marchiori, M., Polleres, A., Schaffert, S. (eds.) *Reasoning Web*. LNCS, vol. 5224, pp. 104–124. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85658-0_3
16. Ambriola, V., Gervasi, V.: Processing natural language requirements. In: *Proceedings 12th IEEE International Conference Automated Software Engineering*, pp. 36–45. IEEE (1997)

17. Ghosh, S., Elenius, D., Li, W., Lincoln, P., Shankar, N., Steiner, W.: ARSE-NAL: automatic requirements specification extraction from natural language. In: Rayadurgam, S., Tkachuk, O. (eds.) NFM 2016. LNCS, vol. 9690, pp. 41–46. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40648-0_4
18. Steffen, B., Gossen, F., Naujokat, S., Margaria, T.: Language-driven engineering: from general-purpose to purpose-specific languages. In: Steffen, B., Woeginger, G. (eds.) Computing and Software Science. LNCS, vol. 10000, pp. 311–344. Springer, Cham (2019). https://doi.org/10.1007/978-3-319-91908-9_17
19. Hunkeler, U., Truong, H.L., Stanford-Clark, A.: MQTT-S-A publish/subscribe protocol for wireless sensor networks. In: 2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE 2008), pp. 791–798. IEEE (2008)
20. Pezoa, F., Reutter, J.L., Suarez, F., Ugarte, M., Vrgoč, D.: Foundations of json schema. In: Proceedings of the 25th International Conference on World Wide Web, ser. WWW 2016. Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee, pp. 263–273 (2016). <https://doi.org/10.1145/2872427.2883029>
21. Internet Engineering Task Force (IETF): The javascript object notation (json) data interchange format (2014). <https://tools.ietf.org/html/rfc7159>
22. Parr, T.: The Definitive ANTLR 4 Reference, 2nd edn. Pragmatic Bookshelf (2013)
23. Koymans, R.: Specifying real-time properties with metric temporal logic. Real-Time Syst. **2**(4), 255–299 (1990)