Towards Verifiable Multi-Agent Interaction Pattern Specification

Alberto Tagliaferro
alberto.tagliaferro@mail.polimi.it
Politecnico di Milano, Milan, Italy

Livia Lestingi
livia.lestingi@polimi.it
Politecnico di Milano, Milan, Italy

Matteo Rossi
matteo.rossi@polimi.it
Politecnico di Milano, Milan, Italy

ABSTRACT

Smart cyber agents play a crucial role in software-intensive systems by monitoring their physical surroundings and making impactful decisions. This paper addresses the challenge of specifying multi-agent patterns, which include interactions with human agents in possibly safety-critical environments. To this end, we introduce the foundations of a domain-agnostic and flexible Domain-Specific Language (DSL) called LIrAs. The language is designed to be accessible to users without programming expertise. LIrAs' semantics are mapped to Deterministic Finite-state Automata, making specifications amenable to formal verification. The DSL is exemplified through an illustrative scenario from the service robotics field.

CCS CONCEPTS

• **Software and its engineering** → **Domain specific languages**; **Formal language definitions**; • **Computer systems organization** → *Embedded and cyber-physical systems.*

KEYWORDS

Multi-Agent Patterns Specification, Domain-Specific Language, Pattern Verification

ACM Reference Format:
Alberto Tagliaferro, Livia Lestingi, and Matteo Rossi. 2024. Towards Verifiable Multi-Agent Interaction Pattern Specification. In *Formal Methods in Software Engineering (FormaliSE) (FormaliSE '24), April 14–15, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3644033.3644379

1 INTRODUCTION

Software-intensive systems are growingly pervasive due to rapidly developing technologies such as assistive robotics, IoT, and intelligent manufacturing systems. Smart cyber agents realize such systems by monitoring their physical surroundings and making decisions that impact the environment in which they operate. Complex systems often feature multiple agents (i.e., *multi-agent* systems) interacting or synchronizing with each other or the environment (e.g., with human agents) to perform their tasks. Such systems are often safety-critical (for example, due to the presence of humans), and failing to complete a specific task may incur significant costs.

FormaliSE '24, April 14–15, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0589-2/24/04
https://doi.org/10.1145/3644033.3644379

1.1 Envisioned Toolchain

Specifying interactive multi-agent tasks is a long-standing software engineering challenge [6]. Specifications should be sufficiently high-level to preserve accessibility and unambiguous to guarantee the dependability of the resulting software components.

This paper addresses this issue by presenting the foundational features of a DSL-based toolchain (see Fig. 1) called LIrAs[1] for specifying multi-agent interaction patterns. LIrAs is developed to be domain-agnostic and flexible with respect to the number and nature of agents involved (e.g., software-based or human) and the number and type of actions constituting the pattern. Agents differ based on the primitive skill set they offer (e.g., moving for a robot or switching on the engine for a quadcopter). In LIrAs, such skills are arranged into patterns. As per Fig. 1, defining the skill set requires the intervention of an expert practitioner, while pattern specification is designed to be accessible to non-expert users.

In LIrAs, the semantics of the synchronization dynamics between agents is based on Deterministic Finite-state Automata (DFA), making this aspect of the specifications amenable to model-to-model conversion and formal verification. Specifically, properties concerning the well-definedness of the synchronizations can then be verified through the Uppaal tool [11]. This stage of the toolchain targets only properties concerning the logical and structural soundness of the pattern (e.g., skills with conflicting goals), thus not involving the physical component of the corresponding cyber-physical system.

Several LIrAs specifications constitute a pattern library. In the following, we intend term *mission* as a sequence of patterns. A so-defined and verified library of patterns can be used in a wider mission specification and formal analysis toolchain external and decoupled from LIrAs, such as the one presented in [12]. The latter envisages a textual DSL (separate from LIrAs) to specify missions involving human and robot agents with a particular focus on human-robot interaction. Such DSL currently exploits human-robot interaction patterns from a pre-determined and fixed set, thus limiting the framework's applicability to real-life scenarios. This shortcoming could be overcome by importing a LIrAs pattern library for specifying the mission. We remark that the specification of the operational environment (e.g., the layout and the points of interest) is assumed to be done independently of LIrAs (i.e., LIrAs patterns are parametric w.r.t. the environment).

