Verification-Oriented Specification of Multi-Agent Interaction Patterns

Alberto Tagliaferro[0009−0004−4923−7831], Livia Lestingi[0000−0001−8724−1541], Matteo Rossi[0000−0002−9193−9560]

{firstname.lastname}@polimi.it
Politecnico di Milano, Milan, Italy

Abstract. Smart cyber agents are pivotal in software-intensive systems such as smart manufacturing, robotics, and the Internet of Things. These agents monitor physical surroundings through sensors and make impactful decisions that influence the environment. Software engineering challenges in this domain include the specification of interactive multi-agent tasks. The general-purpose Domain-Specific Language named LIrAs, Language for Interactive Agents, is a high-level language that allows for unambiguous custom pattern definition. Additionally, LIrAs facilitates interactions with human agents, a safety-critical situation requiring particular attention. This paper lays the foundation for LIrAs specifications translation to Stochastic Hybrid Automaton (SHA). The target SHA model structure follows a three-layer hierarchical structure and makes LIrAs specifications amenable to formal verification, specifically Statistical Model Checking, through the Uppaal tool, capable of including time-dependent physical phenomena, such as human fatigue and robot dynamics.

Keywords: Multi-Agent Patterns Specification · Domain-Specific Language · Stochastic Hybrid Automata.

1 Introduction

Assistive robotics, Internet of Things (IoT), and intelligent manufacturing systems are examples of increasingly widespread and complex software-intensive systems. These systems feature smart cyber agents equipped with sensors to gather environmental data and make decisions influencing their surroundings. Including multiple interacting agents—i.e., Multi-Agent Systems (MASs)—adds complexity to these environments since agents must interact with both the environment and each other to be effective. The involvement of human agents (often unavoidable, if not desirable) is an additional challenge since well-tailored programming and decision-making on the robotic agents' side can spare stakeholders from financial losses or physical harm.

Specifying MAS interactive tasks poses a critical challenge in software engineering [10]. The inherent complexity of these systems often necessitates expert users for task specification, making the process both time-consuming and error-prone. To address this issue, a common solution is developing a Domain-Specific

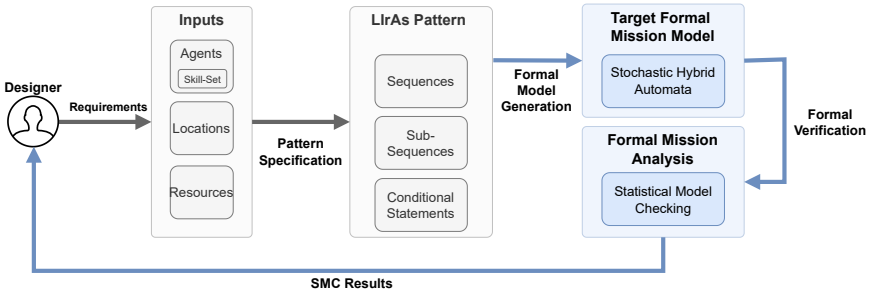


Fig. 1: LIrAs-based toolchain for MAS specification. Pre-existing elements are in grey, while the contributions of this paper are highlighted in blue. The inputs on the left are defined externally from LIrAs and depend on the application field. In the middle, there are the main elements of a LIrAs pattern and on the right, the target mission analysis framework, selected to be compatible with the actual framework for the analysis of Human-Robot Interaction (HRI) missions [16].

Language (DSL) to be used as a high-level language. This approach significantly reduces the complexity of specifications, the time required to develop them, the likelihood of errors, and the amount of technical knowledge needed. Often, DSLs can be translated into formal models, enabling the verification of correctness properties through formal verification techniques.

This work builds upon a DSL-based toolchain for MAS specifications, illustrated in Figure 1, centered around the LIrAs DSL [25]. LIrAs enables the specification of tasks for various types of agents (software-based or human) with a high degree of versatility across different applications. In LIrAs, the interaction patterns are flexible concerning the type and quantity of actions involved since they are not bound to a specific application. LIrAs agents are endowed with basic atomic abilities, referred to as *skills*, such as a translational motion for a robot or a quadcopter. Through LIrAs, it is possible to define *patterns*, as depicted in Figure 1, combining different skills that multiple agents must perform in a coordinated fashion through user-friendly primitives. LIrAs patterns are parametric with respect to the context in which they will be executed—i.e., all aspects related to specifying the operational environment, such as layout or Points of Interest (PoIs), are defined externally from LIrAs. Several LIrAs specifications constitute a custom pattern *library*.

The semantics of LIrAs patterns, specifically the custom synchronization dynamics among different agents, are defined starting from Deterministic Finite-state Automata (DFA), allowing for the verification of basic well-formedness properties, such as ensuring that the pattern can be completed. Interested readers can refer to [25] for the presentation of the fundamental primitives of LIrAs and their mapping to DFA semantics.

However, formal models of the agents' skills involve a broader range of features than can be expressed through DFA (e.g., physical variables, such as a

robot's battery charge, and sources of uncertainty, such as human choices). Furthermore, LIrAs pattern libraries, when *contextualized* within a layout with specific agents instances, lend themselves to richer analyses than basic well-formedness properties, such as the computation of *quality metrics* concerning the effort required on the human agent's side or the timeliness of a contextualized custom pattern. Lestingi et al. [16] introduce a framework for formally modeling and analyzing HRIs. The framework relies on a textual DSL, distinct from LIrAs, specifically tailored for tasks involving both human and robotic agents. However, a limitation of this DSL is its reliance on a finite set of predefined patterns, which may not be sufficient to cover all real-world scenarios.

