Reassessing the Pattern-Based Approach for Formalizing Requirements in the Automotive Domain

Predrag Filipovikj
Mälardalen University
Västerås, Sweden
predrag.filipovikj@mdh.se

Mattias Nyberg
Scania
Södertälje, Sweden
mattias.nyberg@scania.com

Guillermo Rodriguez-Navas
Mälardalen University
Västerås, Sweden
guillermo.rodriguez-navas@mdh.se

***Abstract*—The importance of using formal methods and techniques for verification of requirements in the automotive industry has been greatly emphasized with the introduction of the new ISO26262 standard for road vehicles functional safety. The lack of support for formal modeling of requirements still represents an obstacle for the adoption of the formal methods in industry. This paper presents a case study that has been conducted in order to evaluate the difficulties inherent to the process of transforming the system requirements from their traditional written form into semi-formal notation. The case study focuses on a set of non-structured functional requirements for the Electrical and Electronic (E/E) systems inside heavy road vehicles, written in natural language, and reassesses the applicability of the extended Specification Pattern System (SPS) represented in a restricted English grammar. Correlating this experience with former studies, we observe that, as previously claimed, the concept of patterns is likely to be generally applicable for the automotive domain. Additionally, we have identified some potential difficulties in the transformation process, which were not reported by the previous studies and will be used as a basis for further research.**

I. Introduction

New and emerging technologies are being incorporated into vehicles, leading to a significant increase of the number and complexity of the implemented functions. Currently, in industrial settings, manual inspection (peer review) is the most widespread technique for checking the correctness of the systems' specification [13]. Performing these reviews manually is becoming more difficult as the complexity of the specifications increase. Additionally, the fact that the reviewing peers may be different stakeholders, with different backgrounds and interests, and even working at different companies makes it more difficult to guarantee that the specification was properly understood and validated [6].

In these conditions, it is clear that having tool support for computer-assisted verification of the specifications would speed up the process and be beneficial. The new ISO26262 standard for functional safety of the road vehicles acknowledges this, by advocating the application of formal verification techniques at each level of system abstraction [8].

One of the main problems in order to introduce formal verification techniques in industry is the lack of formal specifications of the systems. The common practice in the automotive industry nowadays is to specify system requirements in natural language, typically using a general purpose text editor. Requirements written like this are ambiguous, cannot be easily processed by computers and provide limited traceability; they are also most likely to be incomplete and/or inconsistent. The most desirable situation would be to have requirements expressed in some kind of logics, for instance with temporal logics such as LTL [5], CTL [5] or TCTL [1]. Requirements expressed as such can be subject to different forms of automated analysis, like consistency checks and others.

It is not realistic to believe that automotive engineers will be able to easily express system requirements using formal notations. First of all, automotive engineers are in general very knowledgeable about disciplines like mechanics, hydraulics, electronics, mechatronics, etc., but they are not so skilled in computer science and discrete mathematics. Teaching every engineer how to specify properties with temporal logics would require a great amount of time, and would be too costly to be feasible. Furthermore, it is not only the engineers who need to understand the requirements. Other stakeholders at different levels of the organization, e.g. customer service or maintenance service, must process the requirements and validate them according to their specific needs [4].

Several researchers believe that this gap can be filled with the help of software tools that will assist the engineers in the process of *automated transformation* of requirements from natural language into temporal logics [2] [9] [11]. This process is often called *formalization of requirements*. There exist some interesting tools on the market providing this functionality [7], but they are not widely accepted. More research is still needed for understanding how the automated formalization of requirements will fit into the development process of an organization.

This paper presents a preliminary study that has been conducted in order to gain further understanding of the benefits, limitations and challenges encountered when formalizing requirements in a realistic setup. This study has been performed in collaboration with Scania, one of the leading Swedish truck manufacturers. The goal of the case study is to take a

978-1-4799-3033-3/14 © 2014 IEEE

RE 2014, Karlskrona, Sweden

small subset of non-structured functional requirements from the E/E systems written in natural language and formalize them using the approach known as Real Time Specification Patterns, developed by Konrad and Cheng [10]. These patterns are based on the set of Specification Pattern System (SPS), initially defined by Dwyer et al. [3], but are enriched with patterns that can capture the timing aspect in the requirements. The work by Konrad and Cheng also proposed a system of restricted English grammar, through which the requirements can be expressed in specification patterns more naturally from a linguistic point of view. These patterns have been applied at least in one case study in the automotive domain [12]. A significant advantage of this approach is that each pattern maps into temporal logics, which makes the transition from the restricted English into formal notation automated. This paper summarizes the knowledge gained from this case study, with special emphasis on the challenges faced during the process, and reports future directions for research.