Within the broader framework, the resulting mission specification is automatically converted into a formal model based on Stochastic Hybrid Automata [5]. Should custom LIrAs interaction patterns be imported, automated generation of the formal model would imply drafting the formal model of each custom skill employed in the newly defined pattern, which requires the input of an expert user. The resulting formal model of the mission is subject to Statistical Model Checking [5] to compute quality metrics

[1] LIrAs stands for "Language for Interactive Agents".

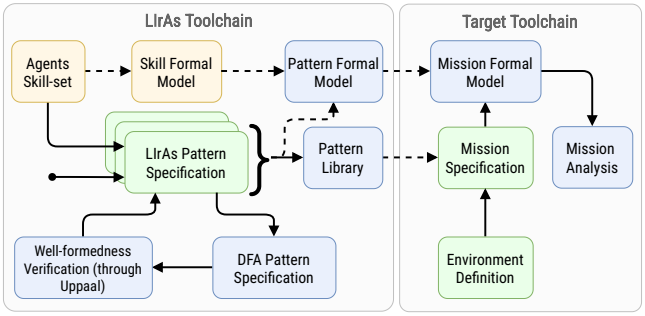

Figure 1: Envisioned toolchain. Automated tasks are in blue, while tasks to be performed by expert practitioners and non-expert users are in yellow and green, respectively. Dashed arrows represent future developments.

concerning the mission (e.g., time to completion and physical effort required on the humans' side) and apply reconfiguration measures, if necessary. In this later stage of the toolchain, given the hybrid nature of Stochastic Hybrid Automata, both the cyber and physical components of the corresponding system can be involved in the formal analysis. Properties to be verified may concern, for example, relevant physical variables with continuous dynamics (e.g., a robot's battery charge) or the incompatibility between the custom-defined patterns and the environment.

1.2 Related Work

Several formal notations have been exploited over the years to model the workflow of agents' tasks, such as Petri Nets [17], automata [10], and Linear Temporal Logic [19]. However, these approaches are meant for automated development toolchains and are, thus, inaccessible to users lacking expertise in formal modeling.

When a human-in-the-loop approach is required, a common practice consists in using a high-level DSL to define specifications that would otherwise be excessively ambiguous in natural language. DSLs for task specification can either be general-purpose [4, 16, 20], or target specific categories of agents, such as autonomous vehicles [18], aerial [14] and medical rehabilitation robots [7]. Specifically, concerning robotics, Nordmann et al. in 2014 surveyed 137 DSLs [15], and significant progress has been made in the field ever since. For example, PsALM [13] is a structured grammar-based DSL that defines 22 patterns for robotic missions. PROMISE [8] envisages the specification of concurrent robot missions through sequences of basic actions instead of predefined patterns. PGCD [2] combines message-passing and motion primitives for robots in a fleet. The MutRoSe framework [9] automates the decomposition of a mission into valid task sequences allocated to different robots in a fleet. PuRSUE [3] envisages a modeling language for human-robot interactions whose specifications are compiled into Timed Game Automata. The language, however, contains lower-level primitives since it targets users with a stronger technical background and could be considered complementary to LIrAs.

LIrAs allows for more flexible specifications by combining agents' primitives into patterns rather than relying on a predetermined set thereof. Therefore, a LIrAs specification is not tied to a specific type of agent: any set of agents (including humans) offering the required skills is compatible with a LIrAs pattern. The latter feature also targets a shared shortcoming of existing languages, i.e., the limited or absent support for patterns involving human agents. Finally, LIrAs targets a level of abstraction high enough to avoid the description of architectural details (patterns are parametric w.r.t., for example, a robot's angular and translational velocities).