The hereby presented vision builds upon both sides by bridging the LIrAs toolchain with the framework for HRI analysis, hereinafter referred to as the *target* framework. LIrAs libraries of *ad-hoc* patterns can be imported into the DSL for HRI mission specification, with *mission* referring to a sequence of LIrAs patterns selected from the pattern library, thus enhancing the flexibility in mission specification.

The target framework models HRIs missions as networks of Stochastic Hybrid Automata (SHA), analyzed through Statistical Model Checking (SMC). SHA capture the physical components of the cyber-physical system under modeling, including factors such as battery charge and discharge or human fatigue, whose time dynamics are represented through differential equations [16]. Additionally, SHA enable the representation of stochastic behavior arising from human free will, capturing the possibility of a human agent acting differently from expectations.

Therefore, for the incorporation of LIrAs patterns into the target framework and mission analysis through SMC to be possible, a model-to-model conversion principle from LIrAs to SHA must be defined. This work introduces the stage of the toolchain where the pattern specification is transformed into a network of SHA, which is then analyzed using SMC. The translation from LIrAs to SHA follows a three-layer hierarchical structure. The top layer manages the synchronization of sub-sequences among agents; the middle one controls the actions' execution; the bottom layer contains the action skill set for the agents belonging to the mission.

The contributions of this paper pave the way towards the computation of quality metrics associated with the mission, such as the time required for completing each pattern and the physical effort required from humans. Once these metrics are computed, reconfiguration measures can be applied to satisfy specific design constraints. Given the hybrid nature of SHA, the components considered in the formal analysis in this newly developed stage of the toolchain encompass both cyber and physical components. Verified properties may encompass physical phenomena over time with continuous dynamics, such as the battery charge of a robot, and may also address compatibility with the environment and custom-defined patterns.

The paper is structured as follows: Section 2 provides an analysis of related work; Section 3 presents the technical background for SHA; Section 4 offers an

illustrative example; Section 5 describes the translation from LIrAs to SHA; Section 6 concludes and presents possible future developments.

2 Related Work

The aim of LIrAs is to encompass a wide range of requirements specification in multi-agent systems, such as robotic applications or smart manufacturing. LIrAs' distinctive features are the possibility to customize patterns instead of relying on a predefined set and to include human agents in a pattern, making it particularly suitable for human-robot interaction patterns.

Various formal notations, such as Petri Nets [23], Automata [14], Linear Temporal Logic (LTL) [26], and Temporal Plan Networks [17] have been used to model the workflow of different agents' tasks. However, these approaches are less accessible to users without expertise in formal modeling. For this, especially in scenarios where a human-in-the-loop approach is essential, a common practice is to employ a high-level DSL for specifying requirements. This allows users to articulate specifications that might otherwise be ambiguous in natural language.

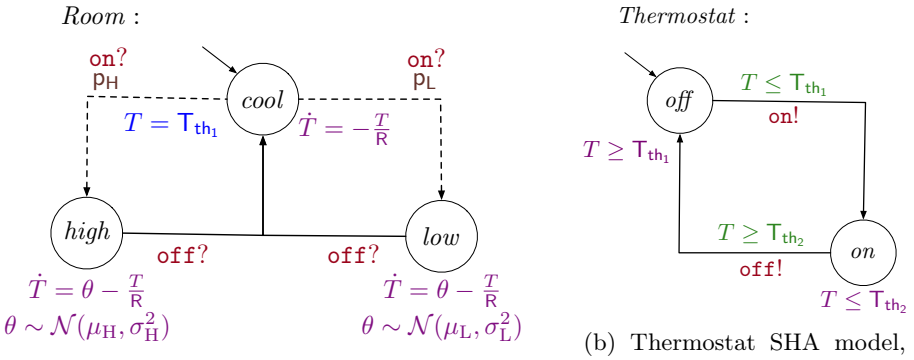
The DSLs developed for task specification can be general-purpose [28,21,5] or bounded to specific agent categories, such as autonomous vehicles [24], multicopters [6,22], or medical rehabilitation robots [12]. In the field of robotics, Nordmann et al. in 2014 surveyed 137 such languages [20]. More recently, PsALM, a grammar-based DSL, was introduced, defining 22 patterns for robotic missions [18]. Additionally, PROMISE focuses on specifying concurrent robot missions through sequences of basic actions without relying on predefined patterns [13]. PuRSUE is a modeling language for human-robot interactions compiled into Timed Game Automata [4]. However, its language features lower-level primitives, as it targets users with a stronger technical background, making it complementary to LIrAs.

LIrAs provides high flexibility in specifications by combining agents' primitives into patterns without being constrained to a predefined set. This unique characteristic enables LIrAs to accommodate any type of agent, including humans, overcoming a common limitation in existing languages [27]: the limited support for patterns involving human agents. In conclusion, LIrAs offers a high level of abstraction, allowing patterns to be parametric to the environment and the various agent characteristics, such as the angular and translational velocities of robots.

3 Background

This section recaps the theoretical foundation of this paper's contribution, concerning SHA and SMC [2,9,11].