II. Description and Setup of the Case Study

As already indicated, Post et al. successfully applied Real Time Specification Patterns in a case study within the automotive domain [12]. Our case study is inspired by that experience but also presents some differences. It is similar in the sense that a set of industrial non-structured requirements will be taken and converted into patterns; we will call this process *patterning* of the requirements. But it differs in the way the requirements are gathered and filtered before the patterning happens.

A. Real Time Specification Patterns

In our case study we use the Real Time Specification Pattern System (RTSP) as defined by Konrad and Cheng [10]. This set of patterns has been chosen because it provides a quantitative notion of time, which is needed for formalization of the real-time requirements. Table I shows these patterns together with their representation in restricted English grammar; the list contains 17 specification patterns. All these patterns share the characteristics of being non-recursive and prone to an automated transformation into temporal logics such as LTL, CTL and TCTL. One should notice that there is no direct mapping of *each* pattern from the RTSP into *every* temporal logics, as there can be patterns with semantics that cannot be expressed in some of the previously mentioned formal notations.

Each pattern is constituted by literal and non-literal terminals. The non-literal terminals can be either boolean expressions describing system properties, or integer values capturing timing aspects. The remaining parts of the pattern are the literal terminals, which cannot be changed. For example in the *Precedence pattern* given as: "it is always the case that if *P* holds, then *S* previously held", *P* and *S* represent non-literal terminals and the remaining of the pattern are literal terminals. For each pattern an extent of program execution for which the requirement holds must be defined. As defined by Dwyer et al. [3], there are five scopes of program execution: *Globally* (the entire program execution), *Before Q* (until the first occurrence of the state/event Q), *After Q* (after the occurrence of the state/event Q), *Between Q and R* (any part of the program execution between the two states/events Q and R) and After Q until R (similar behavior as Between scope except that the execution continues even if the second state/event never occurs). The graphical representation of the scopes is given in Figure 1.

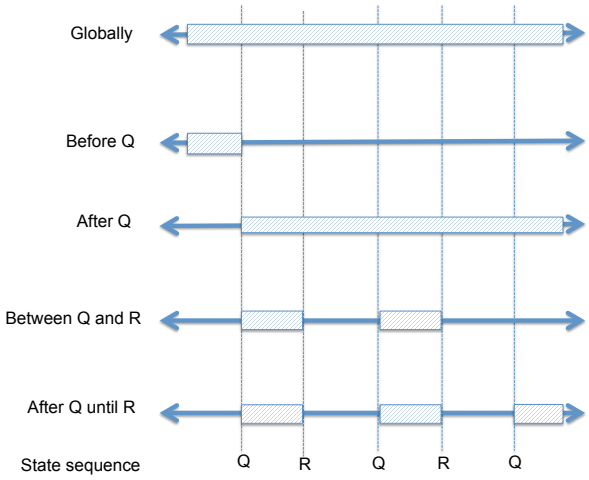

Fig. 1: Pattern scopes [12].

B. Requirements Gathering

In order to collect the requirements for the study, some engineers from Scania were contacted and asked to send requirements documents exemplifying their usual work. In response, we received four documents with a number of non-structured functional requirements written in natural language. Three of the documents were written as MS Word documents, and the other one was written in DOORS. In total, one hundred requirements were gathered. Although this number is small compared to the total number of requirements of a system, the gathered data was representative enough for the first evaluation of the patterning process.

Regarding the contents, one of the documents included requirements relative to user functions, or UFR in Scania terminology, whereas the other three contained requirements describing the behavior at the sub-system level, known as Allocation Element Requirements (AER) in Scania terminology.

After this gathering, the chosen requirements were extracted from the documents and stored in a data sheet, disposing any other information. This guarantees that only the statements marked as requirements are assessed, and not context information or other meta-data related to them. This gives an indirect measure of the quality of the requirement with respect to automated transformation: if a requirement cannot be patterned without knowing other information appearing in the document, it will most likely not be prone to an automated formalization.

