This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2945329, IEEE Transactions on Software Engineering

Specification Patterns for Robotic Missions

Claudio Menghi, Christos Tsigkanos, Patrizio Pelliccione, Carlo Ghezzi, and Thorsten Berger

Abstract—Mobile and general-purpose robots increasingly support our everyday life, requiring dependable robotics control software. Creating such software mainly amounts to implementing their complex behaviors known as missions. Recognizing this need, a large number of domain-specific specification languages has been proposed. These, in addition to traditional logical languages, allow the use of formally specified missions for synthesis, verification, simulation or guiding implementation. For instance, the logical language LTL is commonly used by experts to specify missions as an input for planners, which synthesize the behavior a robot should have. Unfortunately, domain-specific languages are usually tied to specific robot models, while logical languages such as LTL are difficult to use by non-experts. We present a catalog of 22 mission specification patterns for mobile robots, together with tooling for instantiating, composing, and compiling the patterns to create mission specifications. The patterns provide solutions for recurrent specification problems, each of which detailing the usage intent, known uses, relationships to other patterns, and—most importantly—a template mission specification in temporal logic. Our tooling produces specifications expressed in the temporal logics LTL and CTL to be used by planners, simulators or model checkers. The patterns originate from 245 realistic textual mission requirements extracted from the robotics literature, and they are evaluated upon a total of 441 real-world mission requirements and 1251 mission specifications. Five of these reflect scenarios we defined with two well-known industrial partners developing human-size robots. We validated our patterns' correctness with simulators and two different types of real robots.

1 Introduction

Mobile robots are increasingly used in complex environments aiming at autonomously realizing missions [9]. The rapid pace of development in robotics hardware and technology demands software that can sustain this growth [10], [11], [12], [13]. Even though existing solutions are not readily usable [14], in the near future robots will be used for accomplishing tasks of everyday life by end-users with no expertise and knowledge in computer science, robotics, mathematics or logics. Providing techniques that support robotic software development is a major software-engineering challenge [10], [15]. Indeed, as in the mobile application domain, where electrical engineers develop low level hardware components and constructing higher level software components that are executed on mobile devices is a software engineering issue, in the robotic domain robotic engineers develop robots and low level software primitives that allow controlling and managing these robots and developing and defining software that uses those low level primitives is a software engineering issue.

The mission describes the high-level tasks the robotic software must accomplish [16]. Among the different ways of describing missions that were proposed in the literature [17], in this work, we consider declarative specifications [18]. These describe the final outcome the software should achieve—rather than describing how to achieve it—and are prominently used in the robotics domain [17]. Precisely specifying missions and transforming them into a form useful for automatic processing are among the main challenges in engineering robotics software [19], [20]. On the one hand, missions should be defined with a notation that is high-level and user-friendly [16], [21]. On the other hand, to enable automatic processing, the notation should be unambiguous and provide a formal and precise description of what robots should do in terms of movements and actions [22], [23], [24].

Engineering robotics software typically amounts to expressing the robotic mission in natural language (henceforth called *mission requirement*), and then translating mission requirements into more precise *mission specifications*. The latter are often expressed in a domain-specific language, many of which have been proposed over the last decades [17]. These languages are often integrated with development environments used to generate code that can be executed within simulators or real robots [25], [26], [27], [28]. However, these languages are typically bound to specific types of robots and support a limited number and type of missions. Other works especially from the robotics domain, advocate the use of formally specified missions in temporal logics [29], [30], [31], [32]. Unfortunately, specifying missions using temporal logic formulae can be too complex and error-prone for practitioners or engineers. As such, defining robotic missions is generally challenging, as widely recognized in the software-engineering and robotics communities [33], [34], [35], [36].

Mission requirements are often ambiguous, hindering precise and unambiguous specification [36], [37], [38]. Consider the very simple mission requirement "the robot shall visit the kitchen and the office." This can be interpreted as "visit the kitchen" and also that at some point the robot should "visit the office" without a specific order between the visit of the kitchen and the office, or as visit "the kitchen and the office in order." Assume that the correct intended behavior requires that "the kitchen and the office are visited in order," which is a common mission specification problem [39], [40]. When transforming this mission requirement expressed in natural language into a precise mission specification, an expert might come up with the following formula in temporal logic:

$$\phi_1 = \mathcal{F}\left((r\ in\ l_1) \wedge \mathcal{F}(r\ in\ l_2)\right),$$

where $r\ in\ l_1$ and $r\ in\ l_2$ signify that robot r is in the kitchen and office, respectively, and $\mathcal{F}$ denotes *finally*. Now, recall that the actual mission requirement is that the robot reaches the kitchen *before* the office. It is important to highlight that the logical formula still admits that the robot reaches the office

before entering the kitchen, which may be an unintended behavior. In fact, a possible interpretation might also require that a robot should visit the office after having visited the kitchen and that the robot absolutely cannot visit the office before having visited the kitchen. This alternative interpretation requires defining additional behavioral constraints. A correct formula, among others, is the following:

$$\phi_2 = \phi_1 \wedge \big((\neg(r \; in \; l_2)) \; \mathcal{U} \; (r \; in \; l_1)\big),$$

where $\mathcal{U}$ stands for *until*. Further interpretations are also possible. This highlights the ambiguity in natural language requirements formulation, and common mistakes may be introduced when diverse interpretations are given [24], [41], [42], [43]. The additional constraint added in the last interpretation above requires the office to not be visited before the kitchen, recalling a specification pattern for temporal logics known as the *absence pattern* [44]. Rather than conceiving such specifications recurrently in an ad hoc way with the risk of introducing mistakes, engineers could re-use validated solutions to existing mission requirements.

Creating mission specifications that correctly capture mission requirements is hard and error-prone [33], [34], [35], [36], also evident from the examples above. The challenge of defining behavioral properties in logical languages such as LTL, has been recognized by researchers. While precise behavioral specifications in logical languages enable reasoning about behavioral properties [45], [46], their specification is hard and error prone [47], [48]. Practitioners are often unfamiliar with the specification process as well as with the intricate syntax and semantics of logical languages [44]. Specification patterns have become a popular solution to this challenge. For instance, Dwyer et al. [44] introduced patterns for safety properties, which were later extended by Grunske [49] and Konrad et al. [50] to address real-time and probabilistic quality properties, respectively. Autili et al. [51] consolidated and organized these patterns into a comprehensive catalog. Bianculli et al. [52] applied specification patterns to the domain of web services. All these patterns provide template solutions that can be used to specify the respective properties. However, none of these pattern catalogs focuses on the robotic domain to solve the mission specification problem. Our contribution enriches this line of research by focusing on the new emerging domain of mobile robots, whose missions need to be expressed in precise terms by users who are not proficient in formal specifications. It follows a typical research paradigm in engineering that tries to replicate, contextualize, and extend an existing useful method to a different domain, which has its own specificities.

We propose a new set of patterns focusing on robot movement as one of the major aspects considered in the robotics domain [53], [54], [55], as well as on how robots perform actions within their environment. For each pattern we provide its usage intent, known uses, relationships to other patterns, and—most importantly—a template mission specification in temporal logics. The latter relies on LTL and CTL as the most widely used formal specification languages in robotics [27], [29], [30], [32], [56], [57], [58], [59], [60], [61], [62]. The template mission specification can be defined in multiple languages that may have different expressiveness – the patterns we provide lie in the intersection of LTL and CTL. These logical formalisms are sufficiently expressive, since missions that contain explicit time requirements are beyond the scope of this work, and subject of future investigation. Our catalog has been produced by analyzing 245 natural-language mission requirements systematically retrieved from the robotics literature. From these requirements we identified recurrent mission specification problems and conceived solutions organized as patterns. Our patterns provide a formally defined vocabulary that supports robotics developers in defining mission requirements. Relying on the usage of the pattern catalog as a common vocabulary allows mitigating ambiguous natural language formulations [34]. Our patterns also provide validated mission specifications for recurrent mission requirements, facilitating the creation of correct mission specifications.

We implemented the tool PsALM (Pattern bAsed Mission specifier)to further support developers in designing missions. PsALM allows (i) specifying a mission requirement through a structured English grammar, which uses patterns as basic building blocks and operators that allow composing these patterns into complex missions, and (ii) automatically generating specifications from mission requirements. PsALM is robot-agnostic and integrated with Spectra [63] (a robot development environment), a planner [30], NuSMV [64] (a model checker), and Simbad [65] (a simulator for education and research). The pattern catalog and the PsALM tool are available in an online appendix [66].

We validated the correctness of the proposed patterns. The methodology we conceived is generic and can be reused in future work that propose pattern catalogs. Specifically, we characterize all (and only) the set of behaviors that were expected to be admitted by a mission requirement by manually defining an ω-regular expression. This ω-regular expression is compared with the set of behaviors admitted by an LTL formula by using standard language inclusion procedures. To further build confidence for the absence of errors on the definition of the ω-regular expression, we additionally tested patterns correctness on a set of 12 randomly generated models representing buildings where a robot is deployed. We considered ten mission requirements (each obtained by combining three patterns), converted the mission requirements into LTL mission specifications and used those to generate robots' plans. We used the Simbad [65] simulator to verify that the plans satisfied the intended mission requirement. We subsequently generated both LTL and CTL specifications from the considered mission requirements. We verified that the same results are obtained when they are checked on the randomly generated models, confirming the correspondence among the CTL and LTL specifications.

We evaluated the benefits of using our patterns for designing missions. We collected 441 mission requirements in natural language: 436 obtained from robotic development environments used by practitioners (i.e., Spectra [63] and LTLMoP [31], [36]), and five defined in collaboration with two well-known robotics companies developing commercial, human-size service robots (BOSCH and PAL Robotics). We show that most of the mission requirements were ambiguous but expressible using the proposed patterns, and that usage of patterns reduces ambiguities. We then evaluated the coverage of mission specifications. We collected 1229 LTL and 22 CTL mission specifications from robotic development

environments used by practitioners (i.e., Spectra [63] and LTLMoP [31], [36]) and research publications (i.e., [67]) and show that almost all specifications can be obtained using the proposed patterns (1154 over 1251, i.e., $\approx 92\%$). We also generated specifications for five mission requirements defined in collaboration with the two robotic companies and fed them into an existing planner. The produced plans were correctly executed by real robots, namely Tiago and Turtlebot[1] showing the benefits of the patterns support in real scenarios. Finally, we also showed that the LTL formulation of the patterns is within the GR(1) fragment, enabling the use of existing reactive synthesis tools (e.g., [29]).

A small fragment of this work has been published as an extended abstract [68] and a tool-demo paper [69]. In these papers, we presented the initial idea and a description of our toolset for early dissemination. This paper presents our work in its full richness, explaining the methodology, presenting all patterns with their formalization, as well as our evaluation of the patterns' correctness and benefit using real-world mission requirements and mission specifications.

We proceed by presenting background information and important terminology in Section 2, and by describing our research methodology in Section 3. We present our pattern catalog in Section 4, and tool support in Section 5. We evaluate patterns' correctness in Section 6 and their benefits in Section 7. We discuss our findings in Section 8, related work in Section 9, and conclude in Section 10.

2 Background

In this section, we present the terminology used in the remainder and introduce the temporal logic LTL used for defining the patterns' template solutions.

Recall that for communication and further refinement, the requirements of a software system are typically expressed in natural language or informal models. Refining these requirements into more formal representations avoids ambiguity, allowing automated processing and analysis. Such practices also emerged in the robotics engineering domain.

- *Mission Requirement*: a description in a natural language or in a domain-specific language of the mission (also called "task") the robots must perform [37].
- *Mission Specification*: a formulation of the mission in a logical language with a precise semantics [57] .
- *Mission Specification Problem*: the problem of generating a mission specification from a mission requirement.
- *Mission Specification Pattern*: a mapping between a recurrent mission-specification problem to a template solution and a description of the usage intent, known uses, and relationships to other patterns.
- *Mission Specification Pattern Catalog*: a collection of mission specification patterns organized in a hierarchy aiding at browsing and selecting patterns, in order to support decision making during mission specification.

We consider LTL (Linear Temporal Logic) [70] and CTL (Computation Tree Logic) [71], since they are commonly used to express mission specifications in robotics and are utilized extensively by the community (e.g., [32], [56], [58]). A temporal logic specification can be used for several purposes, such as (i) for producing plans through the use of planners, (ii) for analysing the mission satisfaction though the use of model checkers, and (iii) to design a robotic application.