1.3 Paper Contributions

In this research idea paper, we lay the foundations for the LIrAs toolchain, presenting:

- the fundamental features of the LIrAs syntax;
- their formalization through DFA;
- an illustrative application of LIrAs to a use case from the assistive robotics domain.

LIrAs syntax and semantics are outlined in Section 2 and exemplified through the illustrative example; Section 3 concludes and discusses future research directions.

2 LIRAS SYNTAX AND SEMANTICS

We present the foundational features of LIrAs's syntax and its semantics based on DFA.

2.1 Syntax

LIrAs's syntax is exemplified through Listing 1 and Listing 2. *Agents* are the actors realizing LIrAs's specifications. Each agent is characterized by an id and is indicated in the rest of this paper as $a_{id} \in AG$. In the following, the term *action* refers to an atomic skill an agent possesses. Actions are the foundational building blocks of the language and are denoted as `<action{i}>`. An action may be parametric w.r.t. the environment (e.g., moving to a specific point of interest in the environment), and the actual value of each parameter must be specified through an identifier (e.g., moveTo poi1).

Actions can be declared within *conditional* blocks, which, based on their type, capture premature termination, introduce dependencies between agents, and impose time constraints. Table 1 details the grammar for a conditional action block. The DSL assumes that a set AP of atomic predicates (e.g., indicating an agent being in possession of a resource) and a set W of real-valued variables (e.g., capturing the position of an agent) are available. In the following, we use the generic token `<condition>` to indicate conditions consisting of atomic predicates and constraints of the form $w \lesssim \theta$ (where w is a variable and θ is a threshold), possibly combined

Listing 1: Abstract example of LIrAs' syntax, reporting the fundamental elements.

```
<pattern>
<agent1>:
        <l1>: <action1>
        <l2>: <conditional action2>
                <action3>
<agent2>:
        <l1>: <action1>
        <l2>: <conditional action2>
<success> <condition>
<failure> <condition>
```

through the usual Boolean operators ($\neg$, $\wedge$). There are two types of conditional action blocks:

- <action> until <condition>, stating that <action> is interrupted as soon as <condition> is true;
- <action> if <condition> else <action'>, stating that if <condition> is true, then <action> is executed, otherwise <action'> is.

Actions executed by the same agent are grouped into *sequences*. The declaration of agent a_{id}'s sequence begins with <agent_{id}>. Sequences are composed of an ordered list of *sub-sequences*, each syntactically identified by a label of the form <l{i}>, where i is a strictly increasing index. Formally, sub-sequence i executed by agent a_{id} is represented through notation $a_{id}.s_i$. Sub-sequences contain one or more actions that must be executed sequentially. Notation $a_{id}.s_i.x_j$ refers to action j of sub-sequence $a_{id}.s_i$. Action $a_{id}.s_i.x_{j+1}$ can start only if $a_{id}.s_i.x_j$ has been completed. Similarly, $a_{id}.s_{i+1}$ can only start if $a_{id}.s_i$ has been completed. In Listing 1, a_1's sequence includes two sub-sequences labelled as <l1> and <l2>. Sub-sequence <l1> only contains <action1>, while <action2> and <action3> both belong to <l2>, and are executed sequentially.

Sub-sequences belonging to different agents' sequences can share the same label. Synchronization between different agents, indeed, occurs by having sub-sequences with the same label start simultaneously. In Listing 1, Lines 3 and 7 start simultaneously, and the same happens for Lines 4 and 8. By default, $a_{id}.s_1$ begins simultaneously for all agents. If a sub-sequence contains no action, it can either be declared empty or omitted entirely. Both cases correspond to the agent performing a default action (e.g., waiting or moving freely), which can be redefined through the `default` keyword.

