Real-time Specification Patterns

Sascha Konrad  Betty H.C. Cheng
konradsa@cse.msu.edu  chengb@cse.msu.edu
Software Engineering and Network Systems Laboratory
Department of Computer Science and Engineering
Michigan State University
3115 Engineering Building
East Lansing, Michigan 48824 USA

ABSTRACT
Embedded systems are pervasive and frequently used for critical systems with time-dependent functionality. Dwyer et al. have developed qualitative specification patterns to facilitate the specification of critical properties, such as those that must be satisfied by embedded systems. Thus far, no analogous repository has been compiled for real-time specification patterns. This paper makes two main contributions: First, based on an analysis of timing-based requirements of several industrial embedded system applications, we created real-time specification patterns in terms of three commonly used real-time temporal logics. Second, as a means to further facilitate the understanding of the meaning of a specification, we offer a structured English grammar that includes support for real-time properties. We illustrate the use of the real-time specification patterns in the context of property specifications of a real-world automotive embedded system.

Categories and Subject Descriptors
D.2.1 [Requirements/Specifications]: Methodologies; D.2.4 [Software/Program Verification]: Formal methods

General Terms
Verification

Keywords
Patterns, Formal Specification, Real-time Requirements, Embedded Systems

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ICSE’05, May 15–21, 2005, St. Louis, Missouri, USA.
Copyright 2005 ACM 1-58113-963-2/05/0002 ...$5.00.

1. INTRODUCTION
Given the pervasive nature of embedded systems [15] and their use for critical applications (e.g., medical devices, transportation systems), they must typically achieve a high level of robustness and reliability. In addition, embedded systems software usually involves time-dependent functionality. Consequently, methods for developing and modeling embedded systems and rigorously verifying behavior before committing to code are increasingly important. Specification patterns [17] have been used to guide users of finite-state verification tools in expressing system requirements directly in a formal specification language, such as linear-time temporal logic (LTL) \cite{LTL}, computational tree logic (CTL) \cite{CTL}, or graphical interval logic (GIL) \cite{GIL}. However, these specification patterns cannot be used to specify real-time properties, since they do not support quantitative reasoning about time. To the best of our knowledge, no repository has been compiled for real-time specification patterns. This paper introduces a collection of real-time specification patterns that can be used to specify real-time properties for embedded systems; these patterns are intended to preserve the style of, but be complementary to, Dwyer et al.’s patterns \cite{Dwyer}. We provide mappings for the specification patterns in terms of three commonly used temporal logics. To further facilitate the use of both our real-time specification patterns and Dwyer et al.’s patterns, we also introduce a structured English grammar for capturing the specification in terms of natural language.

Several previously published works contain properties specified in real-time temporal logics, but these property specifications differ on several aspects with respect to our objectives. First, they do not offer the completeness of a pattern system. Given the increasing use of Dwyer et al.’s specification patterns, it is clear that the additional information provided by a system of patterns significantly facilitates the specification of critical properties. Second, the properties are usually only specified in terms of one real-time temporal logic. Our assessment of real-time temporal logics indicates that there are a variety of support tools with different approaches to analysis. We have identified a “common” set of properties that are expressible in all three logics, thus enabling a specifier to choose the analysis tool best suited for the properties and models under consideration. And finally, these other real-time specifications do not provide support for scopes similar to those used for the specification patterns.
This paper presents a two-pronged approach for overcoming the perceived difficulty with specifying formal properties of real-time systems: templates for real-time temporal logic specifications and natural language specifications of properties. We analyzed several requirements documents for embedded systems from industrial partners working on automotive and appliance applications. Based on our studies of these requirements, it became clear that many of the requirements were timing-based, which could not be specified in terms of Dwyer et al.’s original specification patterns. As such, we generalized the timing-based requirements into several categories of recurring properties. Then we created mappings of these properties into three real-time temporal logics. Finally, we created a pattern template, largely in the same spirit of the one used by Dwyer et al., in order to facilitate their use. One notable modification is the addition of a field that contains a structured English representation of a given property. This structured English sentence serves as an accompanying handle for each scoped version of a pattern and aids in understanding the property captured by the pattern.

