Applying Restricted English Grammar on Automotive Requirements—Does it Work? A Case Study

Amalinda Post[1], Igor Menzel[1], and Andreas Podelski[2]

[1] Robert Bosch GmbH, Corporate Research, Stuttgart, Germany
{Amalinda.Post,Igor.Menzel}@de.bosch.com
[2] University of Freiburg, Department of Computer Science, Freiburg, Germany
podelski@informatik.uni-freiburg.de

Abstract. [**Context and motivation**] For an automatic consistency check on requirements the requirements have to be formalized first. However, logical formalisms are seldom accessible to stakeholders in the automotive context. Konrad and Cheng proposed a restricted English grammar that can be automatically translated to logics, but looks like natural language. [**Question/problem**] In this paper we investigate whether this grammar can be applied in the automotive domain, in the sense that it is expressive enough to specify automotive behavioral requirements. [**Principal ideas/results**] We did a case study over 289 informal behavioral requirements taken from the automotive context. We evaluated whether these requirements could be formulated in the grammar and whether the grammar has to be adapted to the automotive context. [**Contribution**] The case study strongly indicates that the grammar, extended with 3 further patterns, is suited to specify automotive behavioral requirements of BOSCH.

Keywords: automotive, requirements, formalization, real-time.

1 Introduction

In this work we investigate whether the restricted English grammar provided by Konrad and Cheng [1] suffices to express behavioral requirements taken from the automotive context. The grammar looks like natural language, however it allows an automatic translation into linear time logic (LTL)[2], computational tree logic (CTL)[2], graphical interval logic (GIL)[3], metric temporal logic (MTL)[4], timed computational tree logic (TCTL)[4], and real-time graphical interval logic (RTGIL)[5].

We are only interested in requirements specifying the *behavior* of the system, i.e., we do *not* consider any other kind of requirements. Thus, in the following, every time we speak of "requirements" we mean in fact *behavioral requirements* [6], i.e., requirements specifying the behavior of the system. We consider behavior without exact timing bounds (e.g., "*If the system is in diagnostic mode then previously the diagnostic request* DiagStart *held.*") and behavior with exact timing bounds (e.g., "*If the diagnostic request* IRTest *appears then the infrared lamps are turned off after at most 6 seconds*"). The case study is motivated from automotive development processes. In this context sets of requirements often comprise of several hundred pages. The requirements are mainly written in natural language. There are a lot of involved stakeholders, many of them spread over different companies [7]. Furthermore, requirements specifications are *manually* checked for errors, e.g., by peer reviews [8].

The problem addressed in this work is that such a manual check is a considerable effort since requirements affect each other and cannot be analyzed in isolation [9]. Therefore *automatic* checks are desirable already for small sets of requirements [10,11,12]. However, to allow automatic checks the requirements need to be available in a formal language, such as, e.g., a logic like LTL. In addition to that, the new functional safety standard for Automotive Electronic Systems ISO26262 (currently under development) states that for safety critical systems at least a semi formal specification of safety requirements is highly recommended [13]. Thus, in the automotive domain a need arises for methods developing semi formal or formal requirements.

Unfortunately such formalizations are rarely accessible to the stakeholders who need to read them [14,1]. Thus, before working with a formal specification all the stakeholders would need to be trained in the chosen logic, even beyond company boundaries. In practice this is clearly unrealistic. Another possibility is to develop both a formal and an informal requirements document. But this is double work and it will be difficult to hold both documents consistently over the development time of 2-5 years.

Thus, the idea is to use a language that retains the mathematical rigor but uses a vocabulary and syntax very close to natural English language. Such a language is provided by Konrad and Cheng [1]. They propose a restricted English grammar that represents a *specification pattern system* (SPS), such that every pattern looks like natural language but can be translated into a logical formula in LTL, CTL, TCTL, RTGIL or MTL. The drawback of this language is that it covers only a subset of the statements possible to express in the particular logic. For example the following requirement cannot be expressed in the SPS as the SPS provides no pattern that expresses such a behavior: *"For a codeword with 5 digits, the time between entering the first digit and the last one is less than 5 seconds"*. However, the SPS does not claim to be complete. Thus, the question is not whether *all* possible requirements can be expressed in this SPS but: *Does* in practice *the SPS suffice to express* automotive *requirements?*

If this would be true, then automotive requirements could be expressed in SPS, i.e., they would be still readable for all stakeholders (as they look like natural language) but could be automatically checked by a computer (as they can be automatically translated into logical formulas). To investigate that question, we did a case study over 289 requirements taken from five projects from the automotive domain. The requirements were given in informal prose. For the case study we checked for every requirement in this sample whether it could be reformulated in the SPS without loss of meaning.