The example from the service robotics domain demonstrates how a pattern can be specified using LIrAs (see Listing 2) starting from a natural language specification. We assume that a set of predicates is available: time(x) is true if at least x time units have elapsed since the beginning of the pattern; position(x, y) is true if agent x's position equals y while dist(x, y) returns the Euclidean distance between x and y; possess(x, y) is true if agent x holds resource y; finally, tired(x) is true if agent x requires rest before moving again, and SoC(x) returns the state of charge of agent x.

The scenario involves three agents: two robots (indicated as Robot1 and Robot2) and a Human. The robots' skill set includes stopping, moving to a target, fetching a resource, and delivering a resource. The human's skill set includes moving to a target, following another agent, stopping, and receiving a resource from another agent (the latter must synchronize with a delivering action). We assume that the environment features two points of interest (referred to as room1 and room2) and one resource named item1. Initially, Robot1 waits for the Human to approach (Line 4). Concurrently, Robot2 moves to room1 for patrolling (Line 15). Upon completing this initial movement, Robot1 must guide Human to room1 (Line 5). Throughout this interaction, the robot monitors the distance with the Human, stopping if it exceeds a certain threshold and resuming when the Human gets closer. Human follows Robot1 (Line 10); if tired, it stops and resumes following the robot only when rested. When both Human and Robot1 reach room1, Human waits (Line 12) for Robot1 to fetch item1 and deliver it (Lines 6, 7, and 8). Since its presence in room1 is not necessary once Human is there (Line 16), Robot2 moves to room2 (Line 17). The pattern is successfully completed when Human is in room1 and has item1 (note that the pattern can be completed before Robot2 reaches room2). The pattern fails if it takes more than time t_th to complete or if Robot1's charge (SoC) falls below a threshold.

Table 1: Grammar for a conditional action.

$\psi ::= x$ until $\phi \mid x$ if ϕ else x'	$x, x' \in$ AC (action)
$\phi ::= p \mid w \lesssim \theta \mid \neg\phi \mid \phi \wedge \phi$	$p \in AP$ (atomic predicate)
	$w \in W$ (real-valued variable)
	$\theta \in \mathbb{R}$ (real-valued threshold)

Listing 2: LIrAs interaction pattern for the three agents from the service robotic example.

```
1  PatrolFetchDeliver(Robot1, Robot2, Human,
2  room1, room2, item1, d_th, t_th, c_th)
3  Robot1:
4      l1:  stop
5      l2:  moveTo room1 if dist(Human,Robot1)<d_th else stop
6      l3:  fetch item1
7      l4:  moveTo Human
8      l5:  deliver item1
9  Human:
10     l1:  moveTo Robot1 until dist(Human,Robot1)<d_th
11     l2:  follow Robot1 (room1) if !tired else stop
12    l3-l4: stop
13     l5:  receiveFrom Robot1
14 Robot2:
15     l1:  moveTo room1
16    l2-l5: stop until !position(Human, room1)
17         moveTo room2
18
19 success position(Human, room1) && possess(Human, item1)
20 failure time(t_th) || SoC(Robot1)<c_th
```

Listing 2 shows a broader range of synchronization dynamics among sub-sequences. Sub-sequences can also be grouped into a *continuation*, labeled as <l{i}-{i+k}> (see Listing 2, Line 12). This sub-sequence set starts synchronously with s_i and is only required to end for s_{i+k+1} to start (i.e., no further synchronization is enforced with sub-sequences in the range $[s_{i+1}, s_{i+k}]$). This feature captures the cases when more than two agents are involved and not all sub-sequences must start synchronously for all the agents. In the example in Listing 2, Line 12 features a continuation because the human does not need to synchronize with Robot1 when it has picked up the item and starts moving back to the human to deliver it. Similarly, Robot2 only needs to synchronize with the human when they enter room1, thus Lines 16 and 17 are part of a continuation.