1. Tiago (tiago.pal-robotics.com) and Turtlebot (turtlebot.com).

Table 1
Papers (requirements) analyzed per venue and year

Robotics Venue	2017	2016	2015	2014	2013	Total
Int. Conf. Robotics & Autom.	9(14)	16 (11)	17 (18)	27 (22)	16 (15)	85 (80)
Int. J. of Robotics Research	4(8)	13 (12)	12 (11)	13 (8)	17 (12)	59 (51)
Trans. on Robotics	2(6)	12 (9)	5 (1)	8 (2)	4 (2)	31 (20)
Int. Conf. on Int. Robots & Sys.	10(23)	55 (26)	13 (8)	20 (16)	33 (21)	131 (94)

We now briefly recall LTL's syntax and semantics; a precise treatment can be found in specialized text books (e.g., Baier and Katoen [5]). While CTL has also been considered as a target logic to define patterns' template solutions, it is not introduced explicitly as this paper will use LTL as a reference temporal logic. For additional details on the use of CTL in the formulation of the proposed pattern the interested reader may consult our online appendix [66]. Let π be a set of atomic propositions; LTL's syntax is defined as follows:

$$\text{(LTL)}\ \ \phi ::= \tau \mid \neg\phi \mid \phi \vee \phi \mid \ \mathcal{X}\,\phi \mid \phi\ \mathcal{U}\ \phi \text{ where } \tau \in \pi.$$

The semantics of LTL is defined over an infinite sequence of truth assignments to the propositions π. The formula $\mathcal{X}\,\phi$ expresses that ϕ is true in the next position in a sequence, and the formula $\phi_1\ \mathcal{U}\ \phi_2$ expresses the property that ϕ_1 is true until ϕ_2 holds. The eventually $\mathcal{F}$, always $\mathcal{G}$ and weak until $\mathcal{W}$ operators can be obtained from the $\mathcal{X}$ and $\mathcal{U}$ LTL operators as usual [5].

3 Methodology

We derived our pattern catalog in three main steps: (i) collection of mission requirements, (ii) identification of mission specification problems, and (iii) pattern formulation.

Collection of Mission Requirements. We collected mission requirements from scientific papers in the field of robotics. We additionally considered the software engineering literature, but noted a general absence of robotic mission specifications. We chose major venues based on consultation with domain experts and by considering their impact factor. Specifically, we analyzed mission specifications published in the four major robotics venues [72] over the last five years, in line with similar studies for pattern identification [44], [49], [50]. We analyzed all papers published within a venue with two inclusion criteria (considered in order): (i) the paper title implies some notion of robotic movement-related concept, (ii) the paper contains at least one formulation of a mission requirement involving a robot that concerns movement. When the paper contained more than one mission requirement, each was considered separately.

Altogether we obtained 306 papers, through which, matching our inclusion criteria, we obtained 245 mission requirements. Table 1 shows the venues included in our analysis, together with the number of scientific publications and mission requirements obtained per year. The considered software engineering venues (ICSE, FSE, and ASE) are not present since they did not contain any paper matching the inclusion criteria.

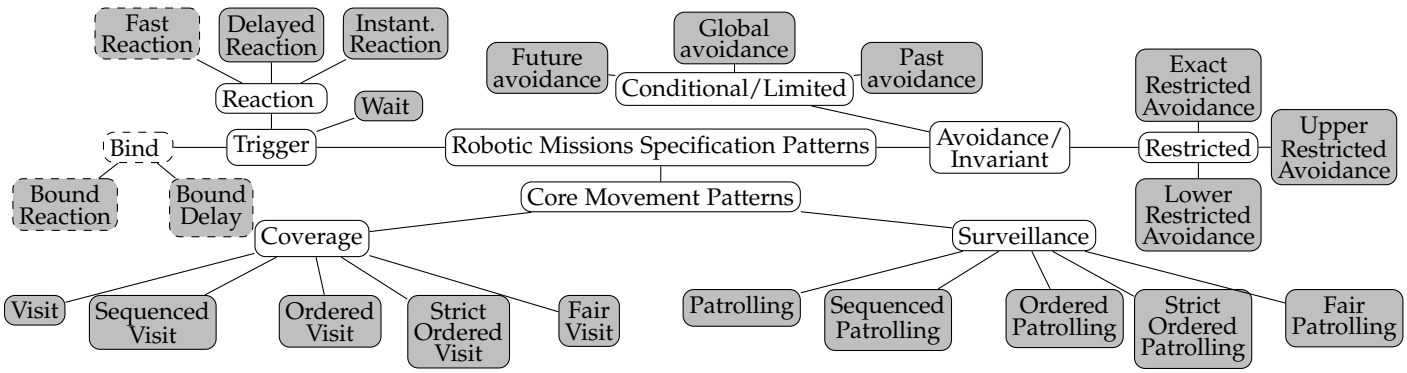


Figure 1. Mission specification pattern catalog. Filled nodes: patterns, non-filled nodes: categories.

Identification of Mission Specification Problems. We identified mission specification problems as follows[2].

- (STEP.1) We divided the collected mission requirements among two of the authors, who labeled them with keywords that capture the mission specification problems they describe. For example, the mission requirement "The robot has to autonomously patrol the site and measure the state of valve levers and dial gauges at four checkpoints in order to decide if some machines need to be shut down" (occuring in Schillinger et al. [73]) was associated with the keywords "patrol," since the robot has to patrol the site, and "instantaneous reaction," since when a valve is reached its level must be checked.
- (STEP.2) We created a graph structure representing semantic relations between keywords. Each keyword is associated with a node of the graph structure. Two nodes were connected if their keywords identify two similar mission specification problems. For example, the keywords "visit" and "reach" are related since in both cases the robot has to visit/reach a location.
- (STEP.3) Since our interest was not a mere classification of actions and movements that are executed by a robots, but rather detecting mission specification problems that concern how actions and movements are executed by a robot behavior over time, nodes that contain keywords that only refer to actions are removed (e.g., balance).
- (STEP.4) Nodes that were connected through edges and contained keywords that identify to the same mission specification problem, e.g., visit and reach, were merged.
- (STEP.5) We hierarchically organized the mission specification problems into a catalog represented through a tree structure that facilitates browsing among mission specification problems.

Pattern Formulation. We formulated patterns by following established practices in the literature [44], [49], [51]. A pattern is characterized by

- (i) a name;
- (ii) a statement that captures the pattern intent (i.e., the mission requirement);
- (iii) a template instance of the mission specification in LTL and CTL;
- (iv) variations describing possible minor changes that can be applied to the pattern;
- (v) examples of known uses;
- (vi) relationships of the pattern to others and;
- (vii) occurrences of the pattern in literature.

For each LTL pattern we also designed a Büchi Automaton (BA) that unambiguously describes the behaviors of the system allowed by the mission specification. The mission specification was designed by consulting specifications encoding requirements already present in the papers surveyed, by crosschecking them, and consulting specification patterns already proposed in the software-engineering literature [51]. If the proposed specification was related to (or corresponded with) one of an already existing pattern, we indicated this in the relationships of the pattern to others, meaning that the pattern presented in the literature is also useful to solve the identified mission specification problem.

4 MISSION SPECIFICATION PATTERNS

In this section, we present our catalog of mission specification patterns[3] and present one of them. Our catalog comprises 22 patterns hierarchically organized into a pattern tree as illustrated in Figure 1. Leaves of the tree represent mission specification patterns. Intermediate nodes facilitate browsing within the hierarchy and aid pattern selection and decision making. Patterns identified by following the procedure described in Section 3 are graphically indicated with a solid border. Patterns represented with a dashed border represent new patterns identified during our evaluation, as explained below in Section 7.1.

We provide a high-level description of all patterns identified, examples of application, and the corresponding LTL mission specifications. The interested reader may refer to our online appendix [66], which contains additional examples, occurrences of patterns in the literature, relations among the patterns, and additional CTL mission specifications.

Preliminaries. To aid comprehension of behavior and facilitate precise pattern definitions, we introduce the following notation. Given a finite set of locations $L = \{l_1, l_2, \ldots, l_n\}$ and robots $R = \{r_1, r_2, \ldots, r_n\}$, $PL = \{r_x \; in \; l_y \mid r_x \in R \text{ and } l_y \in L\}$ is a set of location propositions, each indicating that a robot r_x is in a specific location l_y of the environment. Given a finite set of conditions of the environment $C = \{c_1, c_2, \ldots, c_m\}$, we indicate as $PE = \{s_1, s_2, \ldots, s_m\}$

2. For technical details on this methodology see [66].

3. The pattern catalog in full, accompanied material and tool support is available on our dedicated website: www.roboticpatterns.com

Table 2
Core movement patterns. The symbol $\%$ indicates the mathematical operator modulo.

	Description	**Example**	**Formula** ($l_1, l_2, \ldots$ are location propositions)
Visit	Visit a set of locations in an unspecified order.	Locations l_1, l_2, and l_3 must be visited. $l_1 \rightarrow l_4 \rightarrow l_3 \rightarrow l_1 \rightarrow l_4 \rightarrow l_2 \rightarrow (l_\#)^\omega$ is an example trace that satisfies the mission requirement.	$\bigwedge_{i=1}^{n} \mathcal{F}(l_i)$
Sequenced Visit	Visit a set of locations in sequence, one after the other.	Locations l_1, l_2, l_3 must be covered following this sequence. The trace $l_1 \rightarrow l_4 \rightarrow l_3 \rightarrow l_1 \rightarrow l_4 \rightarrow l_2 \rightarrow (l_{\#\setminus 3})^\omega$ violates the mission since l_3 does not follow l_2. The trace $l_1 \rightarrow l_3 \rightarrow l_1 \rightarrow l_2 \rightarrow l_4 \rightarrow l_3 \rightarrow (l_\#)^\omega$ satisfies the mission requirement.	$\mathcal{F}(l_1 \wedge \mathcal{F}(l_2 \wedge \ldots \mathcal{F}(l_n)))$
Ordered Visit	The sequenced visit pattern does not forbid to visit a successor location before its predecessor, but only that after the predecessor is visited the successor is also visited. Ordered visit forbids a successor to be visited before its predecessor.	Locations l_1, l_2, l_3 must be covered following this order. The trace $l_1 \rightarrow l_3 \rightarrow l_1 \rightarrow l_2 \rightarrow l_3 \rightarrow (l_\#)^\omega$ does not satisfy the mission requirement since l_3 preceeds l_2. The trace $l_1 \rightarrow l_4 \rightarrow l_1 \rightarrow l_2 \rightarrow l_4 \rightarrow l_3 \rightarrow (l_\#)^\omega$ satisfies the mission requirement.	$\mathcal{F}(l_1 \wedge \mathcal{F}(l_2 \wedge \ldots \mathcal{F}(l_n)))$ $\bigwedge_{i=1}^{n-1} (\neg l_{i+1}) \,\mathcal{U}\, l_i$
Strict Ordered Visit	The ordered visit pattern does not avoid a predecessor location to be visited multiple times before its successor. Strict ordered visit forbids this behavior.	Locations l_1, l_2, l_3 must be covered following the strict order l_1, l_2, l_3. The trace $l_1 \rightarrow l_4 \rightarrow l_1 \rightarrow l_2 \rightarrow l_4 \rightarrow l_3 \rightarrow (l_\#)^\omega$ does not satisfy the mission requirement since l_1 occurs twice before l_2. The trace $l_1 \rightarrow l_4 \rightarrow l_2 \rightarrow l_4 \rightarrow l_3 \rightarrow (l_\#)^\omega$ satisfies the mission requirement.	$\mathcal{F}(l_1 \wedge \mathcal{F}(l_2 \wedge \ldots \mathcal{F}(l_n)))$ $\bigwedge_{i=1}^{n-1} (\neg l_{i+1}) \,\mathcal{U}\, l_i$ $\bigwedge_{i=1}^{n-1} (\neg l_i) U (l_i \wedge \mathcal{X}(\neg l_i \,\mathcal{U}(l_{i+1})))$
Fair Visit	The difference among the number of times locations within a set are visited is at most one.	Locations l_1, l_2, l_3 must be covered in a fair way. The trace $l_1 \rightarrow l_4 \rightarrow l_1 \rightarrow l_3 \rightarrow l_1 \rightarrow l_4 \rightarrow l_2 \rightarrow (l_{\#\setminus\{1,2,3\}})^\omega$ does not perform a fair visit since it visits l_1 three times while l_2 and l_3 are visited once. The trace $l_1 \rightarrow l_4 \rightarrow l_3 \rightarrow l_1 \rightarrow l_4 \rightarrow l_2 \rightarrow l_2 \rightarrow l_4 \rightarrow (l_{\#\setminus\{1,2,3\}})^\omega$ performs a fair visit since it visits locations l_1, l_2, and l_3 twice.	$\bigwedge_{i=1}^{n} \mathcal{F}(l_i)$ $\bigwedge_{i=1}^{n} \mathcal{G}(l_i \rightarrow \mathcal{X}((\neg l_i) \,\mathcal{W}\, l_{(i+1)\%n}))$
Patrolling	Keep visiting a set of locations, but not in a particular order. The patrolling problem also appears in literature as *surveillance*.	Locations l_1, l_2, l_3 must be surveilled. The trace $l_1 \rightarrow l_4 \rightarrow l_3 \rightarrow l_1 \rightarrow l_4 \rightarrow l_2 \rightarrow (l_2 \rightarrow l_3 \rightarrow l_1)^\omega$ ensures that the mission requirement is satisfied. The trace $l_1 \rightarrow l_2 \rightarrow l_3 \rightarrow (l_1 \rightarrow l_3)^\omega$ represents a violation, since l_2 is not surveilled.	$\bigwedge_{i=1}^{n} \mathcal{G}\,\mathcal{F}(l_i)$
Sequenced Patrolling	Keep visiting a set of locations in sequence, one after the other.	Locations l_1, l_2, l_3 must be patrolled in sequence. The trace $l_1 \rightarrow l_4 \rightarrow l_3 \rightarrow l_1 \rightarrow l_4 \rightarrow l_2 \rightarrow (l_1 \rightarrow l_2 \rightarrow l_3)^\omega$ satisfies the mission requirement since globally any l_1 will be followed by l_2 and l_2 by l_3. The trace $l_1 \rightarrow l_4 \rightarrow l_3 \rightarrow l_1 \rightarrow l_4 \rightarrow l_2 \rightarrow (l_1 \rightarrow l_3)^\omega$ violates the mission requirement since after visiting l_1, the robot does not visit l_2.	$\mathcal{G}(\mathcal{F}(l_1 \wedge \mathcal{F}(l_2 \wedge \ldots \mathcal{F}(l_n))))$
Ordered Patrolling	Sequence patrolling does not forbid to visit a successor location before its predecessor. Ordered patrolling ensures that (after a successor is visited) the successor is not visited (again) before its predecessor.	Locations l_1, l_2, and l_3 must be patrolled following the order l_1, l_2, and l_3. The trace $l_1 \rightarrow l_4 \rightarrow l_3 \rightarrow l_1 \rightarrow l_4 \rightarrow l_2 \rightarrow (l_1 \rightarrow l_2 \rightarrow l_3)^\omega$ violates the mission requirement since l_3 precedes l_2. The trace $l_1 \rightarrow l_1 \rightarrow l_2 \rightarrow l_4 \rightarrow l_4 \rightarrow l_3 \rightarrow (l_1 \rightarrow l_2 \rightarrow l_3)^\omega$ satisfies the mission requirement	$\mathcal{G}(\mathcal{F}(l_1 \wedge \mathcal{F}(l_2 \wedge \ldots \mathcal{F}(l_n))))$ $\bigwedge_{i=1}^{n-1} (\neg l_{i+1}) \,\mathcal{U}\, l_i$ $\bigwedge_{i=1}^{n} \mathcal{G}(l_{(i+1)\%n} \rightarrow \mathcal{X}((\neg l_{(i+1)\%n}) \,\mathcal{U}\, l_i))$
Strict Ordered Patrolling	The ordered patrolling pattern does not avoid a predecessor location to be visited multiple times before its successor. Strict Ordered Patrolling ensures that, after a predecessor is visited, it is not visited again before its successor.	Locations l_1, l_2, l_3 must be patrolled following the strict order l_1, l_2, and l_3. The trace $l_1 \rightarrow l_4 \rightarrow l_1 \rightarrow l_2 \rightarrow l_4 \rightarrow l_3 \rightarrow (l_1 \rightarrow l_2 \rightarrow l_3)^\omega$ violates the mission requirement since l_1 is visited twice before l_2. The trace $l_1 \rightarrow l_4 \rightarrow l_2 \rightarrow l_4 \rightarrow l_3 \rightarrow (l_1 \rightarrow l_2 \rightarrow l_3)^\omega$ satisfies the mission requirement.	$\mathcal{G}(\mathcal{F}(l_1 \wedge \mathcal{F}(l_2 \wedge \ldots \mathcal{F}(l_n))))$ $\bigwedge_{i=1}^{n-1} (\neg l_{i+1}) \,\mathcal{U}\, l_i$ $\bigwedge_{i=1}^{n} \mathcal{G}(l_{(i+1)\%n} \rightarrow \mathcal{X}((\neg l_{(i+1)\%n}) \,\mathcal{U}\, l_i))$ $\bigwedge_{i=1}^{n-1} \mathcal{G}((l_i) \rightarrow \mathcal{X}(\neg l_i \,\mathcal{U}(l_{(i+1)\%n})))$
Fair Patrolling	Keep visiting a set of locations and ensure that the difference among the number of times locations within a set are visited is at most one.	Locations l_1, l_2, and l_3 must be fair patrolled. The trace $l_1 \rightarrow l_4 \rightarrow l_3 \rightarrow l_1 \rightarrow l_4 \rightarrow l_2 \rightarrow (l_1 \rightarrow l_2 \rightarrow l_1 \rightarrow l_3)^\omega$ violates the mission requirements since the robot patrols l_1 more than l_2 and l_3. The trace $l_1 \rightarrow l_4 \rightarrow l_3 \rightarrow l_4 \rightarrow l_2 \rightarrow l_4 \rightarrow (l_1 \rightarrow l_2 \rightarrow l_3)^\omega$ satisfies the mission requirement since locations l_1, l_2, and l_3 are patrolled fairly.	$\bigwedge_{i=1}^{n} \mathcal{G}(\mathcal{F}(l_i))$ $\bigwedge_{i=1}^{n} \mathcal{G}(l_i \rightarrow \mathcal{X}((\neg l_i) \,\mathcal{W}\, l_{(i+1)\%n}))$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2945329, IEEE Transactions on Software Engineering