SHA features are illustrated through a running example inspired from [8, Section 4]. The system depicted in Figure 2 [16] consists of two distinct components: a heating system (Figure 2b) and a room (Figure 2a). The physical



(a) Room SHA model, receiving events from the thermostat.

(b) Thermostat SHA model, initiating events that influence the room.

Fig. 2: Example of SHA network [16]. Dashed arrows represent probabilistic transitions, characterized by weights (in brown) denoted with $\mathsf{p_H}$ and $\mathsf{p_L}$, while solid arrows are deterministic transitions. Color-coding is as follows: channels are red, probability distributions, flow conditions, exponential rates are purple, guard conditions are green, and updates are blue.

phenomenon measured in this system is the temperature of the room, modeled as a real-valued variable, denoted by T, in the depicted automaton.

The thermostat's model has two locations, *on* and *off*. While in *off*, if the temperature falls below threshold $\mathsf{T_{th_1}}$ (as per invariant $T \geq \mathsf{T_{th_1}}$ labeling the *off* location), the thermostat switches to the *on* location, activating the heating system. Similarly, the thermostat switches *off* the heating system when, while in *on*, the temperature exceeds the threshold $\mathsf{T_{th_2}}$. Three locations model the room temperature's behavior, as illustrated in Figure 2a. The temperature decreases naturally in the *cool* location with the heating system *off*. Locations *high* and *low* represent the room while the heating system is active, with the temperature increasing at different rates.

The temperature evolution follows differential equations, referred to as flow conditions. When the thermostat is *on*, the associated differential equation is $\dot{T} = \theta - \frac{T}{\mathsf{R}}$, while it is $\dot{T} = -\frac{T}{\mathsf{R}}$ when the heating system is *off*. Here, R is a constant, and θ is a randomly distributed parameter whose variability is based on a probability distribution. At the onset of the system, the thermostat is in *off*, and the temperature is initialized to $\mathsf{T_{th_1}}$.

The parameter θ in Figure 2a represents a realization of a normal distribution. When the room is heating at a high rate, it has a standard deviation of σ_{H} and a mean of μ_{H} (denoted as $\mathcal{N}(\mu_{\mathrm{H}}, \sigma^2_{\mathrm{H}})$), where "H" stands for high. The probability distribution is $\mathcal{N}(\mu_{\mathrm{L}}, \sigma^2_{\mathrm{L}})$ when the room is heating slowly, as there is an open window; here, the subscript "L" indicates low. The notation $\theta \sim \mathcal{N}(\mu, \sigma)$ is used to express that a random parameter θ is a realization of a random variable Θ governed by the distribution $\mathcal{N}(\mu, \sigma)$.

SHA switch locations through edges, and edges can be labeled by the events that trigger the corresponding switch. Event labels are either of the form `e`? or `e`! depending on whether the automaton receives or triggers the event, respectively. For example, once the thermostat triggers the `on`! event, the room reacts (label `on`?), realizing the synchronization between the two automata. The decision regarding the heating rate (whether high or low) when `on`! fires is probabilistic with weights $\mathtt{p_H}$ and $\mathtt{p_L}$, respectively.

SHA are amenable to SMC [1] and can be modeled and analyzed through the Uppaal tool [15]. SMC requires a stochastic model M (the SHA network in our case) and a property ψ formulated in Metric Interval Temporal Logic (MITL) over atomic propositions taken from the set AP [3]. These propositions include constraints in the set of symbols W (e.g., $w < 10$) or automata locations (such as *high* in Figure 2a). In our work, the property ψ is of the form $\diamond_{\leq \tau} ap$, where $\diamond$ is the "eventually" operator, $ap \in AP$ and τ is a time bound (i.e., the formula is true if ap holds within τ units of time from the onset). SMC estimates the probability of ψ holding for M—i.e., the value of expression $\mathbb{P}_M(\psi)$ [8].

In this paper, LIrAs specifications are translated into SHA network M. Through SMC, the framework calculates the probability of SHA o eventually reaching the location l within a time-bound τ, expressed as $\mathbb{P}_M(\diamond_{\leq \tau}\ o.l)$.

4 Illustrative Example

In this section, we propose an example of specification using LIrAs, based on the primitives of this language. We first present a MAS specification exemplar in natural language, followed by its associated LIrAs specification, highlighting LIrAs's compact and unambiguous syntax. In Section 5, to describe the contribution of this work, the exemplar is translated to DFA and SHA, and then analyzed through SMC.

The illustrative example focuses on HRI, involving two agents: a `Human` and a `Robot`. In LIrAs, each agent is associated with a set of *actions*, i.e., the atomic skills available to them to complete the mission. Agents operate in a known layout characterized by its geometry and a set of *points of interest*, intended as potential targets of their actions (e.g., significant locations to move to or equipment they can interact with). In this example, the set of actions available to these agents is: `stop`, `moveTo`, and `follow`. The `stop` action simply halts the involved agent, while the `moveTo` action requires a parameter, PoI, indicating the destination towards which the associated agent must move. The `follow` action can have one or two parameters: the mandatory one is the agent to be followed, and the optional second parameter specifies the target destination where the followed agent should bring the followers. A pattern consists of sequences, which dictate the order in which the following sub-sequences are executed by the agent. This element is crucial for enforcing synchronization stages across agents. A more formal explanation of these basic elements, including conditional statements and atomic predicates, is provided in [25].