Our real-time specification patterns contain templates for specifying real-time properties in terms of three commonly-used real-time temporal logics: metric temporal logic (MTL) [11, 54], timed computational tree logic (TCTL) [2], and real-time graphical interval logic (RTGIL) [41]. While there are numerous real-time temporal logics (for representative overviews please refer to [3, 17]), we chose these particular logics for three major reasons. First, in order to facilitate the use of our patterns by users of Dwyer et al.’s specification patterns, we use real-time temporal logics that are directly related to temporal logics used in the original specification patterns: LTL [37], CTL [12], and GIL [46]. Second, these three real-time temporal logics have tool support with different strengths and weaknesses, including model checking and theorem proving capabilities. Therefore, we allow the specifier to choose the tool best suited for the system model at hand. Finally, while MTL and TCTL cover linear time as well as branching time using a text-based representation of properties, RTGIL is a linear-time temporal logic that provides an easy to read graphical representation of a property that is considered more intuitive for non-expert specifiers [16]. In addition to the mappings to the three real-time temporal logics, our structured English grammar can be used to create a natural language representation of a specification contained in a pattern.

The real-time specification patterns have been applied to several industrial projects and served to facilitate the construction of real-time formal properties of embedded systems. In this paper, we illustrate the applicability of our real-time specification patterns to the real-time requirements of an electronically controlled steering system obtained from one of our industrial collaborators, which we analyze using our timing-extended UML formalization framework [30]. The remainder of the paper is organized as follows. Section 2 reviews specification patterns and the three temporal logics used in our patterns. Section 3 describes the real-time specification pattern template and the patterns uncovered thus far. It also provides a classification of the patterns based on the type of property they address. Section 4 describes the application of our approach to the electronically-controlled power-assisted steering system. Section 5 overviews related work. Finally, in Section 6 we present conclusions and future work.

2. BACKGROUND

In this section, we overview the specification patterns by Dwyer et al. and briefly describe the real-time temporal logics used for our patterns, MTL [4, 34], TCTL [2], and RTGIL [41]. Real-time temporal logics extend standard temporal logics with temporal operators that permit quantitative temporal reasoning. Real-time temporal logics that are interpreted over a discrete time domain (such as N) are termed discrete real-time temporal logics (e.g., MTL), while real-time temporal logics that are interpreted over a dense time domain (such as R) are called dense real-time temporal logics (e.g., TCTL and RTGIL). For our purposes, we use temporal logics without past temporal operators, largely because they are rarely supported by formal analysis tools. For each real-time temporal logic, we also briefly mention available tool support, where analysis tools can be categorized into two broad categories [8]. Heterogeneous analysis tools support more than one specification language for modeling a system and properties to be checked against the system. In contrast, homogeneous analysis tools use the same language to model both the system and properties.

2.1 Specification Patterns

Dwyer et al. [17] describe several patterns applicable to software properties for specifications written in different formalisms, such as LTL [37], CTL [12], GIL [46], and quantified regular expressions (QRE) [33]. Specification patterns are categorized into two major groups: occurrence patterns and order patterns. While a given specification pattern may have several scopes of applicability (e.g., globally, before an event/state occurs, after an event/state occurs), the original specification patterns do not include timing information. Henceforth, we refer to the specification patterns by Dwyer et al. as qualitative specification patterns as they specify qualitative properties that are not amenable to quantitative reasoning about time.

2.2 MTL

Metric temporal logic (MTL) [11, 54] is an extension of linear-time temporal logic (LTL) [37] that is interpreted over a discrete time line. MTL assumes the digital-clock (or fictitious-clock) model [2], in which an external, discrete clock progresses at a fixed rate. Although this clock runs asynchronously with other components in a system, those components increment their discrete timer variables synchronously with every tick of the external clock. The ordering of states between two time ticks is known, but not the exact time of occurrence. Despite these limitations, discrete time models have been used successfully for the analysis of a variety of real-time systems [25, 26]. MTL contains time-constrained versions of the always (□), eventually (?), next (□), strong until (U), and weak until (W) operators.