Table 3
Avoidance patterns.

	Description	Example	Formula
Past avoidance	A condition has been fulfilled in the past.	If the robot enters location l_1, then it should have not visited location l_2 before. The trace $l_3 \rightarrow l_4 \rightarrow l_1 \rightarrow l_2 \rightarrow l_4 \rightarrow l_3 \rightarrow (l_2 \rightarrow l_3)^\omega$ satisfies the mission requirement since location l_2 is not entered before location l_1.	$(\neg(l_1))\,\mathcal{U}\,p$, where $l_1 \in L$ and $p \in M$
Global avoidance	An avoidance condition globally holds throughout the mission.	The robot should avoid entering location l_1. Trace $l_3 \rightarrow l_4 \rightarrow l_3 \rightarrow l_2 \rightarrow l_4 \rightarrow l_3 \rightarrow (l_3 \rightarrow l_2 \rightarrow l_3)^\omega$ satisfies the mission requirement since the robot never enters l_1.	$\mathcal{G}(\neg(l_1))$, where $l_1 \in L$
Future avoidance	After the occurrence of an event, avoidance has to be fulfilled.	If the robot enters l_1, then it should avoid entering l_2 in the future. The trace $l_3 \rightarrow l_4 \rightarrow l_3 \rightarrow l_1 \rightarrow l_4 \rightarrow l_3 \rightarrow (l_3 \rightarrow l_2 \rightarrow l_3)^\omega$ does not satisfy the mission requirement since l_2 is entered after l_1.	$\mathcal{G}((c) \rightarrow (\mathcal{G}(\neg(l_1))))$, where $c \in M$ and $l_1 \in PL$
Upper Rest. Avoidance	A restriction on the maximum number of occurrences is desired.	A robot has to visit l_1 at most (less than) 3 times. The trace $l_1 \rightarrow l_4 \rightarrow l_1 \rightarrow l_3 \rightarrow l_1 \rightarrow l_4 \rightarrow l_1 \rightarrow (l_3)^\omega$ violates the mission requirement since l_1 is visited four times. The trace $l_4 \rightarrow l_3 \rightarrow l_1 \rightarrow l_2 \rightarrow l_4 \rightarrow (l_3)^\omega$ satisfies the mission requirement.	$\neg\,\mathcal{F}(\underbrace{l_1 \wedge \mathcal{X}(\mathcal{F}(l_1 \wedge \ldots \mathcal{X}(\mathcal{F}(l_1))))}_{\text{n/n+1}})$, where $l_1 \in L$
Lower Rest. Avoidance	A restriction on the minimum number of occurrences is desired.	A robot should enter location l_1 at least (more than) 3 times. The trace $l_4 \rightarrow l_3 \rightarrow l_2 \rightarrow l_2 \rightarrow l_4 \rightarrow (l_3)^\omega$ violates the mission requirement since location 1 is never entered. The trace $l_1 \rightarrow l_4 \rightarrow l_3 \rightarrow l_1 \rightarrow l_4 \rightarrow l_1 \rightarrow (l_3)^\omega$ satisfies the mission requirement.	$\mathcal{F}(\underbrace{l_1 \wedge \mathcal{X}(\mathcal{F}(l_1 \wedge \ldots \mathcal{X}(\mathcal{F}(l_1))))}_{\text{n/n+1}})$, where $l_1 \in L$
Exact Rest. Avoidance	The number of occurrences desired is an exact number.	A robot must enter location l_1 exactly 3 times. The trace $l_4 \rightarrow l_3 \rightarrow l_2 \rightarrow l_2 \rightarrow l_4 \rightarrow (l_3)^\omega$ violates the mission requirement. The trace $l_1 \rightarrow l_4 \rightarrow l_3 \rightarrow l_1 \rightarrow l_4 \rightarrow l_1 \rightarrow (l_3)^\omega$ satisfies the mission requirement since location l_1 is entered exactly 3 times.	$\underbrace{(\neg(l1))\,\mathcal{U}(l1 \wedge (\mathcal{X}((\neg l1)\,\mathcal{U}(l1 \ldots \wedge (\mathcal{X}((\neg l1)\,\mathcal{U}(l1}_{\text{n}}$ $\wedge(\mathcal{X}(\mathcal{G}(\neg l1))))))))))$, where $l_1 \in L$

Table 4
Trigger patterns.

	Description	Example	Formula
Inst. Reaction	The occurrence of a stimulus instantaneously triggers a counteraction.	When location l_2 is reached the action a must be executed. The trace $l_1 \rightarrow l_3 \rightarrow \{l_2, a\} \rightarrow \{l_2, a\} \rightarrow l_4 \rightarrow (l_3)^\omega$ satisfies the mission requirement since when location l_2 is entered condition a is performed. The trace $l_1 \rightarrow l_3 \rightarrow l_2 \rightarrow \{l_1, a\} \rightarrow l_4 \rightarrow (l_3)^\omega$ does not satisfy the mission requirement since when l_2 is reached a is not executed.	$\mathcal{G}(p_1 \rightarrow p_2)$, where $p_1 \in M$ and $p_2 \in PL \cup PA$
Delayed Reaction	The occurrence of a stimulus triggers a counteraction some time later	When c occurs the robot must start moving toward location l_1, and l_1 is subsequently finally reached. The trace $l_1 \rightarrow l_3 \rightarrow \{l_2, c\} \rightarrow l_1 \rightarrow l_4 \rightarrow (l_3)^\omega$ satisfies the mission requirement, since after c occurs the robot starts moving toward location l_1, and location l_1 is finally reached. The trace $l_1 \rightarrow l_1 \rightarrow \{l_2, c\} \rightarrow l_3 \rightarrow (l_3)^\omega$ does not satisfy the mission requirement since c occurs when the robot is in l_2, and l_1 is not finally reached.	$\mathcal{G}(p_1 \rightarrow \mathcal{F}(p_2))$, where $p_1 \in M$ and $p_2 \in PL \cup PA$
Prompt Reaction	The occurrence of a stimulus triggers a counteraction promptly, i.e. in the next time instant.	If c occurs l_1 is reached in the next time instant. The trace $l_1 \rightarrow l_3 \rightarrow \{l_2, c\} \rightarrow l_1 \rightarrow l_4 \rightarrow (l_3)^\omega$ satisfies the mission requirement, since after c occurs l_1 is reached within the next time instant. The trace $l_1 \rightarrow l_3 \rightarrow \{l_2, c\} \rightarrow l_4 \rightarrow l_1 \rightarrow (l_3)^\omega$ does not satisfy the mission requirement.	$\mathcal{G}(p_1 \rightarrow \mathcal{X}(p_2))$, where $p_1 \in M$ and $p_2 \in PL \cup PA$
Bound Reaction	A counteraction must be performed every time and only when a specific location is entered.	Action a_1 is bound though a delay to location l_1. The trace $l_1 \rightarrow l_3 \rightarrow \{l_2, c\} \rightarrow \{l_1, a_1\} \rightarrow l_4 \rightarrow \{l_1, a_1\} \rightarrow (l_3)^\omega$ satisfies the mission requirement. The trace $l_1 \rightarrow l_3 \rightarrow \{l_2, c\} \rightarrow \{l_1, a_1\} \rightarrow \{l_4, a_1\} \rightarrow \{l_1, a_1\} \rightarrow (l_3)^\omega$ does not satisfy the mission requirement since a_1 is executed in location l_4.	$\mathcal{G}(p_1 \leftrightarrow p_2)$, where $p_1 \in M$ and $p_2 \in PL \cup PA$
Bound Delay	A counteraction must be performed, in the next time instant, every time and only when a specific location is entered.	Action a_1 is bound to location l_1. The trace $l_1 \rightarrow l_3 \rightarrow \{l_2, c\} \rightarrow \{l_1\} \rightarrow \{l_4, 1_1\} \rightarrow \{l_1\} \rightarrow \{l_4, a_1\} \rightarrow (l_3)^\omega$ satisfies the mission requirement. The trace $l_1 \rightarrow l_3 \rightarrow \{l_2, c\} \rightarrow \{l_1\} \rightarrow \{l_4, 1_1\} \rightarrow \{l_1, a_1\} \rightarrow \{l_4\} \rightarrow (l_3)^\omega$ does not satisfy the mission requirement.	$\mathcal{G}(p_1 \leftrightarrow \mathcal{X}(p_2))$, where $p_1 \in M$ and $p_2 \in PL \cup PA$
Wait	Inaction is desired till a stimulus occurs.	The robot remains in location l_1 until condition c is satisfied. The trace $l_1 \rightarrow l_3 \rightarrow \{l_2, c\} \rightarrow l_1 \rightarrow l_4 \rightarrow (l_3)^\omega$ violates the mission requirement since the robot left l_1 before condition c is satisfied. The trace $l_1 \rightarrow \{l_1, c\} \rightarrow l_2 \rightarrow l_1 \rightarrow l_4 \rightarrow (l_3)^\omega$ satisfies the mission requirement.	$(l_1)\,\mathcal{U}(p)$, where $l_1 \in L$ and $p \in PE \cup PA$

Name: Strict Ordered Patrolling

Intent: A robot must patrol a set of locations following a strict sequence ordering. Such locations can be, e.g., areas in a building to be surveyed.