2 Related Work

Various tabular notations aim to provide requirements that are both accessible and suitable for formal analysis. For example, Heitmeyer et al [11] have built a variety of tools for checking consistency, completeness, and safety properties of requirements expressed in the tabular SCR notation. In contrast to the tabular notation, formalisms like B or Z [15] aim to express requirements in a mathematical way. E.g., the formalism Z defines rigid notations for logical operations, quantifiers, sets, and functions. Whereas Z is strictly designed as a specification language, the goal of B is to use mathematical proofs to verify consistency between refinement levels and to enable an automatic transformation of requirements into some executable code. However neither B nor Z offer a natural language representation.

Some research, such as the Attempto Controlled English project [16] and the work of Han [17], attempt to construct formal specifications from natural language requirements. However, automatic interpretation of natural language is still error-prone, as natural language is often vague and the information is often implicitly stated. The use of natural language in the work described here is much less ambitious.

This work investigates whether the restricted English grammar proposed by Konrad and Cheng [1] can be applied to the automotive context. The grammar depicted in Table 1 represents a specification pattern system (SPS) [1] with 16 non-recursive patterns. Every pattern can be mapped to logical statements in, e.g., the logics MTL, TCTL and RTGIL. The patterns without exact timing bounds were originally proposed by Dwyer et al in [18]. Konrad and Cheng extend these patterns in their SPS .

The patterns consist of non-literal terminals (given in a sans serif font) and literal terminals (confined via quotation marks). For example, the *min duration* pattern consists of “it is always the case that once ” P “ becomes satisfied, it holds for at least ”c “time units”, with P, c as non-literal terminals and the rest as literal terminals. The non-literal terminal P denotes a boolean propositional formulae that describes properties of states and is used to capture properties of the system. The non-literal terminal c is instantiated with integer values.

Both Dwyer et al [18] and Konrad [19] evaluate their approach with a case study. However, Dwyer evaluates the applicability only for patterns without exact timing bounds, and Konrad only for patterns with exact timing bounds. To our knowledge there are no case studies that evaluate the applicability of the SPS for mixed requirements.

The work of Dwyer et al has been extended in a number of directions. Grunske extends the patterns to express also probabilistic quality properties [20]. This extension might be useful to express availability or reliability requirements, however in our case study we consider only behavioral requirements, thus we base our work on Konrad’s SPS.

Cobleigh et al. developed the tool PROPEL (PROPerty ELucidator) that aims to guide users through the process of creating property specifications in supporting them in selecting a suited pattern and scope [21]. The tool provides

Table 1. Restricted English grammar given by Konrad and Cheng in [1]

Start	1: property	::=	*scope* "," *specification* "."
Scope	2: scope	::=	"Globally" | "Before" R | "After" Q | "Between" Q " and" R | "After" Q "until" R
General	3: specification	::=	*qualitative type* | *realtime type* | *invariant type*
Quali.	4: qualitative type	::=	*occurrence category* | *order category* | *possibility category*
	5: occurrence category	::=	*absence pattern* | *universality pattern* | *existence pattern* | *bounded existence*
	6: absence pattern	::=	"it is never the case that" P " holds"
	7: universality pattern	::=	"it is always the case that" P " holds"
	8: existence pattern	::=	P " eventually holds"
	9: bounded existence	::=	"transitions to states in which " P " holds occur at most twice"
	10: order category	::=	"it is always the case that if" (*precedence pattern* | *response pattern* | *precedence chain 1-2* | *precedence chain 2-1* | *response chain 2-1* | *response chain 1-2* | *constrained chain*)
	11: precedence pattern	::=	P "holds, then" S "previously held"
	12: precedence chain 1-2	::=	S "holds and is succeeded by" T ", then" P " previously held"
	13: precedence chain 2-1	::=	P "holds then " S " previously held and was preceded by " T
	14: response pattern	::=	P "holds then " S "eventually holds"
	15: response chain 1-2	::=	P " holds" then " S "eventually holds and is succeeded by" T
	16: response chain 2-1	::=	S "holds and is succeeded by " T ", then " P "eventually holds after " T
	17: constrained chain	::=	P "holds "then " S "eventually holds and is succeeded by" T ", where" Z "does not hold between" S "and" T
real time	18: real time type	::=	"it is always the case that" (*duration category* | *periodic category* | *RT Order category*) | *possible real time category*
	19: duration category	::=	"once" P " becomes satisfied, it holds for" (*min duration* | *max duration*)
	20: min duration	::=	"at least" c " time unit(s)"
	21: max duration	::=	"less than" c " time unit(s)"
	22: periodic category	::=	P " holds" *bounded recurrence*
	23: bounded recurrence	::=	" at least every" c "time unit(s)"
	24: RT Order category	::=	"if" P " holds, then" S "holds" (*bounded response* | *bounded invariance*)
	25: bounded response	::=	" after at most" c "time unit(s)"
	26: bounded invariance	::=	" for at least" c "time unit(s)"

both a finite-state automaton representation and a natural language representation. However, the tool currently only supports patterns without exact timing bounds.

