Formal, Model- and Scenario-based Requirement Patterns

Markus Fockel, Jörg Holtmann, Thorsten Koch and David Schmelter
Software Engineering Department, Fraunhofer IEM, Paderborn, Germany

Keywords: Requirement Patterns, Modal Sequence Diagrams, Safety Requirements, Real-time Requirements.

Abstract: Distributed, software-intensive systems such as automotive electronic control units have to handle various situations employing message-based coordination. The growing complexity of such systems results in an increasing difficulty to achieve a high quality of the systems' requirements specifications. Scenario-based requirements engineering addresses the message-based coordination of such systems and enables, if underpinned with formal modeling languages, automatic analyses for ensuring the quality of requirements specifications. However, formal requirements modeling languages require high expertise of the requirements engineers and many manual iterations until specifications reach high quality. Patterns provide a constructive means for assembling high-quality solutions by applying reusable and established building blocks. Thus, they also gained momentum in requirements documentation. In order to support the requirements engineers in the systematic conception of formal, scenario-based requirements specification models, we hence introduce in this paper a requirement pattern catalog for a requirements modeling language. We illustrate and discuss the application of the requirement patterns with an example of requirements for an automotive electronic control unit.

1 INTRODUCTION

Distributed, software-intensive systems such as electronic control units within vehicles have to handle various, often real-time- and safety-critical situations employing message-based coordination (e.g., via bus communication). The growing complexity of such systems also results in an increased size of their requirements specifications. Accompanied by that, a high quality of such specifications is more difficult to achieve if the requirements engineers rely on manual requirements validation techniques. A high quality of a requirements specification encompasses characteristics like completeness and consistency (ISO, 2011).

Scenario-based requirements engineering addresses the message-based coordination of such systems and enables, if underpinned with formal languages, automatic requirements validation techniques for improving the quality of a requirements specification. One of such requirements engineering approaches bases on a recent visual Live Sequence Chart (Damm and Harel, 2001) variant compliant to the UML (Object Management Group, 2015), so-called *Modal Sequence Diagrams* (MSDs) (Harel and Maoz, 2008). The formal semantics of this requirements engineering approach enable automatic requirements validation techniques, like simulation considering assumptions on the environment (Brenner et al., 2013) and real-time requirements (Brenner et al., 2014), as well as a formal consistency check based on the technique of controller synthesis (Greenyer et al., 2013).

However, formal requirements modeling languages like MSDs require a deep knowledge by the requirements engineers and typically require many manual and hence costly iterations until specifications become complete and consistent. That is, the requirements engineers initially conceive a typically underspecified and inconsistent requirements specification. Afterward, they iteratively apply the simulation as well as the consistency check and incrementally improve the underspecified and inconsistent parts of the specification until a reasonable solution is found.

In order to facilitate and systematize this sophisticated and complicated procedure, we introduce in this paper a model- and scenario-based pattern catalog for MSD requirements. The usage of patterns is known to be constructive thanks to assembling solutions by means of reusable building blocks that are proven in practice, so that recurring problems do not need to be solved over and over again (Alexander et al., 1977). Thus, patterns also gained momentum in the area of requirements documentation (e.g., (Chung et al., 2016)). Our MSD requirement pattern catalog consolidates and unifies patterns from three well-known, practice-oriented pattern catalogs (Dwyer et al., 1999;

Fockel, M., Holtmann, J., Koch, T. and Schmelter, D.
Formal, Model- and Scenario-based Requirement Patterns.
DOI: 10.5220/0006554103110318
In *Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2018)*, pages 311-318
ISBN: 978-989-758-283-7
Copyright © 2018 by SCITEPRESS – Science and Technology Publications, Lda. All rights reserved

Konrad and Cheng, 2005; Bitsch, 2001), covering the aspects chronological succession, real-time, and safety, respectively. We illustrate and discuss the application of the MSD requirement patterns with example requirements for an automotive rear door system.

This paper is structured as follows. In Section 2 we discuss existing research on requirement patterns. Section 3 contains the basics on Modal Sequence Diagrams. In Section 4 we introduce the pattern catalog and some pattern examples. In Section 5 we discuss the results, and we conclude in Section 6.

2 RELATED WORK

In this section, we summarize existing research on pattern- and scenario-based engineering and align our contributions with this work.