Template: The following formula encodes the mission in LTL for n locations and a robot r (% is the modulo arithmetic operator):

$$\bigwedge_{i=1}^{n} \mathcal{G}(\mathcal{F}(l_1 \wedge \mathcal{F}(l_2 \wedge ... \mathcal{F}(l_n)))) \bigwedge_{i=1}^{n-1} ((\neg l_{i+1}) \ \mathcal{U} \ l_i) \bigwedge_{i=1}^{n} \mathcal{G}(l_{(i+1)\%n} \rightarrow \mathcal{X}((\neg l_{(i+1)\%n}) \ \mathcal{U} \ l_i))$$

Example with two locations.

$$\mathcal{G}(\mathcal{F}(l_1 \wedge \mathcal{F}(l_2))) \wedge ((\neg l_2) \ \mathcal{U} \ l_1) \wedge \mathcal{G}(l_2 \rightarrow \mathcal{X}((\neg l_2) \ \mathcal{U} \ l_1)) \wedge \mathcal{G}(l_1 \rightarrow \mathcal{X}((\neg l_1) \ \mathcal{U} \ l_2))$$

where l_1 and l_2 are expressions that indicate that a robot r is in locations l_1 and l_2, respectively.

Variations: A developer may want to allow traces in which sequences of *consecutive* l_1 (l_2) are allowed, that is strict ordering is applied on sequences of non consecutive l_1 (l_2). In this case, traces in the form $l_1 \rightarrow (\rightarrow l_1 \rightarrow l_1 \rightarrow l_3 \rightarrow l_2)^\omega$ are admitted, while traces in the form $l_1 \rightarrow (\rightarrow l_1 \rightarrow l_3 \rightarrow l_1 \rightarrow l_2)^\omega$ are not admitted. This variation can be encoded using the following specification:

$$\mathcal{G}(\mathcal{F}(l_1 \wedge \mathcal{F}(l_2))) \wedge ((\neg l_2) \ \mathcal{U} \ l_1) \wedge \mathcal{G}((l_2 \wedge \mathcal{X}(\neg l_2)) \rightarrow \mathcal{X}((\neg l_2) \ \mathcal{U} \ l_1)) \wedge \mathcal{G}((l_1 \wedge \mathcal{X}(\neg l_1)) \rightarrow \mathcal{X}((\neg l_1) \ \mathcal{U} \ l_2))$$

This specification allows for sequences of consecutive l_1 (l_2) since the left side of the implication $l_1 \wedge \mathcal{X}(\neg l_1)$ ($l_2 \wedge \mathcal{X}(\neg l_2)$) is only triggered when l_1 (l_2) is exited.

Examples and Known Uses: A common usage example of the Strict Ordered Patrolling pattern is a scenario where a robot is performing surveillance in a building during night hours. Strict Sequence Patrolling and Avoidance often go together. Avoidance patterns are used to force robots to avoid obstacles as they guard a location. Triggers can also be used in combination with the Strict Sequence Patrolling pattern to specify conditions upon which Patrolling should start or stop.

Relationships: The Strict Ordered Patrolling pattern is a specialisation of the Ordered Patrolling pattern, forcing the strict ordering.

Occurrences: Smith et. al. [74] proposed a mission specification forcing a robot to not visit a location twice in a row before a target location is reached.

Figure 2. The pattern Strict Ordered Patrolling. The catalog in full can be found in the online appendix [66].

a set of environment propositions such that $s_i \in PE$ is true if and only if condition c_i holds. Given a finite set of actions $A = \{a_1, a_2, \ldots, a_m\}$ that the robots can perform, we indicate as $PA = \{r_x \ exec \ a_y \mid r_x \in R \text{ and } a_y \in A\}$ a set of action propositions such that $r_x \ exec \ a_y$ is true if and only if action a_y is performed by robot r_x. We define the set of propositions M of a robotic application as $PL \cup PE \cup PA$. Let $M_x, M_y, M_z \subseteq M$, a trace is an infinite sequence $M_x \rightarrow M_y \rightarrow M_z \ldots$ indicating that M_z holds after M_y, and M_y holds after M_x. For example, $\{r_1 \ in \ l_1\} \rightarrow \{r_1 \ in \ l_2, c_1\} \rightarrow \{c_2, r_2 \ exec \ a_1\} \ldots$ is a trace where the element in position 1 of the trace indicates that the robot r_1 is in location l_1, then the element in position 2 indicates that the robot r_1 is in location l_2 and condition c_1 holds (indicating, for example, that an obstacle is detected), and then the element in position 3 indicates that condition c_2 holds and robot r_2 is executing action a_1. In the following, with a slight abuse of notation, when a set is a singleton we will omit brackets. We use the notation $(M_x \rightarrow \ldots \rightarrow M_y)^\omega$, where $M_x, \ldots, M_y \subseteq M$, to indicate a sequence $M_x \rightarrow \ldots \rightarrow M_y$ that occurs infinitely. We use the notation $l_\#$ to indicate any location, e.g., $r_1 \ in \ l_1 \rightarrow r_1 \ in \ l_\# \rightarrow r_1 \ in \ l_2$ indicates that a robot r_1 visits location l_1, afterwards any location, and then location l_2. We use the notation $l_{\#\setminus K}$, where $K \subset M$, to indicate any possible location not in K, e.g., $r_1 \ in \ l_1 \rightarrow r_1 \ in \ l_{\#\setminus\{l_3\}} \rightarrow r_1 \ in \ l_2$ indicates that r_1 visits l_1, then any location except l_3 is visited, and finally l_2.

Patterns. Patterns are organized in three main groups – core movement (Table 2), triggers (Table 3), and avoidance (Table 4), explained in the following. For simplicity, in Tables 2 and 3, we assume that a single robot is considered during the mission specification and we use the notation l_x as shortcut for $r_1 \ in \ l_x$. The presented examples assume that the environment is made of four locations, namely l_1, l_2, l_3, and l_4.

- *Core movement patterns*. How robots should move within an environment can be divided in two major categories representing locations' coverage and locations' surveillance. Coverage patterns require a robot to reach a set of locations of the environment. Surveillance patterns require a robot to *keep* reaching a set of locations of the environment.
- *Avoidance patterns.* Robot movements may be constrained in order to avoid occurrence of some behavior (Table 3). Avoidance may reflect a condition, possibly over the occurrence of some event. *Conditional avoidance* generally holds globally (i.e., for the entire behavior) and applies when avoidance of locations or obstacles is sought that depends on some condition. For example, a cleaning robot may avoid visiting locations that have been already cleaned. In the *restricted avoidance* case, avoidance does not hold globally but accounts for a number of occurrences of an avoidance case. Depending on the number of occurrences being a maximum, minimum or exact number, *upper*, *exact* or *lower* restricted avoidance is yielded. For example, a cleaning robot may avoid cleaning a room more than three times.
- *Trigger patterns.* Trigger patterns express a robot reactive behaviour based on stimuli, or robot's inaction until a stimulus occurs as described in Table 4.

As an example, the definition of the *Strict Ordered Patrolling* mission specification pattern is presented in Figure 2. The patterns in detail are available in our online appendix [66]. Note that the logic formulation is generic on purpose to allow specifications where a robot can be simultaneously in an area and in one of its sub-areas. An additional logical constraint can be added to disallow these behaviors. For example, the constraint $\mathcal{G}(\neg((r_1 \ in \ l_1) \wedge (r_2 \ in \ l_2)))$ disallows the robot r_1 to be simultaneously in locations l_1 and l_2. Defining such constraints is an orthogonal concern.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2945329, IEEE Transactions on Software Engineering

5 Specification Pattern Tool Support

We now present PsALM [69][4], a tool that supports developers in designing mission requirements through the proposed patterns and the automatic generation of mission specifications. PsALM allows creating complex mission requirements by composing patterns with simple operators and transforming mission requirements (i.e., composed patterns) into mission specifications in LTL or CTL.

Figure 3 illustrates the components of PsALM. Its GUI (1) allows defining robotic missions requirements through a structured English grammar, which uses patterns as basic building blocks and AND and OR logic operators to compose these patterns. The structured English grammar is provided in our online appendix [66]. The `SE2PT` component extracts from a mission requirement the set of patterns that are composed through the AND and OR operators (2). The `PT2LTL` (3) and `PT2CTL` (4) components automatically generate LTL and CTL specifications from these patterns.

The produced LTL specifications can be used in different ways; three possible usages are presented in Figure 3. The LTL formulae are (i) fed into an existing planner and used to generate plans that satisfy the mission specification (5); (ii) converted into Deterministic Büchi automata used as input to the widely used Spectra [63] robotic application modeling tool (6); or (iii) converted into the NuSMV [64] input language to be used as input for model checking (7). The plans produced using the planner are (i) used as inputs by Simbad [65] (10), an autonomous robot simulation package for education and research; and (ii) performed by actual real robots (9), as also illustrated in the following sections. The produced CTL specifications are also converted to the NuSMV [64] input language to be used as input for model checking (7).

To use our pattern-based mission specification and the PsALM prototype tool in practice (as exemplified in Figure 4), a robotics engineer follows four distinct steps:

1) The pattern catalog is consulted and behavior intents relevant to the mission at hand are selected [66]. This step is essential to establish common vocabulary, utilize the unambiguous patterns notation, and provide a precise description of what robots should do in terms of movements and actions during run-time.
2) The mission is defined using an appropriate GUI that allows using patterns as basic building blocks and composing them through a structured English grammar (Figure 4a).
3) Automatically generated CTL or LTL specifications are customized if necessary.
4) Analysis, planning or simulation facilities are invoked through interfacing with NuSMV [64], Spectra [63], Simbad [65] (Figure 4b), or sent to robots for execution (Figure 4c) through LTL planning.

A video showing how PsALM can be used in practice is also made available on a dedicated website [75].

6 Correctness of the Patterns

We now describe the procedure we used to validate the correctness of our patterns. This procedure can be considered a contribution per se since it can be used in validating future patterns be proposed by the research community. The procedure has two phases: (i) testing the compliance between the English formulation and the logical specification of the patterns (Section 6.1) and (ii) testing for errors in the logical specification (Section 6.2).

4. The tool is available at github.com/claudiomenghi/PsAlM

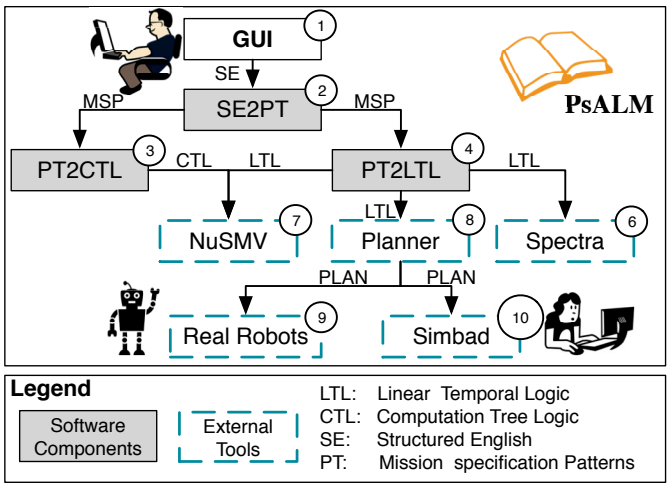


Figure 3. Main components of the PsALM specification tool [69].

6.1 Compliance Testing

We tested the compliance between the English formulation and the target mission specifications (LTL and CTL) as follows. We considered the English formulation and defined an alternative set of specifications into a language that is different from (but comparable to) the one used to express the target mission specifications. We then compared the specifications. If the alternative and the target mission specifications are not equivalent, an error must be present either in the alternative or in the target specification. If the two formulations are equivalent there is no guarantee that the specification is correct, as it might be the case that both the specifications do not represent the intended mission requirement. A further check that we performed to avoid this case is explained in Section 6.2.