3 Planning of the Case Study

3.1 Study Goals and Questions

In order to assess whether the SPS proposed by Konrad and Cheng can be suitably applied in the automotive domain two questions should be raised: first, is the SPS expressive enough to express automotive requirements? And, second, do developers, requirements engineers, and sub-suppliers accept the SPS, in the sense that they think it easy to use and useful?

In this case study we address only the first question. However, in order to ensure that BOSCH requirements engineers and developers can understand requirements formalized in SPS and can apply it to their requirements, we did an initial informal survey. We showed them requirements formalized in SPS, and asked them to explain their meaning to us and to apply the SPS on some of their behavioral requirements. The results indicate that requirements formalized in SPS are easy to understand and SPS is easy to apply. However, many experts asked for tool support. Therefore, we plan to develop a tool and do a further case study to address this question in a bigger context — presuming the present case study indicates the SPS is expressive enough.

Another property we investigate in this case study is the *pattern complexity*: every pattern in the SPS can be expressed in logic, but not every pattern can be expressed in every logic. We define *pattern complexity* as a function that maps a

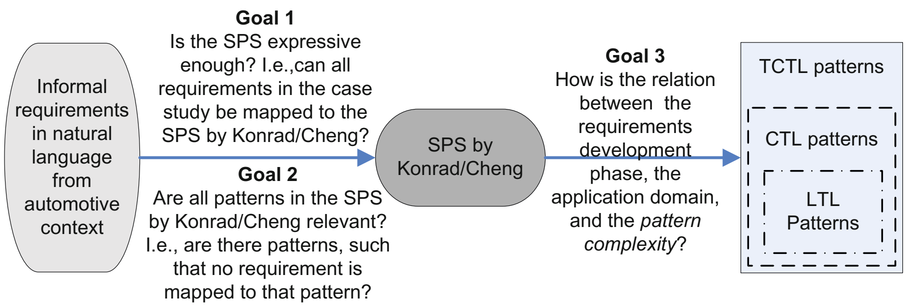

Informal requirements in natural language from automotive context
Goal 1
Is the SPS expressive enough? I.e.,can all requirements in the case study be mapped to the SPS by Konrad/Cheng?
Goal 2
Are all patterns in the SPS by Konrad/Cheng relevant? I.e., are there patterns, such that no requirement is mapped to that pattern?
SPS by Konrad/Cheng
Goal 3
How is the relation between the requirements development phase, the application domain, and the *pattern complexity*?
TCTL patterns
CTL patterns
LTL Patterns

Fig. 1. In the case study we investigate the relation between requirements and SPS, and between requirements and pattern complexity

pattern to the least expressive logic of a given finite set of logics in which it can be expressed. We are interested whether our case study contains subsets with a small *pattern complexity.*

Therefore, we identified four subgoals of the case study:

Goal 1a. Is it possible to express all automotive requirements in the SPS by Konrad and Cheng?

Goal 1b. If there are automotive requirements, that cannot be expressed in the SPS, what is the reason? I.e. could such requirements be expressed in another formalism?

Goal 2. Are all patterns in the SPS by Konrad and Cheng relevant for the automotive domain? I.e. are there patterns that are never needed?

Goal 3. What is the relation between application domain (e.g., human machine interface, engine controller,...), the requirements development phase and the pattern complexity?

As depicted in Figure 1, the subgoals 1a, 1b, and 2 investigate the relation between requirements and SPS, whereas subgoal 3 relates requirements (via the SPS) with pattern complexity.

3.2 Selection of the Sample

Selection criteria for documents. In the first step we selected requirements documents from different BOSCH projects of the automotive domain. To get a representative sampling, we decided to apply stratified sampling over the automotive application domains car multimedia, driving assistance, engine controlling, powertrain development, and catalytic converter development. Moreover, we decided to use projects from different development stages, i.e. platform and customer projects [22] from concept phases to well-known development phases[1]. We then used convenience sampling to select a project out of every stratum.

[1] *Platform projects* develop a collection of reusable artifacts, such as requirements, software components, test plans etc. These artifacts are then reused in *customer projects* in order to build applications.

Each project had several requirements documents, some consisting of more than 100 pages. In order to get a representative sample we asked the corresponding requirements engineers to give us a requirements document such that, first, the document contained behavioral requirements, and, second, the document was representative for the application domain. This way we obtained the following five requirements documents: a document specifying a Human Machine Interface(HMI) D_1 (of a car multimedia project), a document specifying an error-handling concept D_2 (of a driving assistance project), one of a controller of a heater of an oxygen sensor D_3 (of a catalytic converter project), one of a powertrain controller D_4 (of a powertrain project), and one of an engine injection device D_5 (of an engine control project).

Selection criteria for requirements. Every document $D_1, \ldots, D_5$ consisted of several chapters. In the documents process requirements (e.g., "*The testing coverage shall be at least 99% over all code statements*") or other nonfunctional requirements were mostly grouped together into chapters, thus separated from the behavioral requirements. In this work we are only interested in behavioral requirements. Therefore we scanned the chapters of the documents and made a list with the chapters containing such requirements. After that we randomly chose a chapter out of this list, and selected all requirements in the chapter. In this way we obtained the set of requirements R_1 out of $D_1, \ldots, R_5$ out of D_5. R, the union of these sets, consisted of 289 requirements. We randomly chose chapters instead of the requirements itself, as in our experience a requirement should not be interpreted out of its context.