Example heterogeneous analysis tools that support MTL include our recently revised UML formalization framework [39], which uses a Promela representation of UML class and state diagrams with timing information to model a system and uses MTL to specify correctness properties. Additionally, the Temporal Rover [16] supports the inclusion of code generated from MTL formulae into program code (e.g.,
C, C++, Java) to monitor the satisfaction of the MTL specifications at runtime.

2.3 TCTL

Timed computational tree logic (TCTL) [2] is a real-time extension to computational tree logic (CTL) [12]. TCTL is interpreted over a dense time line and contains time-constrained versions of the always (G), eventually (F), strong until (U), and weak until (W) operators, which are either existentially (E) or universally (A) quantified.

Existing heterogeneous analysis tools that support TCTL include UPPAAL [45], HyTech [25], and Kronos [9]. More specifically, timed automata (or in the case of HyTech a more general form, hybrid automata) are used to model a system, and TCTL is used to specify correctness properties. Kronos supports full TCTL, while UPPAAL and HyTech only support quantification at the beginning of a temporal formula, thereby focusing on reachability analysis.

2.4 RTGIL

Real-time graphical interval logic (RTGIL) [41], and its corresponding textual representation, real-time future interval logic (RTFIL), are real-time extensions to graphical interval logic (GIL) [46], and its textual representation, future interval logic (FIL), respectively. RTGIL is a propositional linear-time temporal logic interpreted over dense time. In RTGIL, a time line is used to show the progression of a computation. Intervals can be constructed on this time line; an interval is represented by a segment of the time line delimited by two states and is left-closed and right-open. Intervals are constructed using search patterns with associated target formulae. A search locates the first state in the future from the current position on the time line where the target formula holds (which might be the current state if the formula holds there). Formulae are read from top to bottom and from left to right, and can be combined using standard logical infix operators. Initial properties as well as henceforth or eventuality properties can be assigned to an interval. The only real-time operator supported by RTGIL is the len predicate, \(\text{len}(d, D)\), which implies that the duration of the interval, if it can be constructed, is greater than \(d\) time units and less than or equal to \(D\) time units (\(d\) and \(D\) represent nonnegative rational constants, where \(D\) can also be \(\infty\)).

RTGIL is supported by the RTGIL environment [41], which includes a graphical editor, an automated theorem prover, and a data base and proof manager component. Because the RTGIL environment is a homogeneous analysis tool, a model and its correctness properties are both specified in terms of RTGIL formulae.

2.5 Discussion

A major semantic difference between RTGIL and MTL/TCTL is the way that real-time information is included in RTGIL formulae. While MTL and TCTL construct an interval with a time-bounded operator and denote the states that can occur in this interval, RTGIL constructs an interval by denoting the states that serve as endpoints to the interval and then places a bound on the duration of this interval. Therefore, RTGIL is considered mutually inexpressible to other linear-time real-time temporal logics, such as MTL [46]. Nevertheless, RTGIL is one of the few dense real-time temporal logics known to be decidable [46].

(Another temporal logic known to be decidable is metric interval temporal logic (MITL) [2].)

It is noted that the properties in our pattern system were selected so that they are expressible in terms of all three real-time temporal logics. However, given the differences in the underlying time models, semantics, and expressive power, a property expressed in one real-time temporal logic cannot be considered truly equivalent to the same property expressed in a different temporal logic. There have been attempts to establish equivalence relationships between pairs of the timing models with respect to specific properties. For example, some researchers have investigated how discrete-time semantics can be used to prove, or at least approximate with arbitrary precision, the adherence of a model to some classes of properties in a dense time model [8, 29]. However, this relation cannot be established for all classes of properties.

As with any pattern system, our specification pattern system does not provide a “silver bullet”; to make effective use of these patterns requires a solid understanding of the underlying fundamentals. Our pattern system can be useful in providing strategies and guidance for specifiers to create their own pattern systems tailored to their domain and system models.

3. REAL-TIME PATTERNS

This section gives an overview of the real-time specification patterns uncovered thus far and classifies our pattern system. We also describe our structured English grammar and pattern template, and give an example real-time specification pattern, the Bounded Recurrence Pattern.