Formal, textual property specification *languages* like Computation Tree Logic (CTL) (Emerson and Clarke, 1982) and Timed CTL (TCTL) (Alur et al., 1993) are common in model checking and provide a means to assert the correctness of state-based designs through mathematical reasoning (Baier and Katoen, 2008). Property specification *patterns* provide well-established rules that help property specification language practitioners in creating quality specifications. Autili et al. present a unified catalog of property specification patterns (Autili et al., 2015). They combine four existing catalogs of qualitative (i.e., chronological succession), real-time, and probabilistic patterns. They provide a natural language front-end to specify properties. With MSDs, we provide a model- and scenario-based visual language. Autili et al. include probabilistic patterns. As we focus requirements engineering for message-based communication of safety-critical, software-intensive systems that have to meet requirements to 100%, we do not address such patterns. In contrast, we include safety requirement patterns (cf. (Bitsch, 2001)) in our catalog which Autili et al. do not address.

Konrad and Cheng present a model-based requirement pattern catalog (Konrad and Cheng, 2002). Their catalog targets embedded systems and utilizes UML for modeling structural and behavioral requirements aspects. The catalog encompasses ten patterns that address typical embedded system use cases, e.g., for specifying requirements on actuators/sensors (structural pattern) or on fault handling (behavioral pattern). Konrad et al. present a second catalog on object analysis patterns in (Konrad et al., 2004) that is comprised of seven patterns. This second catalog is based on (Konrad and Cheng, 2002) and targets the system analysis phase. Especially, they show how to apply automatic analysis with a property specification language and a model checker based on object models created with these patterns. Konrad et al. provide patterns for use cases of requirements engineering for embedded systems. In contrast, our catalog targets the message-based communication of these systems. Moreover, they do not address real-time requirements and analysis techniques.

Requirement boilerplates, e.g., by (Mavin et al., 2009) and (Pohl and Rupp, 2015), provide patterns for requirements in controlled natural language. Their aim is to mitigate ambiguities and make requirements in natural language more amenable to automation. For example, Arora et al. present an approach to check requirements specifications for boilerplate conformance using Natural Language Processing (Arora et al., 2014). In contrast to our pattern catalog, Mavin et al. as well as Pohl and Rupp address requirements on a more abstract level, for arbitrary systems.

Scenario-based visual languages related to MSDs are presented in (Autili et al., 2007) and (Zhang et al., 2010). With Property Sequence Charts (PSCs), Autili et al. introduce a graphical notation to ease the specification of properties in, e.g., CTL. Similarly to MSDs, PSCs extend UML Interactions. Zhang et al. introduce Timed Property Sequence Charts (TPSCs) that are a timed extension of PSCs. Looking at the available language features, (T)PCSs and MSDs are quite similar. However, (T)PCSs are designed as a visual notation for property specification languages that require existing knowledge about the system states, which is, however, typically not available in early phases of requirements engineering. MSDs specify requirements on the message-based communication of a software system. Knowledge about system states is assumed to not exist yet and is not required. Thereby, MSDs enable automatic analysis techniques in early phases of requirements engineering.

3 MODAL SEQUENCE DIAGRAMS (MSD)

This section introduces basic concepts of the formal, model- and scenario-based RE approach based on Modal Sequence Diagrams (MSDs) (Holtmann et al., 2016). An MSD requirements specification consists of a UML class diagram and a set of MSDs. The UML class diagram is used to define the structure of the system under development and its environment. Furthermore, it specifies the possible messages each system can receive by operations of the defined classes.

Our running example consists of the three classes ESC, RearDoorSystem, and RearDoorMechanics.

The RearDoorSystem is the system under development. It is an electronic control unit that opens/closes and locks/unlocks a vehicle's rear door electronically. The latter is done by actuating the RearDoorMechanics. In addition, the RearDoorSystem is informed by the electronic stability control (ESC) whenever the vehicle starts or stops. The RearDoorSystem needs this information for safety reasons: the rear door shall not open while the vehicle is moving. To prevent the rear door from opening while the vehicle is moving, the RearDoorSystem shall automatically lock the rear door as soon as the vehicle starts moving. This automatic safety locking is detailed in the following requirements:

Requirement 1: Once the ESC informs the RearDoorSystem that the vehicle starts moving, the RearDoorSystem shall eventually lock the RearDoorMechanics before the ESC informs the RearDoorSystem that the vehicle stopped moving.

Requirement 2: Once the ESC informs the RearDoorSystem that the vehicle starts moving, the RearDoorSystem shall lock the RearDoorMechanics after at most 3s.