After that we preprocessed the initial data set: we deleted 11 headings, and 22 statements, that were either mere descriptions but not real requirements (e.g., "*Most drivers prefer a smooth deceleration*"), or requirements that did not specify behavior (e.g., "*All failure thresholds shall be defined and documented by the developers.*"). Furthermore we deleted 11 redundant requirements. In order to ensure that these redundant requirements, headings and statements could be safely deleted, we discussed every deletion with 2 further experts.

After that our sample consisted of 245 informal requirements in prose.

3.3 Case Study Design

The setting of the case study is depicted in Figure 2: As input we used 245 informal requirements. These are analyzed in a content analysis with the category system defined below.

The category system consists of three main groups: *phenomenon requirements, requirements expressible in SPS* and *requirements not expressible in SPS*. The category *requirements expressible in SPS* is further refined in the patterns defined in [1].

We defined a requirement as *phenomenon requirement* if it specified not behavior but data. An example for such a requirement is *"An error is built up as following: error name (2 Byte), error status (1Byte), odometer value when the error occurred (2Byte)"*. Note that a *phenomenon requirement* cannot be

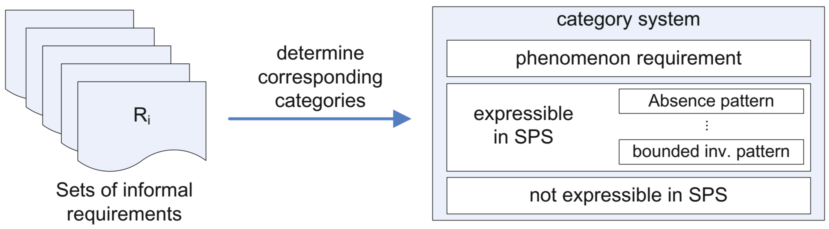


Fig. 2. case study design

mapped to a pattern, instead the data it specifies is mapped to non-literal terminals. Thus, *phenomenon requirements* are *indirectly* expressed in the SPS.

We defined a requirement as *expressible in SPS* if and only if the SPS provides one or more suitable patterns and there is an assignment to the non-literal terminals of these patterns, such that the conjunction of the instantiated patterns expresses the meaning of the requirement.

We defined a requirement as *not expressible in SPS*, if it was not a *phenomenon requirement* and could not be reformulated in the SPS without loss of meaning.

We then asked a requirements engineer to determine for every requirement in the sample whether it was a *phenomenon requirement*, a *requirement expressible in SPS* or a *requirement not expressible in SPS*. For *requirements expressible in SPS* the engineer should further give instantiated patterns expressing the meaning of the requirement.

The majority of the informal requirements could be reformulated into exactly one instantiated pattern. However, for some requirements a conjunction of multiple instantiated patterns was needed to express the meaning of the initial requirement. Thus, we obtained a set size for the resulting requirements *expressible in SPS*, *phenomena* and *not expressible in SPS* of 307. The following statistics concerning subgoal 1 and subgoal 2 relate to the initial number of requirements(245), the statistics for the subgoal 3 relates to the 307 SPS requirements.

4 Analysis of the Results

4.1 Goal 1: Expressivity of the SPS

In a first step we investigated whether it is possible to reformulate all requirements of the case study in SPS. Therefore we first measured how many requirements could be expressed in the SPS by Dwyer et al, which is limited to patterns without exact timing bounds. After that we compared these results with the measurements for the SPS by Konrad and Cheng. Figure 3 depicts the results.

The figure shows that for the requirements in the case study the extension of Konrad and Cheng strongly reduces the number of *not expressible* requirements. However, 39 requirements could not be reformulated. 25 of these requirements needed a branching time concept, not provided in the given patterns.