We do not claim our real-time specification pattern system to be complete; the patterns represent commonly occurring types of real-time properties found in several requirements documents for appliances and automotive embedded systems applications. Our pattern system is intended to provide strategies for how to specify real-time properties in a formal specification language, where the properties are amenable to automated analysis. An analysis of requirement documents from different domains may lead to additional or different sets of recurring properties. In addition, due to the selected temporal logics, certain properties are not expressible, e.g., properties that only apply to one path of execution (MTL and RTGIL formulae are always universally quantified) or certain properties with strict timing constraints (due to the limited expressiveness of the len predicate in RTGIL). Therefore, different real-time specification pattern systems are possible and should be investigated. Selecting a different set of temporal logics may lead to a different set of specification patterns. This paper shows how one such system can be developed.

3.1 Pattern Repository and Classification

We collected 80 real-time system requirements from 9 requirement documents for embedded systems provided by industrial collaborators (such as an ABS system and an electronically controlled steering system); the number of qualitative requirements in these documents was considerably larger. These requirements were analyzed and generalized into specification patterns. The collection of patterns were then classified into three broad categories of real-time properties: duration (captures properties that can be used to place bounds on the duration of an occurrence), periodic (describes properties that address periodic occurrences), and
real-time order (captures properties that place time bounds on the order of two occurrences).

Figure 1 overviews the real-time specification patterns uncovered thus far and lists example requirements from real-world systems that can be captured by the corresponding real-time specification pattern. Scopes used in Dwyer et al.'s specification patterns (i.e., “Globally”, “Before R”, “After Q”, “Between Q and R”, “After Q until R”) are also applicable for our real-time specification patterns. Figure 2 shows a specification pattern hierarchy comprising our real-time specification patterns and the qualitative specification patterns by Dwyer et al. (in the shaded box).

3.2 Structured English

In order to facilitate the use of specification patterns, we have developed a structured English grammar (shown in Figure 3) that supports both qualitative and real-time specification patterns. In the grammar, literal terminals are delimited by quotation marks (" "), non-literal terminals are given in a sans serif font, and non-terminals are given in italics. The start symbol of the grammar is property and the language \( L(G) \) of the grammar is finite, since the grammar is non-circular and has no repetitions. Each sentence (or string) \( s \) with \( s \in L(G) \) serves as a handle that accompanies a scoped formula of a qualitative or real-time specification pattern. Therefore, the grammar aids in understanding the meaning of a property without needing to analyze the temporal logic representation. Our structured English grammar is similar in spirit to the structured English grammar for clocked computational tree logic (CCTL) by Flake et al. [10]. In contrast to Flake et al.'s grammar, whose main objective is to support the specification of CCTL properties, our grammar is intended to be general enough to support translations of untimed and timed properties to three different untimed temporal logics and three different real-time temporal logics, respectively, as well as handle scopes. Specifically, our grammar supports specifications using LTL, CTL, and GIL for Dwyer et al.'s specification patterns [4] and MTL, TCTL, and RTGIL for our real-time specification patterns. Using our grammar, users are able to derive natural language representations of qualitative and real-time properties specified in terms of temporal logics. Feedback from industry has indicated that a structured English representation is less intimidating than the temporal logic notation. Experience has shown that direct exposure to formal specifications often hinders the pervasive use of formal methods [12].

The structured English grammar is organized according to our classification of patterns given in Figure 4. In general, the process for using our grammar to create a natural language representation is as follows: Initially, the user chooses the scope of the specified property (globally, before, after, between, or after-untill), followed by the type (qualitative or real-time). Then the category of the specified property is selected (duration, periodic, or real-time order for real-time properties, and occurrence or order for qualitative properties). The final structured English sentence is constructed by choosing the corresponding specification pattern. In the grammar, “precede” and “succeed” denote strict past and future, respectively; while “held previously” and “hold eventually” denote non-strict past and future, respectively. For example, “\( S \) eventually holds” is satisfied if \( S \) holds in the current state, while this is not the case with “\( S \) succeeds”.

For example, the user wants to create a natural language representation for a bounded response property captured by the following MTL formula:

\[ \square((x = 0) \rightarrow \diamond_{\leq 6}(y = 1)). \]

Initially, the user derives the following sentence from the grammar (grammar rules are given in parentheses):

“Globally, it is always the case that if \( P \) holds, then \( S \) holds after at most \( c \) time unit(s).” (Grammar: 1, 2, 3, 18, 24, 25).