Listing 1.1: LIrAs specification of the motivating example.

```
Human:
  1: moveTo poi1
  2: moveTo poi2
Robot:
  1: moveTo poi1 if position(Human,poi1)
                 else stop
  2: follow Human (poi2)
```

Firstly, the two agents must reach the `poi1`, but the `Robot` will start its movement only once the `Human` has reached the destination. Notice that the DSL introduced in [16] does not envisage a pre-determined pattern for independent movement as in this case, hence justifying the specification a LIrAs custom pattern for this example. This is reached by using an if/else conditional statement and the `position` atomic predicate. Once both agents have reached the destination, the `Human` starts moving towards the `poi2`; meanwhile, the `Robot` follows. This simple specification is captured in LIrAs as shown in Listing 1.1, which is parametric to all the elements concerning the operational environment.

5 Translating LIrAs patterns to SHA

This section firstly introduces the SHA structure for a LIrAs pattern, then presents the DFA model capturing the illustrative example, followed by the translation of the LIrAs code to SHA. Finally, it reports the results from the validation of the SHA network.

5.1 Hierarchical LIrAs Pattern Structure

The SHA will be structured using the three-layer hierarchical design of Figure 3, which thanks to its modularity is suitable for scenarios with different domains, and not only for HRI. This structure follows a top-to-bottom approach and consists of the Pattern Orchestrator, the Sequence Orchestrator, and the Action Set. These three categories comprise automata, with adjacent layers communicating through synchronized transitions, global variables, and guards.

At the top level of Figure 3, the Pattern Orchestrator oversees the pattern by synchronizing all the sub-sequences of the involved agents. The orchestrator provides the conditioned starting signal for the passage from one sub-sequence to the next, communicating via shared synchronized transitions with the layer below, the Sequence Orchestrator. Each sub-sequence i is associated with a channel, s_i; more precisely, in the Pattern Orchestrator the transition label is of the form $s_i!$ (sending), while in the Sequence Orchestrator, it is $s_i?$ (receiving). Once the Pattern Orchestrator provides the starting commands, it no longer intervenes in

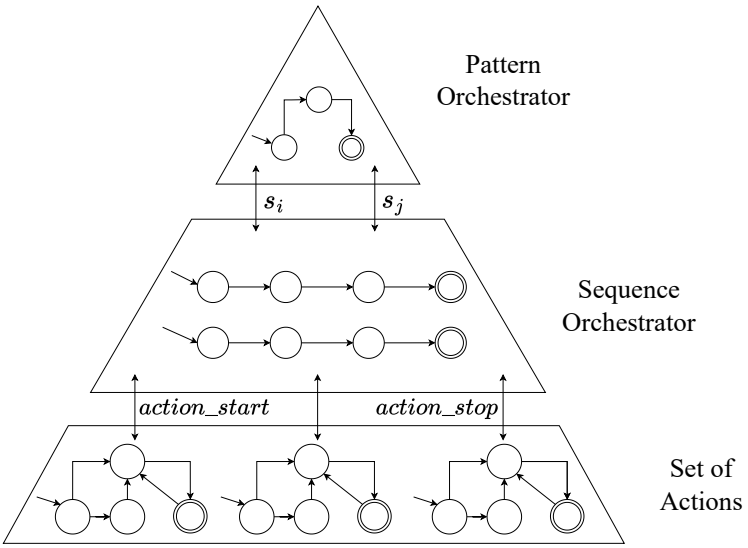


Fig. 3: Schematic representation of the three-layer hierarchical structure of the SHA in the LIrAs framework. The double arrows indicate the sharing of information between layers.

executing that sub-sequence. For example, in Listing 1.1, the Pattern Orchestrator allows Line 2 and Line 5 to start simultaneously (firing channel s_1), and once both end, it provides the starting signal for Line 3 and Line 7 (channel s_2). However, LIrAs is limited to series-parallel networks. While this might be true from the Pattern Orchestrator's perspective, the managed sub-sequences can incorporate choices, dependencies, and loops through conditional statements and atomic predicates, thus adding more flexibility to the DSL.

Each mission corresponds to one Pattern Orchestrator. The number of orchestrator locations is one for each sub-sequence associated with the pattern, plus an initial and a final location. In Listing 1.1, having two sub-sequences, the number of locations for the Pattern Orchestrator is four.

The middle layer of Figure 3 consists of as many automata as the number of agent sequences within the pattern; thus, there are only two for Listing 1.1. This layer forms the Sequence Orchestrator and communicates with the other two layers by sending and receiving synchronization signals, or by updating and checking global variables that indicate the execution status of actions and sub-sequences. In Listing 1.1, there are no sub-sequences with more than one action. However, if such sub-sequences exist, the Sequence Orchestrator must manage the start and stop conditions for the sequential execution of actions in the bottom layer. The only exception is in Line 5, where two actions are due to a conditional statement. In this case, the Sequence Orchestrator manages both the atomic predicate and the corresponding decision, as discussed in more detail in

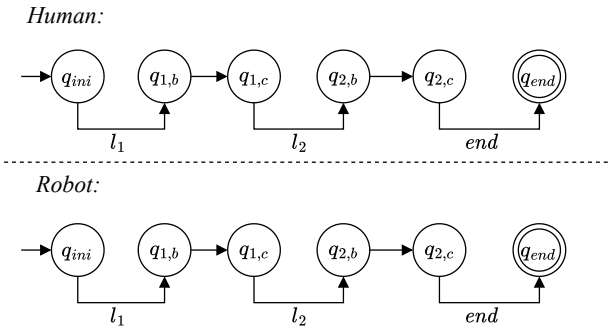


Fig. 4: DFA network of the example discussed in Section 4. Each action corresponds to two locations: $q_{1,b}$ when executing and $q_{1,c}$ when completed.