A set of sequence declarations, one for each agent $a_{id} \in AG$, with $|AG| \geq 1$, constitutes a *pattern*, whose identifier is indicated through token <pattern> in Listing 1. The execution of a pattern is considered complete when the last subsequence of each agent has been completed. It is also possible to specify sufficient conditions for successful (e.g., each agent's goal has been achieved) or unsuccessful (e.g., an agent is unable to proceed due to exhausted battery) termination of the pattern through tokens <success> and <failure> followed by a condition expressed according to Table 1.

2.2 Semantics

As detailed in the following, the semantics of synchronization among agents in LIrAs patterns defined according to this syntax can be mapped to DFA. DFA are defined as follows.

Definition 2.1. A DFA is a 5-tuple $(A, Q, q_{ini}, F, \delta)$ where A is the *alphabet* (i.e., a set of symbols); Q is the set of states, with $q_{ini} \in Q$ as the initial state; $F \subseteq Q$ is the subset of final states; $\delta : Q \times A \rightarrow Q$ defines the state transitions labeled with symbols in A.

A LIrAs pattern can be converted into a DFA *network*. The conversion is exemplified through the DFA network in Fig. 2, modeling the pattern in Listing 2.

Each agent in set AG corresponds to a standalone DFA, and the DFA in the network synchronize over symbols in A. DFA modeling agent $a_{id} \in AG$ has an initial state $a_{id}.q_{ini}$ modeling the situation in which no subsequence has started and a final state $a_{id}.q_{end}$ modeling the case in which all subsequences have ended. For each subsequence $a_{id}.s_i$, the DFA has two states (indicated as $a_{id}.q_{i,b}$ and $a_{id}.q_{i,c}$), modeling a_{id} being *busy* executing $a_{id}.s_i$ and a_{id} having *completed* $a_{id}.s_i$ and waiting for the following synchronization, respectively. Notice that a $(a_{id}.q_{i,b}, a_{id}.q_{i,c})$ pair is defined even when $a_{id}.s_i$ consists only of the default action, but not for continuations (e.g., for subsequence $a_2.s_3$). In the latter case, state $a_2.q_{3,c}$ is not necessary because, due to the continuation, a_2 does not need to synchronize with other agents to start $a_2.s_4$.

Symbols $l_i \in A$ are defined for each index $i \in \mathbb{N}$ such that subsequence $a_{id}.s_i$ is defined for some $a_{id} \in AG$. Internal action ε and symbol *end* marking the end of a sequence are also available. Transition function δ then defines, for all $(a_{id}.q_{i,b}, a_{id}.q_{i,c})$ pairs, a transition labeled with internal action ε from $a_{id}.q_{i,b}$ to $a_{id}.q_{i,c}$ capturing the completion of $a_{id}.s_i$. Furthermore, a transition labeled with l_j is defined between each $(a_{id}.q_{i,c}, a_{id}.q_{j,b})$ pair such that no other subsequence $a_{id}.s_k$ with $i < k < j$ exists, marking the start of $a_{id}.s_j$. If $a_{id}.s_i$ is the last for a_{id}, an edge labeled with *end* from $a_{id}.q_{i,c}$ to $a_{id}.q_{end}$ is defined.

In this preliminary stage, guard conditions deriving from conditional action blocks (including time constraints) are not involved in the mapping to DFA. The same holds for early termination conditions due to success and failure. In the future, we plan on incorporating all these aspects into the model-to-model conversion of LIrAs

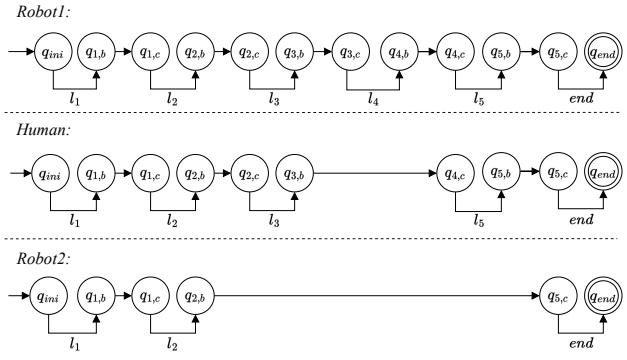

Figure 2: DFA network representing the illustrative example scenario.