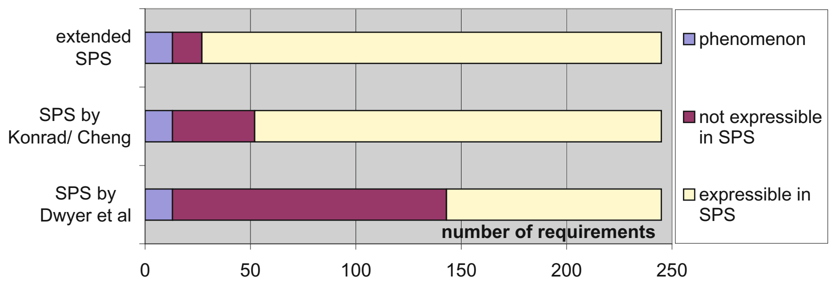


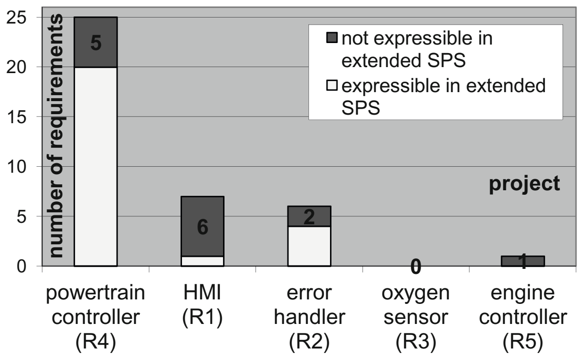
Fig. 3. The number of *not expressible* requirements is strongly reduced by Konrad and Chengs' patterns. It can be further reduced in extending the grammar with possibility patterns.

The branching concept was needed to allow the specification of *possible* behavior. Consider the requirement *If the gear is in P then it must be possible to start the engine.* In the later development phases this requirement will certainly be split into further more precise and deterministic requirements, e.g, *If the gear is in P and the ignition is turned on then the system starts the engine. If the gear is in P and the ignition is turned off, then the engine stays off....* However, in the early development phases it is desirable to allow also a less precise specification, as the information to specify the requirements in the deterministic way is probably not yet known. Thus, for early development phases we think that an extension with branching time concept patterns is needed.

Therefore, we propose to extend the grammar by Konrad and Cheng with *possibility patterns* depicted in Table 2. With the help of these patterns it is possible to reduce the number of *not expressible* requirements from 39 to 14, as depicted in Figure 3. Figure 4 depicts that in greater detail for $R_1, \ldots, R_5$: every bar represents the number of requirements, that were *not expressible* in the SPS by Konrad and Cheng. E.g., 25 requirements of R_4 were *not expressible.* The bars are further divided into two classes: the number of requirements that could be expressed by the new possibility patterns is depicted in light gray, the number of requirements that are still *not expressible* is depicted in dark gray. Note that in the case study the *possibility patterns* were especially needed for the requirements of the projects in an early development phase, particularly in R_4. This is plausible as in later development phases such requirements will be split into more precise requirements.

Table 2. Grammar of the extended patterns

possibility pattern	::=	"if" P "holds then there is at least one execution sequence such that" S "eventually holds"
possible bounded response pattern	::=	"if" P "holds then there is at least one execution sequence such that" S "holds after at most" c "time unit(s)"
possible bounded invariance pattern	::=	"if" P "holds then there is at least one execution sequence such that" S "holds for at least" c "time unit(s)"



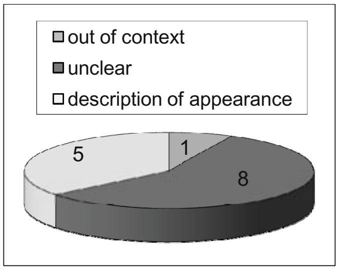


Fig. 4. The left Figure depicts the number of requirements *not expressible* in the language by Konrad/Cheng but expressible via extended patterns. On the right a further classification of requirements *not expressible in extended SPS.*

However, even with this extension it was not possible to find a reformulation in SPS for 14 requirements. Thus, we investigated what reasons made a requirement *not expressible.* We identified three reasons, depicted in Figure 4.

One *not expressible* requirement was only not expressible because it was out of the system context: *"If failures are detected in multiple electronic control units (ECUs) the same classification of faults shall be used in all ECUs"*. This requirement was given to a project developing a single ECU, but clearly this requirement cannot be solved on the ECU level as in this context it is not known what faults are detected by other ECUs. However, on the context level of the whole car the requirement could be expressed in SPS.

Five requirements could not be formalized as they did not specify any behavior. Instead they described the appearance of the product, e.g., *"the warning icon is an image of two cars with a star in between"*. As in this work we wanted to investigate only *behavioral* requirements, these requirements were wrongly selected into our sample. Thus, for the result of this case study (limited to behavioral requirements) these five requirements should be ignored.

Finally, the majority of requirements became *not expressible* because the meaning of the requirement was not clear or just too vague. Neither the requirements engineer who formalized the requirements in the first step, nor the evaluators understood what the requirement wanted to specify, thus, it was clearly impossible to formalize it. Examples for such requirements are *"The drag torque and the activation torque depend on the operating state."*(how?) and *"warning in central line of vision"*(how is the "line of vision" determined? E.g., adaptive to the size of the driver?). Thus, in these 8 cases, the problem was not that the SPS was not expressive enough but instead that the requirements were unclear. So, in fact the SPS helped to identify requirements that needed to be revised.

Thus, we come to the conclusion that the extended version of the SPS is well suited to express *behavioral* requirements from the automotive context. The majority of the behavioral requirements could be directly expressed in the SPS by [1]. With an extension of only three patterns, all behavioral requirements could be expressed as long as their meaning was clear. As only three further patterns were needed, the case study indicates that Dwyer et al may be right with their belief that *in practice* only some few patterns are needed to express requirements. The effort to extend the SPS was low, i.e., we had to add only three patterns with their formalization. Furthermore the SPS helped to identify requirements that needed to be revised. Thus it seems that for *behavioral* automotive requirements the SPS is well suited.