Section 5.3. LIrAs is currently limited to managing actions that do not include multitasking, meaning multiple actions cannot be executed simultaneously by the same agent. However, we plan to add this feature to the DSL with a suitable grammar.

Finally, the bottom layer of Figure 3 collects all the expert-defined SHA representing the tasks executable by each agent. In Listing 1.1, only three automata are required: one for the human `moveTo` action, and two for the robot `moveTo` and `follow` actions. The `stop` action does not require a dedicated automaton as it simply halts the robot by terminating the execution of an action or movement. The primary focus for this layer is understanding why and how it must communicate with the Sequence Orchestrator. It receives start and stop commands for initiating and concluding the execution of an action, whether at the natural end of the task or due to a conditional statement that necessitates an early interruption. Sensor measurements are passed from the action layers to the Sequence Orchestrator to determine when an action can end or to compute the values of atomic predicates. These considerations imply that the actions do have not a predetermined duration for their execution, but they depend on the passed parameters like the PoI to be reached during a `moveTo` action, and the measurement of sensors. The random time required by each action is compensated with the reprogrammable *action_end* locations [25] at the end of each sub-sequence in the Sequence Orchestrator, where we can set a default behavior to be followed while waiting for the next synchronization stages.

5.2 DFA Representation

The DFA in Figure 4 formalizes the synchronization among agents in the illustrative example. This does not include conditional statements, which involve physical variables and are, thus, only handled by the SHA model.

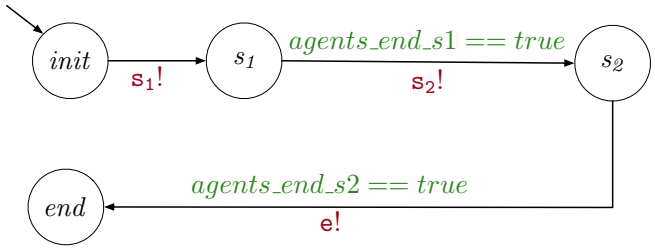


Fig. 5: Simplified automaton of the Pattern Orchestrator for the illustrative example in Section 4.

In Figure 4, both agents start from the initial location q_{ini}. From here, the transition labeled l_1 fires simultaneously, moving both agents to the busy location $q_{1,b}$ for the first action in the first sub-sequence (the `moveTo` actions in our example). After completing these actions, both agents transition to the complete locations $q_{1,c}$ for the first sub-sequences, as there is only one action in it. This sequence of events repeats for the second sub-sequences with the transition labeled l_2. A labeled transition can fire only if all agents sharing this label can do so simultaneously. This strategy ensures the synchronization requirements imposed by LIrAs in a multi-agent environment.

In the model of Figure 4, the following Timed Computation Tree Logic (TCTL) properties can be verified. As captured by Formula (1), it is verified that there exists a path such that eventually all agents reach the final state.

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

On the other hand, the property that, for all paths, eventually all agents reach the final state—captured by Formula (2)—is not verified for the network because an agent can remain indefinitely in any state, and not only in the q_{end} location.

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

5.3 SHA Representation

We now present the SHA[1] associated with the illustrative example from Section 4, one for each layer depicted in Figure 3.

The Pattern Orchestrator is depicted in the concise representation of Figure 5, managing transitions between sub-sequences in the layer below using labels $\mathtt{s}_1!$ and $\mathtt{s}_2!$ for the first and second sub-sequences, respectively. Guards, shown

[1] The complete Uppaal model associated with the illustrative example is available at https://zenodo.org/records/12667806.

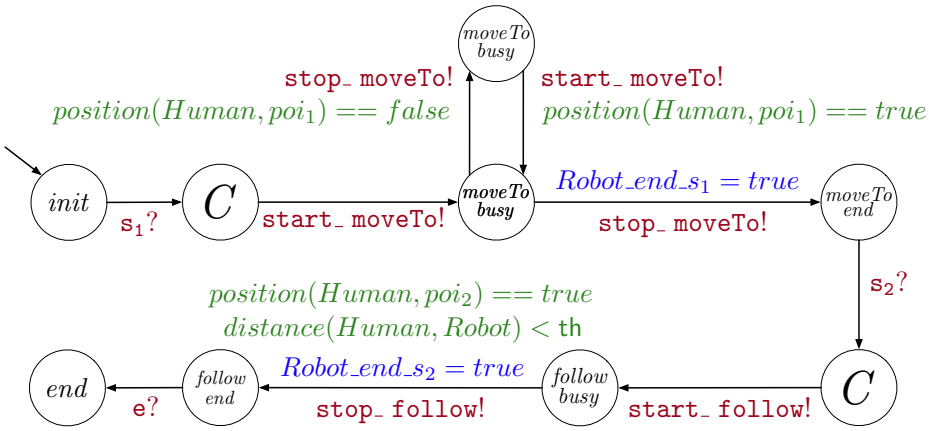


Fig. 6: Simplified automaton of the Sequence Orchestrator for the robot in the illustrative example in Section 4. Locations marked with C denote committed locations, which trigger instantaneously upon reaching them.

in green, impose conditions on these transitions to ensure that all agents complete their sub-sequences before progressing. In the example involving a human and a robot, each agent must await authorization from the Pattern Orchestrator upon completing a sub-sequence. This authorization is granted based on the Pattern Orchestrator's monitoring of all agents through global variables. These global variables are $a\_end\_s_i$, $\forall a \in Agents$ and $\forall s_i \in Subsequences$. Hence, in the example these variables are: $robot\_end\_s_1$, $robot\_end\_s_2$, $human\_end\_s1$ and $human\_end\_s_2$, which in Figure 5 are represented compactly through $agents\_end\_s_1$ and $agents\_end\_s_2$.