A difference with respect to the previous case study is that the requirements were extracted from the documents by the researchers (not by the engineers) as they were provided,

TABLE I: Restricted English grammar patterns [10].

Start	property	*scope , specification .*
Scope	scope	*Globally | Before R | After Q | Between Q and R | After Q until R*
General	specification	*qualitativeType | realtimeType*
Qualitative	qualitativeType	*occurrenceCategory | orderCategory*
	occurrenceCategory	*absencePattern | universalityPattern | existencePattern | boundedExistencePattern*
	absencePattern	*it is never the case that P holds*
	universalityPattern	*it is always the case that P holds*
	existencePattern	*P eventually holds*
	boundedExistencePattern	*transitions to states in which P holds occur at most twice*
	orderCategory	*it is always the case that if P holds (precedencePattern | precedenceChainPattern1-2 | precedenceChainPattern2-1 | responsePattern | responseChainPattern1-2 | responseChainPattern2-1 | constrainedChainPattern1-2)*
	precedencePattern	*, then S previously held*
	precedenceChainPattern1-2	*and is succeeded by S , then T previously held*
	precedenceChainPattern2-1	*, then S previously held and was preceded by T*
	responsePattern	*, then S eventually holds*
	responseChainPattern1-2	*, then S eventually holds and is succeeded by T*
	responseChainPattern2-1	*and is succeeded by S , then T eventually holds after S*
	constrainedChainPattern1-2	*, then S eventually holds and is succeeded by T , where Z does not hold between S and T*
Real-time	realtimeType	*it is always the case that (durationCategory | periodicCategory | realtimeOrderCategory)*
	durationCategory	*once P becomes satisfied, it holds for (minDurationPattern | maxDurationPattern)*
	minDurationPattern	*at least c time unit(s)*
	maxDurationPattern	*less than c time unit(s)*
	periodicCategory	*P holds boundedRecurrencePattern*
	boundedRecurrencePattern	*at least every c time unit(s)*
	realtimeOrderCategory	*if P holds, then S holds (boundedResponsePattern | boundedInvariancePattern)*
	boundedResponsePattern	*after at most c time unit(s)*
	boundedInvariancePattern	*for at least c time unit(s)*

specifically without any filtering or preprocessing. In contrast, Post et al. [12] only considered behavioral requirements, and were preprocessed before the formalization. We proceed differently, because our goal is to investigate to what extent the considered specification patterns can help the verification effort. Since engineers have to verify all kinds of requirements, behavioral and non behavioral, and their starting point is typically the existing documentation, we preferred to stay as close as possible to those conditions.

C. Requirements Patterning

The patterning of the requirements was performed sequentially, one by one, by accessing only the information available in the data sheet. It was preceded by a phase in which the researchers studied the specification patterns and prepared a list containing all the patterns described in [12], which would be consulted during the process. None of the researchers involved in the process had previous experience with the specification patterns.

The patterning process consisted in the following tasks: i) identifying which pattern should be applied to the requirement and after that, ii) writing the requirement in restricted English grammar according to the chosen pattern. Since the goal of the exercise was to assess the expressiveness and adequacy of the patterns, there was no need to proceed further and obtain the expressions in temporal logics. For the purpose of our research, we claim that a requirement is *formalizable* if there is a pattern that captures its semantics.

In some occasions, the level of ambiguity of a requirement would make it difficult to discriminate between possibly applicable patterns. In such cases, we contacted the responsible engineer and discussed the meaning of that particular requirement. This was needed for instance for determining the scope of validity of a requirement or for understanding the exact order of events. These requirements are still considered formalizable.

Some requirements could not be expressed through patterns for the reasons that will be described in Section III. Such requirements are said to be *non-formalizable*.

III. Analysis of the Results

In this section we present details about the results achieved after the patterning process.

A. Pattern Expressiveness

The results from the formalization process are presented in Figure 2. Around 70% of the requirements could be formalized with patterns; among the remaining 30% of requirements non-formalizable with patterns, there is an important group, called *phenomenon* requirements, which can still be formalized by

other means, as it will be described later on in this section. Therefore, the proportion of non-formalizable patterns is 6%.