Requirement 3: Once the ESC informs the RearDoorSystem that the vehicle starts moving, the RearDoorSystem may not unlock the RearDoorMechanics before the ESC informs the RearDoorSystem that the vehicle stopped moving.

These three requirements on the RearDoorSystem are specified in the MSD in Figure 1. An MSD basically consists of lifelines and messages. Lifelines refer to structural entities defined in a UML class diagram. The MSD depicted in Figure 1 encompasses the three lifelines :ESC, :RearDoorMechanics, and :RearDoorSystem. Messages, depicted by arrows between lifelines, define requirements on the communication between objects. A concrete message exchange between two objects (send and receive) is called message event.

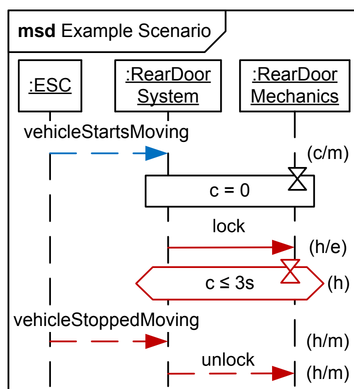

Figure 1: Example requirements 1 to 3 in one MSD.

Messages have a temperature and an execution kind. The temperature of a message can be cold (c) or hot (h) visualized by blue and red arrows in Figure 1. A cold message may be sent/received after any preceding and before any subsequent message of the same MSD, but it is not required to occur (e.g., vehicleStartsMoving depicted in Figure 1). A hot message, on the contrary, has to strictly occur in the order as specified in the MSD (e.g., vehicleStoppedMoving depicted in Figure 1). If any other message of the same MSD occurs when the hot message is expected, the MSD is violated (i.e., the requirement is not fulfilled). If any other message of the same MSD occurs when a cold message is expected, the MSD is terminated/discarded (but the requirement is not violated). The execution kind of a message can either be executed (e) or monitored (m) depicted by solid and dashed arrows in Figure 1, respectively. A monitored message can be observed during the execution of the MSD but its occurrence is not required. An executed message, on the contrary, is required to occur during the execution of an MSD. If it is not sent/received, the MSD is violated. In the MSD in Figure 1, the message lock is hot and executed to specify the requirement that :RearDoorSystem has to send the message in order to operate safely (cf. Requirement 1 and 2). We label the messages in our figures accordingly with (c/m), (c/e), (h/m), and (h/e) for non-color printing.

The MSD Example Scenario contains a clock reset and a clock condition to enable the specification of time dependent communication behavior. A clock reset enables requirements engineers to reset the current value of a clock variable to 0. After a clock reset, the value of the clock variable will increase with time, indicating the time that passed since the last reset. This allows requirements engineers to specify relative timing. Graphically, clock resets are represented by rectangular boxes with a solid border and a sketch of an hourglass in the upper right. The label of a clock reset always has the form <Clock Variable> = 0, where <Clock Variable> is the name of the clock variable to reset (e.g., c=0 depicted in Figure 1).

A clock condition enables requirements engineers to specify that an MSD may only advance under certain time conditions. Graphically, a clock condition is represented by a convex hexagon with parallel opposing edges including an hourglass in the upper right corner. It contains a textual expression in the form <Clock Variable> <Relational Operator> <Natural Number>, where a clock with the name <Clock Variable> is compared with an integer <Natural Number> by using an operator <Relational Operator> $\in \{<,\leq,=,\geq,>\}$ (e.g., $c \leq 3s$ depicted in Figure 1). Like messages, clock conditions have a temperature. The

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

formula of a clock condition must be a Boolean formula and may refer to any clock variables which are bound before the evaluation of the clock condition. A condition is fulfilled if and only if its formula evaluates to true. The MSD may only advance past the condition if it is fulfilled. If the Boolean formula of a cold clock condition evaluates to false, the MSD is terminated/discarded. On the contrary, if the Boolean formula of a hot clock condition evaluates to false, the MSD only advances if the formula evaluates to true due to the passing of time. However, if the Boolean formula can never be fulfilled, the MSD is violated.