Figure 6 illustrates the automaton of the Sequence Orchestrator for the robotic agent. The number of locations in Figure 6 is determined as follows: two for the initial and final locations, two locations per action, two for the busy and end locations, one for each committed location (i.e., equivalent to having an invariant $t \leq 0$ on the location and a guard $t \geq 0$ for each exit edge from that location) per sub-sequence, and one for each `if/else` conditional statement. This layer synchronizes with the Pattern Orchestrator through transitions labeled with $\mathtt{s}_1?$ and $\mathtt{s}_2?$ to know when to initiate sub-sequences. It updates the global variable representing the agent's sub-sequence status—i.e., $Robot\_end\_s1 = true$ and $Robot\_end\_s2 = true$ (the blue text in Figure 6). These annotations indicate when an agent's sub-sequence concludes, and it is used by the Pattern Orchestrator to determine the starting commands for the next sub-sequence.

The robot's Sequence Orchestrator also communicates with the bottom layer of Figure 3 using labels `start_action!` and `stop_action!`. These labels respectively initiate and terminate the execution of a task. The execution process can

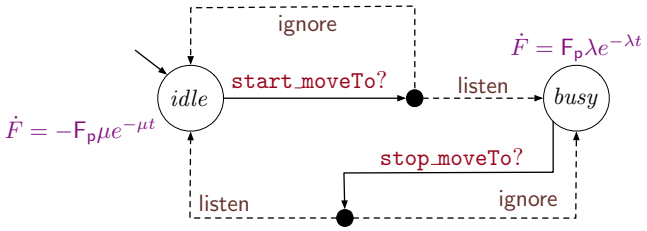


Fig. 7: Simplified SHA of the `moveTo` action for the human agent [16]. Dashed arrows represent probabilistic transitions, and purple equations denote differential equations used to compute the evolution of human fatigue.

be influenced by guards from Figure 6, which facilitate the completion of tasks and evaluate atomic predicates in conditional statements.

All the custom-defined skills for each agent in the pattern are consolidated in the bottom layer of Figure 3, labeled as Action Set. Figure 7 presents the core features of the SHA modeling of the human agent's `moveTo` action. We recall that LIrAs specifications define the orchestration of expert-defined SHA in the Action Set layer into a SHA network, so even if the orchestration itself can be described through DFA (see Section 5.2), the overall model is SHA.

The `moveTo` SHA includes features capturing human decision-making and physical variables related to human physiology. Specifically, the SHA includes probabilistic edges capturing the fact that, once a command is received by the Sequence Orchestrator (e.g., through label `start_moveTo`?), the human may choose to ignore it with weight $\mathsf{ignore} \in \mathbb{R}_{\geq 0}$ or abide by it with weight $\mathsf{listen} \in \mathbb{R}_{\geq 0}$. Each location models a specific human behavioral state. Specifically, locations differ based on the time dynamics of a physiological variable—i.e., human physical fatigue modeled through a real-valued variable F. Fatigue decreases gradually in the *idle* location, indicating rest, and increases during movement phases (i.e., while in *busy*). Each location is, thus, labeled with a flow condition constraining the evolution of F while in such a location. In this specific case, flow conditions $\dot{F} = -\mathsf{F_p}\mu e^{-\mu t}$ and $\dot{F} = \mathsf{F_p}\lambda e^{-\lambda t}$ exploit Merletti et al.'s model of physical fatigue [19], where $\mu, \lambda \in \mathbb{R}_{\geq 0}$ represent the recovery and fatigue rates, respectively, and $\mathsf{F_p} \in \mathbb{R}_{\geq 0}$ is the residual fatigue from the previous cycle. Interested readers may refer to [16] for the description of the human action's SHA.

This work's specific action set selection is due to the contextualization of LIrAs within the specific target framework. We recall that LIrAs is agnostic concerning the application domain and agents' skill sets and that the expressiveness of SHA extends to any generic agent with comparable features (e.g., choices subject to uncertainty and physical variables with non-linear time dynamics).