Phenomenon Requirements: The term *phenomenon requirement* was coined by Post et al. to refer to a requirement that does not express system behavior, but gives information about data or the system configuration. These requirements cannot be mapped into a pattern, but can be expressed by means of non-literal terminals [12]. An example of Scania phenomenon requirement is:

"The signal totalFuelLevel shall receive its value from externalTotalFuelLevelIn."

Some of the difficulties encountered in the patterning process of the phenomenon requirements is that often the scope of execution is not clearly defined. Another issue with these requirements is that sometimes they do not explicitly include sufficient data to be patterned. Let us take, for example, the requirement below, which appeared in one of the requirement documents. Note that this requirement is in fact composed by three requirements of different type, which can be extracted and considered as separate entities.

"Signal lowFuelLevelWarning shall be set to Active when input totalFuelLevel is below a predefined level. This level shall be 10% for tank size equal to or below 900 liters and 7% for tank sizes larger than 900 liters. The tank size is determined by the parameters fuelTankSizeLeft and fuelTankSizeRight."

The statement above decomposes in the following requirements:

1) *"Signal lowFuelLevelWarning shall be set to Active when input totalFuelLevel is below a predefined level."*
2) *"This level shall be 10% for tank size equal to or below 900 liters and 7% for tank sizes larger than 900 liters."*
3) *"The tank size is determined by the parameters fuelTankSizeLeft and fuelTankSizeRight."*

Only the first statement is a behavioral requirement, since it captures the behavior of the system after the *lowFuelLevelWarning* signal reaches some threshold (indicated as the *predefined level*). The second requirement is a phenomenon requirement used in addition to the previous requirement in order to accurately define what *predefined level* means. There is no pattern to represent this requirement, but it can be formalized using non-literal statements.

The third requirement is also a phenomenon requirement, but it cannot be formalized. It simply indicates that there is a relationship between the tank size and the parameters *fuelTankSizeLeft* and *fuelTankSizeRight*, but the exact relationship is not given. It is not possible to formalize such requirement, unless the missing relationship is defined with assistance from the engineers. Interestingly, when asked about the missing information in the requirement, the engineer answered that this information was omitted because it was considered trivial. It is well known that the existence of this *"domain knowledge"* introduces ambiguity and represents a challenge for the specification of requirements in general. For the case of formalization of requirements, there is a need to convert this domain knowledge into formal expressions, which is also challenging.

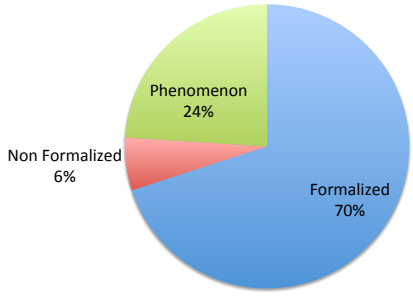

Fig. 2: Formalization results.

Formalized Requirements: The second group of requirements includes the requirements that were successfully expressed with patterns. These requirements can be further divided into two big categories:

1) Requirements formalized without help of the engineers.
2) Requirements formalized with the help of the engineers.

The requirements falling into the first category typically looked like the following:

1) *"If lowFuelLevelWarningParam = 0, output lowFuelLevelWarning shall be set to Take No Action."*
2) *"When DTC sensorShortToGround or sensorShortToBattery is set, the status of input signal fuelLevelSensor shall be set to Error"*
3) *"The CMS shall send a valid value in totalFuelLevel within 2 seconds from when the ECU starts."*

In these examples, the behavior is clearly stated and the scope of the program execution can be identified, so the corresponding pattern can be found without requiring further assistance. We expressed the first two requirements with a *Response* pattern, and the third one with a *Bounded Response* pattern:

1) *"Globally, it is always the case that if (lowFuelLevelWarningParam = 0) holds, then (lowFuelLevelWarning = 'Take No Action') eventually holds."*
2) *"Globally, it is always the case that if (DTC sensorShortToGround or sensor ShortToBattery is set) holds, then (fuelLevelSensor = Error) eventually holds."*
3) *"Globally, it is always the case that if (ECU was started) holds, then (CMS sent valid signal to totalFuelLevel) holds after at most 2 seconds."*