4 MSD REQUIREMENT PATTERNS

Our formal, model- and scenario-based requirement pattern catalog consists of 86 patterns categorized into 19 classes shown in Table 1. These model-based patterns are derived from textual patterns that were identified in requirements specifications from industry (Dwyer et al., 1999; Konrad and Cheng, 2005; Bitsch, 2001). The column "Src." in Table 1 denotes from which textual catalog the pattern classes originate. Dwyer et al. (D) defined patterns about the chronological succession (occurrence and order) of properties that shall hold (Dwyer et al., 1999), Konrad and Cheng (K) defined real-time patterns (Konrad and Cheng, 2005), and Bitsch (B) defined safety patterns (Bitsch, 2001). The column "RT" denotes whether a pattern class can be used to specify requirements with continuous real-time (x) or discrete chronological succession (-). The column "S" denotes whether a pattern class can be used to specify safety requirements. The categorization of the pattern classes into *Occurrence* and *Order* follows the structure proposed in (Dwyer et al., 1999) and (Autili et al., 2015).

The classes 1 to 4 contain requirement patterns about the occurrence of an event within certain (time) bounds. The classes 5 to 7 are comprised of patterns about the occurrence of a condition that shall hold within certain (time) bounds. The classes 8 and 9 contain patterns about the required absence of events within certain (time) bounds. Pattern class 10 describes patterns that combine the occurrence (necessity) and absence (permission) of events within certain time bounds. The classes 11 to 16 contain requirement patterns about the response to (chains of) events within certain (time) bounds. The classes 17 to 19 contain patterns about events that are required to precede (chains of) events within certain bounds.

Dwyer et al. define five scopes to decompose each pattern class into the same bounds. We adopted these scopes. For example for the existence class, an event shall occur globally, before an event q, after an event q, between two events q and r, or after an event q until an event r occurs. Konrad and Cheng use the same five scopes for the pattern classes they define. Bitsch uses four different scopes to specify that within certain (time) bounds a proposition is *necessary*, under certain conditions necessary (*conditional guarantee*), *permitted*, or *necessary & permitted* to hold. He defines 52 patterns in total. All of his pattern classes not listed in Table 1 are instances of (a combination of) the other patterns (which is denoted by an x in the "S" column in Table 1 and shown in (Fockel et al., 2017), see also Section 5). We combined all pattern classes and scopes such that our catalog contains 86 distinct patterns.

Table 1: Requirement pattern classes.

Pattern Class		**Src.**	**RT**	**S**	**#**
Occurrence					
1.	Existence	D	-	x	5
2.	Bounded Existence	D	-	-	5
3.	Bounded Recurrence	K	x	-	5
4.	RT-Safety - Necessary	B	x	x	2
5.	Universality	D	-	x	5
6.	Minimum Duration	K	x	-	5
7.	Maximum Duration	K	x	-	5
8.	Absence	D	-	x	5
9.	RT-Safety - Permitted	B	x	x	2
10.	RT-Safety - Nec.&Perm.	B	x	x	2
Order					
11.	Response	D	-	x	5
12.	Response Chain 1-*n*	D	-	-	5
13.	Response Chain *n*-1	D	-	-	5
14.	Bounded Response	K	x	x	5
15.	Bounded Invariance	K	x	-	5
16.	Constrained Chain	D	-	-	5
17.	Precedence	D	-	-	5
18.	Precedence Chain 1-*n*	D	-	-	5
19.	Precedence Chain *n*-1	D	-	-	5
				Total:	86

Legend: RT: Real-Time; S: Safety;
D: Dwyer et al.; K: Konrad and Cheng; B: Bitsch

Dwyer et al., Konrad and Cheng, and Bitsch specified their patterns in formal, textual property specification languages like CTL (Emerson and Clarke, 1982) and TCTL (Alur et al., 1993), common in model checking (Baier and Katoen, 2008). We use the formal, model- and scenario-based requirements specification language Modal Sequence Diagrams (Holtmann et al., 2016). In the following, we explain two model-based requirement patterns. The complete catalog can be found in our technical report (Fockel et al., 2017). For each pattern, we provide its MSD representation, a Büchi automaton (Büchi, 1966), and the pattern's application to the rear door system example (Requirement 2 and Requirement 3).

Absence Pattern. The MSD in Figure 2 specifies the chronological succession requirement pattern *Absence (After q until r)* in model-based form. The right of the figure shows its representation as Büchi automaton. The transitions represent events (i.e., the execution of messages) and the states denote the possible MSD execution states between events. s0 is the start state before the initial event q of the MSD. Σ denotes the set of all possible events.