5.4 Validation

In the following, we present the results from the validation of the SHA network, obtained through the manual translation from LIrAs to automata using

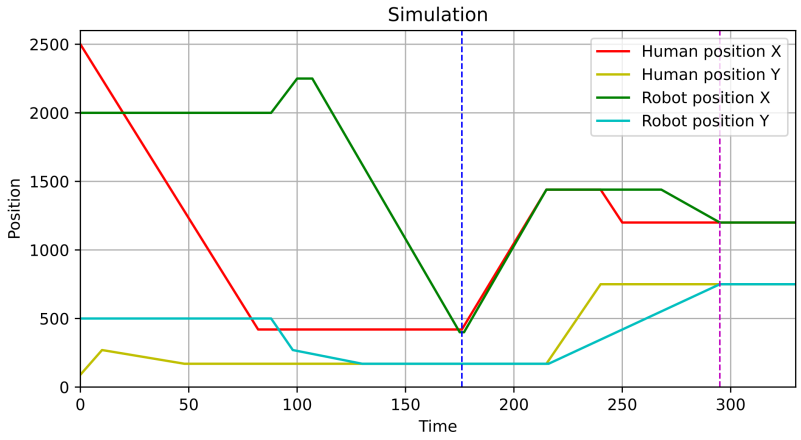


Fig. 8: Simulation of agents' behavior captured by Listing 1.1.

the Uppaal tool. This translation, based on the described methodology, aims to assess the feasibility of automating the entire process. Additionally, translating into SHA is necessary to analyze a pattern specified with LIrAs and verify SHA properties that provide relevant information about the specification. This analysis is performed for simplicity just for the illustrative example; other examples, which show similar results, are reported in [25]. Specifically, Figure 8 illustrates a simulation run to demonstrate the behavior of the full system, followed by the results of the SMC query.

The simulation starts by executing the first sub-sequence, having the two agents in different locations. For the first 85 units of time, where a time unit corresponds to a second, the robotic agent is not moving, due to the `if/else` conditional statement, until the human does not reach the first PoI. After that, the human concludes the first sub-sequence and the robot starts moving. When the robot reaches the first PoI, too (around $175s$), the first sub-sequence ends also for it. Now the Pattern Orchestrator is allowed to provide the start for the second sub-sequence, corresponding to the blue dashed line in the simulation of Figure 8, where the robot starts the `follow` action until both agents reach the second PoI. After that, at around time 300 (marked with the pink dashed line), the pattern concludes having reached the final location in all the orchestrator automata.

Thanks to the Uppaal tool we use SMC to compute the probability of desired properties holding. Expression $\mathbb{P}_M(\psi)$ amounts to a confidence interval for the probability of ψ holding for M, which Uppaal computes through the Clopper-Pearson method [7]. Specifically, given hyperparameter ϵ, Uppaal concludes the verification experiment when the Clopper-Pearson confidence interval is smaller than ϵ. All results reported in the following are run with default value $\epsilon = 0.05$.

Therefore, the highest confidence interval that can be obtained (approximating to near certainty of property ψ holding) is $[0.95, 1]$.

The first properties checked are $\mathbb{P}_M(\diamond_{\leq 300}\ o.\text{end})$, $\mathbb{P}_M(\diamond_{\leq 300}\ h.\text{end})$, and $\mathbb{P}_M(\diamond_{\leq 300}\ r.\text{end})$; they correspond to the probability for the Pattern Orchestrator (o), the Sequence Orchestrator for the `Human` (h), and the one for the `Robot` (r), respectively, to reach the final location within $300s$. The resulting confidence interval is [0.95, 1], signifying the likelihood of these properties holding for M. We use the same time bound ($300s$) for all three properties because the shared synchronization label and status guards lead to the synchronous firing of the three terminal locations.

Since the probability $\mathbb{P}_M(\diamond_{\leq 175}\ o.\text{start}_{s2})$ has a confidence interval ranging from 0.95 to 1, we can assert that the first sub-sequence concludes within 175 seconds with near certainty. Given that the entire specification, which has two sub-sequences, completes within 300 seconds, it is apparent that $\texttt{s}_1$ and $\texttt{s}_2$ have similar durations. The first sub-sequence is slower between the two due to the idle action introduced by the `if/else` conditional statement in the robotic agent's sub-sequence, as seen in Lines 5 and 6 of Listing 1.1.

Lastly, since we do not want an automaton to reach the *end* location while the others do not, we can check the probability $\mathbb{P}_M((\diamond_{\leq 300}\ (o.\text{end} \wedge \neg h.\text{end}) \vee (o.\text{end} \wedge \neg r.\text{end}))$ is violated. It represents the probability of reaching the terminal location in the Pattern Orchestrator while not in the Sequence Orchestrators (for both the `Human` and the `Robot`). The resulting confidence interval in this case is [0, 0.05], stemming from the guards and synchronization in the last transition, being almost zero we can assert that this property, as desired, is violated.

6 Conclusions and Future Developments

This paper builds upon the toolchain relying on LIrAs (whose syntax and semantics are presented in [25]) by demonstrating the translation of LIrAs specifications to SHA through an illustrative example and showcasing how SMC applies to the resulting formal model.

Through this translation process, we can derive a complex formal model based on SHA from a high-level specification with LIrAs, which would otherwise require an expert to generate. This model can then be verified with Uppaal by checking properties as reported in Section 5 and others. The validation presented in Section 5 demonstrates that our model efficiently represents the specification created with LIrAs, showing the probability of satisfying the desired properties.

Although the translation is currently performed manually, we plan to automate it in the future by leveraging the three-layer hierarchical structure presented in Section 5.1. This will simplify the validation of this approach on more examples. Additionally, we intend to extend this validation analysis to different fields to ensure the domain-agnostic capabilities of LIrAs in MAS applications. Finally, we aim to conduct a user study to assess the user-friendliness and clarity of LIrAs for non-expert users.