However, some of the requirements that could be formalized without engineers' assistance raised an interesting concern. It may happen that different patterns express semantics that are not easily distinguishable by a non-expert. This, combined with the inherent ambiguity of the natural language in

which the requirements are written, may result in a wrong selection of a pattern; where wrong means "not conveying the intention of the person writing the requirement". Such cases of incorrect disambiguation of requirements are particularly interesting because they may yield the results of the associated verification activities useless. As part of a larger process, the pattern selection should be validated by other means. However, analyzing the problem of validating the correctness of the pattern selection is out of the scope of this paper, and should be addressed only in a larger empirical case study with engineers taking part in the patterning process actively. In this work, a requirement is considered formalizable if there is at least one pattern that expresses the meaning of the requirement correctly as perceived by the researchers. For instance, let us consider the following requirement:

"When parkingBrakeApplied has status 'Error' or 'Not Available' the replacement value 'Not Set' shall be used"

To the best of our knowledge, the semantics of the above requirement can be expressed with either *Response* pattern with *Global* scope or with an *Universality* pattern with restricted scope *After Q*. For clarification, we present the requirement expressed through both patterns, followed by their corresponding TCTL representation:

- *"Globally, it is always the case that if (parkingBrakeApplied = 'Error' or parkingBrakeApplied = 'Not Available') holds, then (parkingBrakeApplied = 'Not Set') eventually holds."*

```
AG[((parkingBrakeApplied = 'Error') OR
(parkingBrakeApplied = 'Not Available')) →
AF(parkingBrakeApplied = 'Not Set')]
```

- *"After ((parkingBrakeApplied = 'Error') or (parkingBrakeApplied = 'Not Available')), it is always the case that (parkingBrakeApplied = 'Not Set') holds."*

```
AG[((parkingBrakeApplied = 'Error')
OR (parkingBrakeApplied = 'Not Available'))
→ AG(parkingBrakeApplied = 'Not Set')]
```

With the information given in the requirement alone, it is difficult to determine which of the patterns expresses the correct behavior to the full extend. The problem is that both patterns capture behaviors that are similar, but not equivalent, for instance because they are within different scopes. Potential causes of these problems are: the ambiguity of the requirement and the apparent overlapping between different patterns. The implications of choosing one pattern over the other are only evident in the following stages of the verification process, which are not considered in this case study.

Regarding the requirements for which assistance from the engineers was required, the most common difficulties were determining the scope of execution and understanding the underlying meaning. According to the Dwyer et al. [3], there are five possible execution contexts for the patterns. Interestingly, none of them captures the moment of entering a particular state or executing a specific event. For example, let us consider the following requirement:

"Output signal lowFuelLevelWarning shall have the initial value 'Not Active' at start-up."

This requirement imposes that the lowFuelLevelWarning signal shall receive some value when the system is *"in"* some specific state (*start-up*). None of the scopes captures the moment when the system is in a particular state but only *Before* and *After*, so we had to consult with the engineer in order to reformulate the requirement and expressed it with one of the defined scopes. For this particular requirement the *After Q* scope was applied.

Another problem identified was that sometimes concepts from different level of abstractions were used, without specifying the meaning of the high level concepts. This problem may be related to the absence of phenomenon requirements that complement the semantics of the requirement, but it needs further investigation. For example, the following requirement is apparently well specified, but contains ambiguity:

"At the shut down, the last value of the totalFuelLevel shall be stored until next start-up."

In this requirement, concepts from different levels of abstraction are present: the *shut-down* and *start-up* states/events belong to a higher abstraction level and are not properly specified, so their meaning is ambiguous. More importantly, it is not clear what *"the last value"* in this context is, and how and where it should be stored. For such requirements patterning is impossible unless an engineer disambiguate them.

***Non Formalized Requirements**:* The third type of requirements are the non formalized requirements. According to our experience, high complexity, high level of ambiguity and lack of information are the main reasons impeding the patterning of such requirements. We present one representative from this group:

"Signal totalFuelLevel shall be output of a filter that includes information from both FLS_vol and fuelRate to achieve a stable signal. The filter shall be implemented with a Kalman algorithm given by equations (2-4) with a feedback gain K."

In this requirement, the relationship between the input and the output parameters has been defined with an analytical expression that is not present in the text. There is no approach that deals with transformation of analytical expressions into logic, so that the relationship between the input and the output parameters in this requirement cannot be formalized. For this reason, the requirement was classified as not formalizable.