However, for requirements that are not behavioral requirements this claim does not hold. In fact the requirements specifying the appearance of the product were nearly impossible to formalize *in any formalism.* Thus, we think this is an indication that not all kind of requirements can be formalized. It seems that *behavioral* requirements are suitable for formalization, but for the other kinds of requirements this needs to be investigated. I.e., methods are needed that separate between requirements that can be formalized and requirements that need to be validated via other methods.

4.2 Goal 2: Pattern Relevance

Next, we evaluated whether all patterns in the SPS by Konrad and Cheng are relevant for the automotive application domain. This question needs to be asked to find a minimal pattern set. We assume that a less complex SPS with fewer patterns is easier and faster to use for developers than one with many patterns. However, we still want to express all automotive requirements. Thus, we evaluated whether the SPS by Konrad and Cheng contains patterns that were never needed in the case study.

We identified six patterns that were not needed to express any requirement in the case study: the *bounded existence, precedence chain 1-2, precedence chain 2-1, response chain 1-2, response chain 2-1* and *constrained chain* pattern. Thus, the case study indicates that for the system level in the automotive application domain these patterns might be omitted.

4.3 Goal 3: Pattern Complexity versus Application Domain

The SPS by Konrad and Cheng (and also the extended version) can be automatically transformed to the logics LTL, CTL, GIL, MTL, TCTL, and RTGIL. However, not every pattern can be translated into every logic. All provided patterns can be translated into TCTL. But, e.g., only the patterns without any reference to quantitative time or *possible* behavior can be expressed in LTL.

For LTL, CTL and TCTL tool support is available [23,24], therefore we focus in the following work on these logics. However, even for this subset of logics, there is a trade off between expressivity of the logic and its computational class. E.g., a consistency check for a set of requirements can be transformed into the satisfiability problem. However, the satisfiability problem is undecidable for TCTL [4], while decidable for LTL and CTL. Thus we are interested in the *pattern complexity* and its relation to the application domain (e.g., human machine interface, engine controller,...), and the requirements development phase. We define pattern complexity as a function that maps a pattern to the least expressive logic

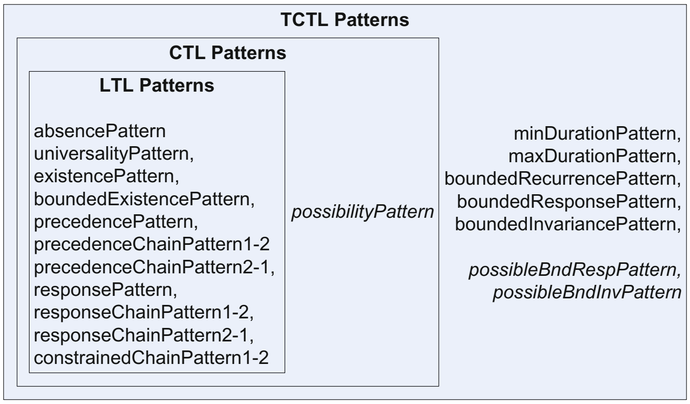


Fig. 5. The patterns are classified according to the least expressive logic they can be expressed in. The extended patterns are depicted in italic font.

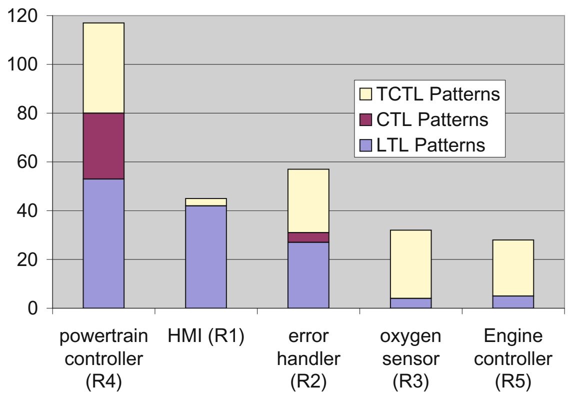


Fig. 6. TCTL-Patterns were needed in every project

∈ {LTL, CTL, TCTL} in which it can be expressed. The result of the mapping is depicted in Figure 5.

Note that all LTL Patterns can be expressed in CTL as well, and both CTL and LTL Patterns can be also expressed in TCTL. Generally, LTL is no subset of CTL. However, in this case all patterns that can be expressed in LTL can be expressed in CTL as well. We counted for every set of requirements $R_1, ..., R_5$ the number of LTL, CTL and TCTL Patterns.

As visible in Figure 6, TCTL Patterns were needed in every project. Only in the set of requirements of the earliest development phase the number of TCTL Patterns was negligible. This indicates that for the specification of the whole system functionality the need for TCTL Patterns is inevitable. However, it might be the case that individual components of the system could be solely expressed in less complex patterns. Further investigations are needed to prove or refute that thesis.