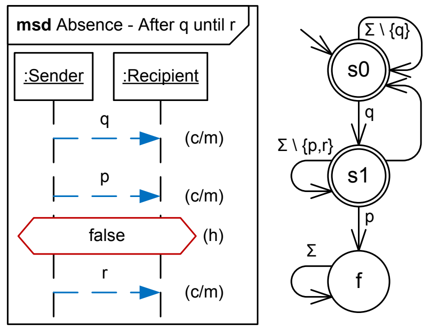

Figure 2: Absence (After q until r) pattern.

The pattern specifies that if the event q occurs, the event p may not occur until the event r occurred. The initial message q in the MSD denotes the if-condition. If q is followed by the event p, the execution of the MSD ends in the hot condition false. It always evaluates to false and produces a violation that cannot be avoided by any further event (non-accepting state f in Figure 2 cannot be left). In this case, the requirement specified by this MSD is not fulfilled because p occurred after q and before r. The position of the hot condition false also prevents that the message r specified below will ever be reached. That message is only part of this MSD to force a discarding of the MSD if the event r occurs before p (in state s1). In that case, the requirement is fulfilled because p did not occur before r. All messages in the pattern are monitored because they are not required to occur. We only argue about the absence of events. In addition, all messages are cold because we do not require a certain order of occurrence, we forbid one order by the hot false condition. For instance, the event q may be repeated arbitrarily often before r (self-transition on state s1).

The MSD in Figure 3 shows the application of the pattern to specify Requirement 3 introduced in Section 3. The rear door shall not be unlocked while the vehicle is moving. Thus, the message unlock may not be sent after the message vehicleStartsMoving until the message vehicleStoppedMoving is received.

Bounded Response Pattern. The MSD in Figure 4 specifies the real-time requirement pattern *Bounded Response (Globally)* in model-based form. The pattern is also part of the safety patterns of Bitsch. Thus, the x in Table 1 in the safety column of this pattern. The pattern specifies that if the event p occurs, the event s has to occur within k time units.

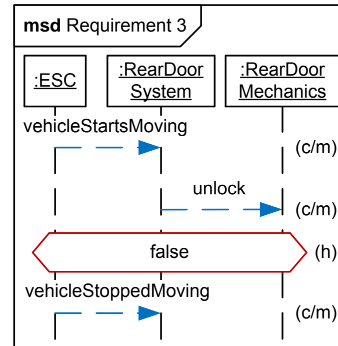

Figure 3: Absence pattern applied to example.

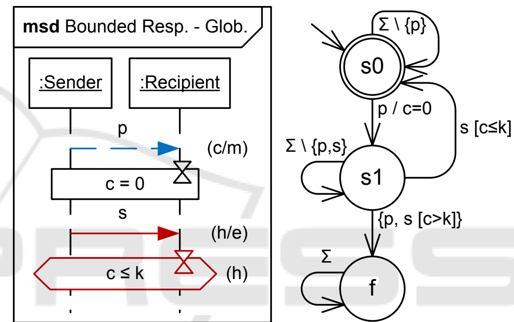

Figure 4: Bounded response (Globally) pattern.

The initial message p in the MSD denotes the if-condition. It is directly followed by a clock reset that resets the clock c to 0. The executed message s denotes that s has to eventually occur. Thus, the state s1 is non-accepting and needs to be left. The message is also hot, so it has to occur before the following clock condition. That hot clock condition specifies that once it is reached, the clock c has to have a value lower or equal to k (i.e. s1 has to be left via the transition s [c≤k]). If this is not possible, s occurred too late and the requirement is violated. In this case, the Büchi automaton stays in one of the non-accepting states s1 or f.

The MSD in Figure 5 shows the application of the pattern to specify Requirement 2 introduced in Section 3. The rear door shall be locked once the vehicle starts moving. Thus, the message lock shall be sent within 3 seconds after vehicleStartsMoving.

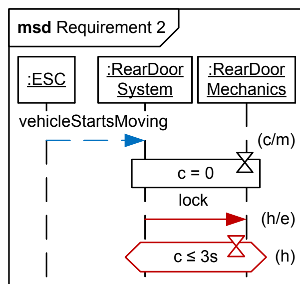

Figure 5: Bounded response pattern applied to example.

5 DISCUSSION

In this paper, we present a formal, model- and scenario-based requirement pattern catalog. It is based on three existing catalogs of patterns about chronological succession (Dwyer et al., 1999), real-time (Konrad and Cheng, 2005), and safety (Bitsch, 2001). The authors of those catalogs provide patterns for property specification languages like CTL and TCTL. The focus of these languages is on checking the correctness of an existing software. They specify state-based requirements using conditions. For example, "After being in a state where condition *c* holds, the system shall enter a state where condition *d* holds until condition *e* holds". This requires existing knowledge of possible system states, or a state-based design model has to be defined alongside the requirements.