This sentence corresponds to the following globally scoped MTL template from the bounded response pattern:

\[ \square(P \rightarrow \diamond_{\leq 6}S). \]

Non-literal terminals need to be instantiated for the natural language representation. \( P \) and \( S \) denote boolean propositional formulae that describe properties of states and are used to capture properties of the system as well as to denote states that serve as bounds for scopes. For real-time properties, \( c \) also needs to be instantiated with the integer values that are used with the constrained operators in a real-time temporal logic formula. After replacing \( P \) with \( x = 0 \), \( S \) with \( \gamma = 1 \), and \( c \) with \( 6 \), the final natural language representation for the aforementioned property is obtained:

“Globally, it is always the case that if \( (x = 0) \) holds, then \( (\gamma = 1) \) holds after at most \( 6 \) time unit(s).”.

A complementary approach for using the natural-language-based derivation process and corresponding tool support is described elsewhere [32].

3.3 Pattern Template

Dwyer et al. [17] used a variation of the design pattern template developed by Gamma et al. [21]. We base our template for the real-time specification patterns on Dwyer et al.'s template and provide similar information. Additionally, we include a field “Structured English Specification” in our template. This field contains an unscoped structured English sentence according to the grammar presented in Section 3.2. Our real-time specification pattern template contains the following fields:

- **Pattern Name and Classification**: The pattern name serves as a handle for the pattern's use and describes the nature of the pattern. The classification denotes if the pattern belongs to the duration, periodic, or real-time order category.
- **Structured English Specification**: The structured English sentence captures the property. The sentence is given in its unscoped version; a scope will be added as a prefix to the sentence when the pattern is instantiated.
- **Pattern Intent**: A short description of properties for which the pattern is applicable.
- **Real-time Temporal Logic Mappings**: Contains mappings of the pattern to (1) MTL, (2) TCTL, and (3) RTGIL for each of the five possible scopes.
- **Examples and Known Uses**: Gives example instantiations of the pattern and describes common situations where the pattern is useful in real-world scenarios.
- **Relationships**: Describes relations to other qualitative as well as real-time specification patterns. Additionally, this field contains information about other commonly used properties and techniques that are related to this pattern.

\( ^1 \)Note that the Precedence Chain Patterns, Response Chain Patterns, and Constrained Chain Pattern from Dwyer's specification patterns are not closed under stuttering [11]. Therefore, these specification patterns should not be used for model checking with partial order reduction.
### Real-time Specification Pattern Example

Figure 4 contains an example real-time specification pattern, the **Bounded Recurrence Real-time Specification Pattern**. The example contains the complete mappings for both MTL and TCTL. For RTGIL, only the mappings with scope “Globally” and “Between $Q$ and $R$” are given due to space constraints. In this example, the RTGIL mappings demonstrate the readability of these formal properties for non-experts in formal methods. While the MTL and TCTL mappings of the scope “Between $Q$ and $R$” are lengthy and may be difficult to interpret, the graphical representation of RTGIL greatly aids in understanding what property the specification pattern captures.

### CASE STUDY

In order to validate the applicability of our approach, we applied it to several examples from industry. In this paper, we present an electronically controlled steering (ECS) system whose requirements involve untimed as well as real-time properties.

#### 3.4 Real-time Specification Pattern Example

The ECS system is intended to supplement the benefits provided by traditional hydraulic power steering. This all-electric and engine-independent system eliminates the traditional hydraulic system’s power steering pump, hoses, and hydraulic fluid, as well as the drive belt and pulley on the engine. Instead, the ECS system uses an electric-motor power assist mechanism to provide responsive power steering. The system provides variable assistance with turning the wheels based on the current speed of the car and the amount of torque (turning force) applied to the steering wheel by the driver.

### 4. CASE STUDY

In order to validate the applicability of our approach, we applied it to several examples from industry. In this paper, we present an electronically controlled steering (ECS) system whose requirements involve untimed as well as real-time properties.