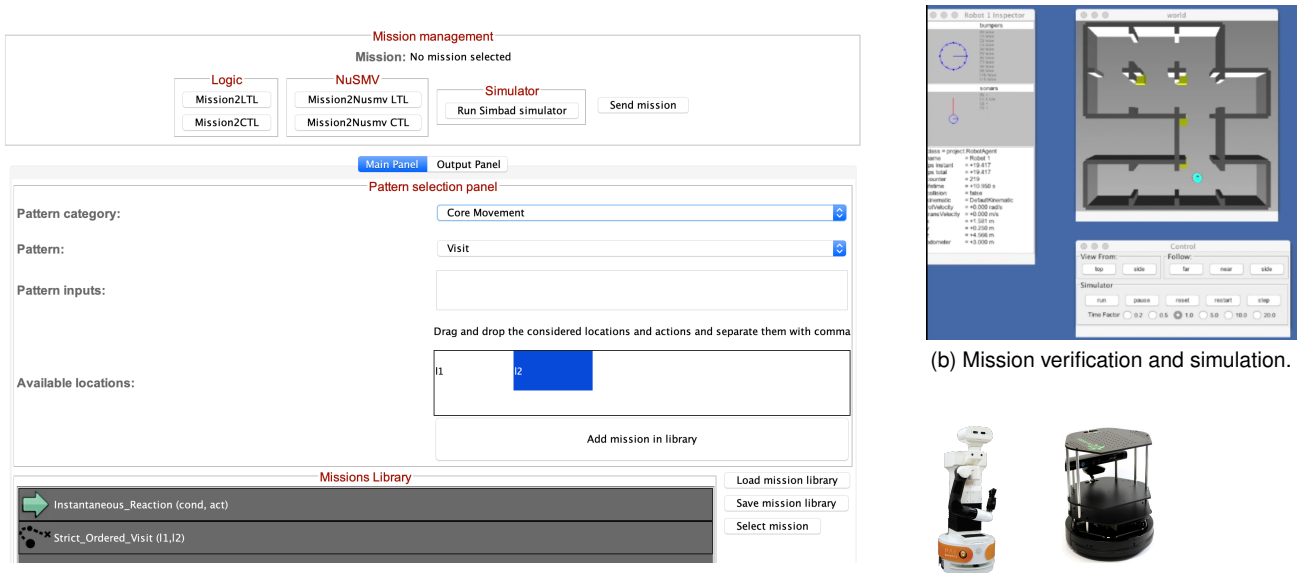
For LTL specifications we proceeded as follows:

- (Step.A1) We considered a pattern's English description and wrote a ω-regular expression [5], which encodes all (and only) the behaviors of the system that are admitted by the specification;
- (Step.A2) We converted the LTL specification into a Büchi automaton (BA), which was subsequently converted into a ω-regular expression;
- (Step.A3) We checked the equivalence among the ω-regular expressions.

We performed those steps using SPOT [76]. The testing activity did not reveal any non-compliance between the English formulation and the logical specification of the patterns, i.e., all the ω-regular expressions were equivalent.

While a similar procedure can also be applied in the case of the CTL specifications, e.g., by considering a restriction of First Order Logic, in our case this check was not necessary since the CTL mission specifications correspond with the LTL specifications where all the temporal operators are scoped

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2945329, IEEE Transactions on Software Engineering



(a) Pattern-based mission specification.

(b) Mission verification and simulation.

(c) Execution on robots.

Figure 4. Specification of dependable robotic missions with PsALM [69].

with the branch quantifier "for all" ($\forall$). Since, given a CTL formula, if there exists an equivalent LTL formula, it can be obtained by dropping all the branch quantifiers [5], the CTL formulation is compliant with the LTL specification.

The steps previously described can be used to check compliance among English formulations and mission specifications in future pattern catalogs proposed by the research community.

6.2 Checking for Errors through Testing

We performed a further experiment to check whether for some patterns we had formulated two equivalent specifications, but both of them do not represent the mission requirement. This has been performed through testing. We randomly created scenarios and assigned to a robot a mission obtained by combining a set of patterns. We synthesized a plan for the robot and observed the robot performing the plans. We checked whether the robot was executing the plans according to the mission requirement described by the patterns. This has been performed through the following steps:

- (STEP.B1) Scenarios were randomly defined over a map. Testing by exploiting a set of randomly generated models is a widespread technique to evaluate artifacts in both software [77], [78], [79] and robotic engineering communities [80], [81], [82], [83], [84].
- (STEP.B2) To test the LTL specifications, randomly generated mission requirements were created by combining a core movement, a trigger, and an avoidance pattern, and by ensuring that each pattern in our catalog is used for at least one requirement. Then, (a) the corresponding LTL specification was negated; (b) the specification and the model of the scenario was encoded in a model checker; and (c) the model checker was used to check whether the models contained a path that satisfied the mission specification (i.e., violates its negation). If a path was present, a simulator was used to execute the produced plan. Subsequently, we checked whether the plan execution was correct with respect to the intent of the mission specification patterns: when we expected a plan to not be present in the given model, the model checker was not able to compute it, and, when a plan was expected to be present, it was computed by the model checker. Finally, we simulated the generated plans by using the Simbad robot simulator and checked whether the robots achieve their mission requirements.
- (STEP.B3) To build confidence that errors do not exist in the CTL specification with respect to the LTL ones, an extra testing activity was performed. The comparison among the LTL versus the CTL expressions was performed since given a CTL formula ϕ, and an LTL formula ψ obtained by eliminating all the path quantifiers from ϕ, either the two formula are equivalent (i.e., $\phi \equiv \psi$) or it does not exist any LTL formula that is equivalent to ϕ [5]. Our testing activity aims at determining whether CTL and LTL formulation are not equivalent. The same models considered in step (STEP.B2) were used and LTL and CTL specifications were generated for the considered mission requirements. The LTL and CTL specifications, as well as the model of the scenario, were considered and a model checker was used to check whether the verification of the specifications returned the same results, i.e., if the LTL specification was satisfied, also the CTL specification was satisfied (and viceversa).

While the proposed process can be reused in future works, the random generation procedure for creating test cases is problem-specific and must be defined on a per-domain basis. In the following, we describe the results of executing the process described above.

- (STEP.B1) We generated 12 scenarios representing floor plan structures containing 16 locations, where a robot is deployed. We synthesized plans to be executed by robots from combinations of mission specification patterns

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2945329, IEEE Transactions on Software Engineering

Table 5
Results of the verification procedure for checking presence of errors. For step B2 columns contain the number of times a plan is found ($\top$) and not found ($\bot$). For step B3 and B3′ columns contain the number of times the mission requirement is satisfied ($\top$) and violated ($\bot$).

	B2		B3		B3′	
Mission Requirement	$\top$	$\bot$	$\top$	$\bot$	$\top$	$\bot$
OrdPatrol,UpperRestAvoid,Wait	2	10	1	11	1	11
FairVisit,ExactRestAvoid*,DelReact	5	7	0	12	4	8
StrOrdVisit,GlobalAvoid,InstReact	3	9	1	11	1	11
SeqVisit,FutAvoid,BindDel*	1	11	0	12	2	10
OrdVisit,PastAvoid,InstReact	3	9	1	11	1	11
Visit,LowRestAvoid,BindReact	3	9	1	11	1	11
StrictOrdPatrol,FutAvoid,Wait	1	11	1	11	1	11
Patrol,LowRestAvoid,InstReact	3	9	1	11	1	11
FairPatrol,ExactRestAvoid*,DelReact	3	9	0	12	4	8
SeqPatrol,UpperRestAvoid,FastReact*	1	11	0	12	2	10

and checked whether the plan execution is correct with respect to patterns' intent. The plan has been generated by allocating 12 traversable locations and 4 locations that cannot be crossed, on a 4×4 matrix. The identifiers $l_0, l_1, \ldots, l_{11}$ are randomly assigned to the traversable locations. In 6 of the 12 scenarios, the robot can move among adjacent cells that are traversable, while it cannot move within not crossable locations. In the other 6 scenarios, the robot can cross adjacent cells from a cell with coordinates $[i, j]$ to a traversable one of $[i, j + 1 \bmod j]$ (and similarly to the j axis). Conditions and actions specify whether a box is present in a location ($cond$ in the following), and the capability of the robot in changing its color (act in the following). We randomly selected 4 traversable locations in which $cond$ is true and 4 locations in which act can be performed, that is the robot can wrap around to the other side of the grid.

- (STEP.B2) In total we generated 10 mission requirements (Table 5). Core movement patterns are parametrized with locations l_1, l_2. The patterns upper, exact, and lower restricted avoidance are parametrized by forcing the robot to visit location l_3, at most, exactly, and at least 2 times, respectively. The pattern global avoidance forces the robot to not visit l_3, while the future and past avoidance force the robot to not visit l_3 after and before condition $cond$ is satisfied, i.e., a room that contains a box is visited. The wait pattern forces the robot to wait in location l_4 if a box is not present. The other trigger patterns are parametrized with the action act that must be executed by the robot in relation with the occurrence of condition $cond$.
  To test the LTL specifications, we followed the indications provided in (STEP.B2) and made use of the NuSMV [64] model checker and of Simbad [65] to simulate the robot executing the plan.
  We did not find any error in the proposed LTL specifications. The column labeled ExpV1 in Table 5 marked with $\top$ (respectively $\bot$) contains the number of cases in which a plan was (was not) present.
- (STEP.B3) We used the NuSMV [64] model checker: Table 5's column B3 contains the number of cases in which the mission requirement was satisfied ($\top$) and not satisfied ($\bot$). Mission requirements were generally not satisfied, since to be satisfied they have to hold on all the paths of the models. The testing activity did not reveal any error as NuSMV always returned the same results for LTL and CTL specifications. Since in several cases the mission requirement was always not satisfied, and we also wanted to test cases in which the mission requirement was satisfied, we relaxed the mission requirements, by removing the patterns marked with the * symbol in Table 5. Table 5's column B3′ shows that by relaxing the mission requirements there were cases in which the mission requirements were actually satisfied. Also in these cases the testing activity did not reveal any error in the LTL and CTL specifications as NuSMV always returned the same results.

Table 6
Coverage of the proposed mission requirements. Lines contain the total number of mission requirements (MR), the number of not expressible (NE) and ambiguous (A) mission requirements and the number of requirements that lead to a consensus (C) and no consensus (NC). Columns labeled with Spectra and MP contain the number of requirements extracted from LTLMoP.

	Spectra												
	1	2	3	4	5	6	7	8	9	10	11	MP	**Total**
MR	29	2	22	5	1	159	4	32	47	53	74	8	436
NE	3	0	0	0	0	47	0	0	7	1	8	0	66
A	3	0	2	1	0	35	0	10	12	32	7	0	102
C	13	0	11	2	1	29	4	8	11	8	20	5	112
NC	10	2	9	2	0	48	0	14	17	12	39	3	156

7 EVALUATION

We evaluated how *effective* our pattern catalog is in capturing mission requirements and producing mission specifications. We investigated three questions:

- **RQ1**: To what extent are real-world, natural-language mission requirements expressible using our pattern catalog? (Section 7.1)
- **RQ2**: To what extent are real-world mission specifications expressible using our pattern catalog? (Section 7.2)
- **RQ3**: Does the pattern catalogue support the formulation of mission requirements and specifications in a set of real-world scenarios defined in collaboration with our industrial partners? (Section 7.3)

To cover both aspects and to answer the question above, we performed three experiments.

7.1 Formulation of Mission Requirements (RQ1)

We collected mission requirements in natural language from available requirements produced from Spectra [63] and LTLMoP [31], [36]. Spectra is a tool that supports the design of robotic applications. LTLMoP is a software package assisting in the development, implementation, and testing of robot controllers. We checked how the pattern catalog may have supported developers in the definition of the mission requirements.

In the case of Spectra, we extracted 428 mission requirements from Spectra files of 11 robotic applications. Note that these mission requirements are realistic, since they were defined for and executed with real robots. Videos are available online [85]. The number of mission requirements

Table 7
Number of occurrences of each pattern in the formulation of mission requirements.

Pattern	Occ	Pattern	Occ	Pattern	Occ	Pattern	Occ
Visit	25	SeqVisit	1	OrdVisit	1	InstReact	127
PastAvoid	60	DelReact	50	Wait	3	FutAvoid	48
StrictOrdPat	1	GlobAvoid	25	ExactRest	1		

(MR) per robotic application is reported in Table 6. In the case of LTLMoP, 8 requirements were extracted from the corresponding research papers [31], [36] (Table 6 MP column).

Each mission requirement was independently analyzed by two of the authors. The authors checked whether it is possible to express the mission requirement using the mission specification patterns. If one the authors stated that the requirement is not expressible, it is marked as not expressible (NE). The number of not expressible mission requirements is presented in Table 6 in the row NE. If at least one of the authors found the mission requirement is ambiguous the author marked it with the flag A. Otherwise, the mission requirement is labeled with the mission specification patterns needed to express the mission requirement. Then, the mission specification patterns used to express the mission requirement are considered. If the authors used the same mission specification patterns to express the mission requirement, a consensus is reached. The number of mission requirements that leads to consensus (respectively no consensus) is indicated in the row labeled C (respectively NC). The number of occurrences of each pattern is indicated in Table 7.

The results show that most of the mission requirements (370 of 436, i.e., $\approx 84\%$) were expressible using the pattern catalog, which we consider a reasonable coverage for a pattern catalog. The 66 mission requirements that are not covered suggested the introduction of new patterns identified in Figure 1 with a dashed border. It also shows that the pattern catalog is effective in real scenarios.