B. Pattern Frequency

The distribution of the patterns used for formalizing the requirements in this case study is given in Figure 3. Note that the distribution includes only the requirements formalized with patterns, and not the phenomenon requirements. From a total of seventeen patterns presented by Konrad and Cheng [10], seven were used in this case study. The distribution of the

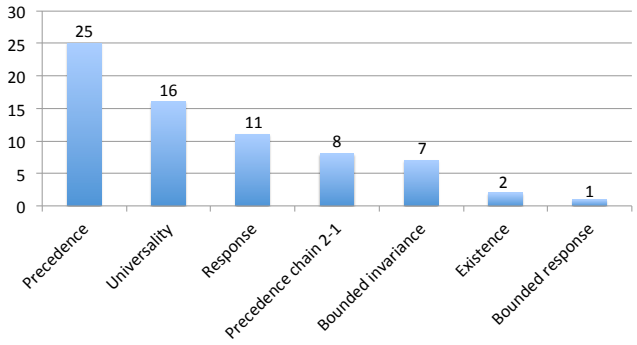

Fig. 3: Pattern frequency.

patterns is coherent with Post et al. [12] case study; this is an indication that a small number of patterns are enough to express the majority of the requirements in the automotive domain. The pattern most frequently used is *Precedence* followed by *Universality*. It is interesting is that the *Precedence chain 2-1* pattern was used in our case study, but not in Post et al's.

Another interesting difference between our results and the ones presented in the previous case study is the ratio between the *Precedence* and the *Response* patterns. In our case, most of the properties are expressed by using the *Precedence* pattern, whereas Post et al's case study uses *Response* pattern more often. These two patterns are very similar but not equivalent. This is due to the fact that the *Response* allows effects to occur without causes, while the *Precedence* allows causes to occur without effect. A number of factors can influence the application of both types of patterns, and we believe that the ambiguity of the requirements as well as the understanding of the underlying industrial systems are the most common ones. Our hypothesis is that within the verification phase, it will be possible to understand which one conveys better the requirement as originally intended by the engineer.

IV. Reflection on the Experience

According to our observations and expectations, the patterning process resulted into a number of benefits. One is an apparent reduction of ambiguity, due to the fact that the patterns tend to capture the *"domain knowledge"* information, which is usually omitted in the requirements expressed in natural language. Additionally, in our conversations with the engineers, the patterns provided a useful support for discussing about the meaning of certain requirements. This is expected to impact positively on the communication between stakeholders, and also to improve testability of requirements.

The case study was completely focused on testing the expressiveness of the specification patterns on a functional requirements from the electrical and electronic systems installed in trucks. Driven by this specific goal, we did not take any precise measures about the effectiveness of the approach represented as patterning time per requirement. However, according to our observations, the patterning process was conducted in a reasonable time frame, thus making it applicable in industrial settings.

The case study also revealed certain limitations of the approach. Using patterns based on restricted English grammar eliminates the need to use mathematical expressions for formalizing the requirements, but the process still requires a significant effort. We observed that as the patterning process advanced, the researchers involved in the experiment became more proficient in using the patterns and also on the understanding of the requirements, which resulted in reducing the patterning time per requirement. But finding ways of softening this learning curve is desirable.

The main challenge of the patterning process is understanding the meaning of the patterns and the scopes of execution over which the patterns hold. Post et al. [12] reported similar problems with respect to choosing the scope. Their experiment showed that the engineers favor the *Global* scope because of its simplicity. According to them, this happens because the *Global* scope captures the entire program execution, and thus it is easier to grasp. Our case study, however, showed the opposite tendency. The researchers tend to understand the patterns better and have an inclination to use as much of the defined patterns and scopes as possible. Therefore, even though the extensive use of the *Global* scope of execution can be accepted in most of the cases, finding more precise definition of the scope became a priority.

Although we believe that using restricted scopes is desirable, the implications of the effect from this decision is only visible in the next phases of the verification process. It remains as an open problem that will need further investigation.

V. Conclusion

This paper has discussed a case study intended to assess the suitability of using the specification patterns [10] for formalizing E/E requirements in the automotive domain. The goal was to evaluate whether such patterns provide enough expressiveness and are, at the same time, easy to apply. Even though the case study was only a preliminary study of the problem, with some inherent limitations in size, a few interesting conclusions have been drawn.