#### 4.1 ECS System Description

The ECS system uses an electric-motor power assist mechanism to provide responsive power steering. The system provides variable assistance with turning the wheels based on the current speed of the car and the amount of torque (turning force) applied to the steering wheel by the driver.

#### 4.2 Applying Real-time Specification Patterns

We demonstrate how to use our real-time specification patterns and how to convert prose requirements of embedded systems into formal specifications, focusing on real-time requirements. A list of the time-sensitive requirements for the ECS system is given in Figure 5.

In the ECS study, we use object analysis patterns for em-
bedded systems to create timed UML models. Object analysis patterns contain structural and behavioral information to guide the creation of UML models of the system. In addition, they contain property templates to aid in formally capturing properties to which the system must adhere. The timed UML models are analyzed with the real-time extension of our modeling and analysis framework.\footnote{Our timing extension assumes the digital-clock (or fictitious-clock) model as the underlying model of time and uses the Spin model checker for verification and validation purposes.}

**Real-time**

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>18: realtimeType</td>
<td>“it is always the case that” (durationCategory</td>
</tr>
<tr>
<td>19: durationCategory</td>
<td>“once” P “becomes satisfied, it holds for” (minDurationPattern</td>
</tr>
<tr>
<td>20: minDurationPattern</td>
<td>“at least” c “time unit(s)”</td>
</tr>
<tr>
<td>21: maxDurationPattern</td>
<td>“less than” c “time unit(s)”</td>
</tr>
<tr>
<td>22: periodicCategory</td>
<td>P “holds” boundedRecurrencePattern</td>
</tr>
<tr>
<td>23: boundedRecurrencePattern</td>
<td>“at least every” c “time unit(s)”</td>
</tr>
<tr>
<td>24: realtimeOrderCategory</td>
<td>“if” P “holds, then” S “holds” (boundedResponsePattern</td>
</tr>
<tr>
<td>25: boundedResponsePattern</td>
<td>“after at most” c “time unit(s)”</td>
</tr>
<tr>
<td>26: boundedInvariancePattern</td>
<td>“for at least” c “time unit(s)”</td>
</tr>
</tbody>
</table>

Figure 3: Structured English grammar

2(c) from Figure 3. In addition, we present only properties that are unscoped and assume that the system is always in a mode of normal operation. For illustrative purposes, we abstract Requirement 1(c) in Figure 3 as follows: “Operational checks must be done every 10 milliseconds.”

1. **Requirement 1(c): “Operational checks must be done every 10 milliseconds.”**

This requirement describes a periodic occurrence as it denotes that something has to happen every 10 milliseconds. We assume that an operational check happening more often than every 10 milliseconds is considered correct system behavior. Therefore, it is appropriate to use the Bounded Recurrence Pattern. One time unit corresponds to 0.5 milliseconds in the fine-grained model. Thus, 10 milliseconds are captured as 20 time units. This property is captured by the following globally scoped MTL formula of the Bounded Recurrence Real-time Specification Pattern:

$$\Box(\forall_{\leq 20}(\text{in}(\text{OperationalCheck})))$$

where \(\text{in}(\text{OperationalCheck})\) denotes that the system is in state OperationalCheck, in which it performs
Bounded Recurrence Real-time Specification Pattern

- **Pattern Name and Classification**
  Bounded Recurrence: Periodic Real-time Specification Pattern

- **Structured English Specification**
  *scope*, if it is always the case that $P$ holds at least every $c$ time unit(s).

- **Pattern Intent**
  This pattern describes the periodic satisfaction of a propositional formula. Intuitively, it captures the property that in every $c$ time unit(s), the proposition $P$ has to hold at least once. The proposition $P$ holding more often than every $c$ time units or holding continuously is considered a correct behavior in this pattern.