patterns, thus requiring an extension of the semantics to Timed Automata. In light of this, we use Uppaal for our analysis[2], even if the models do not currently capture time. Also, the properties regarding the well-definedness of patterns that are verified on DFA networks are expressed in Timed Computation Tree Logic (TCTL), although the temporal features of the logic are not currently exploited. As a result, Uppaal considers all the transitions in the DFA networks non-deterministic w.r.t. the firing time.

Figure 2 shows the DFA specification for the illustrative example. Notice that, due to the continuations of Human and Robot2, the former does not need to synchronize over symbol l_4, and the same stands for the latter and symbols l_3, l_4, and l_5. For the network in Fig. 2, the TCTL property captured by Formula (1)—which states that there exists a path such that eventually all agents reach the final state—is verified.

$$\text{EF} \bigwedge_{a_{id} \in AG} a_{id}.q_{end} \tag{1}$$

The property captured by Formula (2) (i.e., "for all paths, eventually all agents reach the final state"), instead, does not hold for the network since an agent can remain indefinitely in any state other than q_{end}.

$$\text{AF} \bigwedge_{a_{id} \in AG} a_{id}.q_{end} \tag{2}$$

Including timing constraints and conditional action blocks in the model extends the range of verifiable properties, potentially highlighting issues arising from unsatisfiable action completion conditions (e.g., a location being unreachable). For the network in Fig. 2, examples of issues that may arise from the verification of well-formedness properties are item1 being in an unreachable position (thus, the action on Line 6 never terminates) or the entire pattern always failing to complete within time t_th.

3 FUTURE RESEARCH OUTLOOK

In the future, we plan on releasing LIrAs as a fully-fledged toolchain by implementing all the steps envisaged in Fig. 1. To this end, it is necessary to extend LIrAs's semantics to Timed Automata, incorporating guards and timing constraints in the model-to-model conversion and significantly extending the range of verifiable properties. State-of-the-art testing techniques could be employed to explore the domain of the involved variables in the absence of a specific environment definition.

We also plan on assessing the effectiveness of LIrAs when serving as the basis for a broader mission analysis framework through realistic scenarios (see, for example, those collected in [1]). This requires developing an automated approach to compile LIrAs patterns into Stochastic Hybrid Automata, potentially starting from formal model snippets corresponding to individual skills.

Finally, a user study is necessary to ascertain the user-friendliness of the language to non-expert users with a technical skillset (e.g., practitioners without a background in programming) and domain experts (e.g., logistic experts in need of deploying a multi-agent mission in their facility).

[2]Example Uppaal models are available at: https://doi.org/10.5281/zenodo.10556557.

REFERENCES

[1] Mehrnoosh Askarpour, Christos Tsigkanos, Claudio Menghi, Radu Calinescu, Patrizio Pelliccione, Sergio García, Ricardo Caldas, Tim J von Oertzen, Manuel Wimmer, Luca Berardinelli, et al. 2021. Robomax: Robotic mission adaptation exemplars. In *Intl. Symp. on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 245–251.

[2] Gregor B Banusić, Rupak Majumdar, Marcus Pirron, Anne-Kathrin Schmuck, and Damien Zufferey. 2019. PGCD: robot programming and verification with geometry, concurrency, and dynamics. In *ACM/IEEE Intl. Conf. on Cyber-Physical Systems*. 57–66.

[3] Marcello M Bersani, Matteo Soldo, Claudio Menghi, Patrizio Pelliccione, and Matteo Rossi. 2020. PuRSUE-from specification of robotic environments to synthesis of controllers. *Formal Aspects of Computing* 32 (2020), 187–227.

[4] Matthew L Bolton, Radu I Siminiceanu, and Ellen J Bass. 2011. A systematic approach to model checking human–automation interaction using task analytic models. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 41, 5 (2011), 961–976.