The results of applying specification patterns to our set of requirements can be considered satisfactory. Among the total set of requirements collected initially, 70% were formalized with patterns. Considering only the behavioral requirements, 92% of them were formalized. According to our observations, the patterning process reduced the ambiguity of the requirements and showed promising potential as a support to communication.

Our experience indicates that the patterns in their current form (a theoretical framework) are difficult to introduce in the industrial process. The patterns must be accompanied with support material, such as graphical representations and examples, which will make them more comprehensible for the engineers. Specifically, both the identification of patterns and the selection of scope of execution can be difficult sometimes, and may lead to wrong patterning; some extra information

is needed to avoid this problem. In addition, providing user-friendly tool support will certainly favor industrial acceptance of the methodology and is most desired.

As a conclusion, the preliminary results showed that there is a strong need for conducting a more extensive case study including more engineers from different departments and more Scania E/E systems. To further increase the relevance of the results, we are also working towards including other companies from the automotive domain into the case study. In the new setup we expect to gain deeper understanding about the patterning process of the requirements, with a particular emphasis on identifying the engineers' needs. This data will be used for providing a tool to adequately support the process with a high level of correctness. However, even with an appropriate tool support, wrong interpretations may still happen in this initial phase, and the causes should be identified and investigated. When possible, these cases should be correlated to the engineers' background. The future case study will focus on accurately measuring the time required for patterning the requirements, in order to be able to draw statistical data such as formalization time per requirement. Additional effort is also needed for identifying the right metrics to study the quality of the requirements achieved through formalization as well as measuring (qualitatively) the engineers' perception about the benefits achieved through such formalization.

ACKNOWLEDGMENT

This work was funded by the Swedish Governmental Agency for Innovation Systems (VINNOVA) under project 2013-01299.

REFERENCES

[1] R. Alur. *Techniques for automatic verification of real-time systems.* PhD thesis, Stanford University, Stanford. 1992.

[2] R. L. Cobleigh, G. S. Avrunin, and L. A. Clarke. *User guidance for creating precise and accessible property specifications.* In Proc. of ACM SIGSOFT Found. on Soft. Eng. (FSE). 2006. pp. 208-218.

[3] M. B. Dwyer, G. S. Avrunin, J. C. Corbett. *Patterns in property specifications for finite-state verification.* In Proceedings of the 21st international conference on Software engineering (ICSE '99). ACM, New York, NY, USA. 1999. pp.411–420.

[4] M. Elizabeth, C. Hull, K. Jackson, J. Dick. *Requirements Engineering, Third Edition.* Springer. 2011.

[5] E. A. Emerson. *Temporal and modal logic*. In Handbook of Theoretical Computer Science. Amsterdam, Netherlands: Elsevier, 1995. pp. 995–1072.

[6] N. Heumesser, F. Houdek. *Experiences in managing an automotive requirements engineering process.* In: RE, IEEE Computer Society. 2004. pp. 322–327.

[7] H. J. Holberg, U. Brockmeyer. *ISO 26262 compliant verification of functional requirements in the model-based software development process.* White paper. Embedded World Exhibition and Conference. 2011.

[8] International Organization for Standardization. *ISO/DIS 26262-1 - Road vehicles Functional safety.* International Organization for Standardization / Technical Committee 22 (ISO/TC 22). Geneva, Switzerland. 2009.

[9] S. Konrad and B. H. Cheng. *Facilitating the construction of specification pattern-based properties*. In Proc. of the IEEE Int. Req. Eng. Conf. (RE). 2005. pp. 329-338.

[10] S. Konrad, B. Cheng. *Real-time specification patterns*. In Proceedings of 27th International Conference on Software Engineering. 15–21 May 2005. pp. 372–381.

[11] S. P. Overmyer, B. Lavoie, and O. Rambow. *Conceptual modeling through linguistic analysis using LIDA*. In Proc. of the IEEE Int. Conf. on Soft. Eng. (ICSE). 2001. pp. 401-410.

[12] A. Post, I. Menzel, J. Hoenicke, A. Podelski. *Automotive behavioral requirements expressed in a specification pattern system: a case study at BOSCH.* In: Requirements Engineering Journal 17. March 2012. pp. 19–33.

[13] GS. Walia, J.C. Carver. *A systematic literature review to identify and classify software requirement errors.* Inf. Softw. Technol. 2009. pp. 51(7):1087-1109.