- **Real-time Temporal Logic Mappings**

  - **MTL:**
    
    | Scope               | MTL Expression |
    |--------------------|----------------|
    | Globally:          | $\Box(\Diamond \leq c P)$ |
    | Before $R$:        | $\Diamond R \rightarrow ((\Diamond \leq c (P \lor R)) \lor R)$ |
    | After $Q$:         | $\Box(Q \rightarrow \Box(\Diamond \leq c P))$ |
    | Between $Q$ and $R$: | $\Box((Q \land \neg R \land \Diamond R) \rightarrow ((\Diamond \leq c (P \lor R)) \lor R))$ |
    | After $Q$ until $R$: | $\Box((Q \land \neg R) \rightarrow ((\Diamond \leq c (P \lor R)) \lor R))$ |

  - **TCTL:**
    
    | Scope               | TCTL Expression |
    |--------------------|-----------------|
    | Globally:          | $\AG(\AF \leq c P)$ |
    | Before $R$:        | $\AG((\AF \leq c (P \lor R)) \lor \AG(\neg R)) \lor R)$ |
    | After $Q$:         | $\AG(Q \rightarrow \AG(\AF \leq c P))$ |
    | Between $Q$ and $R$: | $\AG((Q \land \neg R) \rightarrow \AG((\AF \leq c (P \lor R)) \lor \AG(\neg R)) \lor R))$ |
    | After $Q$ until $R$: | $\AG((Q \land \neg R) \rightarrow \AG((\AF \leq c (P \lor R)) \lor R))$ |

  - **RTGIL:**
    
    (a) Globally

    (b) Between $Q$ and $R

    (The scopes "After $Q", "Before $R", and "After $Q$ until $R" for RTGIL have been elided due to space constraints and can be found in [31].)

- **Examples and Known Uses**
  This pattern is commonly used in embedded systems, as these systems commonly perform periodic tasks. For example, a watchdog has to become active at least every $c$ time unit(s) and verify that certain system constraints are not violated. Additionally, embedded systems often have to perform specific services periodically, such as sending a heartbeat (a message denoting that the embedded system is functioning correctly) to other embedded systems using a communication device, *e.g.*, a controller area network (CAN) bus [47].

- **Relationships**
  The untimed version of the bounded recurrence property, expressed as $\Box(\Diamond P)$ in LTL, can be found in several publications (such as [28]). It is commonly used to specify the absence of non-progress cycles in a system.

Figure 4: Bounded Recurrence Real-time Specification Pattern
1. Fine-Grained ECS Requirements (Time Granularity 0.5 milliseconds)
   (a) The input torque value must be converted to an assisting torque value at least every 500 microseconds (or 0.5 milliseconds).
   (b) The conformity of the torque sensors to within five percent of each other must be verified every ten milliseconds.
   (c) Operational checks must be done every 10 milliseconds. These checks include an external watchdog verification, a RAM verification, and a flash memory verification.

2. Coarse-Grained ECS Requirements (Time Granularity 100 milliseconds)
   (a) Once every second, a fault status report must be sent over a controller area network (CAN) communication link.
   (b) Upon power up, the malfunction indicator light must be illuminated for three seconds.
   (c) In case of a system shutdown, the assisting torque should be gradually ramped down over two seconds.

Figure 5: Time-sensitive requirements of the ECS system

This MTL formula translates to the following structured English sentence:

“Globally, it is always the case that if \((\text{RampDownInitiated} = \text{true})\) holds, then \((\text{AssistTorque} \neq 0)\) holds for at least 19 time unit(s).” (Grammar: 1, 2, 3, 18, 24, 26).

5. RELATED WORK

Due to the nature of our two-pronged approach, we cover a number of different types of related projects, ranging from natural-language-based approaches for formal specifications to other formats for real-time properties.

Several projects have explored a natural-language-based approach to formal specifications. Flake et al. [19] introduced a syntax and semantics for a subset of English to specify clocked computational tree logic (CCTL) properties based on the specification patterns by Dwyer et al. In contrast to the work presented in this paper, they focus exclusively on CCTL and do not include other real-time temporal logics or support for scopes. Their grammar is also not entirely based on a pattern system. Numerous approaches [6, 10, 18, 20, 27, 40, 42] construct formal specifications in different forms (such as temporal logics, OO-based representations, Prolog specifications), from natural language to support a variety of tasks, ranging from completeness and consistency checking to formal validation and verification. Often, informal specifications are initially mapped to an intermediate representation, at which point context dependencies and ambiguities are resolved. The result is then further mapped into the targeted formal specification language(s). While these approaches allow the use of an only slightly restricted natural language (a completely unrestricted language is considered undesirable for practical and technical reasons [42]), this type of extraction requires advanced natural language processing approaches and techniques to deal with imprecision and ambiguities inherent to natural language specifications. In general, this type of extraction is more ambitious than our objectives for creating natural language representations. In conclusion, none of the aforementioned approaches combines the completeness of a pattern system, the support for real-time properties, amenability for formal validation and verification with a wide variety of for-
nal validation and verification tools, and the accessibility of a natural language representation.