In contrast, the focus of MSDs lies on requirements on message-based communication, i.e., requirements on message events that occur at simulation- or run-time (cf. Section 3). Such a requirement has the form "Once the event *q* occurs, the system shall trigger event *p* before event *r* occurs". Knowledge about internal system states is assumed to not yet exist and is not required. Furthermore, MSDs organize requirements in sets of use cases of the system to be developed. It would be challenging to define system states valid for all scenarios during the early development phase of requirements specification.

Nevertheless, we base our work on the existing, well-known pattern catalogs for two reasons: Firstly, Dwyer et al. suggested that their state-condition-based formulas can be translated into event-based formulas by using two events for specifying that a condition starts or stops to hold (Dwyer et al., 1999). Secondly, the two catalogs (Dwyer et al., 1999) and (Konrad and Cheng, 2005) have been translated into visual notations before, to provide a more intuitive specification means (Autili et al., 2007; Zhang et al., 2010).

In the following, we discuss our findings from translating the three pattern catalogs into the MSD requirements language (Holtmann et al., 2016).

Each MSD Defines its Own Clocks. The semantics of timed MSDs (Brenner et al., 2014) specify that clocks only exist within the scope of a single MSD (cf. clock c in Figure 4). Thus, the clock variable of one MSD cannot be used in clock conditions of other MSDs. Hence, a requirement pattern that consists of a cascade of MSDs and reasons about one time interval from beginning to end, cannot be specified using a single clock variable.

Fortunately, it is possible to specify all real-time patterns (from (Konrad and Cheng, 2005) and (Bitsch, 2001)) by MSDs. In four cases we use time triggered messages (i.e., events that are enforced after a clock condition evaluates to true) to join cascading MSDs. In future work, MSD semantics could be extended to allow global clocks that span multiple MSDs.

Each MSD Has to Start with an Initial Message: The semantics of MSDs specify that the execution of an MSD is triggered by an event that is listened for by the first message (that is always cold and monitored) of the MSD. Thus, MSDs always represent if-then requirements: if the initial message/event occurs, then the succeeding part of the MSD shall (not) occur.

For the before-scope requirement patterns, this means that we have to add an event that represents the start of the system: "If the system starts, then the system shall... before the event *q* occurs". However, we argue this is viable because a software-intensive system always has to be started somehow (e.g., by a vehicle's ignition or a power button).

Strictness Requirements (Hot Messages) Forbid Event Recurrences: If an MSD contains a hot message, no other message of that MSD may occur when the hot message is anticipated. For example, in Figure 1, neither the message vehicleStartsMoving before the hot message lock nor vehicleStoppedMoving and unlock after it are allowed to occur when lock is anticipated. The state-condition-based property specification patterns typically reason about conditions. For example, "After condition *c* holds, condition *d* has to eventually hold afterward". They do not consider how long *c* holds or if it is interrupted for a while until *d* eventually holds.

We adapted the patterns to the event-based formalism. For example, "After the event *q* occurs, the event *p* has to eventually occur". Intuitively, one might interpret this requirement such that *p* has to occur after every *q*. However, the original property specification pattern would allow the event *q* to recur multiple times before *p* has to eventually occur (the condition *c* may be interrupted). If we obey the formal pattern interpretation, most requirement patterns require mul-

tiple MSDs to avoid unwanted dependencies between events (cf. (Fockel et al., 2017)). If we make the intuitive assumption (and state it explicitly), the patterns can be applied in simpler form leading to condensed MSDs (e.g., Figure 1). Both interpretations can be expressed with MSDs. The separate MSDs for the formal interpretation decompose each pattern into reduced subrequirements that fulfill the singularity quality characteristic imposed by ISO 29148: "The requirement statement includes only one requirement with no use of conjunctions." (ISO, 2011). For example, the existence pattern (used for Requirement 1) is decomposed into two MSDs.

MSD Patterns are Closer to their Realization than Property Specification Patterns: The property specification patterns specify conditions that shall hold for certain states of a system. Thus, it is assumed that a state-based design model of the system exists, which can be directly verified against test requirements specified through the property specification patterns (e.g., by model checking). MSDs are used to specify requirements on the communication of systems and do not require an existing state-based design model. They describe scenario-based requirements that a system to be developed shall fulfill.