102 mission requirements were ambiguous, meaning that different interpretations can be given to the proposed mission requirement. For these requirements, alternative combinations of patterns have been proposed by the authors to express the mission requirement. Each of these alternatives represents a possible way of expressing it in a non-ambiguous manner. For 156 mission requirements, while the authors judged that the requirement was not ambiguous, different pattern combinations were proposed. The combinations of patterns encode possible ways of expressing the mission requirement in a non-ambiguous manner.

7.2 Formulation of Mission Specifications (RQ2)

We analyzed the mission specifications contained in the Spectra examples collected for answering RQ1 . We collected 1216 distinct LTL mission specifications and we analyzed each of these specifications.[5] We verified whether it is possible to obtain the mission specifications starting from the proposed patterns, by performing the following steps.

- (STEP.1) For each property we automatically checked whether it was an instance of a mission specification pattern or a simple combination of mission specification patterns through a script that matches pattern occurrences within mission specifications. Results are shown in Table 8. Among 1216 mission specifications 424 were obtainable from the proposed patterns.
- (STEP.2) We considered the mission specifications that did not match any of the proposed patterns. 127 of these mission specifications are simple statements on the initial state of the system (no temporal operator is used) and, thus, did not match any of the proposed patterns. 442 mission specifications concern properties that refer to variation of the trigger patterns, that we have added to the pattern catalogue. 224 mission specifications still did not match any of the proposed patterns. After analysis, 155 among them were expressed using past temporal operators[6], which are not used in the mission specifications proposed in this work. In Step 3 we checked whether these specifications might be reformulated without the past operators. 69 of these mission specifications, while they could be rewritten using the proposed patterns, they are written as complex LTL formulae and thus they do not match any of our patterns or combination of them. As these specifications did not "directly" match any of our patterns (or combinations of them), we did not consider them as covered.
- (STEP.3) We considered the 155 mission specifications expressed using past temporal operators and we designed mission specifications for them. We found that 129 of the proposed LTL formulae match one of the proposed patterns, while 26 are complex LTL formulae that did not match any combination of the patterns. Thus, the final coverage of the proposed pattern catalog is 92%.

We then analyzed 13 mission specifications expressed in the form of LTL properties and 22 PCTL properties considered by Ruchkin et al. [67]. The PCTL properties wrap an LTL formula into a CTL formula which is scoped with the probabilistic operator ($\mathcal{P}$). To check for the pattern presence we transform PCTL formulae into CTL by replacing the probabilistic operator ($\mathcal{P}$) with the universal quantifier ($\forall$) and by scoping temporal operators of the LTL formulae with the universal quantifier. Removing the probabilistic operator in front of the PCTL formula generated an approximation of the original formula. Scoping the LTL formulae with the universal quantifier was allowed as the analyzed formulae belong to the intersection among LTL and CTL [7]. Given the small number of LTL and CTL mission specifications we manually checked the presence of patterns in the formulae (Step 1 in Table 8). The results reported in the Table 8 (Columns labeled with [67]) show that 2 among these specifications did not match any pattern. The results show that the pattern system was able to generate almost all mission specifications (1154 of 1251, i.e., $\approx 92\%$).

5. This number differs from the one of RQ1, since some specifications were not related with a mission requirement in natural language.

6. The LTL formulae can be formulated using the future or past operators. We used future operators as commonly done and dictated by reuse of robotic components (e.g., planners), needed for the real robots of BOSCH/PAL Robotics. Formulations using past operators can be obtained using LTL formulae equivalence.

7. This has been checked using a simple state-of-the-art procedure [5].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2945329, IEEE Transactions on Software Engineering

Table 8
Pattern occurrence in the formulation of mission specifications for the considered mission specifications.

		LTL		CTL
	Pattern	Spectra	[67]	[67]
Step 1	Instantaneous reaction	318	0	0
	Visit	52	0	0
	Patrolling	0	1	0
	Strict Ordered Visit	0	9	18
	Wait	0	1	2
	Avoidance/Invariant	21	0	0
	Visit and Instantaneous reaction	18	0	0
	Strict Ordered Visit and Global Avoidance	0	0	1
	Reaction chain (chain of instantaneous reactions)	15	0	0
	Non matching	792	1	1
Step 2	Init	127	-	-
	Fast reaction	379	-	-
	Bound reaction	36	-	-
	Bound delay	27	-	-
	Non matching for past	155	-	-
	Actual non matching	69	-	-
Step 3	Fast reaction	103	-	-
	Bound delay	26	-	-
	Actual non matching	26	-	-

7.3 Usage in Real-World Robotic Scenarios (RQ3)

We checked how the pattern catalog supports the formulation of mission requirements and the generation of mission specifications in real-world robotic scenarios. To this end, we defined five scenarios (Table 9) in collaboration with our industrial partners BOSCH and PAL Robotics.

The pattern catalog supported the formulation of mission requirements using the patterns listed in Table 9 for the different scenarios. In all the scenarios, PsALM allowed the automatic creation of LTL mission specifications from the mission requirements. The mission specifications were then executed by the robots by relying on existing planners (see Figure 3). In our case, we used an existing planner developed by Meng and Dimarogonas [3] as this was a requirement of the Co4Robots project [4] which was funding part of this work. However, the proposed patters are agnostic with respect to the LTL planner that is used to synthesize plans. Videos of the robots performing the described missions are available in our dedicated website [66]. The pattern catalog effectively supported the creation of mission requirements and specifications in realistic, industry-sourced scenarios.

8 Reflection

The pattern catalog is effective in supporting developers in defining mission requirements and in generating mission specifications. Sections 7.1 and 7.3 show that the pattern catalog effectively supports the definition of mission requirements and that helps in reducing ambiguities in available mission requirements. Sections 7.2 and 7.3 show that the pattern catalog effectively supports the generation of mission specifications. Section 7.3 shows how the pattern catalog can be used to generate precise, unambiguous, and formal mission specifications in industry-sourced scenarios.

Methodology. The number of mission requirements analyzed is in line with other approaches in the field [44], [49], [50], [51], [52]. These requirements usually come from exemplar scenarios used to provide evaluation about effectiveness of research-intensive works. As such, we believe that the scope of the pattern system is quite wide. Our study is certainly not exhaustive, as (i) formal specification in robotic application spreads, and (ii) the types of mission specifications change over time. As shown in the evaluation, patterns will grow over time as specifications that do not belong to the catalog emerge.

Patterns. While the presented patterns are mainly conceived to address needs of robotic mission specification, they are more generic and can be applied when the need is to specify some "ordering" among events or action execution. Rather than requiring robots reaching a set of locations, coverage and surveillance patterns may also include propositions that refer to generic events. In this sense, the proposed patterns can be considered as an extension of the property specification patterns [44], [89] that explicitly address different ordering among the occurrence of a set of events. While in this paper we proposed a direct encoding in LTL and CTL, they may also be expressed in terms of standard property specification patterns. The instantaneous reaction pattern may be obtained from the response pattern scoped with the global operator. The precedence chain and the response chains patterns [44], [89] (that illustrate the 2 cause-1 effect and 1 cause-2 effects chain), can be composed with the precedence and response patterns [44], [89] to specify different ordering among a set of events.

Evaluation. The Spectra tool only supports specifications captured by the GR(1) LTL fragment used to describe three types of guarantees: initial, safety, and liveness. Initial guarantees constraint the initial states of the environment. Safety guarantees start with the temporal operator $\mathcal{G}$ and constraint the current and next state. Liveness guarantees start with the temporal operators $\mathcal{G}\,\mathcal{F}$ and may not include the $\mathcal{X}$ operator. These constraints justify the prevalence of patterns presented in Tables 7, 8, and 5. While the proposed patterns can be expressed using deterministic Büchi automata, which can be translated into GR(1) formulae [29], a manual encoding of the proposed patterns in GR(1) is complex and error prone. This is confirmed by the fact that analysis on the standard property specification patterns that can be expressed in GR(1), and an automatic procedure to map these patterns on formulae that are in the GR(1) fragment has been recently conducted [29]. The BA generated from the LTL formulations associated with the patterns proposed in this work are deterministic, and thus can be used as assumptions or guarantees of the GR(1) formula [1], [2]. Thus, the automatic procedure presented in by Maoz et al. [29] can be integrated in PsALM to generate Spectra formulae.

9 Related work

Temporal logic specification patterns are a well-known solution to support developers in requirement specification [44], [49], [50], [51], [90], [91], [92]. Foundational work by Dwyer et al. [44] has been extended by Konrad and Cheng [50] to express real-time properties, by Grunske [49] to address real time and probabilistic properties, and Autili et al. [51] that proposed a comprehensive catalog. Further use, refinement, and extensions were developed, for events [90] or automata-based specification [91], [92]. Property specification patterns use in specific domains has been investigated in the literature,

Table 9
Mission specification patterns for checking the usage in real-world robotic scenarios. Labels SC1, SC2, . . . , SC5 identify the considered scenarios. The column patterns contains the patterns used to formulate the mission requirement.

Scenario	Informal Description	Patterns
SC1	A robot is deployed within a supermarket and reports about the absence of sold items within a set of locations (i.e. $l1$, $l2$, $l3$, and $l4$). Furthermore, if in location $l4$ (where water supplies are present) a human is detected, it has to perform a collaborative grasping action and help the human in placing new water supplies.	Ordered Patrolling, Instantaneous Reaction
SC2	Three robots are deployed within an hospital environment: a mobile platform (Summit [86]), a manipulator (PA10 [87]) and a mobile manipulator (Tiago [88]), identified in the following as MP, M, and MM, respectively. The robot M is deployed in hospital storage; when items (e.g., towels) are needed by a nurse or doctors, M has to load them on the MP. MP should reach the location where the nurse is located. If the item is heavy (e.g., heavy medical equipment), MM should reach the location where the nurse is to help unloading the equipment. When MP and MM are not required for shipping items they are patrolling a set of locations to avoid unauthorized people entering restricted areas of the hospital (e.g., radiotherapy rooms).	Patrolling, Instantaneous Reaction, Ordered Visit, Wait
SC3	A robot is developed within a university building to deliver coffee to employees. The robot reaches the coffee machine, uses the coffee machine to prepare the coffee and delivers it to the employees.	Strict Ordered Visit, Instantaneous Reaction
SC4	A robot is deployed within a shop to check the presence of intruders during night time. It has to iteratively check for intruders and report on their presence	Patrolling, Instantaneous Reaction
SC5	A robot is deployed within a company to notify employees in presence of a fire alarm. If a fire is detected, the robot is send to different areas of the company to ask employees to leave the building.	Visit, Instantaneous Reaction

including service-based applications [52], safety [93] and security [94]. Patterns have also been considered in the robotic domain [95], [96], [97].

For example, Rothwell et al. [98], [99] proposed a specification pattern editor for vehicle mission planning. However, the work is not focusing on proposing and collecting mission specification patterns for the robotic domain. Differently, we proposed patterns, discuss their variations, and provided example traces that do and do not fulfill the patterns.

Domain-Specific Languages (DSLs) [21], [28], [100], [101], [102] have been proposed for various purposes including production and analysis of behavior descriptions, property verification, and planning. However, features incorporated within DSLs are usually arbitrarily chosen by relying on the domain-specific experience of robotic engineers. Instead, specification patterns presented in this paper are collected from missions encountered in the scientific literature, evaluated in industrial uses, and aim at supporting a wide range of robotic needs. We believe that the presented patterns consist of basic building blocks that can be reused within existing and new robotic DSLs. Moreover, support for developers on solving the mission specification problem is also provided in the literature by graphical tools that simplify the specification of LTL formulae [22], [23], [24]. Our work is complementary with those; graphical logic mission specifications can also be integrated within PsALM.

Reasoning on system behavior is the main driver electing temporal languages and logics as the most widely adopted specification formalisms. Other system concerns may dictate specification in different system domains. Since robot movement occurs in space, we identify this concern as additionally relevant. Spatial reasoning [103] has been traditionally considered in diverse domains such as safety properties [104], including query languages [105] and logics for the analysis of spatial data [106], [107], where the focus has been mainly in relations that exist between regions, lines, and points of a spatial model [108], [109]. In addition, spatial logics have also been studied in the context of process calculi [110], graph databases [111] or hybrid automata [112]. We identify investigation of complex spatial robotic behaviours as an important avenue for future work, especially regarding specification of spatio-temporal behavior [113]. Sun et al. [104] propose a combination of metric and spatial logics for verification of safety properties in cyber-physical systems. Shao et al. [112] present a composition of a topological and temporal logic over hybrid automata. A combination of CTL and SLCS is developed [114] to study bike sharing systems, while runtime verification of spatio-temporal behaviors of complex systems is studied by Nenzi et al. [115]. Signal Temporal Logic (STL) [6] and Metric Temporal Logic (MTL) [116] is also used in the literature to express missions that contian explicit timing constraints [7], [8].

10 Conclusions

In this paper, we proposed a pattern catalog for mission specification of mobile robots. We identified patterns by analyzing mission requirements that have been systematically collected from scientific publications. We further presented PsALM, a tool that uses the proposed patterns to concretely support robotic developers in designing complex missions. We evaluated the support provided by the pattern catalog in the definition of real-world missions, as well as the correctness of the mission specifications.