References

1. Agha, G., Palmskog, K.: A survey of statistical model checking. ACM Transactions on Modeling and Computer Simulation (TOMACS) **28**(1), 1–39 (2018)
2. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical computer science **126**(2), 183–235 (1994)
3. Alur, R., Feder, T., Henzinger, T.A.: The benefits of relaxing punctuality. Journal of the ACM (JACM) **43**(1), 116–146 (1996)
4. Bersani, M.M., Soldo, M., Menghi, C., Pelliccione, P., Rossi, M.: PuRSUE-from specification of robotic environments to synthesis of controllers. Formal Aspects of Computing **32**, 187–227 (2020)
5. Bolton, M.L., Siminiceanu, R.I., Bass, E.J.: 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), 961–976 (2011)
6. Bozhinoski, D., Di Ruscio, D., Malavolta, I., Pelliccione, P., Tivoli, M.: Flyaq: Enabling non-expert users to specify and generate missions of autonomous multicopters. In: 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE). pp. 801–806. IEEE (2015)
7. Clopper, C.J., Pearson, E.S.: The use of confidence or fiducial limits illustrated in the case of the binomial. Biometrika **26**(4), 404–413 (1934)
8. David, A., Larsen, K.G., Legay, A., Mikučionis, M., Poulsen, D.B.: Uppaal smc tutorial. International journal on software tools for technology transfer **17**, 397–415 (2015)
9. David, A., Larsen, K.G., Legay, A., Mikučionis, M., Poulsen, D.B., Van Vliet, J., Wang, Z.: Statistical model checking for networks of priced timed automata. In: Formal Modeling and Analysis of Timed Systems: 9th International Conference, FORMATS 2011, Aalborg, Denmark, September 21-23, 2011. Proceedings 9. pp. 80–96. Springer (2011)
10. Dragule, S., Gonzalo, S.G., Berger, T., Pelliccione, P.: Languages for specifying missions of robotic applications. Software Engineering for Robotics pp. 377–411 (2021)
11. Feo-Arenis, S., Vujinović, M., Westphal, B.: On implementable timed automata. In: International Conference on Formal Techniques for Distributed Objects, Components, and Systems. pp. 78–95. Springer (2020)
12. Forbrig, P., Bundea, A.N.: Modelling the collaboration of a patient and an assisting humanoid robot during training tasks. In: Intl. Conf. HCII. pp. 592–602. Springer (2020)
13. García, S., Pelliccione, P., Menghi, C., Berger, T., Bures, T.: High-level mission specification for multiple robots. In: ACM SIGPLAN Intl. Conf. on Software Language Engineering. pp. 127–140 (2019)
14. Lacerda, B., Lima, P.: 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 (2009)
15. Larsen, K.G., Pettersson, P., Yi, W.: UPPAAL in a nutshell. Int. J. on Softw. Tools for Tech. Transf. **1**(1-2), 134–152 (1997)
16. Lestingi, L., Zerla, D., Bersani, M.M., Rossi, M.: Specification, stochastic modeling and analysis of interactive service robotic applications. Robotics and Autonomous Systems **163**, 104387 (2023)

17. Levine, S.J., Williams, B.C.: Watching and acting together: Concurrent plan recognition and adaptation for human-robot teams. Journal of Artificial Intelligence Research **63**, 281–359 (2018)
18. Menghi, C., Tsigkanos, C., Pelliccione, P., Ghezzi, C., Berger, T.: Specification patterns for robotic missions. IEEE Transactions on Software Engineering (2019)
19. Merletti, R., Conte, L.L., Orizio, C.: Indices of muscle fatigue. Journal of Electromyography and Kinesiology **1**(1), 20–33 (1991)
20. Nordmann, A., Hochgeschwender, N., Wrede, S.: A survey on domain-specific languages in robotics. In: International conference on simulation, modeling, and programming for autonomous robots. pp. 195–206. Springer (2014)
21. Paterno, F., Mancini, C., Meniconi, S.: ConcurTaskTrees: A diagrammatic notation for specifying task models. In: Intl. Conf. on Human-Computer Interaction. pp. 362–369. Springer (1997)
22. Ruscio, D.D., Malavolta, I., Pelliccione, P., Tivoli, M.: Automatic generation of detailed flight plans from high-level mission descriptions. In: Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems. pp. 45–55 (2016)
23. Salimifard, K., Wright, M.: Petri net-based modelling of workflow systems: An overview. European journal of operational research **134**(3), 664–676 (2001)
24. Silva, D.C., Abreu, P.H., Reis, L.P., Oliveira, E.: Development of a flexible language for mission description for multi-robot missions. Information Sciences **288**, 27–44 (2014)
25. Tagliaferro, A., Lestingi, L., Rossi, M.: Towards verifiable multi-agent interaction pattern specification. In: Intl. Conf. on Formal Methods in Software Engineering. pp. 122–126. ACM (2024)
26. Tumova, J., Dimarogonas, D.V.: Multi-agent planning under local ltl specifications and event-based synchronization. Automatica **70**, 239–248 (2016)
27. Van, T.N., Fredivianus, N., Tran, H.T., Geihs, K., Huynh, T.T.B.: Formal verification of ALICA multi-agent plans using model checking. In: Intl. Symp. on Information and Communication Technology. pp. 351–358 (2018)
28. Van Der Aalst, W.M., Ter Hofstede, A.H.: YAWL: yet another workflow language. Information systems **30**(4), 245–275 (2005)