We argue that MSD requirements better serve the purpose of finding a possible realization than requirements formulated in a property specification language. In conclusion, MSD requirement patterns support the ISO 29148 requirement quality characteristics feasibility and verifiability (i.e., testability).

Bitsch's Safety Patterns are Instances of (Combinations of) the Chronological Succession and Real-time Patterns: When translating the safety patterns (Bitsch, 2001) into MSDs, we discovered that most of them can be expressed using the same MSDs as we used for the chronological succession patterns and real-time patterns (Dwyer et al., 1999; Konrad and Cheng, 2005). The bounded response pattern (cf. Figure 4) is an example that is contained in the real-time pattern catalog and the safety pattern catalog. We denote these mapping relations in the safety column (S) of Table 1. The concrete instance-of-mapping for each safety pattern is contained in our technical report (Fockel et al., 2017).

There are three pattern classes, namely *RT-Safety - Necessary*, *RT-Safety - Permitted*, and *RT-Safety - Necessary & Permitted* that could not be specified as instances of patterns of the other catalogs. We list these separately in Table 1. The first two pattern classes were defined as new (non-safety) patterns called *Time-constrained existence* and *Time-constrained absence* independently from Bitsch by Autili et al. (Autili et al., 2015). The third pattern class is the combination of the first two.

Summary: In summary, we showed that MSDs can be used to specify chronological succession, real-time, and safety requirement patterns. However, their semantics also can hinder the intuitive specification of requirements (e.g., concerning clocks, initial messages, and strictness). On the contrary, the strict and constructive use of the requirement patterns fosters requirements specification quality by supporting the ISO 29148 requirement quality characteristics singularity, feasibility, and verifiability. In addition, the support for automatic requirements verification (Greenyer et al., 2013) fosters the requirement quality characteristic consistency. Furthermore, the support for requirements simulation (Brenner et al., 2013; Brenner et al., 2014) helps to identify missing requirements and assumptions to facilitate the requirement quality characteristic completeness. Moreover, Bitsch suggests to use Life Sequence Charts or similar languages for complex requirement patterns: "The practical benefit of this kind of approaches is obvious but the correct use of these notations still have to be learned" (Bitsch, 2001). We argue that the use of our pattern catalog for MSDs (which are based on Life Sequence Charts) facilitates the learning process. In addition, the transition and traceability to the model-based design is easier if requirements are also specified in a model-based way.

6 CONCLUSIONS

In this paper, we introduced a model- and scenario-based pattern catalog that is tailored to the MSD requirements specification language (Harel and Maoz, 2008; Holtmann et al., 2016). The requirement pattern catalog consolidates and unifies requirement patterns from three well-known requirement pattern catalogs (Dwyer et al., 1999; Konrad and Cheng, 2005; Bitsch, 2001) that cover the aspects chronological succession, real-time, and safety, respectively. We illustrated and discussed the application of the MSD requirement patterns by means of an automotive example, whereas we present the full catalog encompassing altogether 86 patterns in (Fockel et al., 2017).

The requirement patterns facilitate and systematize the conception of complete and consistent MSD specifications. Moreover, our requirement pattern catalog is practice-oriented due to the fact that the three underlying requirement pattern catalogs were extracted from real-world requirements specifications.

In contrast to the three underlying catalogs that apply derivatives of property specification languages (Baier and Katoen, 2008) that are difficult to apply correctly (Autili et al., 2007) and require a state-based design model, our pattern catalog bases on the visual notion of MSDs that can be directly simulated and formally checked for consistency already at requirements level.

In future work we plan to evaluate our formal, model- and scenario-based requirement pattern catalog by applying it in further industry case studies. Furthermore, we want to extend the catalog by further types of patterns like security requirement patterns (Spanoudakis et al., 2007).

REFERENCES

Alexander, C., Ishikawa, S., and Silverstein, M. (1977). *A Pattern Language*, volume 2 of *Center for Environmental Structures Series*. Oxford University Press.

Alur, R., Courcoubetis, C., and Dill, D. (1993). Model-checking in dense real-time. *Information and Computation*, 104(1):2 – 34.

Arora, C., Sabetzadeh, M., Briand, L. C., and Zimmer, F. (2014). Requirement boilerplates: Transition from manually-enforced to automatically-verifiable natural language patterns. In *4th Int. Workshop on Requirements Patterns (RePa 2014)*.