Currently, our patterns can be combined by using simple AND and OR logical operators. By using these simple operators our coverage was adequate for the great majority of the mission requirements and specification collected from literature. However, specification can be extended to support patterns' nesting. Analyzing how patterns can be nested within each other may increment coverage, something which is the subject of future investigation. Future extensions of our mission specification pattern catalog will also consider time, space, and probability and therefore the mapping to other logics, such as STL [6], MTL [116], TCTL [117], and CSL [118]. Moreover, we will consider extensions of the patterns to collaborative or team aspects of robotic missions. Although the formal languages adopted so far constitute the majority

of mission specification, we will also investigate the use of spatial logics [103], [106], [107], [110], [119] to express more complex spatial robotic behaviors and conduct user studies.

ACKNOWLEDGEMENTS

This work has received funding from the European Research Council under the European Union's Horizon 2020 research and innovation programme (grant agreements No 731869 and No 694277).
We thank Domenico Bianculli for insightful comments. We are grateful for feedback provided by the anonymous TSE reviewers.

REFERENCES

[1] N. Piterman, A. Pnueli, and Y. Sa'ar, "Synthesis of reactive (1) designs," in *International Workshop on Verification, Model Checking, and Abstract Interpretation*. Springer, 2006, pp. 364–380.
[2] R. Bloem, B. Jobstmann, N. Piterman, A. Pnueli, and Y. Sa'ar, "Synthesis of reactive (1) designs," *Journal of Computer and System Sciences*, vol. 78, no. 3, pp. 911–938, 2012.
[3] M. Guo and D. V. Dimarogonas, "Multi-agent plan reconfiguration under local ltl specifications," *The International Journal of Robotics Research*, vol. 34, no. 2, pp. 218–235, 2015.
[4] http://www.co4robots.eu/, "Co4robots," 2019.
[5] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT press, 2008.
[6] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*. Springer, 2004, pp. 152–166.
[7] Y. V. Pant, H. Abbas, R. A. Quaye, and R. Mangharam, "Fly-by-logic: Control of multi-drone fleets with temporal logic objectives," in *International Conference on Cyber-Physical Systems*. IEEE, 2018.
[8] R. Mangharam, "Fly-by-logic: A tool for unmanned aircraft system fleet planning using temporal logic," in *NASA Formal Methods*. Springer, 2019, p. 355.
[9] IFR, "World Robotic Survey," https://ifr.org/ifr-press-releases/news/world-robotics-survey-service-robots-are-conquering-the-world-, 2016.
[10] D. Brugali, *Software engineering for experimental robotics*. Springer, 2007, vol. 30.
[11] E. A. Lee, "Cyber physical systems: Design challenges," in *Symposium on Object Oriented Real-Time Distributed Computing (ISORC)*. IEEE, 2008, pp. 363–369.
[12] J. Pérez, N. Ali, J. A. Carsı, I. Ramos, B. Álvarez, P. Sanchez, and J. A. Pastor, "Integrating aspects in software architectures: Prisma applied to robotic tele-operated systems," *Information and Software Technology*, vol. 50, no. 9-10, pp. 969–990, 2008.
[13] N. Gamez and L. Fuentes, "Architectural evolution of famiware using cardinality-based feature models," *Information and Software Technology*, vol. 55, no. 3, pp. 563–580, 2013.
[14] D. Bozhinoski, D. D. Ruscio, I. Malavolta, P. Pelliccione, and I. Crnkovic, "Safety for mobile robotic system: a systematic mapping study from a software engineering perspective," *Journal of Systems and Software (JSS), to appear*, 2019.
[15] D. Brugali and E. Prassler, "Software engineering for robotics," *IEEE Robotics Automation Magazine*, vol. 16, no. 1, pp. 9–15, March 2009.
[16] C. Lignos, V. Raman, C. Finucane, M. Marcus, and H. Kress-Gazit, "Provably correct reactive control from natural language," *Autonomous Robots*, vol. 38, no. 1, pp. 89–105, 2015.
[17] A. Nordmann, N. Hochgeschwender, and S. Wrede, "A survey on domain-specific languages in robotics," in *Simulation, Modeling, and Programming for Autonomous Robots*. Springer, 2014.
[18] M. Broy, "Declarative specification and declarative programming," in *Software Specification and Design*. IEEE, 1991.
[19] J. F. Kramer and M. Scheutz, "Development environments for autonomous mobile robots: A survey," *Autonomous Robots*, vol. 22, pp. 101–132, 2007.
[20] S. Maniatopoulos, M. Blair, C. Finucane, and H. Kress-Gazit, "Open-world mission specification for reactive robots," in *International Conference on Robotics and Automation (ICRA)*. IEEE, 2014.
[21] D. Bozhinoski, D. D. Ruscio, I. Malavolta, P. Pelliccione, and M. Tivoli, "Flyaq: Enabling non-expert users to specify and generate missions of autonomous multicopters," in *Automated Software Engineering (ASE)*. IEEE, 2015.
[22] I. Lee and O. Sokolsky, "A graphical property specification language," in *High-Assurance Systems Engineering Workshop*. IEEE, 1997.
[23] M. H. Smith, G. J. Holzmann, and K. Etessami, "Events and constraints: A graphical editor for capturing logic requirements of programs," in *International Symposium on Requirements Engineering*. IEEE, 2001.
[24] S. Srinivas, R. Kermani, K. Kim, Y. Kobayashi, and G. Fainekos, "A graphical language for ltl motion and mission planning," in *International Conference on Robotics and Biomimetics (ROBIO)*. IEEE, 2013.
[25] R. Arkin, "Missionlab v7. 0," 2006.
[26] T. Balch, "Teambots," 2004. [Online]. Available: www.teambots.org
[27] S. Maoz and Y. Sa'ar, "Aspectltl: an aspect language for ltl specifications," in *International conference on Aspect-oriented software development*. ACM, 2011.
[28] D. D. Ruscio, I. Malavolta, P. Pelliccione, and M. Tivoli, "Automatic generation of detailed flight plans from high-level mission descriptions," in *Model Driven Engineering Languages and Systems*, ser. MODELS. ACM, 2016.
[29] S. Maoz and J. O. Ringert, "GR(1) synthesis for LTL specification patterns," in *Foundations of Software Engineering (FSE)*. ACM, 2015.
[30] M. Guo and D. V. Dimarogonas, "Multi-agent plan reconfiguration under local LTL specifications," *The International Journal of Robotics Research*, 2015.
[31] C. Finucane, G. Jing, and H. Kress-Gazit, "LTLMoP: Experimenting with language, temporal logic and robot control," in *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2010, pp. 1988–1993.
[32] C. Menghi, S. Garcia, P. Pelliccione, and J. Tumova, "Multi-robot ltl planning under uncertainty," in *Formal Methods*, K. Havelund, J. Peleska, B. Roscoe, and E. de Vink, Eds. Cham: Springer International Publishing, 2018, pp. 399–417.
[33] X. C. Ding, M. Kloetzer, Y. Chen, and C. Belta, "Automatic deployment of robotic teams," *IEEE Robotics Automation Magazine*, vol. 18, no. 3, pp. 75–86, Sept 2011.
[34] Y. Endo, D. C. MacKenzie, and R. C. Arkin, "Usability evaluation of high-level user assistance for robot mission specification," *Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 34, no. 2, pp. 168–180, 2004.
[35] S. Maoz and J. O. Ringert, "On the software engineering challenges of applying reactive synthesis to robotics," in *Workshop on Robotics Software Engineering*, ser. RoSE '18. ACM, 2018.
[36] W. Wei, K. Kim, and G. Fainekos, "Extended LTLvis motion planning interface," in *International Conference on Systems, Man, and Cybernetics*. IEEE, 2016.
[37] C. Lignos, V. Raman, C. Finucane, M. Marcus, and H. Kress-Gazit, "Provably correct reactive control from natural language," vol. 38, no. 1, Jan 2015, pp. 89–105.
[38] V. Raman, C. Lignos, C. Finucane, K. C. Lee, M. Marcus, and H. Kress-Gazit, "Sorry dave, i'm afraid i can't do that: Explaining unachievable robot tasks using natural language," University of Pennsylvania Philadelphia United States, Tech. Rep., 2013.
[39] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Temporal-logic-based reactive mission and motion planning," *Transactions on robotics*, vol. 25, no. 6, pp. 1370–1381, 2009.
[40] C. Yoo, R. Fitch, and S. Sukkarieh, "Online task planning and control for fuel-constrained aerial robots in wind fields," *The International Journal of Robotics Research*, vol. 35, no. 5, pp. 438–453, 2016.
[41] U. S. Shah and D. C. Jinwala, "Resolving ambiguities in natural language software requirements: a comprehensive survey," *ACM SIGSOFT Software Engineering Notes*, 2015.
[42] N. Kiyavitskaya, N. Zeni, L. Mich, and D. M. Berry, "Requirements for tools for ambiguity identification and measurement in natural language requirements specifications," *Requirements engineering*, 2008.
[43] J. O. Ringert, B. Rumpe, and A. Wortmann, "A requirements modeling language for the component behavior of cyber physical robotics systems," *arXiv preprint arXiv:1409.0394*, 2014.

[44] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett, "Patterns in property specifications for finite-state verification," in *International Conference on Software Engineering (ICSE)*. IEEE, 1999.

[45] E. A. EMERSON, "{CHAPTER} 16 - temporal and modal logic," in *Formal Models and Semantics*, ser. Handbook of Theoretical Computer Science, J. V. LEEUWEN, Ed. Elsevier, 1990, pp. 995 – 1072.

[46] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Temporal-logic-based reactive mission and motion planning," *Transactions on Robotics*, vol. 25, no. 6, pp. 1370–1381, 2009.

[47] G. J. Holzmann, "The logic of bugs," in *Foundations of Software Engineering (FSE)*. ACM, 2002.

[48] M. Autili, P. Inverardi, and P. Pelliccione, "Graphical scenarios for specifying temporal properties: An automated approach," *Automated Software Engg.*, vol. 14, no. 3, 2007.

[49] L. Grunske, "Specification patterns for probabilistic quality properties," in *International Conference on Software Engineering (ICSE)*. IEEE, 2008.

[50] S. Konrad and B. H. Cheng, "Real-time specification patterns," in *International conference on Software engineering (ICSE)*. IEEE, 2005.

[51] M. Autili, L. Grunske, M. Lumpe, P. Pelliccione, and A. Tang, "Aligning qualitative, real-time, and probabilistic property specification patterns using a structured english grammar," *Transactions on Software Engineering*, vol. 41, no. 7, pp. 620–638, 2015.

[52] D. Bianculli, C. Ghezzi, C. Pautasso, and P. Senti, "Specification patterns from research to industry: a case study in service-based applications," in *International Conference on Software Engineering (ICSE)*. IEEE, 2012.

[53] R. A. Brooks *et al.*, "Intelligence without reason," *Artificial intelligence: critical concepts*, vol. 3, pp. 107–63, 1991.

[54] D. Brugali and M. Reggiani, "Software stability in the robotics domain: issues and challenges," in *International Conference on Information Reuse and Integration*. IEEE, 2005.

[55] D. Brugali, "Stable analysis patterns for robot mobility," in *Software Engineering for Experimental Robotics*. Springer, 2007, pp. 9–30.

[56] A. Ulusoy, S. L. Smith, X. C. Ding, C. Belta, and D. Rus, "Optimal multi-robot path planning with temporal logic constraints," in *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2011.

[57] G. E. Fainekos, A. Girard, H. Kress-Gazit, and G. J. Pappas, "Temporal logic motion planning for dynamic robots," *Automatica*, vol. 45, no. 2, pp. 343–352, 2009.

[58] M. Guo, K. H. Johansson, and D. V. Dimarogonas, "Revising motion planning under linear temporal logic specifications in partially known workspaces," in *International Conference on Robotics and Automation (ICRA)*. IEEE, 2013.

[59] E. M. Wolff, U. Topcu, and R. M. Murray, "Automaton-guided controller synthesis for nonlinear systems with temporal logic," in *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2013.

[60] H. Kress-Gazit, "Robot challenges: Toward development of verication and synthesis techniques [errata]," *IEEE Robotics & Automation Magazine*, vol. 18, no. 4, pp. 108–109, 2011.

[61] S. Maoz and J. O. Ringert, "Synthesizing a lego forklift controller in GR(1): A case study," in *Proceedings Fourth Workshop on Synthesis (SYNT)*, 2015.

[62] S. Maoz and J. O. Ringert, "On well-separation of GR(1) specifications," in *Foundations of Software Engineering (FSE)*. ACM, 2016.

[63] S. Maoz and J. O. Ringert. Spectra. http://smlab.cs.tau.ac.il/syntech/spectra/. Accessed: 2018-06-20.

[64] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri, "NuSMV: A new symbolic model verifier," in *Computer Aided Verification (CAV)*. Springer, 1999.

[65] L. Hugues and N. Bredeche, "Simbad: an autonomous robot simulation package for education and research," in *International Conference on Simulation of Adaptive Behavior*. Springer, 2006.