5 Threats to Validity

In this section, we analyze threats to validity defined in Neuendorf [25], Krippendorff [26], and Wohlin [27].

5.1 External Validity

Sampling validity [25]. This threat arises if the sample is not representative for the requirements. In order to minimize this threat we used the selection procedure described in Section 3.2, thus getting representative requirements of every application domain. A limitation of the case study is that we only used requirements of BOSCH projects. Thus we cannot extend our results to the whole automotive domain but only for BOSCH's automotive domain.

Interaction of selection and treatment [27]. This threat arises if the requirements engineer in this study (see Section 3.3) is not representative for BOSCH requirements engineers. However, the reliability analysis in Section 5.4 suggests that the application of the patterns is sufficiently independent of the evaluator.

5.2 Internal Validity

Selection [27]. This threat arises due to natural variation in human performance. The requirements engineer in this study (see Section 3.3) could have been especially good in formalization. The reliability analysis in Section 5.4 suggests that the application of the patterns is sufficiently independent of the evaluator. Thus variation in human performance is probably not an issue.

5.3 Construct Validity

Experimenter expectancies [27]. Expectations of an outcome may inadvertently cause the evaluators to view data in a different way. However, the evaluators have no benefit from a good or bad outcome for the applicability of the SPS as they did not invent it. Thus, such psychological effects probably did not affect the evaluators.

Semantic validity [26]. This threat arises if the analytical categories of texts do not correspond to the meaning these texts have for particular readers. In the case study the categories are clearly defined in Section 3.3 and through the patterns in the SPS [1]. However, when reformulating an informal requirement in SPS there are several possibilities to instantiate a pattern with a phenomenon that shall correspond to the description in the text. Thus, in the formalization the requirements engineer might invent phenomena without clearly defining their meaning. Discussion with experts showed that a data dictionary is a potential candidate to minimize this threat.

Face validity [25]. Face validity is the extent to which a measure addresses the desired concept, i.e. the question whether it measures what it is supposed to measure. In order to ensure face validity we discussed with experts, without mention of the case study, whether the instantiated patterns are a good representation of their concept of behavioral requirements. The discussion indicated that the patterns seem to capture typical behavioral requirements.

5.4 Conclusion Validity

Intercoder reliability [25]. Unreliable coding limits the chance to make valid conclusions based on the results. In order to minimize this threat, especially for evaluators with varying backgrounds, we asked a requirements engineer with high experience in the application domain as well as an individual with low experience in the application domain and external to the project to code a set of 30 requirements. The requirements were randomly chosen out of the sample defined in Section 3.2. In order to minimize the threat of different interpretations of the phenomena we gave both evaluators a data dictionary. The results yielded a reliability of 0.86/0.86 according to Cohen's Kappa/Scott's pi. In consequence the measure seems to be very reliable.

Even though the reliability is good we were interested in the reasons for different codings. We identified that the requirements engineer tended to use his domain knowledge when formalizing the requirements. E.g, consider the following informal requirement: *"If the locally measured voltage is not available, the voltage value as received from the bus shall be used."*. Using the phenomena *localVoltageNotAvailable, busVoltage* and *internalVoltage* with their obvious meanings, the evaluator with low experience in the application domain expressed the requirement as following *"Globally, it is always the case that if* localVoltageNotAvailable *holds, then* internalVoltage == busVoltage *holds as well"*. However, the requirements engineer specified instead *"Globally, it is always the case that if* localVoltageNotAvailable & errorGetsActive & ¬ busOff *holds, then* errorVoltage == busVoltage *holds as well"*. The engineer used the additional system knowledge, that (first) the locally measured voltage is only needed to be stored if an error appears, and (secondly) that if the bus is off, then this requirement does not apply.

Thus, differing knowledge of the system context might lead to unreliable results. We believe that this threat can be further minimized if the coders discuss their interpretation of the informal requirements prior to the formalization.

6 Conclusion

This case study investigates the question whether in practice the SPS suffices to express automotive behavioral requirements. Based on the results of Section 4.1 we think that at least for automotive requirements of BOSCH this question can be answered with yes.

The belief of Dwyer et al is that some few patterns suffice to express the majority of the properties of a system. This is only a belief and cannot be proven. Nevertheless, we think that Dwyer et als', Konrad's and our case study confirm that belief: in every case study the majority of requirements could be reformulated in the SPS. Furthermore, the distribution of the patterns indicates the same: some few patterns are extensively used whereas a lot of patterns are only sparsely used. Thus, it seems that in practice some few patterns suffice to express automotive behavioral requirements of BOSCH. However, further studies with requirements of other automotive suppliers are needed to refute or strengthen that belief for the whole automotive domain.

Furthermore the case study indicates that for BOSCH the SPS may be even reduced, as six patterns were not needed. It might be beneficent to investigate whether developers can apply such a less complex SPS even faster.