Other related work addresses the definition and use of specification-oriented patterns. Smith et al. developed Propel [35], where they extended the specification patterns by Dwyer et al. [14] to address important and subtle aspects about a property, such as what happens in a cause-effect relation if the cause recurs before the effect has occurred. Differing from our approach, these extensions do not offer support for including real-time information. In addition, the extended specification patterns are captured in terms of a finite-state automata based extension, and no temporal logic representations are currently offered. As our specification patterns can also be captured in terms of timed automata, extensions similar to the ones used in Propel could also be investigated (an algorithm that translates MTL formulae to timed Büchi automata is presented in [33]). Smith et al. also offer disciplined natural language templates that help a specifier to precisely capture a property in a natural language. The natural language templates are similar to the natural language representations that can be derived from our grammar. Nevertheless, the natural language templates focus on capturing the subtle properties of individual specification patterns, while our grammar covers the entire pattern system. Darmont et al. [12] introduced a library of generic refinement patterns that enable developers to construct detailed requirements from goals without having to understand the underlying mathematics of temporal logic refinements. More recently, Letier et al. [35] described goal patterns to specify commonly occurring goals that include real-time information, but these goal patterns do not incorporate scopes and focus on one particular real-time temporal logic only. Similar to the idea of real-time specification patterns, the MT toolset [13] offers real-time logic (RTL) templates with placeholders in strategic places.

Finally, examples of how to capture real-time properties amenable for formal analysis can be found in numerous papers [12, 20, 21, 24, 33, 34]. But in the aforementioned approaches, temporal logic formulae are never extended and structured into a pattern system in the spirit of Dwyer et al. [14]. In addition, the examples given are usually limited to one particular temporal logic and do not incorporate any scopes.

6. CONCLUSIONS

Our work with real-time specification patterns has yielded two main contributions. First, our pattern system enables developers to express common real-time properties of a system in a precise manner that can be analyzed using model checking or theorem proving techniques. In this paper, we demonstrated that although the three real-time temporal logics we have chosen differ considerably in their syntax and semantics, it is still possible to specify several commonly occurring real-time requirements in all three logics. This multi-pronged approach enables specifiers to use the analysis tool and specification language best suited for the model and property under consideration. Second, the structured English grammar greatly facilitates the understanding of the meaning of a temporal logic property, thereby making our approach more accessible to non-experts in formal methods. The grammar we provide can be used to create natural language representations for qualitative and real-time properties of a system.

Several directions for future work are possible. Tool support is being developed that, in combination with our formalization framework for embedded systems [30, 39], guides the user in deriving, instantiating, and analyzing qualitative and real-time properties [32]. This guidance is provided in the form of a syntax-guided approach to property derivation based on our structured English grammar, and property instantiation is based on information automatically extracted from a selected system model. The real-time specification pattern repository could also be extended with other commonly used real-time specification techniques beyond the temporal logics addressed in this paper, such as AS-TRAL [22], RTL [20], RTTL [11], TILCO [35], TPSTL [15, 16], and timed automata [2, 33]. We expect that different combinations of temporal logics will lead to different patterns. Finally, to extend the real-time pattern repository, requirements documents from other application domains, such as telecommunication and mobile computing systems, could be analyzed in an effort to uncover additional real-time specification patterns.

Acknowledgements

The authors are grateful for feedback regarding this work and answers to our questions received from Laura K. Dillon, Stavros Tripakis, Sergio Yovine, and Laura A. Campbell.

7. REFERENCES


temporal logic specifications. ACM Transactions on Programming Languages and Systems, (2):244–263, April 1986.