[66] "Accompanied material and data for this paper," http://www.roboticpatterns.com/, 2018.

[67] I. Ruchkin, J. Sunshine, G. Iraci, B. Schmerl, and D. Garlan, "IPL: An integration property language for multi-model cyber-physical systems," in *International Symposium on Formal Methods*. Springer, 2018.

[68] C. Menghi, C. Tsigkanos, T. Berger, P. Pelliccione, and C. Ghezzi, "Property specification patterns for robotic missions," in *International Conference on Software Engineering: Companion Proceeedings*, 2018.

[69] C. Menghi, C. Tsigkanos, T. Berger, and P. Pelliccione, "PsALM: Specification of dependable robotic missions," in *International Conference on Software Engineering (ICSE): Companion Proceeedings*, 2019.

[70] A. Pnueli, "The temporal logic of programs," in *Foundations of Computer Science*. IEEE, 1977.

[71] M. Ben-Ari, A. Pnueli, and Z. Manna, "The temporal logic of branching time," *Acta informatica*, 1983.

[72] "Google Scholar Robotic Venues," https://scholar.google.com/citations?view_op=top_venues&hl=en&vq=eng_robotics, 2017.

[73] P. Schillinger, S. Kohlbrecher, and O. von Stryk, "Human-robot collaborative high-level control with application to rescue robotics," in *International Conference on Robotics and Automation (ICRA)*. IEEE, 2016.

[74] S. L. Smith, J. Tumová, C. Belta, and D. Rus, "Optimal path planning for surveillance with temporal-logic constraints," *The International Journal of Robotics Research*, vol. 30, no. 14, pp. 1695–1708, 2011.

[75] https://www.youtube.com/watch?v=ib2hKuRO6n4, "PsALM video." 2018.

[76] A. Duret-Lutz, A. Lewkowicz, A. Fauchille, T. Michaud, E. Renault, and L. Xu, "Spot 2.0 — a framework for LTL and ω-automata manipulation," in *Automated Technology for Verification and Analysis*. Springer, 2016.

[77] P. Saadatpanah, M. Famelis, J. Gorzny, N. Robinson, M. Chechik, and R. Salay, "Comparing the Effectiveness of Reasoning Formalisms for Partial Models," in *Workshop on Model-Driven Engineering, Verification and Validation*. ACM, 2012.

[78] M. Famelis, R. Salay, and M. Chechik, "Partial Models: Towards Modeling and Reasoning with Uncertainty," in *International Conference on Software Engineering (ICSE)*. IEEE, 2012.

[79] C. Menghi, P. Spoletini, M. Chechik, and C. Ghezzi, "Supporting verification-driven incremental distributed design of components," in *Fundamental Approaches to Software Engineering*. Springer, 2018.

[80] C. Menghi, S. García, P. Pelliccione, and J. Tumova, "Towards multi-robot applications planning under uncertainty," in *International Conference on Software Engineering (ICSE): Companion Proceeeding*, 2018.

[81] C. Menghi, S. Garcia, P. Pelliccione, and J. Tumova, "Multi-robot LTL planning under uncertainty," in *International Symposium on Formal Methods (FM)*. Springer, 2018.

[82] G. Best, J. Faigl, and R. Fitch, "Multi-robot path planning for budgeted active perception with self-organising maps," in *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016.

[83] B. Takács and Y. Demiris, "Multi-robot plan adaptation by constrained minimal distortion feature mapping," in *International Conference on Robotics and Automation (ICRA)*. IEEE, 2009.

[84] A. Stentz, "Map-based strategies for robot navigation in unknown environments," in *AAAI spring symposium on planning with incomplete information for robot problems*, 1996, pp. 110–116.

[85] Syntech. http://smlab.cs.tau.ac.il/syntech/lego/. Accessed: 2018-06-20.

[86] Robotnik. https://www.robotnik.eu/mobile-robots/summit-xl-hl/. Accessed: 2018-06-20.

[87] Mitsubishi. https://robotik.dfki-bremen.de/en/research/robot-systems/mitsubishi-pa-10-7c.html. Accessed: 2018-06-20.

[88] P. robotics. http://tiago.pal-robotics.com/. Accessed: 2018-06-20.

[89] "Order Specification Patterns," http://patterns.projects.cs.ksu.edu/documentation/patterns/order.shtml.

[90] D. O. Paun and M. Chechik, "Events in linear-time properties," in *International Symposium on Requirements Engineering*. IEEE, 1999.

[91] D. Remenska, T. A. C. Willemse, J. Templon, K. Verstoep, and H. Bal, "Property specification made easy: Harnessing the power of model checking in uml designs," in *International Federated Conference on Distributed Computing Techniques*. Springer, 2014.

[92] K. C. Castillos, F. Dadeau, J. Julliand, B. Kanso, and S. Taha, *A Compositional Automata-Based Semantics for Property Patterns*. Springer, 2013.

[93] F. Bitsch, "Safety patterns - the key to formal specification of safety requirements," in *International Conference on Computer Safety, Reliability and Security*, 2001.

[94] G. Spanoudakis, C. Kloukinas, and K. Androutsopoulos, "Towards security monitoring patterns," in *Symposium on Applied Computing*. ACM, 2007.

[95] C. Côté, D. Létourneau, F. Michaud, and Y. Brosseau, "Software design patterns for robotics: Solving integration problems with

marie," in *Workshop of Robotic Software Environment, IEEE International Conference on Robotics and Automation*, 2005.
[96] N. M. Nasrabadi, "Pattern recognition and machine learning," *Journal of electronic imaging*, vol. 16, no. 4, p. 049901, 2007.
[97] J. Buchli and A. J. Ijspeert, "Distributed central pattern generator model for robotics application based on phase sensitivity analysis," in *International Workshop on Biologically Inspired Approaches to Advanced Information Technology*. Springer, 2004, pp. 333–349.
[98] C. Rothwell, A. Eggert, M. J. Patzek, G. Bearden, G. L. Calhoun, and L. R. Humphrey, "Human-computer interface concepts for verifiable mission specification, planning, and management," in *AIAA Infotech@ Aerospace (I@ A) Conference*, 2013, p. 4804.
[99] C. Rothwell, M. Patzek, and L. Humphrey, "Verifiable task assignment and scheduling controller," 711 Human Performance Wing Wright-Patterson AFB United States, Tech. Rep., 2017.
[100] D. C. Schmidt, "Guest editor's introduction: Model-driven engineering," *Computer*, vol. 39, no. 2, pp. 25–31, 2006.
[101] F. Ciccozzi, D. D. Ruscio, I. Malavolta, and P. Pelliccione, "Adopting MDE for specifying and executing civilian missions of mobile multi-robot systems," *Journal of IEEE Access*, vol. 2, no. 1, 2016.
[102] S. Adam, M. Larsen, K. Jensen, and U. P. Schultz, *Towards Rule-Based Dynamic Safety Monitoring for Mobile Robots*. Springer, 2014, pp. 207–218.
[103] M. Aiello, I. Pratt-Hartmann, J. van Benthem *et al.*, *Handbook of spatial logics*. Springer, 2007, vol. 4.
[104] H. Sun, J. Liu, X. Chen, and D. Du, "Specifying cyber physical system safety properties with metric temporal spatial logic," in *Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2015.
[105] J. Kennedy, P. Barclay *et al.*, "A survey of query languages for geographic information systems," 1996.
[106] C. H. Papadimitriou, D. Suciu, and V. Vianu, "Topological queries in spatial databases," in *Symposium on Principles of database systems*. ACM, 1996.
[107] R. S. Bivand, E. Pebesma, and V. Gómez-Rubio, *Spatial Data Import and Export*. Springer, 2013.
[108] M. J. Egenhofer and J. Herring, "Categorizing binary topological relations between regions, lines, and points in geographic databases," *The*, vol. 9, pp. 94–1, 1990.
[109] M. J. Egenhofer, A. U. Frank, and J. P. Jackson, "A topological data model for spatial databases," in *Symposium on Large Spatial Databases*. Springer, 1989, pp. 271–286.
[110] L. Cardelli, P. Gardner, and G. Ghelli, "A spatial logic for querying graphs," in *Automata, Languages and Programming*. Springer, 2002.
[111] L. Cardelli and L. Caires, "A spatial logic for concurrency," in *Theoretical Aspects of Computer Software (TACS)*, vol. 1, 2001, pp. 1–37.
[112] Z. Shao and J. Liu, *Spatio-temporal Hybrid Automata for Cyber-Physical Systems*. Springer, 2013, pp. 337–354.
[113] C. Tsigkanos, T. Kehrer, and C. Ghezzi, "Modeling and verification of evolving cyber-physical spaces," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, 2017*, 2017, pp. 38–48.
[114] V. Ciancia, D. Latella, M. Massink, and R. Pakauskas, "Exploring spatio-temporal properties of bike-sharing systems," in *Self-Adaptive and Self-Organizing Systems Workshops (SASOW), 2015 IEEE International Conference on*. IEEE, 2015, pp. 74–79.
[115] L. Nenzi, L. Bortolussi, V. Ciancia, M. Loreti, and M. Massink, "Qualitative and quantitative monitoring of spatio-temporal properties," in *Runtime Verification*. Springer, 2015, pp. 21–37.
[116] R. Koymans, "Specifying real-time properties with metric temporal logic," *Real-time systems*, vol. 2, no. 4, pp. 255–299, 1990.
[117] R. Alur, "Techniques for automatic verification of real-time systems," Ph.D. dissertation, Stanford, CA, USA, 1992.
[118] A. Aziz, K. Sanwal, V. Singhal, and R. K. Brayton, "Verifying continuous time markov chains," in *International Conference on Computer Aided Verification (CAV)*. Springer-Verlag, 1996.
[119] R. Kontchakov, A. Kurucz, F. Wolter, and M. Zakharyaschev, "Spatial logic+ temporal logic=?" in *Handbook of spatial logics*. Springer, 2007, pp. 497–564.

Claudio Menghi is an Associate Researcher at the Interdisciplinary Centre for Security, Reliability and Trust (SnT), at the University of Luxembourg. After receiving his PhD at Politecnico di Milano, he was post-doctoral researcher at Chalmers | University of Göteborg, Sweden. His current research interests lie in software engineering, with a special interest in cyber physical systems (CPS), and formal verification.

Christos Tsigkanos is university assistant at the Technical University of Vienna, Austria. Previously, he was post-doctoral researcher at Politecnico di Milano, Italy where he also received his PhD (2017). His current research interests lie in the intersection of distributed systems and software engineering, and include self-adaptive and dependable systems, requirements engineering and formal verification.

Patrizio Pelliccione is Associate Professor at the Chalmers University of Technology and University of Gothenburg, Sweden, Department of Computer Science and Engineering and Associate Professor at the University of L'Aquila, Italy (double affiliation). He got his PhD in 2005 at the University of L'Aquila (Italy) and from February 1, 2014 he is Docent in Software Engineering, title given by the University of Gothenburg. His research topics are mainly in software engineering, software architectures modelling and verification, autonomous systems, and formal methods. He has co-authored more than 120 publications in journals and international conferences and workshops in these topics. He has been on the program committees for several top conferences and is a reviewer for top journals in the software engineering domain. He is very active in European and National projects. In his research activity he has collaborated with several industries such as Volvo Cars, Volvo AB, Ericsson, Jeppesen, Axis communication, Systemite, Thales Italia, Selex Marconi telecommunications, Siemens, Saab, TERMA, etc. More information is available at http://www.patriziopelliccione.com.

Carlo Ghezzi is an Emeritus Professor in the Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Italy. He is past president of Informatics Europe. He has been the editor in chief of the ACM Transactions on Software Engineering and Methodology and associate editor of the IEEE Transactions on Software Engineering. He is currently an associate editor of the Communications of the ACM and the Science of Computer Programming. His research has been mostly focusing on different aspects of software engineering. He co-authored more than 200 papers and eight books. He coordinated several national and international research projects and has been a recipient of an ERC Advanced Grant. He received the ACM SIGSOFT Outstanding Research Award (2015) and the Distinguished Service Award (2006). He is a fellow of the ACM and the IEEE, a member of the European Academy of Sciences and the Italian Academy of Sciences.

Thorsten Berger is an Associate Professor in Software Engineering at Chalmers University of Technology and University of Gothenburg in Sweden. His research focuses on model-driven software engineering, program analysis, and empirical software engineering. He develops methods and tools for engineering highly configurable software. Thorsten Berger received the PhD degree in computer science from the University of Leipzig in Germany in 2013, supported by a scholarship from the German National Academic Foundation. He worked as a Postdoctoral Fellow at the University of Waterloo in Canada and the IT University of Copenhagen in Denmark. He received grants from the Swedish Research Council (competitive early-career grant), the Wallenberg Autonomous Systems Program, Vinnova Sweden (EU ITEA project), and the European Union (H2020 project). He received best-paper awards at the 2015 ACM SIGPLAN conference on MODULARITY and the 2013 European Conference on Software Maintenance and Reengineering (CSMR, now IEEE SANER). His service was recognized with a distinguished reviewer award at the 2018 IEEE/ACM ASE conference.