[5] Alexandre David, Kim G Larsen, Axel Legay, Marius Mikučionis, Danny Bøgsted Poulsen, Jonas Van Vliet, and Zheng Wang. 2011. Statistical model checking for networks of priced timed automata. In *Intl. Conf. on Formal Modeling and Analisys of Timed Systems*. Springer, 80–96.

[6] Swaib Dragule, Sergio García Gonzalo, Thorsten Berger, and Patrizio Pelliccione. 2021. Languages for specifying missions of robotic applications. *Software Engineering for Robotics* (2021), 377–411.

[7] Peter Forbrig and Alexandru-Nicolae Bundea. 2020. Modelling the collaboration of a patient and an assisting humanoid robot during training tasks. In *Intl. Conf. HCII*. Springer, 592–602.

[8] Sergio García, Patrizio Pelliccione, Claudio Menghi, Thorsten Berger, and Tomas Bures. 2019. High-level mission specification for multiple robots. In *ACM SIGPLAN Intl. Conf. on Software Language Engineering*. 127–140.

[9] Eric Bernd Gil, Genaína Nunes Rodrigues, Patrizio Pelliccione, and Radu Calinescu. 2023. Mission specification and decomposition for multi-robot systems. *Robotics and Autonomous Systems* 163 (2023), 104386.

[10] Bruno Lacerda and Pedro Lima. 2009. LTL plan specification for robotic tasks modelled as finite state automata. In *Proc. of Workshop ADAPT–Agent Design: Advancing from Practice to Theory, Workshop at AAMAS*, Vol. 9.

[11] Kim G Larsen, Paul Pettersson, and Wang Yi. 1997. UPPAAL in a nutshell. *Int. J. on Softw. Tools for Tech. Transf.* 1, 1-2 (1997), 134–152.

[12] Livia Lestingi, Davide Zerla, Marcello M Bersani, and Matteo Rossi. 2023. Specification, stochastic modeling and analysis of interactive service robotic applications. *Robotics and Autonomous Systems* 163 (2023), 104387.

[13] Claudio Menghi, Christos Tsigkanos, Patrizio Pelliccione, Carlo Ghezzi, and Thorsten Berger. 2019. Specification patterns for robotic missions. *IEEE Transactions on Software Engineering* (2019).

[14] Martin Molina, Ramon A Suarez-Fernandez, Carlos Sampedro, Jose Luis Sanchez-Lopez, and Pascual Campoy. 2017. TML: a language to specify aerial robotic missions for the framework Aerostack. *International Journal of Intelligent Computing and Cybernetics* 10, 4 (2017), 491–512.

[15] Arne Nordmann, Nico Hochgeschwender, and Sebastian Wrede. 2014. A survey on domain-specific languages in robotics. In *International conference on simulation, modeling, and programming for autonomous robots*. Springer, 195–206.

[16] Fabio Paterno, Cristiano Mancini, and Silvia Meniconi. 1997. ConcurTaskTrees: A diagrammatic notation for specifying task models. In *Intl. Conf. on Human-Computer Interaction*. Springer, 362–369.

[17] Khodakaram Salimifard and Mike Wright. 2001. Petri net-based modelling of workflow systems: An overview. *European journal of operational research* 134, 3 (2001), 664–676.

[18] Daniel Castro Silva, Pedro Henriques Abreu, Luís Paulo Reis, and Eugénio Oliveira. 2014. Development of a flexible language for mission description for multi-robot missions. *Information Sciences* 288 (2014), 27–44.

[19] Jana Tumova and Dimos V Dimarogonas. 2016. Multi-agent planning under local LTL specifications and event-based synchronization. *Automatica* 70 (2016), 239–248.

[20] Wil MP Van Der Aalst and Arthur HM Ter Hofstede. 2005. YAWL: yet another workflow language. *Information systems* 30, 4 (2005), 245–275.