Moreover the case study shows that the SPS can be easily adapted to a certain application domain. The adaptation to the BOSCH automotive domain required the addition of three further patterns to the SPS and their translation to the logical formalisms. This effort stays reasonable as only few further patterns are needed. Last, regarding the pattern complexity, it seems that on system level there are always requirements that can be solely expressed in TCTL Patterns. It would be interesting to investigate whether this is also true on component level.

Concluding, the results indicate that it will be worth developing tool support to allow the next stage of evaluation. In the next stage it should be evaluated whether *in practice* automotive requirements engineers accept the strictures of SPS and how strong they rate the benefit of formal reasoning.

References

1. Konrad, S., Cheng, B.H.C.: Real-time specification patterns. In: ICSE 2005: Proc. 27th Int. Conf. Softw. Eng., pp. 372–381. ACM, New York (2005)
2. Emerson, E.A.: Temporal and modal logic. In: Handbook of Theoretical Computer Science, vol. B, pp. 995–1072. Elsevier Science Publishers, Amsterdam (1990)
3. Ramakrishna, Y.S., Melliar-Smith, P.M., Moser, L.E., Dillon, L.K., Kutty, G.: Interval logics and their decision procedures. TCS 170(1-2), 1–46 (1996)
4. Alur, R.: Techniques for automatic verification of real-time systems. PhD thesis, Stanford University, Stanford, CA, USA (1992)
5. Moser, L.E., Ramakrishna, Y.S., Kutty, G., Melliar-Smith, P.M., Dillon, L.K.: A graphical environment for the design of concurrent real-time systems. ACM Trans. Softw. Eng. Methodol. 6(1), 31–79 (1997)
6. Davis, A.M.: Software requirements: objects, functions, and states. Prentice-Hall, Inc., Upper Saddle River (1993)
7. Heumesser, N., Houdek, F.: Experiences in managing an automotive requirements engineering process. In: RE, pp. 322–327. IEEE Computer Society, Los Alamitos (2004)
8. Walia, G.S., Carver, J.C.: A systematic literature review to identify and classify software requirement errors. Inf. Softw. Technol. 51(7), 1087–1109 (2009)
9. Dahlstedt, A.G., Persson, A.: Requirements interdependencies - moulding the state of research into a research agenda. In: REFSQ, pp. 71–80 (2003)
10. Heimdahl, M.P.E., Leveson, N.G.: Completeness and consistency analysis of state-based requirements. In: IEEE Trans. on SW Engineering, pp. 3–14 (1995)
11. Heitmeyer, C.L., Jeffords, R.D., Labaw, B.G.: Automated consistency checking of requirements specifications. ACM Transactions on Software Engineering and Methodology 5(3), 231–261 (1996)

12. Yu, L., Su, S., Luo, S., Su, Y.: Completeness and consistency analysis on requirements of distributed event-driven systems. In: TASE, Washington, pp. 241–244 (2008)
13. ISO26262: Road vehicles - Functional safety, Part 8, Baseline 17 (2010)
14. Hall, A.: Realising the benefits of formal methods. Journal of Universal Computer Science (J. UCS) 13(5), 669–678 (2007)
15. Abrial, J.R.: Formal methods in industry: achievements, problems, future. In: ICSE, pp. 761–768. ACM, New York (2006)
16. Kuhn, T.: AceRules: Executing rules in controlled natural language. In: Marchiori, M., Pan, J.Z., de Sainte Marie, C. (eds.) RR 2007. LNCS, vol. 4524, pp. 299–308. Springer, Heidelberg (2007)
17. Han, B., Gates, D., Levin, L.: From language to time: A temporal expression anchorer. In: TIME, pp. 196–203 (June 2006)
18. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: ICSE, pp. 411–420. ACM, New York (1999)
19. Konrad, S.: Model-driven Development and Analysis of High Assurance Systems. PhD thesis, Michigan State University, East Lansing, MI (October 2006)
20. Grunske, L.: Specification patterns for probabilistic quality properties. In: ICSE, pp. 31–40. ACM, New York (2008)
21. Cobleigh, R.L., Avrunin, G.S., Clarke, L.A.: User guidance for creating precise and accessible property specifications. In: FSE, pp. 208–218. ACM, New York (2006)
22. Pohl, K., Böckle, G., Linden, F.J.v.d.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer, USA (2005)
23. Behrmann, G., David, A., Larsen, K.G.: A tutorial on uppaal, pp. 200–236 (2004)
24. Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, p. 359. Springer, Heidelberg (2002)
25. Neuendorf, K.A.: Content Analysis Guidebook. Sage Publications, Thousand Oaks (2002)
26. Krippendorff, K.H.: Content Analysis: An Introduction to Its Methodology, 2nd edn. Sage Publications, Inc., Thousand Oaks (2003)
27. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in software engineering: an introduction. Kluwer Academic Publishers, Norwell (2000)