Autili, M., Grunske, L., Lumpe, M., Pelliccione, P., and Tang, A. (2015). Aligning qualitative, real-time, and probabilistic property specification patterns using a structured english grammar. *IEEE Transactions on Software Engineering*, 41(7):620–638.

Autili, M., Inverardi, P., and Pelliccione, P. (2007). Graphical scenarios for specifying temporal properties: an automated approach. *Automated Software Engineering*, 14(3):293–340.

Baier, C. and Katoen, J.-P. (2008). *Principles of Model Checking*. MIT Press.

Büchi, J. R. (1966). Symposium on decision problems: On a decision method in restricted second order arithmetic. *Studies in Logic and the Foundations of Mathematics*, 44:1–11.

Bitsch, F. (2001). Safety patterns – the key to formal specification of safety requirements. In *20th Int. Conference on Computer Safety, Reliability and Security (SAFECOMP 2001)*, pages 176–189.

Brenner, C., Greenyer, J., Holtmann, J., Liebel, G., Stieglbauer, G., and Tichy, M. (2014). ScenarioTools real-time play-out for test sequence validation in an automotive case study. In *13th Int. Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2014)*.

Brenner, C., Greenyer, J., and Panzica La Manna, V. (2013). The ScenarioTools play-out of modal sequence diagram specifications with environment assumptions. In *12th Int. Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2013)*.

Chung, L., Kopczyńska, S., Leite, J. C. S. d. P., Supakkul, S., and Zhao, L. (2016). Welcome to the 6th int. workshop on requirements patterns (RePa). In *2016 IEEE 24th Int. Requirements Engineering Conf. Workshops (REW)*, pages 276–277.

Damm, W. and Harel, D. (2001). LSCs: Breathing life into message sequence charts. *Formal Methods in System Design*, 19:45–80.

Dwyer, M. B., Avrunin, G. S., and Corbett, J. C. (1999). Patterns in property specifications for finite-state verification. In *1999 Int. Conference on Software Engineering (ICSE 99)*, pages 411–420.

Emerson, E. A. and Clarke, E. M. (1982). Using branching time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2(3):241–266.

Fockel, M., Holtmann, J., Koch, T., and Schmelter, D. (2017). Model-based requirement pattern catalog. Technical Report tr-ri-17-354, Fraunhofer IEM / Heinz Nixdorf Institute.

Greenyer, J., Brenner, C., Cordy, M., Heymans, P., and Gressi, E. (2013). Incrementally synthesizing controllers from scenario-based product line specifications. In *ESEC/FSE*, pages 433–443. ACM.

Harel, D. and Maoz, S. (2008). Assert and negate revisited: Modal semantics for UML sequence diagrams. *Software & Systems Modeling*, 7(2):237–252.

Holtmann, J., Fockel, M., Koch, T., Schmelter, D., Brenner, C., Bernijazov, R., and Sander, M. (2016). The MechatronicUML Requirements Engineering Method: Process and Language. Technical Report tr-ri-16-351, Fraunhofer IEM / Heinz Nixdorf Institute.

ISO (2011). ISO 29148:2011: Systems and software engineering – life cycle processes – requirements engineering.

Konrad, S. and Cheng, B. H. C. (2002). Requirements patterns for embedded systems. In *IEEE Joint Int. Conference on Requirements Engineering*.

Konrad, S. and Cheng, B. H. C. (2005). Real-time specification patterns. In *27th Int. Conference on Software Engineering (ICSE 2005)*, page 372.

Konrad, S., Cheng, B. H. C., and Campbell, L. A. (2004). Object analysis patterns for embedded systems. *IEEE Transactions on Software Engineering*, 30(12):970–992.

Mavin, A., Wilkinson, P., Harwood, A., and Novak, M. (2009). Easy approach to requirements syntax (EARS). In *17th IEEE Int. Requirements Engineering Conference (RE 2009)*, pages 317–322.

Object Management Group (2015). OMG Unified Modeling Language (OMG UML), version 2.5.

Pohl, K. and Rupp, C. (2015). *Requirements Engineering Fundamentals*. Rocky Nook, Inc.

Spanoudakis, G., Kloukinas, C., and Androutsopoulos, K. (2007). Towards security monitoring patterns. In *2007 ACM Symposium on Applied Computing (SAC 07)*, pages 1518–1525. ACM.

Zhang, P., Li, B., and Grunske, L. (2010). Timed property sequence chart. *Journal of Systems and Software*, 83(3):371–390.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS