Specification, Stochastic Modeling and Analysis of Interactive Service Robotic Applications

Livia Lestingi[a,*], Davide Zerla[a], Marcello M. Bersani[a], Matteo Rossi[b]

[a] *Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Via Ponzio 34/5, 20133 Milan, Italy.*
[b] *Dipartimento di Meccanica, Politecnico di Milano, Via Privata Giuseppe La Masa 1, 20156 Milan, Italy.*

Abstract

Assistive robotic systems are quickly becoming a core technology for the service sector as they are understood capable of supporting people in need of assistance in a wide variety of tasks. This step poses a number of ethical and technological questions. The research community is wondering how service robotics can be a step forward in human care and aid, and how robotics applications can be realized in order to put the human role at the forefront. Therefore, there is a growing demand for frameworks supporting robotic application designers in a "human-aware" development process. This paper presents a model-driven framework for analyzing and developing human-robot interactive scenarios in non-industrial settings with significant sources of uncertainty. The framework's core is a formal model of the agents at play—the humans and the robot—and the robot's mission, which is then put through verification to estimate the probability of completing the mission. The model captures non-trivial features related to human behavior, specifically the unpredictability of human choices and physiological aspects tied to their state of health. To foster the framework's accessibility, we present a verification tool-agnostic Domain-Specific Language that allows designers lacking expertise in formal modeling to configure the interactive scenarios in a user-friendly manner. We compare the formal analysis outputs with results obtained by deploying benchmark scenarios in the physical environment with a real mobile robot to assess whether the formal model adheres to reality and whether the verification results are accurate. The entire development pipeline is then tested on several scenarios from the healthcare setting to assess its flexibility and effectiveness in the application design process.

1. Introduction

Breakthrough technological advancements are shaping the future of the service sector. Innovations brought by the phenomenon known as Industry 4.0, such as IoT, pervasive sensorization, Cloud Computing, and Collaborative Robotics, are now spreading to non-industrial settings with significant projected impacts on our everyday lives. Most importantly, highly sophisticated robotic systems under development today are bound to transform the job market once they become commercially available. The uptake of such solutions poses a number of problems which range from technological challenges to ethical and societal implications. A recent study on the future of employment indeed estimates that specific jobs, such as receptionists, information clerks, healthcare support workers, and personal care aides, will be taken over by robots with probabilities ranging from 60% to 90% [1]. In addition, the presence of robots in healthcare has increased in recent years and shows an accelerating trend [2]. The use and penetration of robotics for human care and aid is evidenced by the presence European calls and projects[1], technology companies[2], and market analysis reports [3, 4].

*Corresponding author
Email addresses: livia.lestingi@polimi.it (Livia Lestingi), davide.zerla@mail.polimi.it (Davide Zerla), marcellomaria.bersani@polimi.it (Marcello M. Bersani), matteo.rossi@polimi.it (Matteo Rossi)

[1] Examples are the Harmony (https://harmony-eu.org) and the EnrichME (https://cordis.europa.eu/project/id/643691) projects.
[2] Examples are Kompai Robotics (https://kompairobotics.com) and Labrador Systems (https://labradorsystems.com).

All these initiatives and companies agree that the use of robots in care can increase the quality of services provided, although robots are not a substitute for humans, but a tool to improve their actions. For instance, a study by the American Nurses Association [5] showed that robots can support and augment nursing care delivery, improves nurse productivity, increases time with patients, encourages positive emotional responses. Despite these evidences, the investigation of the extent of such a technological and societal shift is an ongoing research. This work attempts to answer the question on the feasibility of such a step by addressing the analysis from the software engineering standpoint and, in particular, sheds light on the development of collaborative service robot applications in healthcare.

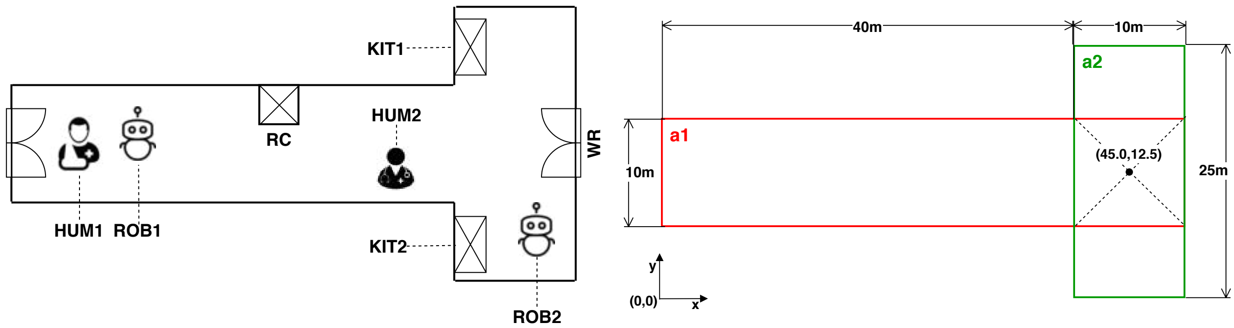
State-of-the-art technologies dealing with sensing, manipulation, and reasoning capabilities make it feasible for robots to perform complex jobs. Nowadays, a robot may be adequately equipped to sense multiple aspects of its surroundings, efficiently detect obstacles, grasp and manipulate fragile objects, perform surgery, and make decisions in delicate situations. However, these skills usually constitute silos of software, whose integration and reuse are challenging tasks. The EFFIROB project [6], which analyzed the profitability of developing a new service robot application, has estimated that up to 80% of the total cost comes from software development and maintenance.

More generally, software engineering techniques for robotics are not mature yet to handle the complexity and changeability of service settings [7]. Service robots operate in unconstrained environments where humans, who they frequently interact with, constitute a significant source of uncertainty. Decisions made at an early design stage of the application determine up to 90% of the overall life-cycle costs [8], and considerable sources of uncertainty can hinder their validity. Therefore, it is of paramount importance to provide designers with frameworks to develop applications that are simultaneously **reliable** and **flexible** with respect to the variability of the environment [9]. Frameworks should also limit the gap between the developer's knowledge and the prerequisites needed to access them, removing the barriers that are due to the lack of specialized skills in the developers.

1.1. Model-Driven Framework

Designing robotic applications to be deployed in delicate environments where robots will closely interact with humans is a challenging task, requiring a strong technical background both in robotics and software design. Our work contributes to this line of research by proposing a **model-driven framework** to develop interactive service robot applications. Target *users* of the framework, called hereafter robotic application designers (or, simply, application designers), are professional figures managing the logistics of service facilities where robotic applications will be deployed, such as clinical workflow analysts [10]. The framework targets robotic applications set in known layouts, featuring a wheeled mobile robot and one (or multiple) humans requesting a service that requires interaction or coordination with the robot. While the geometry of the layout is known, humans are a source of uncertainty as they may make unpredictable choices and stray from the plan while interacting with the robot. Applications eligible for analysis come, though not exclusively, from the healthcare and assisted living settings, where people might be in pain or discomfort. Therefore, the development process encompasses features of human **physiological** (i.e., physical fatigue) and **behavioral** aspects, such as the unpredictability of the human decision-making process.

Within the scope of the framework, interactions between a human and a robot conform to high-level "*patterns*" identifying recurring contingencies in assistive applications. Throughout the paper, we use term "*action*" to indicate an "*atomic behaviour that is executed by any actor in a scene*" [11]. Each interaction pattern is a sequence of actions (e.g., move until a certain event occurs, stop, wait for the human to be close). Although there is no standard definition of robotic "*mission*", with this term we refer to a sequence of interaction patterns identifying the desired behavior of the robot [12] performed in a specific layout. A sequence of missions constitutes a Human-Robot Interaction (HRI) "*scenario*". Hence, in the scope of this work, we understand a robotic application as the realization of a scenario through real or virtual agents. The framework exploits formal analysis to provide the robotic application designer with reliable insights into the outcome of each mission (each analyzed individually) constituting the scenario. Given the initial configuration of a scenario (e.g., positions of the agents, battery charge), the application designer receives an estimation of how likely the associated missions are to end in *success* (dually, in *failure*) and the physical effort each mission imposes on human subjects.



(a) Agents (represented in their starting locations) and POIs of the running example.

(b) Layout for the running example, highlighting the two areas, their dimensions ([m]), and the intersection point.

Figure 1: Graphical representation of the example scenario configuration.

Example. Fig. 1 shows the setup of a possible scenario. The layout is a T-shaped corridor made up of two rectangular areas (see Fig. 1b): a horizontal one and a perpendicular vertical one, whose intersection is centered in point $(45.0, 12.5)$. The corridor features four Points Of Interest (POIs), i.e., significant locations within the layout, also represented in Fig. 1a: the robot's recharge station (RC), two cupboards containing medical kits (referred to as KIT1 and KIT2), and the door leading to the waiting room (WR). There are four agents in the scenario: two humans (HUM1 and HUM2) and two robots (ROB1 and ROB2). Robot ROB1 is a Tiago[3] with initial battery charge equal to 40% of the total capacity and ROB2 is a Turtlebot3 WafflePi[4] with 90% of the total capacity. HUM1 is a young patient with average walking speed 80cm/s while HUM2 is a healthy elderly doctor with average walking speed 100cm/s. The designer assesses two alternative missions to determine which one is most likely to succeed within a given time bound and with no harm for the humans: the first mission features ROB1 leading HUM1 to the waiting room, then delivering KIT2 to HUM2. The second mission features ROB2 following HUM2 to fetch KIT1, then leading HUM1 to the waiting room.

The framework's workflow (shown in Fig. 2) is structured into three phases:

(1) **design-time analysis**: the robotic application designer configures the scenario through a specification language. Starting from the configuration, a formal model of the scenario is automatically generated together with a set of properties. Such properties are subject to verification to estimate quality measures of the scenario (e.g., the probability of success);

(2) **deployment**: when the design-time results are deemed acceptable, the application designer deploys the scenario either in a physical environment or simulated environment; to enable deployment, the formal model is converted into executable code communicating with the deployment environment through a middleware layer;

(3) **reconfiguration**: the application designer examines quality metrics of the scenario estimated from data collected during deployment and applies reconfiguration measures, if necessary (e.g., in the first mission of the example, ROB2 might be deployed instead of ROB1 because of the higher charge level).

1.2. Contributions

This paper builds upon the results presented in [13, 14, 15, 16] by presenting:

1. A custom **Domain-Specific Language** (DSL) for scenario configuration. Since application designers possibly lack a strong background in software development, formal modeling, or formal analysis, the DSL, which is a lightweight textual notation, provides a friendly interface to the configuration phase.

[3] Technical specifications available at: https://pal-robotics.com/robots/tiago/.
[4] Technical specifications available at: https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/.

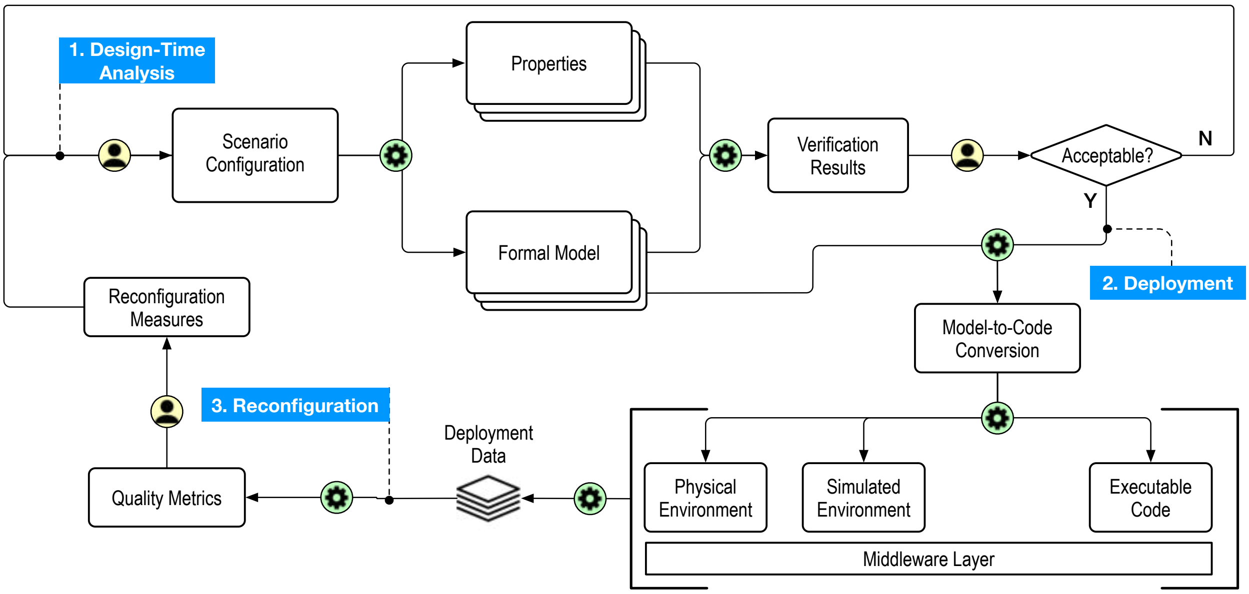


Figure 2: Diagram representing the model-driven framework operational workflow. Yellow circles mark actions performed by the *user* (i.e., the application designer) and green circles correspond to the *automated* tasks. The beginning of each phase of the framework is marked in blue and numbered according to the execution order.

2. A modeling pattern to incorporate a **stochastic** characterization of fatigue into the formal model of human behavior. In [13, 14], we presented an early version of the model formalizing an interactive robotic scenario as a Network of Hybrid Automata, with probabilistic edges capturing human choices made out of free will. This paper details the refined version of the formal model as a Network of **Stochastic Hybrid Automata** (SHA). Specifically, automata capturing human agents are endowed with probability distributions characterizing random parameters of the fatigue phenomenon, incorporating into the formal analysis the physiological variability between individuals belonging to different age groups or in different states of health. This source of variability (only briefly mentioned in [16]), in addition to the probabilistic characterization of human behavior, is accounted for by the formal analysis performed through **Statistical Model Checking** (SMC) via the Uppaal tool [17].

3. A model of the robot's **battery** charge and discharge dynamics refined (compared to [13, 14]) to fit the behavior of the robotic agent used for the experimental validation. The model fitting procedure increases the accuracy of the SHA modeling the battery when compared against field observations.

4. An extensive **experimental validation** to assess whether the developed formal model adheres to reality. Experimental scenarios are built by using elements that recur in 24 real-world exemplars existing in the literature and addressing service robotics in healthcare. Design-time estimations are compared with data collected by deploying benchmark applications implementing a Digital-Twin architecture. The validation activity aimed at assessing the accuracy of the formal model and the SMC results when deploying the application in a physical environment with a real robotic platform and, if necessary, with virtual human agents controlled by a real operator. We exploit a statistical technique based on Clopper-Pearson confidence intervals [18] to estimate the mission success probability range observed in reality and critically compare such results with those obtained by performing SMC experiments. Furthermore, we illustrate the application of the whole model-driven development framework to case studies capturing assistive tasks in a healthcare setting to assess how it supports the application designer from early design to reconfiguration.

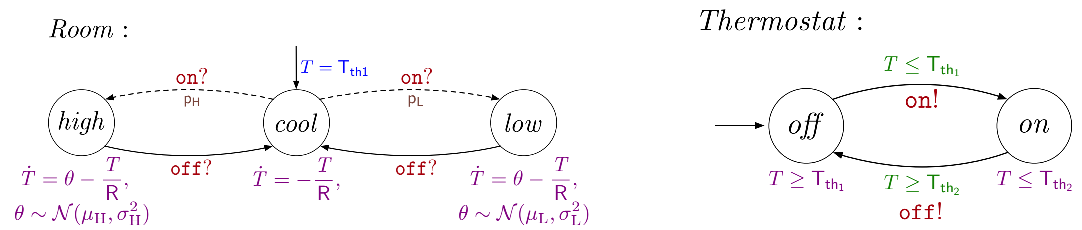
The paper is structured as follows: Section 2 illustrates the theoretical background of the work; Section 3 illustrates the design-time analysis phase of the framework, specifically introducing the conceptual model of the framework's domain and the DSL; Section 4 presents the refined SHA Network; Section 5 illustrates the deployment and reconfiguration phases of the framework; Section 6 presents and discusses the experimental validation results; Section 7 surveys related works in the literature; Section 8 concludes. For the interested reader, Appendix A reports a detailed presentation of SHA semantics, while DSL specifications used for the experimental validation are reported in Appendix B.

2. Background

This section illustrates the pre-existing theoretical concepts constituting the foundation of our work. Specifically, we provide a formal definition of Stochastic Hybrid Automata (SHA) and illustrate their features through a running example inspired by [17, Section 4]. We adopt SHA to model the robotic application, as SHA can capture complex temporal dynamics of physical phenomena, such as the fatigue of human agents and the battery charge/discharge for the robots, but also the digital aspects of the application, such as its operating state and logical behavior evolving over the time.

Example 2.1. The example captures a system composed by a room with a heating system, whose model is shown in Fig. 3a, and the thermostat controlling its temperature, shown in Fig. 3b. The room temperature is the main physical phenomenon of the system, which is modeled by real variable T in the automaton in Fig. 3a. The thermostat is modeled through two different operating states *on* and *off*, and it evolves by alternating states *on*, which makes the room warmer, and *off*, thus, letting the room temperature decrease naturally. When the thermostat is *off*, as soon as temperature T decreases below a threshold $\mathsf{T}_{\mathsf{th}_1}$, hence the condition $T \geq \mathsf{T}_{\mathsf{th}_1}$ labeling location *off* does not hold (resp., exceeds a threshold $\mathsf{T}_{\mathsf{th}_2}$, hence the condition $T \leq \mathsf{T}_{\mathsf{th}_2}$ labeling location *on* does not hold), the thermostat switches the heating on (resp., off). The triggering of the event, indicated with symbol `on`!, makes the thermostat modify its operating state, hence moving to *on* (resp., deactivate the heating, hence moving to *off*). The room temperature is modeled in three different situations, that are represented by the automaton in Fig. 3a: the one for which the temperature decreases due to the absence of heating, i.e., *cool*, and two situations for which the temperature increases at different rates, i.e., *high* and *low*, when the heating is on. The temperature grows according to differential equation $\dot{T} = \theta - \frac{T}{\mathsf{R}}$ when the thermostat is on and decreases according to $\dot{T} = -\frac{T}{\mathsf{R}}$ when it is off, where R is a constant and θ is a randomly distributed parameter, i.e., whose value depends on a probability distribution. At the onset of the system, the thermostat is off, hence the room is cooling down, and the temperature is conventionally initialized with value $\mathsf{T}_{\mathsf{th}_1}$. This value allows the thermostat to spend a non null amount of time in location *cool*, where constraint $T \geq \mathsf{T}_{\mathsf{th}_1}$ limits the temperature inferiorly. The initialization of T is realized by the update on the edge entering location *cool*. When event `on`! (resp., `off`!) is fired by the thermostat, the room simultaneously reacts to it; hence, both the thermostat and the room modify their state at the same time, i.e., they synchronize. Reacting to the event, indicated with symbol `on`? (resp., `off`?), causes the room temperature to rise and the automaton to change location to either *high* or *low* (resp., the room temperature to decrease and the automaton to move to *cool*). The room can be heated at a high or low rate (e.g., if a window is closed or open, respectively): the choice is made probabilistically when the automaton synchronizes with event `on`!. The probability weights are known and amount to p_{H} and p_{L}, respectively. Parameter θ in Fig. 3a is a realization of a Normal distribution with mean μ_{H} and standard deviation σ_{H} (indicated as $\mathcal{N}(\mu_{\mathrm{H}}, \sigma_{\mathrm{H}}^2)$) when the room is heating quickly because the window is closed (the subscript "H" stands for "high" rate of heating). Conversely, when the room is heating slowly because the window is open, the probability distribution is $\mathcal{N}(\mu_{\mathrm{L}}, \sigma_{\mathrm{L}}^2)$ (subscript "L" stands for "low" rate of heating). Throughout the paper, we express that a random parameter θ is a realization of random variable Θ governed by distribution $\mathcal{N}(\mu, \sigma)$ through notation $\theta \sim \mathcal{N}(\mu, \sigma)$.

Thorough investigation on SHA is given in the following [19, 20, 21]. Let Z be a set of symbols; we indicate with $\Gamma(Z)$ the set of conjunctions of constraints of the form $\chi_1 \sim \chi_2$, where $\sim$ is a relation in $\{<, =\}$ and χ_i $(i \in \{1, 2\}\})$ is an arithmetical term defined by the sum of the elements in Z and $\mathbb{N}$ (e.g., $z_1 + z_2 + 3$, with $z_1, z_2 \in Z$). By definition, $\Gamma(Z)$ includes the logical constants `true` and `false`, defined as trivially true formulae (e.g., $0 = 0$) or trivially false formulae (e.g., $0 = 1$). We indicate with $\Xi(Z)$ the set of updates on



(a) Room SHA model. The SHA receives events from the thermostat to start heating at different rates (locations *high* and *low*) or cooling naturally (location *cool*).

(b) Thermostat SHA model. The SHA is in charge of triggering the events that cause the room to start (event `on`) or stop heating (`off`).

Figure 3: Example of SHA network. Dashed arrows model probabilistic transitions with weights (in brown) p_H and p_L and solid arrows represent transitions with weight 1. Flow conditions, probability distributions, and exponential rates are in purple, channels in red, and edge conditions in green, respectively.

elements of Z. An *update* in $\Xi(Z)$ (for example, $z' = z + 2$) is a constraint where free variables are elements of Z (e.g., $z \in Z$) and of its primed version Z' (e.g., $z' \in Z'$). We indicate the set of non-negative real numbers with $\mathbb{R}_+$ and, with $\mathbb{R}^Z$, the set of assignments to variables of Z (i.e., *valuations*).

Definition 1. *A Stochastic Hybrid Automaton is a tuple $\langle L, W, \mathcal{F}, \mathcal{D}, \mathcal{I}, C, \mathcal{E}, \mu, \mathcal{P}, l_{\text{ini}} \rangle$, where:*

1. *L is the set of* ***locations*** *and $l_{\text{ini}} \in L$ is the initial location;*
2. *W is the set of real-valued* ***variables*** *of which clocks $X \subseteq W$, dense-counter variables $V_{\text{dc}} \subseteq W$, and constants $K \subseteq W$ are subsets;*
3. *$\mathcal{F} : L \to \{(\mathbb{R}_+ \cup (\mathbb{R}_+ \times \mathbb{R})) \to \mathbb{R}^W\}$ is the function assigning a set of* ***flow conditions*** *to each location, where flow conditions are (continuous) functions with one or two parameters, which assign a valuation to every time instant in $\mathbb{R}_+$ or to every pair constituted by a time instant in $\mathbb{R}_+$ and a constant value in $\mathbb{R}$, depending on the location;*
4. *$\mathcal{D} : L \rightharpoonup \{\mathbb{R} \to [0,1]\}$ is the* partial *function assigning a* ***probability distribution*** *from $\{\mathbb{R} \to [0,1]\}$ to locations which feature flow conditions with two parameters;*
5. *$\mathcal{I} : L \to \Gamma(W)$ is the function assigning a (possibly empty) set of* ***invariants*** *to each location;*
6. *C is the set of* ***channels****, including the internal action ϵ;*
7. *$\mathcal{E} \subset L \times C_{!?} \times \Gamma(W) \times \wp(\Xi(W)) \times L$ is the set of* ***edges****, where $C_{!?} = \{\mathsf{c}! \mid \mathsf{c} \in C\} \cup \{\mathsf{c}? \mid \mathsf{c} \in C\}$ is the set of* events *involving channels in C. Given an edge $(l, c, \gamma, \xi, l') \in \mathcal{E}$, l (resp. l') is the outgoing (resp. ingoing) location, c is the* edge event*, γ is the* edge condition *and ξ is the* edge update*. For each $l \in L$, $\mathcal{E}(l) \subseteq C_{!?} \times \Gamma(W) \times \wp(\Xi(W)) \times L$ is the set of edges outgoing from l (for each $(c, \gamma, \xi, l') \in \mathcal{E}(l)$ then $(l, c, \gamma, \xi, l') \in \mathcal{E}$ and viceversa);*
8. *$\mu : (L \times \mathbb{R}^W) \to \{\mathbb{R}_+ \to [0,1]\}$ is the function assigning a* ***probability distribution*** *from $\{\mathbb{R}_+ \to [0,1]\}$ to each* configuration *of the SHA, where configurations are (l, v_{var}) pairs constituted by a location $l \in L$ and a valuation $v_{\text{var}} \in \mathbb{R}^W$;*
9. *$\mathcal{P} : L \rightharpoonup \{(C_{!?} \times \Gamma(W) \times \wp(\Xi(W)) \times L) \rightharpoonup [0,1]\}$ is the* partial *function assigning a discrete* ***probability distribution*** *from $\{(C_{!?} \times \Gamma(W) \times \wp(\Xi(W)) \times L) \rightharpoonup [0,1]\}$ to locations such that, for each $l \in L$, $\mathcal{P}(l)$ is defined if, and only if, $\mathcal{E}(l)$ is non-empty; also, the domain of the distribution is $\mathcal{E}(l)$ (hence, $\sum_{\substack{\alpha=(\mathsf{c}!,\gamma,\xi,l') \in \mathcal{E}(l) \\ \mathsf{c} \in C}} \mathcal{P}(l)(\alpha) = 1$ holds).*

In SHA, real-valued variables (i.e., a generalization of clocks) can evolve in time according to generic expressions referred to as *flow conditions* [19]. The flow conditions constraining the evolution over time of variables in W are defined through sets of Ordinary Differential Equations (ODEs). This feature makes SHA a suitable formalism to model systems with complex dynamics, as it is possible to model through flow conditions, for example, laws of physics or biochemical processes. ODEs constraining clocks (for which $\dot{x} = 1$ holds for all $x \in X$), dense-counter variables, and constants (where $\dot{v} = 0$ holds for all $v \in V_{\mathrm{dc}} \cup K$) are special cases of flow conditions.

If a variable $\theta \in V_{\mathrm{dc}}$ is an independent term for a flow condition $f \in \mathcal{F}(l)$ on location $l \in L$, i.e., $f = f(t, \theta)$, and θ is interpreted as a randomly distributed parameter, then f is a **stochastic process** [22]. We limit the analysis to flow conditions depending on *at most one* random parameter, as per Definition 1, which is enough to model human-robot interaction within the scope of our work. For example, in the SHA shown in Fig. 3a the room temperature is modeled by real-valued variable $T \in W$. When temperature is decreasing, it is constrained by the flow condition $\dot{T}(t) = -T(t)/\mathrm{R}$, where function $\dot{T}(t) \in \mathcal{F}(\mathit{cool})$ depends on time only and has solutions in $\mathbb{R}$. When temperature is increasing, it evolves according to flow condition $\dot{T}(t, \theta) = \theta - T(t, \theta)/\mathrm{R}$, depending both on time and random parameter θ. The domain of $\dot{T}(t, \theta)$ is, thus, $\mathbb{R}_+ \times \mathbb{R}$ and its solutions belong to $\mathbb{R}$. Interested readers are referred to Appendix A for a detailed presentation of SHA semantics.

SHA are eligible for Statistical Model Checking (SMC) [23]. SMC requires a model M with stochastic features (the SHA network), and a property ψ expressed, in our case, in Metric Interval Temporal Logic (MITL) over atomic propositions, belonging to set AP [24], which represent, for instance, constraints over set W (e.g., $w < 10$), or automata locations (e.g., *cool* in Fig. 3a). SMC experiments are carried out with the Uppaal tool. Unlike exhaustive model-checking, SMC does not explore the state space but it applies statistical techniques to a set of *traces* entailed by the formal model to estimate the *probability* of the desired property holding. More specifically, we compute the value of expression $\mathbb{P}_M(\psi)$ to estimate the probability of ψ holding for M [17]. Property ψ, in our framework, is of the form $\diamond_{\leq\tau}\mathsf{ap}$, where $\diamond$ is the metric "eventually" operator and $\mathsf{ap} \in \mathrm{AP}$. Formula $\diamond_{\leq\tau}\mathsf{ap}$ is true if ap holds within τ times units from time instant 0, i.e., the onset of the system. If the value of $\mathbb{P}_M(\psi)$ is compared against a threshold $\vartheta \in [0, 1]$, i.e., formula $\mathbb{P}_M(\psi) \sim \vartheta$ is evaluated, where $\sim \in \{\leq, \geq\}$, the result of the SMC experiment is a Boolean value indicating whether the probability of ψ holding for M is $\sim$ than ϑ (thus, `true`) or not (yielding `false`), which is calculated through *hypothesis testing*. Otherwise, the SMC experiment returns a *confidence interval* $[\mathrm{p}_{\min}, \mathrm{p}_{\max}]$ of property ψ holding for M, with $\mathrm{p}_{\min}, \mathrm{p}_{\max} \in [0, 1]$, which is calculated according to the Clopper-Pearson method [18]. To determine when the generated set of traces is sufficient to conclude the experiment, Uppaal checks the length of interval $\epsilon = (\mathrm{p}_{\max} - \mathrm{p}_{\min})/2$. Uppaal stops generating new traces when $\epsilon \leq \epsilon_{\mathrm{th}}$ holds, where ϵ_{th} is an experimental parameter indicating the maximum desired estimation error. The smaller ϵ_{th}, the more accurate the estimation must be and, thus, more traces are required. Similarly, by applying the same Monte Carlo-based simulation, it is possible to calculate the expected value of distributions defined by means of functions max (maximum) and min (minimum) when they are applied to stochastic processes, such as expressions of variables in W. Given an upper bound τ, formulae $\mathrm{E}_{M,\tau}[\mathsf{max}(v)]$ and $\mathrm{E}_{M,\tau}[\mathsf{min}(v)]$, with $v \in W$, indicate respectively the expected value of the maximum and minimum value of variable v along executions that last *at most* τ time units. For example, with the model of Fig. 3, we can compute the probability of getting to operational state *high* within 10 seconds since the onset of the system by evaluating the formula $\mathbb{P}_M(\diamond_{\leq\tau}\mathit{high})$ with $\tau = 10$ and the expected value of the maximum and minimum temperature in the room by computing $\mathrm{E}_{M,\tau}[\mathsf{max}(T)]$ and $\mathrm{E}_{M,\tau}[\mathsf{min}(T)]$. In our framework, the SHA network M modeling an interactive scenario is put through SMC (without specifying a probability bound ϑ) to estimate the probability of success (corresponding to expression $\mathbb{P}_{\mathrm{M}}(\diamond_{\leq\tau}\mathsf{scs})$, where Boolean variable scs becomes true when the mission is completed), and to estimate the (average of the) maximum value of the fatigue of human agents (corresponding to formula $\mathrm{E}_{M,\tau}[\mathsf{max}(F)]$, where F is a real-value variable for the human fatigue) and of the minimum battery charge of the robot serving in a scenario (corresponding to formula $\mathrm{E}_{M,\tau}[\mathsf{max}(C)]$, where C is a real-value variable for the battery charge).

3. Design-Time Analysis of HRI Scenarios

This section illustrates the design-time analysis phase (phase **1** in Fig. 2), introducing the conceptual model of the scenarios which underpins the DSL and the DSL developed to specify HRI scenarios.

As represented in Fig. 2, the design-time analysis phase begins by configuring the scenario through a custom DSL (task "Scenario Configuration" in Fig. 2). The DSL file is automatically processed to generate the SHA network and set of properties according to the user's specifications. The designer specifies the characteristics of the robots, the involved humans, the geometrical representation of the environment layout, and the robotic missions.

Our framework features a predefined (yet extensible) set of high-level patterns identifying recurring interaction contingencies in assistive applications (e.g., a robot which follows a human). A *service* corresponds to an interaction pattern; therefore, for each service, the robot must perform the actions implied by the associated pattern (e.g., retrieve an object and deliver it back to the human). For each mission, the robot must provide services requested by the human in the order specified by the designer. Since the framework is not tied to a specific robot manufacturer or model, robotic platforms available in the fleet may not be pre-programmed to perform all required tasks. Therefore, the framework envisages an *ad-hoc* robot controller, hereinafter referred to as "*orchestrator*", in charge of monitoring the state of the system and sending commands to the robotic agent and suggestions to human subjects in conformity with the interaction patterns. Moreover, it is paramount to take into account the robot's level of charge and charge/discharge cycles (whose parameters vary between among different battery models), which may impact the duration of the mission (thus, its probability of success within a certain time range).

Under these premises, each mission is modeled by a SHA network featuring the following automata:

A. $\mathcal{A}_{\mathrm{h}_i}$ with $i \in [0, \mathsf{N_h} - 1]$ modeling the $\mathsf{N_h}$ humans involved in the scenario;

B. $\mathcal{A}_{\mathrm{r}}$ modeling the mobile robot;

C. $\mathcal{A}_{\mathrm{b}}$ modeling the robot's battery;

D. $\mathcal{A}_{\mathrm{o}}$ modeling the orchestrator.

The tool then automatically verifies through Uppaal the specified properties. At the end of the design-time analysis phase, the designer manually examines the verification results and assesses whether they satisfy their quality criteria, for example if the probability of success is sufficiently high or the expected value of fatigue is not excessive for any human. If results are not acceptable, the designer modifies the scenario (e.g., missions, environment layout, fatigue profiles) and repeats the analysis. If results are acceptable, the application can move forward to deployment or simulation.

3.1. Conceptual Model of HRI Scenarios

The framework covers human-robot interaction scenarios with specific characteristics. Mainly, scenarios must take place in a known layout (thus, robotic missions carried out in unknown environments do not fall within the scope of this work) and the service sequence does not change when the application is already running. Fig. 4 shows the conceptual model (represented as a Class Diagram) for the scenarios capturing the main entities they are composed of and their relations. The diagram, described in detail in the following, constitutes the conceptual foundation of the DSL and the working assumptions that underlie the formal model. Configuring a specific scenario through the DSL to be formally verified is equivalent to defining an instance (i.e., an Object Diagram) of the conceptual model.

A Scenario comprises at least one robotic mission. Each Mission is *set in* a known Layout, which we represent as a *composition* of one or multiple two-dimensional rectangular areas. Each Area is a *composition* of four corner Points, each characterized by a pair of Cartesian coordinates x and y. A Layout also includes a relevant subset of points, called Points Of Interest (POI), that can be the target of an action, such as room entrances, cupboards, and the robot's recharge station. Missions, areas and POIs are identified through attribute name.

A mission is a sequence of services *requested* by a human and *provided* by a robot (specifically, humans are served according to their id). Each Service conforms to an interaction pattern (attribute ptrn) and has a target POI. Patterns (i.e., the items of enumeration InteractionPattern) group common interaction contingencies and are listed in the following:

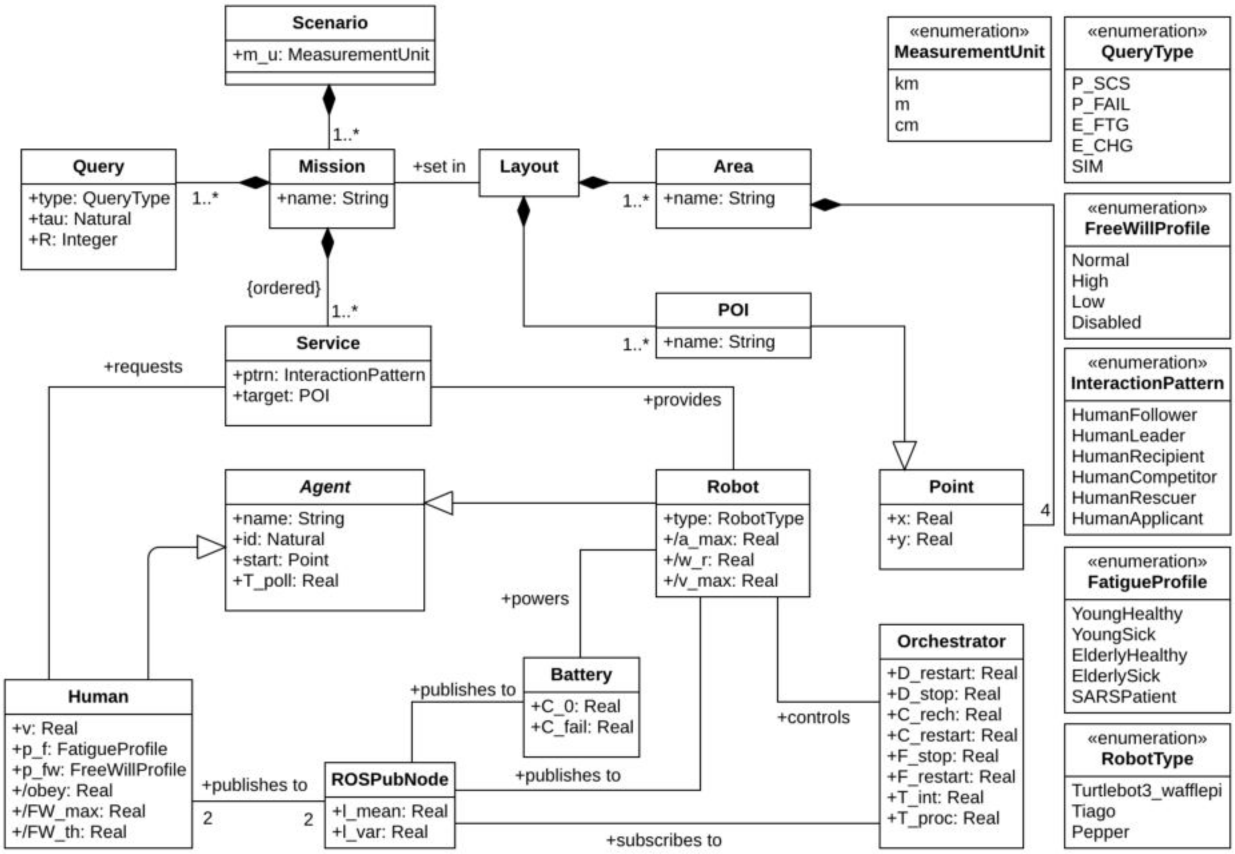


Figure 4: Class Diagram representing the entities constituting a scenario. Throughout the paper, when referring to a *class* of the model rather than the abstract concept it represents, its name is capitalized, uses Sans-Serif font (e.g., "Scenario" rather than "scenario"), and italicized for abstract classes (e.g., "*Agent*"). Attributes whose identifier has a subscript are reported in the diagram with an underscore for visualization purposes.

P1. HumanFollower: the human *follows* the robot to a specific destination (attribute target in Service) of which they do not know the precise location. For example, a patient looking for the waiting room or a doctor's office conforms to this pattern. The human follows the robot but, if they decide to stop walking, the robot also stops and waits for the human to get closer again. The robot signals that the service has been completed when both the robot and the human are close to the destination.

P2. HumanLeader: the human has to *lead* the robot to a specific destination of which they know the precise location (attribute target in Service), for example a nurse requiring the robot to escort them while carrying tools or medications. The human can decide when to start or stop walking and the robot follows accordingly. The human is in charge of signaling when the service has been completed when both the human and the robot have reached the destination.

P3. HumanRecipient: the human *waits* for the robot to fetch an item from a specific location (attribute target in Service) and bring it back to the human, for example a doctor requiring the robot to fetch a tool or a medication from a colleague and bring it back to their office. While the robot fetches and delivers the object, the human is free to move around (and the robot adjusts the delivery destination accordingly). The human is in charge of determining whether the service has been provided when they have successfully collected the item from the robot.

P4. HumanCompetitor: the human and the robot *compete* to fetch a critical resource (for example, a medical kit during an emergency). Both agents move to the location of the resource (captured by attribute target) to reach it as quickly as possible. The competition ends when either of the agents reaches the target location (effectively *winning* the competition). The human may autonomously decide to stop walking at any time.

P5. HumanRescuer: the pattern captures the robot requiring human intervention to complete a task, such as pressing a button to call the elevator or opening a closed door. In this case, the robot will emit audible or visible signals to notify its need for human support. The human autonomously decides to support the robot, move to the robot's current location (captured by attribute target), perform the required action and conclude the interaction.

P6. HumanApplicant: the pattern captures the human requiring the robot's support in performing a certain task that implies timely or close-contact interaction, such as feeding a patient or administering medication. In this case, as soon as the service starts, the human waits for the robot to approach their current location (attribute target). When the robot is sufficiently close, the action requiring synchronization starts. The human may autonomously decide to interrupt the action and resume at any time.

Agents enact the mission. Abstract class *Agent* has a name, id, and starting position start within the layout. In case of human agents, the id attribute determines the order in which humans are served. In case of robotic agents, the id attribute determines the order in which missions are assigned to robots in the fleet in case of multi-robot missions [16]. Agents are endowed with sensors that share a new reading every $\mathsf{T_{poll}}$ instants. Within the scope of our framework, there are two possible *specializations* of an *Agent*: humans and robots. For each robot, attribute type from enumeration RobotType defines its commercial model (e.g., "TurtleBot3" or "Tiago"). We assume that a Robot moves with a trapezoidal velocity profile, whose maximum acceleration $\mathsf{a_{max}}$, linear velocity $\mathsf{v_{max}}$, and angular velocity ω_r are *derived* from attribute type. Each Robot is *powered by* a lithium Battery with initial charge $\mathsf{C_0}$. Class Battery's attribute $\mathsf{C_{fail}}$ corresponds to the lowest voltage under which the device must not move to prevent the battery pack from being damaged.

SHA modeling human behavior include a model of physical fatigue. Each Human has a $\mathsf{p_f}$ attribute determining their fatigue profile (see the FatigueProfile enumeration in Fig. 4), which determines their proneness to fatigue and recovery based on physiological factors. We distinguish subjects by age (Young/Elderly) and state of health (Healthy/Sick) or whether they are affected by a severe respiratory syndrome that hinders their ability of deambulation (SARSPatient), obtaining five possible fatigue profiles. Attribute v specifies the average walking speed. Since human behavior is unpredictable in a real setting, our model includes a probabilistic approximation of human haphazard behavior (e.g., the possibility to ignore a robot's instruction or start and stop freely during the interaction). Therefore, a Human also features attribute $\mathsf{p_{fw}}$ from which attributes obey, $\mathsf{FW_{max}}$, and $\mathsf{FW_{th}}$ determining the probability with which such behavior manifests itself are derived. The $\mathsf{p_{fw}}$ attribute has four possible values (corresponding to the elements of the FreeWillProfile enumeration): Normal, High, Low, or Disabled. The latter results in human free will being entirely ignored at design-time, which may only be used for a preliminary test of the scenario setup.

Agents and batteries are equipped with sensors that during deployment share data with the orchestrator over dedicated topics handled by the middleware layer (based on ROS, see Fig. 2)[41]. The SHA network features a model of ROS publisher nodes (i.e., instances of class ROSPubNode in Fig. 4) mimicking the delay with which messages are processed and published. These delays are normally distributed with mean $\mathsf{l_{mean}}$ and variance $\mathsf{l_{var}}$ [25].

The Orchestrator monitors the state of the system by *subscribing* to sensor readings' topics of the human's position, fatigue, robot's position and battery charge. The Orchestrator periodically checks the state of the system against its policies every $\mathsf{T_{int}}$ time units and processes data for $\mathsf{T_{proc}}$ time units. While processing, it checks sensor-collected data against a set of thresholds: $\mathsf{D_{stop}}$ and $\mathsf{D_{restart}}$ determine the human-robot distance that causes the robot to stop and wait or restart, respectively; $\mathsf{C_{rech}}$ and $\mathsf{C_{restart}}$ correspond to the battery charge levels that cause the robot to start or stop recharging, respectively; $\mathsf{F_{stop}}$ and $\mathsf{F_{restart}}$ correspond to the human fatigue levels inducing the human to stop and rest or resume the action, respectively.

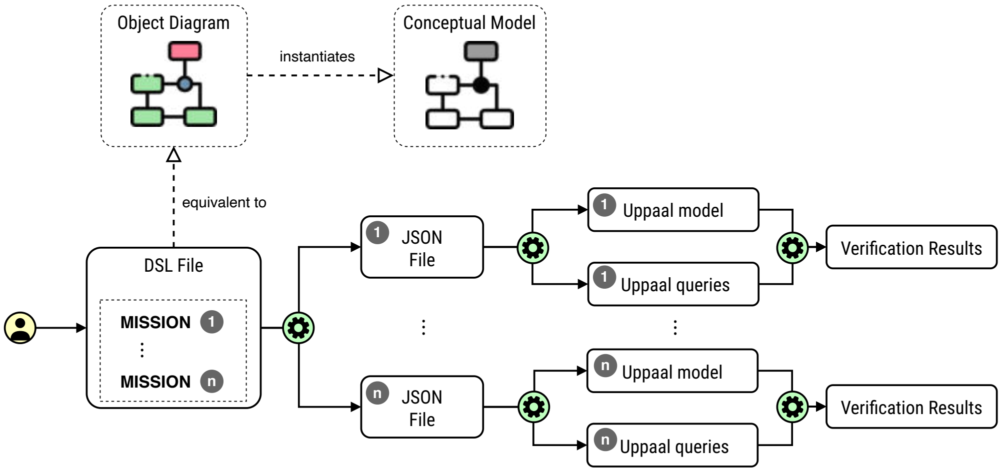


Figure 5: Diagram representing the process that translates a DSL file into Uppaal models. Solid arrows represent operational tasks while dashed arrows represent conceptual equivalences. Solid arrows are marked to distinguish the actions performed by the user from those performed automatically. Boxes are numbered to identify the relations between defined missions and the output files of each phase.

Finally, for each scenario, the analyst is interested in quality metrics to be computed referred to as "*queries*". The framework currently supports three query types (i.e., probability calculation, expected value calculation, and generation of traces) supported by Uppaal. However, different queries and tools can easily be embraced. For each Query, it is necessary to specify its type, time bound τ and—optionally—the maximum number of system traces generated to verify the property, i.e., attribute R. We remark that R should only be specified to limit performance issues during preliminary testing while it is normally advisable to let the verification tool compute the number of runs necessary to perform verification with the required confidence level. The different query types (modeled by enumeration QueryType) allow the designer to estimate:

Q1. the probability of the mission ending with success (item P_SCS) within the time bound. Success occurs when all services have been completed;

Q2. the probability of the mission ending in failure (item P_FAIL) within the time bound. Failure occurs either when the robot is fully discharged and cannot move autonomously or the human is fully fatigued (note that the mission not ending in success due to an insufficient time bound does not constitute a failure, thus the results of a P_FAIL and P_SCS query do not necessarily sum to 1);

Q3. the expected maximum value of fatigue (item E_FTG) for all humans within the time bound;

Q4. the expected minimum battery charge (item E_CHG) value within the time bound;

Q5. one or multiple (i.e., specified as parameter R) system traces to have a more detailed overview of how the system behaves during the execution of the mission (item SIM).

3.2. Domain-Specific Language

As represented in Fig. 5, configuring a scenario through the DSL is semantically equivalent to defining an Object Diagram of the conceptual model in Fig. 4. Therefore, the developed DSL features primitives allowing for the creation of instances of concrete scenarios that reflect the conceptual model. Each primitive is presented in the upcoming subsections through an example.

Fig. 5 shows how DSL files are converted into SMC experiment instances. Each DSL file defines a single scenario, which includes the layout geometry, the points of interest, the agents, and the mission, and represents a well-formed DSL model if specific properties are met (e.g., rooms have non-null area, agents are located within the boundaries of the environment, etc.). Well-formedness properties are automatically verified by the translator every time a DSL file undergoes the conversion process. Each mission in the DSL model is subject to formal verification separately, requiring, thus, a separate formal model. The conversion process features an intermediate phase: a JSON file containing the mission's characteristics is generated for each mission. Each JSON file is then converted into two files, one with the Uppaal model and one with the queries to perform the SMC experiment. The intermediate JSON notation, which is a lightweight and well-established standard, decouples the DSL from the specific verification tool and makes the framework flexible to the introduction of different verification tools or different DSLs. JSON files are also exploited to automatically set up the deployment environment.

We illustrate the DSL features and how they can be exploited to model the illustrative scenario in Section 1.1. The DSL does not have a specific statement for objects of class Scenario because each file inherently instantiates a single scenario, possibly including several missions. Every mission in a scenario consists of four independent sections, each one identified by the keyword **define**, concerning: layout definition, list of agents in the scene, list of services, and list of queries to be computed. Orchestrator and ROS nodes are instantiated automatically when the specification is translated into the model to be used for verification.

3.2.1. Layout, Areas and POIs

While modeling the HRI scenario, the user must specify the layout where the agents will operate. The DSL allows users to model different layouts (such as different building floors or different sections of the same floor) through statement:

define layout

which includes a non-empty list of areas and POIs.

The DSL captures all layouts made up of adjacent rectangular areas (i.e., it does not capture curved or diagonal walls), each defined as:

area id **in** (x_1, y_1) (x_2, y_2)

where coordinate pairs (x_1, y_1) and (x_2, y_2) define one of the area's diagonal segments from which the other two corner points are automatically inferred upon generating the SHA network to save manual effort on the user's side. Areas' corners are validated to ensure that they correctly identify a diagonal, i.e., that $x_1 \neq x_2$ and $y_1 \neq y_2$ hold. Layout-related declarations are validated to check whether there are disconnected areas (i.e., all areas must be *reachable* from any point in the layout) and that no pair of areas overlap entirely.

POIs with their coordinates are declared through the following statement:

poi id **in** (x, y)

Although verification tools only handle adimensional variables, the DSL requires the specification of the length measurement units to ensure that the layout size is consistent with the robot's speed. The measurement unit is specified through the following statement, where $\mathsf{m_u} \in \{\mathrm{km}, \mathrm{m}, \mathrm{cm}\}$:

param measurement_unit $\mathsf{m_u}$

The specification of the layout in Fig. 1 is given in Listing 1: the layout features two areas (a1 and a2) and three POIs corresponding to the recharge station (RC), the waiting room entrance (WR), KIT1, and KIT2. All coordinates are expressed in meters (m), consistently with Fig. 1b.

3.2.2. Agents

Each mission must feature a mobile robot and at least one human requiring assistance, specified in two independent sections that are identified through keywords **define robots** and **define humans**, respectively. The DSL allows designers to declare each available robot through statement:

robot name **in** (x, y) **id** id **type** type **charge** $\mathsf{C_0}$

Listing 1 DSL section defining layout areas and POIs.

```
param measurement_unit m
define layout:
  area A1 in (0.0, 17.5) (40.0, 7.5)
  area A2 in (40.0, 25.0) (50.0, 0.0)
  poi RC in (25.0, 17.5)
  poi WR in (49.5, 12.5)
  poi KIT1 in (40.5, 21.25)
  poi KIT2 in (40.5, 3.75)
```

Listing 2 DSL section defining the agents and their features.

```
define robots:
  robot ROB1 in (10.0, 12.5) id 1 type tiago charge 40
  robot ROB2 in (45.0, 3.5) id 2 type turtlebot3_wafflepi charge 90

define humans:
  human HUM1 in (5.0, 12.5) id 1 speed 80 is young_sick freewill low
  human HUM2 in (35.0, 9.0) id 2 speed 100 is elderly_healthy freewill normal
```

where parameters name and id univocally identify the robot, C_0 defines its initial level of charge, and coordinates (x, y) define its starting position. As per Section 3.1, while generating the formal model, the robot's type determines the translational and rotational speeds, and the acceleration (attributes $\mathsf{v}_{\mathsf{max}}$, ω_{r}, and $\mathsf{a}_{\mathsf{max}}$) in Fig. 4) based on the model's technical specifications. This feature of the DSL saves non-technical users the effort of retrieving these data when they might be more familiar with the type of robot available in the facility.

Each human is declared through the following statement:

$$\textbf{human}\ \mathsf{name}\ \textbf{in}\ (x, y)\ \textbf{id}\ \mathsf{id}\ \textbf{speed}\ \mathsf{v}\ \textbf{is}\ \mathsf{p_f}\ \textbf{freewill}\ \mathsf{p_{fw}} \tag{1}$$

where the name univocally identifies the human and the id determines the serving order (thus, it is also required to be unique). Coordinates (x, y) determine each human's starting location. Parameters v, $\mathsf{p_f}$, and $\mathsf{p_{fw}}$ define the walking speed, fatigue and free will profiles as described in Section 3.1. The user chooses the values of $\mathsf{p_f}$ and $\mathsf{p_{fw}}$ out of a pre-determined list, corresponding to the enumerations in Fig. 4.

Both the robot and human declaration blocks are validated to ensure that no pair of agents share the same id (within the same *Agent* generalization) nor the same name (also across different *Agent* generalizations). The DSL is developed under the simplifyng hypothesis that agents occupy a single point in space (corresponding to their center of gravity). This modeling choice is dictated by the need to keep the DSL (and, thus, the formal model) as simple as feasible and spare the designer from defining the three-dimensional envelope of the agents' bodies. The framework assumes that refined collision avoidance routines are already implemented at a lower level within the robotic platform and the DSL validator only checks that no pair of agents have the same center of gravity (i.e., coordinates (x, y)).

The agents from the running example are defined as per Listing 2. There are two robots available (ROB1 and ROB2) of different types (thus, they will have different speeds), and two humans (HUM1 and HUM2), of which one has a Young/Sick fatigue profile and low free will, whereas the second one is Elderly/Healthy and has normal free will profile.

3.2.3. Missions

Designers can declare multiple **missions** and associate them with a layout and a set of agents. Each mission is assigned to a single robot and verification experiments resulting from each mission declaration (see

Listing 3 DSL section defining the mission (i.e., the sequence of services).

```
define mission m1 for ROB1:
  do robot_leader for HUM1 with target WR
  do robot_transporter for HUM2 with target KIT2

define mission m2 for ROB2:
  do robot_follower for HUM2 with target KIT1
  do robot_leader for HUM1 with target WR
```

Listing 4 DSL section defining the set of queries.

```
 define queries of mission m1:
   compute probability_of_success with duration 120 runs 300
   compute expected_fatigue with duration 120 runs 50

 define queries of mission m2:
   compute probability_of_success with duration 100 runs auto
   compute probability_of_failure with duration 100 runs auto

```

Fig. 5) are performed separately. A mission is declared as in the following, where parameter m is the name of the mission and r is the name of the robot it is assigned to (association *provides* in Fig. 4).

define mission m **for** r

As described in Section 3.1, each mission consists of a sequence of services and each service adheres to one of the interaction patterns described in Section 3.1. As per Statement (1) and the conceptual model presented in Section 3.1, humans are declared independently of the interaction pattern, which is specified when declaring the service as in the following:

do ptrn **for** h **with target** poi

where ptrn can be either robot_leader, robot_follower, robot_transporter, robot_competitor, robot_applicant, or robot_rescuer (corresponding to the HumanFollower, HumanLeader, HumanRecipient, HumanCompetitor, HumanRescuer, and HumanApplicant patterns, respectively), h is the name of the human requesting the service (association *requests* in Fig. 4), and poi instantiates attribute target in Fig. 4. Each service declaration is validated to ensure that h and poi refer to existing human agents and POIs.

The two missions associated with the running example are declared as in Listing 3. In mission m1, ROB2 has to lead HUM1 to POI WR and then deliver KIT2 to HUM2. In m2, ROB2 has to follow HUM2 to KIT1, then lead HUM1 to WR.

3.2.4. Queries

Finally, the designer specifies which experiments to perform for each mission. The DSL captures the set of queries in Fig. 4 and described in Section 3.1. A query is declared through the following statement:

compute query **with duration** τ **runs** R

where query can be either probability_of_success, probability_of_failure, expected_charge, expected_fatigue, or simulation. Parameter τ corresponds to the time bound, while R is the bound on the number of traces generated for the SMC experiment, whose value must be set to `auto` if the user wants the verification tool to compute the required number of runs.

A possible set of queries for missions 1 and 2 from the running example is given in Listing 4: the SMC experiments will estimate the probability of success and maximum fatigue value for all humans for `m1`, and probabilities of failure and success (with no bound on runs) for `m2`.

4. Formal Modeling HRI with Uncertain Human Behaviors

In this section, we illustrate the modeling approach we have adopted to *map* aspects of the real system to SHA features. Subsequently, we present in more detail the automata constituting the SHA network, i.e., the humans, the robot and its battery, and the orchestrator.

The high-level goal of the SHA network is to capture the agents' behavior based on their current *operating state* (e.g., the human resting or walking). For every agent in the scenario and automaton $\mathcal{A}$ modeling its behavior, defined as in Definition 1, every operating state of the agent corresponds to a *location* in L. SHA capture the evolution of relevant quantitative attributes of the real system, such as human fatigue and battery level of charge. Each physical attribute, characterizing a human or a component, corresponds to a real-valued variable in set $W \setminus \{X \cup V_{\mathrm{dc}} \cup K\}$ of their modeling automata and flow conditions $\mathcal{F}(l)$, associated with a location l, reproduce the set of ODEs constraining the evolution of real-valued variables in that specific operating state.

The switch between two operating states consists of an *edge* between the two corresponding locations. Recurrent features of the specific systems that our modeling approach targets identify two *types* of switches, which we refer to as *controllable* or *uncontrollable*. We remark that we use these two terms in a manner that is specific to our framework, and they are not part of the standard terminology of the formalism (for example, they are unrelated to the notion of controllable and uncontrollable edges in Timed Game Automata [26]); instead, they are merely aliases for specific edges recurring in our SHA (i.e., subsets of $\mathcal{E}$ from Definition 1). A controllable switch occurs if, and only if, a specific event fires and a synchronization among two or more automata occurs: for example, the robot starts accelerating when the orchestrator issues the command to start moving. For this reason, all the edges modeling a controllable switch are defined with a channel in $C \setminus \{\epsilon\}$. Conversely, uncontrollable switches occur "naturally" in the original system due to the evolution of the physical variables at play: for example, the human unavoidably stops moving when their fatigue level reaches the maximum endurable threshold. Therefore, they are defined with event $\epsilon!$, i.e., by means of the internal action. In every automaton of the networks capturing the targeted systems, a location l with an outgoing edge modeling an uncontrollable switch is endowed with a set of invariants of the form $w \leq \mathsf{k}_2$, where $w \in W$ is the real-valued variable subject to the constraint and $\mathsf{k}_2 \in K$ is its maximum allowable value (e.g., $F \leq 1$, with $w = F$ and $\mathsf{k}_2 = 1$, constrains the value of the human fatigue to be less or equal than 1). The outgoing edge has condition $w \geq \mathsf{k}_1$, such that $\mathsf{k}_1 \in K$ and $\mathsf{k}_2 \geq \mathsf{k}_1$ hold. If $\mathsf{k}_2 > \mathsf{k}_1$ holds, the edge fires with probability distributed uniformly over interval $[\mathsf{k}_1, \mathsf{k}_2]$, as explained in Appendix A. If $\mathsf{k}_1 = \mathsf{k}_2$ holds, the edge fires with probability 1 when $w = \mathsf{k}_1 = \mathsf{k}_2$ holds (e.g., the edge from *on* to *off* in Fig. 3 where $w = T$ and $\mathsf{k}_1 = \mathsf{k}_2 = \mathsf{T}_{\mathsf{th}_2}$).

Since the orchestrator controls the robots by using the digital observations made by sensors on humans and robots, the SHA modeling physical dynamics (i.e., humans and robots) feature dense-counter variables (set V_{dc}) as the discrete (i.e., digital) equivalents of real-valued variables. Dense counters are periodically updated every $\mathsf{T}_{\mathsf{poll}} \in K$ time units (where $\mathsf{T}_{\mathsf{poll}}$ corresponds to the refresh period of the specific sensor) through updates in $\Xi(W)$ that are compatible with the ODEs modeling the dynamics of the physical attributes in each location. To this end, every SHA in the network uses a clock $t_{\mathrm{upd}} \in X$ to measure the time elapsed between two consecutive measurements and to trigger an update. Therefore, when $t_{\mathrm{upd}} = \mathsf{T}_{\mathsf{poll}}$ holds for an automaton $\mathcal{A}$, hence when time $\mathsf{T}_{\mathsf{poll}}$ has elapsed since the last measurement, then $\mathcal{A}$ uncontrollably switches to a *committed* location. A committed location is equivalent to an ordinary location with invariant $t \leq 0$ and all incoming edges with update $t = 0$ for some $t \in X$: therefore, time cannot elapse while in these locations [27]. In that location, the dense counters modeling the latest sensor readings are immediately notified to the orchestrator by firing an event over a dedicated channel that triggers the publishing routine (the corresponding modeling pattern is described in detail in [15, Section IV]).

In the forthcoming sections, we present the automata constituting the SHA network, i.e., the humans, the robot and its battery, and the orchestrator, primarily focusing on the human behaviors modeled by the patterns Human Follower, Human Leader, and Human Recipient, which this work extends with respect to [13, 14] with a stochastic characterization of physical fatigue, and a factorization of the common modeling pattern capturing the periodic sensor reading update. Comparable SHA for the Human Applicant, Human Rescuer, and Human Competitor patterns are introduced in [16]. We also present a refined model of the robot's battery, which fits the real platform used for the experimental validation. The robot-modeling SHA is presented in detail in [15] and briefly described here to preserve the self-containedness of this paper. Moreover, we report the high-level structure of the orchestrator (introduced in [13] and extended in [16]) with refined mission-management policies.

Controllable switches realize the interactions among the automata and are obtained by means of synchronization channels. Since the orchestrator implements the control logic that governs the agents, the issuing of a command by the orchestrator is modeled through a synchronization between the orchestrator and the automaton modeling the agent that reacts to the command. Hence, all the channels in (the automaton modeling) the orchestrator are labeled with !, whereas (the automata modeling) the humans and the battery are defined with edges having the channels labeled with ?. Hence, The modeling has been realized by considering one single robot serving one individual at a time, even if several agents (i.e., humans and robots) can participate in the scenario. In fact, each pattern models the interaction between one pair of agents, a robot and a human, and missions are finite sequences of interactions. Since there is always only one active robot and a single served human at a time, channels representing synchronizing events between the orchestrator and the agents are not specific to a single instance of a human or a robot. A dense counter in the orchestrator, with finite domain, identifies the human currently interacting with the robot and it is evaluated by every automata modeling the humans to allow or deny the firing of the synchronization events with the orchestrator. In particular, all the edges in the automaton modeling a human agent include a condition which evaluates to true when the dense counter indicating the currently served human is equal to the value id uniquely identifying it (see Section 3.1) and, hence, the automaton. Moreover, even if the modeling of multiple robots serving multiple humans simultaneously is possible in theory, adding this feature would cause the models to increase in complexity. The information flow that the orchestrator realizes by issuing commands to agents, through events via channels, is as follows. The orchestrator

- informs the human to start or stop walking via channels $\texttt{cmd\_h}_{\text{start}}$ and $\texttt{cmd\_h}_{\text{stop}}$;
- makes the robot move or stop via channels $\texttt{cmd\_r}_{\text{start}}$ and $\texttt{cmd\_r}_{\text{stop}}$. Moreover, it starts the battery charging through channel $\texttt{cmd\_b}_{\text{start}}$ and interrupts the charging with channel $\texttt{cmd\_b}_{\text{stop}}$, hence restoring the robot back to the mission-defined interaction.

4.1. Human-Robot Interaction Patterns Model

In all interaction patterns, the SHA modeling humans differentiate between operating states based on how fatigue evolves (i.e., whether individuals are recovering or not) and how humans are interacting with the robot (e.g., they are leading the action or waiting for a robot's action). Hence, all SHA modeling humans feature real-valued variable $F \in W$, capturing physical fatigue, and a dense counter $\mathrm{f} \in V_{\mathrm{dc}}$ capturing the digital counterpart of F. Besides physical fatigue, for each human, suitable sensors also periodically refresh their position within the building. The position is modeled by dense counters $\mathrm{h}_{\mathrm{pos}_\mathrm{x}}$ and $\mathrm{h}_{\mathrm{pos}_\mathrm{y}}$ capturing a pair of Cartesian coordinates. Therefore, the portion of SHA modeling the update of periodic sensor readings is present in all the operating states of the human, generically indicated as op, and is hereinafter referred to as $\langle \mathsf{op} \rangle\_\mathrm{pub}_h$. In the following sections, for a clock $t \in X$, we use notation $\{t\}$ to represent update $t = 0$ (e.g., we write $\{t_{\mathrm{upd}}\}$ instead of $t_{\mathrm{upd}} = 0$).

The $\langle \mathsf{op} \rangle\_\mathrm{pub}_h$ pattern is shown in Fig. 6. In the following, we use label op when describing the high-level structure of the pattern, while it is replaced by descriptive labels when referring to a specific instance of the pattern (e.g., $\langle \mathsf{stand} \rangle\_\mathrm{pub}_h$ and $\langle \mathsf{walk} \rangle\_\mathrm{pub}_h$). The automaton features three locations: an ordinary location $h_{\langle \mathsf{op} \rangle}$ and two committed locations h_{pub_1} and h_{pub_2}. Location $h_{\langle \mathsf{op} \rangle}$ captures the human's behavior while in a specific state (e.g., $h_{\langle \mathsf{stand} \rangle}$ and $h_{\langle \mathsf{walk} \rangle}$). To this end, $h_{\langle \mathsf{op} \rangle}$ is endowed with invariants $\mathcal{I}(h_{\langle \mathsf{op} \rangle})$, flow conditions

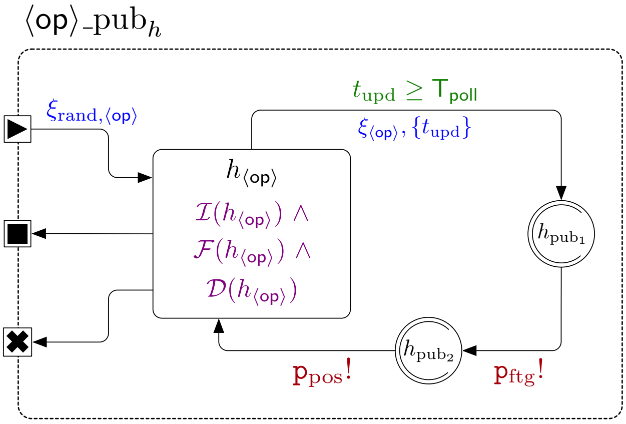


Figure 6: SHA modeling the $\langle\mathsf{op}\rangle$_pub$_h$ pattern, color-coded as in Fig. 3. Ports are marked by symbols "▶", "■", and "×".

$\mathcal{F}(h_{\langle\mathsf{op}\rangle})$, and probability distributions $\mathcal{D}(h_{\langle\mathsf{op}\rangle})$. For all instances of $\langle\mathsf{op}\rangle$_pub$_h$, $(t_{\text{upd}} \leq \mathsf{T}_{\mathsf{poll}}) \in \mathcal{I}(h_{\langle\mathsf{op}\rangle})$ holds. The combination of this invariant with condition $t_{\text{upd}} \geq \mathsf{T}_{\mathsf{poll}}$ on the edge to h_{pub_1} forces the SHA to switch to the committed location when $t_{\text{upd}} = \mathsf{T}_{\mathsf{poll}}$ holds. Upon switching, the set of updates $\xi_{\langle\mathsf{op}\rangle} \subset \Xi(W)$ (e.g., $\xi_{\langle\mathsf{stand}\rangle}$ and $\xi_{\langle\mathsf{walk}\rangle}$) updates dense counters f, $\text{h}_{\text{pos}_\text{x}}$, and $\text{h}_{\text{pos}_\text{y}}$. The effect of $\xi_{\langle\mathsf{op}\rangle}$ varies depending on the specific state of the human. Since h_{pub_1} and h_{pub_2} are committed, the new values are immediately shared with the orchestrator by firing an event through channels $\mathtt{p}_{\text{ftg}}$ first and $\mathtt{p}_{\text{pos}}$ right after.

Edges entering and leaving the $\langle\mathsf{op}\rangle$_pub$_h$ pattern are represented through *ports* (coherently with [15, 16]). Ports are not part of the formalism, but a visualization expedient for the edges entering and leaving the sub-automaton (arrows in and out of a port constitute the *same* transition). SHA *enter* a submachine through the port marked by symbol "▶" (i.e., *start*) and *leave* a submachine through ports marked by symbols "■" (i.e., *end*) and "×" (i.e., *fail*), indicating whether the operating state op ended (or stopped momentarily) or the entire mission ended with failure (e.g., because the human is too fatigued), respectively.

Edge conditions, channels, and updates characterizing edges through ports vary depending on the specific $\langle\mathsf{op}\rangle$_pub$_h$ instance. The only exception is update $\xi_{\text{rand},\langle\mathsf{op}\rangle}$ on the edge through the start port. Update $\xi_{\text{rand},\langle\mathsf{op}\rangle}$ is featured by *all* instances of $\langle\mathsf{op}\rangle$_pub$_h$ since it determines the stochastic properties of human fatigue when a human behaves while in a specific operating state $\langle\mathsf{op}\rangle$ and the way these properties are determined is the same for every instance of $\langle\mathsf{op}\rangle$_pub$_h$.

Human fatigue is a complex phenomenon driven by a wide range of factors: our approach focuses on muscular fatigue due to physical strain. As discussed by Liu et al. [28], a muscle can be seen as a reservoir of motor units. When physical exertion is required, motor units progressively activate and eventually cause *fatigue* due to biochemical processes. The muscle can, then, *recover* from fatigue if it is put to rest [28, 29]. Our approach exploits the model proposed by Konz [29, 30], described by Eq.2, for which human action undergoes alternate fatigue and recovery cycles, each one modeled by an exponential function. Fatigue and recovery are expressed by means of function parameters, called fatigue rates, which depend on several factors such as the age of the subject that the model represents, their health condition, etc. Each cycle is associated with an index i uniquely identified, given time t, by function $j : \mathbb{R}_+ \to \mathbb{N}$ (thus, $i = j(t)$ holds). We indicate the timestamp at which cycle i ends by t_i. During both fatigue and recovery, fatigue $F(t)$ for cycle i depends on the residual value $F(\mathsf{t}_{i-1})$ from the previous cycle ended at time t_{i-1}. Parameters λ_i and ρ_i are the fatigue and recovery rates for cycle i.

$$F(t) = \begin{cases} 1 - (1 - F(\mathsf{t}_{i-1})) \cdot e^{-\lambda_i (t - \mathsf{t}_{i-1})} & \text{(fatigue)} \\ F(\mathsf{t}_{i-1}) \cdot e^{-\rho_i (t - \mathsf{t}_{i-1})} & \text{(recovery)} \end{cases} \quad (2)$$

Full recovery occurs when $F(t) = 0$ holds, whereas condition $F(t) = 1$ models the case in which the muscle has reached the maximum level of *endurance*. Liu et al. [28] argue that the fatigue $F(t)$ can be seen as ratio $M_F(t)/\mathrm{M}_0$, where M_0 is the total amount of motor units, and $M_F(t)$ is the amount of fatigued units at time t. Therefore, $F(t) = M_F(t)/\mathrm{M}_0 = 1$ holds when *every* unit composing a muscle is fatigued. Running experiments on a pool of subjects have shown how a Normal distribution is a good fit to capture the variability of rates λ_i and ρ_i in the fatigue model [31]. Furthermore, the variability of the fatigue rates for an individual subject between different exertion cycles has been observed in [32]. The SHA modeling the human in a scenario embeds this variability by means of probability distributions, as the automaton is not representative for a single specific individual, but it represents a set of subjects with similar physical characteristics. Therefore, we approximate the complexity of the fatigue phenomenon by considering each λ_i (resp., ρ_i) as a sample of distribution $\mathcal{N}(\mu_\lambda, \sigma_\lambda^2)$ (resp., $\mathcal{N}(\mu_\rho, \sigma_\rho^2)$), whose mean and variance depend on the fatigue profile that characterizes the class of humans under analysis.

By construction, every operating state of a human agent is associated with a specific fatigue profile, i.e., it is either a fatigue state or a recovery state. Hence, for every instance of $\langle\mathsf{op}\rangle\_\mathrm{pub}_h$ function $\mathcal{D}(h_{\langle\mathsf{op}\rangle})$ is defined. Upon entering an $\langle\mathsf{op}\rangle\_\mathrm{pub}_h$, update $\xi_{\mathrm{rand},\langle\mathsf{op}\rangle}$ computes the fatigue/recovery rate to be considered while the automaton is in location $h_{\langle\mathsf{op}\rangle}$. To this end, every automaton modeling a human features two dense counters $\lambda, \rho \in V_{\mathrm{dc}}$, which store the current fatigue/recovery rates. Every time update $\xi_{\mathrm{rand},\langle\mathsf{op}\rangle}$ is executed, it generates a new sample of $\mathcal{D}(h_{\langle\mathsf{op}\rangle})$ and assigns it to ρ, if $h_{\langle\mathsf{op}\rangle}$ is a recovery state, otherwise to λ. The sample is generated through the Box-Müller algorithm [17]. Update $\xi_{\mathrm{rand},\langle\mathsf{op}\rangle}$ is given in Eq.3, where rate equals λ if $h_{\langle\mathsf{op}\rangle}$ is a fatigue state and ρ otherwise; $\mu_{\langle\mathsf{op}\rangle}$ and $\sigma_{\langle\mathsf{op}\rangle}$ are the mean and standard deviation of $\mathcal{D}(h_{\langle\mathsf{op}\rangle})$; u_1 and u_2 are independent realizations of uniform distribution $\mathcal{U}(0,1)$.

$$\xi_{\mathrm{rand},\langle\mathsf{op}\rangle}: \quad \mathsf{rate} \;=\; \mu_{\langle\mathsf{op}\rangle} + \sigma_{\langle\mathsf{op}\rangle}\sqrt{-2\ln(u_2)}\cos(2\pi u_1) \tag{3}$$

The values of ρ and λ determine the temporal evolution of the real-valued variable F and its digital counterpart f while the automaton is in $h_{\langle\mathsf{op}\rangle}$. In a given operating state $\langle\mathsf{op}\rangle$, if fatigue increases, flow condition $\mathcal{F}(h_{\langle\mathsf{op}\rangle})$ corresponds to the derivative of Eq.2(fatigue), indicated as f_{ftg} in Eq.4; otherwise, fatigue decreases and $\mathcal{F}(h_{\langle\mathsf{op}\rangle})$ is equal to the derivative of Eq.2(recovery), indicated as f_{rec} in Eq.5. Both equations depend on two terms other than ρ and λ, i.e., clock $t_{\mathrm{phase}} \in X$ and dense counter $\mathrm{F_p} \in V_{\mathrm{dc}}$. Clock $t_{\mathrm{phase}} \in X$ measures the total amount of time the automaton spends in location $h_{\langle\mathsf{op}\rangle}$ and dense counter $\mathrm{F_p} \in V_{\mathrm{dc}}$ is the residual value of fatigue at the end of the previous fatigue/recovery cycle, realized by a different $\langle\mathsf{op}\rangle\_\mathrm{pub}_h$ instance. Both are updated when a new fatigue/recovery cycle begins, i.e., every time the SHA modeling the human enters an instance of $\langle\mathsf{op}\rangle\_\mathrm{pub}_h$ and $\xi_{\mathrm{rand},\langle\mathsf{op}\rangle}$ is carried out: clock t_{phase} is reset and variable $\mathrm{F_p}$ is updated with F.

$$\dot{F} = f_{\mathrm{ftg}}(t_{\mathrm{phase}}, \lambda) = \mathrm{F_p}\lambda e^{-\lambda t_{\mathrm{phase}}} \tag{4}$$

$$\dot{F} = f_{\mathrm{rec}}(t_{\mathrm{phase}}, \rho) = -\mathrm{F_p}\rho e^{-\rho t_{\mathrm{phase}}} \tag{5}$$

Dense counter f, on the other hand, is not associated with a flow in location $h_{\langle\mathsf{op}\rangle}$, because it models the digital equivalent of the physical attribute F. For this reason, the temporal evolution of f is calculated explicitly via the update $\xi_{\langle\mathsf{op}\rangle}$, which computes a new value for f by applying the update in Eq.6, every $\mathsf{T_{poll}}$ time units. The primed version f′ indicates the new value of f after the computation of the expression, which depends on the operating state $\langle\mathsf{op}\rangle$. Unlike Eq.2, the equations in Eq.6 model fatigue in a single cycle and are expressed in terms of the amount of time elapsed from the beginning of the current fatigue/recovery cycle. Conversely, Eq.2 depends on the absolute time t and instant $\mathtt{t}_{i-1}$, the latter indicating the end of the cycle that precedes the current one. Hence, if τ is the amount of time elapsed from the beginning of a cycle, Eq.2 can be rewritten in terms of τ by applying the identity $t - \mathtt{t}_{i-1} = \tau$, and fatigue after τ time units from the beginning of the current fatigue/recovery cycle is $\bar{F}(\tau) = F(\mathtt{t}_{i-1} + \tau)$. For $\tau = 0$, fatigue $\bar{F}(0)$ is equal to the residual value $F(\mathtt{t}_{i-1})$, which is $\mathrm{F_p}$. At the end of the $(k+1)$-th sensor refresh, lasting $\mathsf{T_{poll}}$ time units each, the fatigue is $F(k\mathsf{T_{poll}} + \mathsf{T_{poll}})$. The final expressions are obtained by considering that, before computing $\xi_{\langle\mathsf{op}\rangle}$, f amounts to $\mathrm{F_p}e^{-\rho k\mathsf{T_{poll}}}$, in case of recovery, and to $1-(1-\mathrm{F_p})e^{-\lambda k\mathsf{T_{poll}}}$ otherwise (i.e., the fatigue after k refresh cycles).

$$\mathrm{f}' = \bar{F}(k\mathsf{T_{poll}} + \mathsf{T_{poll}}) = \begin{cases} 1-(1-\mathrm{F_p})e^{-\lambda(k\mathsf{T_{poll}}+\mathsf{T_{poll}})} = 1-(1-\mathrm{F_p})e^{-\lambda k\mathsf{T_{poll}}}e^{-\lambda\mathsf{T_{poll}}} = 1-(1-\mathrm{f})e^{-\lambda\mathsf{T_{poll}}} & \text{(fatigue)} \\ \mathrm{F_p}e^{-\rho(k\mathsf{T_{poll}}+\mathsf{T_{poll}})} = \mathrm{F_p}e^{-\rho k\mathsf{T_{poll}}}e^{-\rho\mathsf{T_{poll}}} = \mathrm{f}e^{-\rho\mathsf{T_{poll}}} & \text{(recovery)} \end{cases} \tag{6}$$

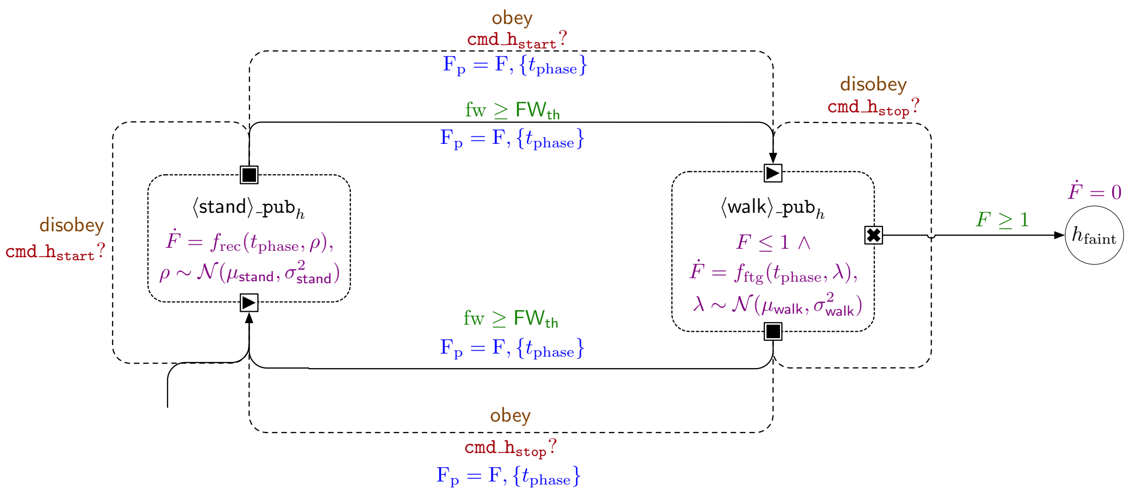


Figure 7: SHA modeling human behavior when adhering to the HumanFollower pattern. Color-coding is the same as Fig. 3.

Compared to [13, 14], we extend SHA modeling humans by introducing $\mathcal{D}(h_{\langle\mathsf{op}\rangle})$, $\xi_{\langle\mathsf{op}\rangle}$, and $\xi_{\mathrm{rand},\langle\mathsf{op}\rangle}$ in all $\langle\mathsf{op}\rangle$_pub$_h$ instances. Enriching the SHA with these features strengthens the results obtained with SMC as they account not only for the uncertainty due to human autonomy, but also for the natural variability of the fatigue phenomenon. The extension, therefore, leads to more reliable estimations of the fatigue levels reached by subjects involved in the scenario, including an estimation of their variability ranges.

In the following, we present the individual SHA modeling the three interaction patterns, all featuring multiple instances of the hereby presented $\langle\mathsf{op}\rangle$_pub$_h$ pattern.

4.1.1. Human Follower

An instance of the SHA modeling the human follower pattern is generated for each service specified through the DSL with ptrn = HumanFollower. The SHA, hereinafter referred to as $\mathcal{A}_{\mathrm{hf}}$ and shown in Fig. 7, features two instances of the $\langle\mathsf{op}\rangle$_pub$_h$ pattern: one capturing the recovery phase while standing ($\langle\mathsf{stand}\rangle$_pub$_h$) and one for the fatigue phase while walking ($\langle\mathsf{walk}\rangle$_pub$_h$). Fatigue decreases while resting (in $\langle\mathsf{stand}\rangle$_pub$_h$) and increases while walking (while in $\langle\mathsf{walk}\rangle$_pub$_h$). Therefore, $\mathcal{F}(h_{\langle\mathsf{stand}\rangle})$ equals $f_{\mathrm{rec}}(t, \rho)$ (see Eq.5) and $\mathcal{F}(h_{\langle\mathsf{walk}\rangle})$ equals $f_{\mathrm{ftg}}(t, \lambda)$ (see Eq.4). Values ρ and λ are realizations of $\mathcal{N}(\mu_{\mathsf{stand}}, \sigma^2_{\mathsf{stand}})$ and $\mathcal{N}(\mu_{\mathsf{walk}}, \sigma^2_{\mathsf{walk}})$, respectively. Table 1 shows the internal updates, ξ_{stand} and ξ_{walk}, respectively, later described in detail. $\mathcal{A}_{\mathrm{hf}}$ also features a deadlock location h_{faint} capturing the case in which the human reaches full exhaustion causing the failure of the mission. If the mission fails because the human has reached location $h_{\langle\mathsf{faint}\rangle}$, modeling the evolution of fatigue is no longer relevant. Therefore, location $h_{\langle\mathsf{faint}\rangle}$ is endowed with flow condition $\dot{F} = 0$.

While walking (thus, while in location $h_{\langle\mathsf{walk}\rangle}$), the SHA periodically updates variables $\mathrm{h_{pos_x}}$ and $\mathrm{h_{pos_y}}$. As described in Section 3.1, we assume that humans walk at average speed $\mathsf{v} \in K$. Dense counter h_γ captures the human's orientation with respect to the x-axis. We assume that the human can rotate instantly while following their trajectory, and, thus, no location is necessary to capture the delay caused by rotation. Variable h_γ is periodically updated while walking through function upd_orientation(), which computes the new orientation required (primed dense counter h'_γ) to head towards the following point of the trajectory. Therefore, every $\mathsf{T_{poll}}$ time instants, the x-y coordinates increase by $\mathsf{vT_{poll}}\cos(\mathrm{h}_\gamma)$ along the x-axis and $\mathsf{vT_{poll}}\sin(\mathrm{h}_\gamma)$ along the y-axis. While standing (in $h_{\langle\mathsf{stand}\rangle}$), as per Table 1, the human does not move, thus the values of $\mathrm{h_{pos_x}}$ and $\mathrm{h_{pos_y}}$ do not change. The mechanism capturing human free will and the corresponding dense counter fw is presented later in this section. The periodic sensor refresh mechanism does not apply to location h_{faint} (which is not, thus, part of an $\langle\mathsf{op}\rangle$_pub$_h$ instance) since, once the mission has failed, the orchestrator no longer requires up-to-date sensor measurements.

The switch between $h_{\langle\mathsf{stand}\rangle}$ and $h_{\langle\mathsf{walk}\rangle}$ (and viceversa) is controllable and triggered by events through

Table 1: Updates for the SHA modeling the $\mathsf{HumanFollower}$ and $\mathsf{HumanLeader}$ patterns.

Symbol	Updates	Description
ξ_{stand}	$\mathrm{f}' = \mathrm{f}e^{\rho \mathsf{T}_{\mathsf{poll}}}$; $\mathrm{h}'_{\gamma} = \mathrm{h}_{\gamma}$; $\mathrm{h}'_{\mathrm{pos_x}} = \mathrm{h}_{\mathrm{pos_x}}$; $\mathrm{h}'_{\mathrm{pos_y}} = \mathrm{h}_{\mathrm{pos_y}}$; $\mathrm{fw}' = \mathsf{roll\_dice}()$;	Resting phase
ξ_{walk}	$\mathrm{f}' = 1 - (1-\mathrm{f})e^{-\lambda \mathsf{T}_{\mathsf{poll}}}$; $\mathrm{h}'_{\gamma} = \mathsf{upd\_orientation}()$; $\mathrm{h}'_{\mathrm{pos_x}} = \mathrm{h}_{\mathrm{pos_x}} + \mathsf{v}\mathsf{T}_{\mathsf{poll}}\cos(\mathrm{h}_{\gamma})$; $\mathrm{h}'_{\mathrm{pos_y}} = \mathrm{h}_{\mathrm{pos_y}} + \mathsf{v}\mathsf{T}_{\mathsf{poll}}\sin(\mathrm{h}_{\gamma})$; $\mathrm{fw}' = \mathsf{roll\_dice}()$;	Fatiguing phase

channels $\mathtt{cmd\_h}_{\text{start}}$ and $\mathtt{cmd\_h}_{\text{stop}}$. The orchestrator sends to the SHA modeling the human events through these channels when it detects that the interaction between the human and the robot must start ($\mathtt{cmd\_h}_{\text{start}}$) or stop ($\mathtt{cmd\_h}_{\text{stop}}$). Upon switching between $h_{\langle \mathsf{stand} \rangle}$ and $h_{\langle \mathsf{walk} \rangle}$, the SHA updates the value of variable $\mathrm{F_p}$ (see the updates on entering $\langle \mathsf{op} \rangle\_\mathrm{pub}_h$ instances). To capture the unpredictability of human behavior, the edges between $h_{\langle \mathsf{stand} \rangle}$ and $h_{\langle \mathsf{walk} \rangle}$ and back have specific features modeling human free will. In the literature, there exist several proposals on how to model the free will phenomenon [33]. We exploit the results on the free will phenomenon and randomness in [34] to model human *haphazard* choices probabilistically. Specifically, as in Fig. 7, the controllable edges between $h_{\langle \mathsf{stand} \rangle}$ and $h_{\langle \mathsf{walk} \rangle}$ and the two self-loops complementing them are associated with a probability distribution such that $\mathsf{obey} + \mathsf{disobey} = 1$, where $\mathsf{obey}, \mathsf{disobey} \in K$ are two constants. The values are the probabilities with which, when the orchestrator fires an event over $\mathtt{cmd\_h}_{\text{start}}$ (resp., $\mathtt{cmd\_h}_{\text{stop}}$), the human abides by it and switches to $h_{\langle \mathsf{walk} \rangle}$ (resp., $h_{\langle \mathsf{stand} \rangle}$) or ignores it and stays in the same location. The specific value of obey derives from attribute $\mathsf{p_{fw}}$ of class Human introduced in Section 3 (then, $\mathsf{disobey} = 1 - \mathsf{obey}$ holds). Value $\mathsf{disabled}$ for attribute $\mathsf{p_{fw}}$ implies $\mathsf{obey} = 1$ (hence, $\mathsf{disobey} = 0$).

Manifestations of human free will do not exclusively occur concomitantly with the orchestrator's instructions. As shown in Fig. 7, two additional uncontrollable edges connect $h_{\langle \mathsf{stand} \rangle}$ and $h_{\langle \mathsf{walk} \rangle}$. These edges capture the possibility that humans *may* decide to start or stop walking haphazardly at any time during the execution of the mission. Therefore, these edges are not associated with any event occurring in the system, but they only depend on the value of edge condition $\mathrm{fw} \geq \mathsf{FW_{th}}$, where $\mathsf{FW_{th}} \in K$ is a constant. As per Table 1, the value of dense counter fw is updated every $\mathsf{T_{poll}}$ time units through function $\mathsf{roll\_dice}()$, which generates a random value from $[0, \mathsf{FW_{max}}]$ where $\mathsf{FW_{max}} \in K$ is a constant. The uncontrollable edge fires if, and only if, the generated value of fw is greater or equal than constant $\mathsf{FW_{th}} \leq \mathsf{FW_{max}}$. Parameters $\mathsf{FW_{th}}$ and $\mathsf{FW_{max}}$ derive from attribute $\mathsf{p_{fw}}$ and determine the frequency of human haphazard actions.

4.1.2. Human Leader

The SHA modeling the leader pattern, shown in Fig. 8, shares most features with the model described in Section 4.1.1. Locations $h_{\langle \mathsf{stand} \rangle}$ and $h_{\langle \mathsf{walk} \rangle}$ (within $\langle \mathsf{stand} \rangle\_\mathrm{pub}_h$ and $\langle \mathsf{walk} \rangle\_\mathrm{pub}_h$) capture the human resting and walking constraining real-valued variable F through the flow conditions in Eq.5 and Eq.4. While in these locations, sensor readings are periodically modified by updates ξ_{stand} and ξ_{walk} in Table 1. When fatigue exceeds the maximum threshold, the SHA switches to deadlock location $h_{\langle \mathsf{faint} \rangle}$.

The distinguishing feature of this pattern is that the switch from $h_{\langle \mathsf{stand} \rangle}$ to $h_{\langle \mathsf{walk} \rangle}$ is purely based on the free will mechanism and not on orchestrator's instructions. As a matter of fact, the leader *autonomously* decides when to start the action. Therefore, the edge to $h_{\langle \mathsf{walk} \rangle}$ is not tied to any event fired through any channel. Dense counter fw appearing in the edge condition is periodically randomly updated as described in Section 4.1.1. On the other hand, while the leader is free to also stop walking at any time irrespective of the robot's decisions (through the solid edge from $h_{\langle \mathsf{walk} \rangle}$ to $h_{\langle \mathsf{stand} \rangle}$), the orchestrator may exceptionally instruct the human to stop walking through channel $\mathtt{cmd\_h}_{\text{stop}}$ when their fatigue reaches an alarming value. As with all other orchestrator commands, the edges triggered by such events are probabilistic and depend on

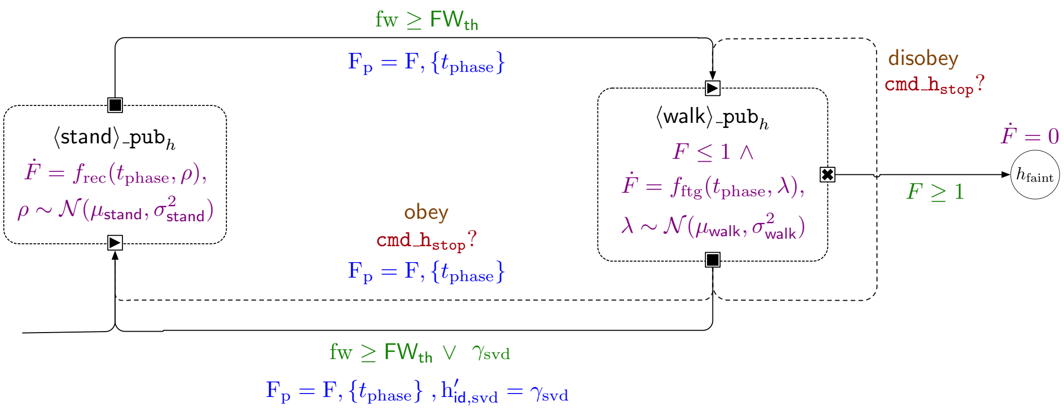


Figure 8: SHA modeling the HumanLeader pattern, color-coded as in Fig. 3.

probabilities obey and disobey (see Fig. 8) governing whether the human abides by the instruction or ignores it and stays in the same location.

Finally, unlike in the follower pattern, the leader marks the end of the service by updating the Boolean dense counter $\mathrm{h}_{\mathsf{id},\mathrm{served}} \in V_{\mathrm{dc}}$, which is also used by the orchestrator to determine whether the robot may move on to serve the following human or stop if the mission is complete. The condition that determines whether the service is complete is indicated as γ_{svd} and corresponds to Formula 7 (see also Fig. 8). The service is considered complete if both the human and the robot are within a specific range of the destination, corresponding to attribute target of class Service in Fig. 4. Dense counters $\mathrm{r}_{\mathrm{pos}_\mathrm{x}}$ and $\mathrm{r}_{\mathrm{pos}_\mathrm{y}}$ represent the Cartesian coordinates of the robot within the layout [15].

$$\sqrt{(\mathrm{h}_{\mathrm{pos}_\mathrm{x}} - \mathsf{target.x})^2 + (\mathrm{h}_{\mathrm{pos}_\mathrm{y}} - \mathsf{target.y})^2} \leq \mathsf{vT}_{\mathsf{poll}} \wedge \sqrt{(\mathrm{h}_{\mathrm{pos}_\mathrm{x}} - \mathrm{r}_{\mathrm{pos}_\mathrm{x}})^2 + (\mathrm{h}_{\mathrm{pos}_\mathrm{y}} - \mathrm{r}_{\mathrm{pos}_\mathrm{y}})^2} \leq \mathsf{vT}_{\mathsf{poll}} \tag{7}$$

As per Fig. 8, when condition γ_{svd} holds and the edge from $h_{\langle\mathsf{walk}\rangle}$ to $h_{\langle\mathsf{stand}\rangle}$ is taken, dense counter $\mathrm{h}_{\mathsf{id},\mathrm{served}}$ is set to 1 (Boolean values are encoded by 0 and 1). If the edge is taken because $\mathrm{fw} \geq \mathsf{FW}_{\mathsf{th}}$ holds, $\mathrm{h}_{\mathsf{id},\mathrm{served}}$ is set with the value of γ_{svd} (that, possibly, can be 0 if the service has not been completed yet).

4.1.3. Human Recipient

The recipient pattern captures a human needing the robot to fetch an object and deliver it back to their current location. While the robot moves to the object's physical location (i.e., attribute target of the corresponding Service) and travels back, the human is free to move around. Therefore, the SHA modeling human behavior for this pattern (shown in Fig. 9) features three operational states, corresponding to as many instances of the $\langle\mathsf{op}\rangle\_\mathrm{pub}_h$ pattern. Instance $\langle\mathsf{stand}\rangle\_\mathrm{pub}_h$ captures the human standing still, as described in Section 4.1.1 and Section 4.1.2. Similarly, $\langle\mathsf{walk}\rangle\_\mathrm{pub}_h$ captures the human walking out of free will while waiting for the robot. Additionally, the recipient pattern features location $h_{\langle\mathsf{exe}\rangle}$ (within pattern $\langle\mathsf{exe}\rangle\_\mathrm{pub}_h$) representing that the robot has reached the human while carrying the object and the human has to collect it. During the handover, neither the robot nor the human can move, thus $\mathcal{F}(h_{\langle\mathsf{exe}\rangle})$ equals f_{rec}. Ordinary location $h_{\langle\mathsf{faint}\rangle}$ captures the human having reached the maximum fatigue level and, as in previously presented patterns, it is endowed with flow condition $\dot{F} = 0$.

While the robot is busy fetching the object, the human can autonomously decide to move at any time. Therefore, the edges from $h_{\langle\mathsf{stand}\rangle}$ to $h_{\langle\mathsf{walk}\rangle}$ and back depend on dense counter fw, which is periodically updated as described in Section 4.1.1. The orchestrator starts the handover when the human is ready to deliver the object to the robot, by firing an event through channel $\mathtt{cmd\_h}_{\mathrm{start}}$. In this case, whether the human is walking (thus, in $\langle\mathsf{walk}\rangle\_\mathrm{pub}_h$) or idle (in $\langle\mathsf{stand}\rangle\_\mathrm{pub}_h$), they receive the instruction through channel $\mathtt{cmd\_h}_{\mathrm{start}}$ to switch to $\langle\mathsf{exe}\rangle\_\mathrm{pub}_h$ for the synchronization phase. As with the previous SHA modeling human behavior, there is a certain probability that the human ignores the orchestrator's commands as dictated by weights obey and disobey.

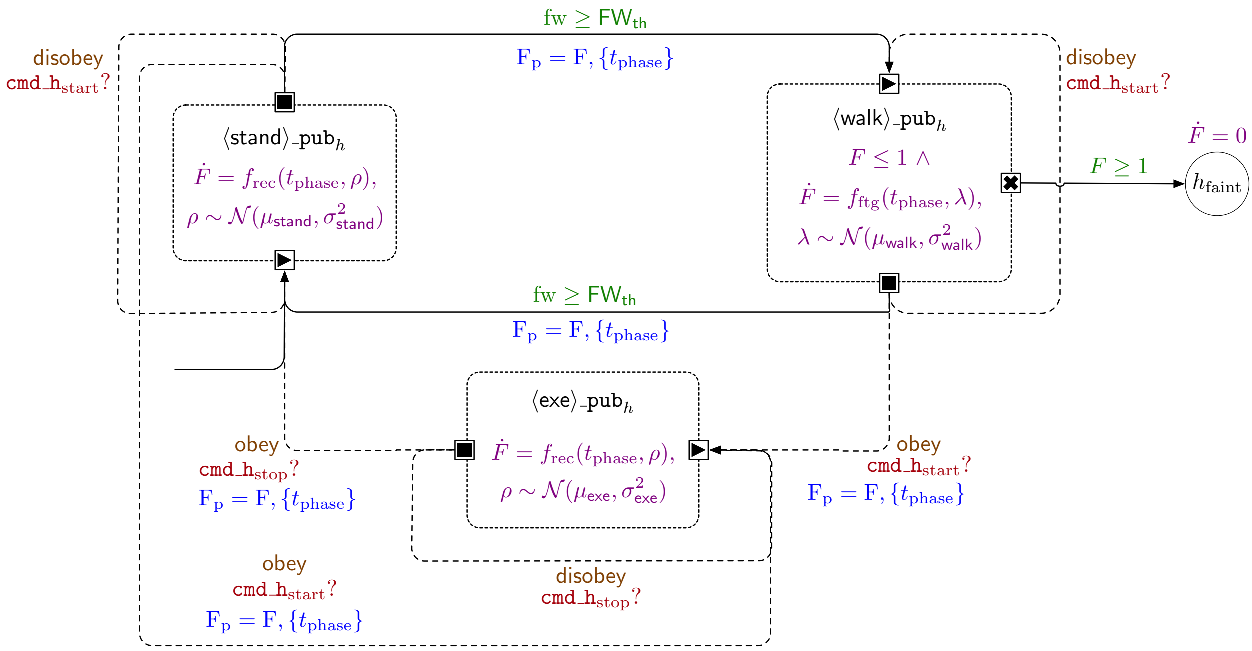


Figure 9: SHA modeling the HumanRecipient pattern, color-coded as in Fig. 3.

The orchestrator gives the human time to pick up the object and then fires an event through $\mathtt{cmd\_h}_{\text{stop}}$ to mark the end of the service, which the human may follow or ignore. On the other hand, no edge governed by variable fw enters or leaves $\langle\mathsf{exe}\rangle\_\mathrm{pub}_h$, since it would not capture "*rational*" behaviors. As a matter of fact, such edge entering $\langle\mathsf{exe}\rangle\_\mathrm{pub}_h$ would capture the human collecting the object before the robot is sufficiently close. Similarly, a free-will edge leaving $\langle\mathsf{exe}\rangle\_\mathrm{pub}_h$ would capture the human deliberately suspending the synchronization phase, possibly dropping the item. The introduction of these categories of erroneous behaviors will be investigated as a further extension of the SHA modeling humans. We remark that, instead, the possibility that the human still needs time to complete the synchronization after command $\mathtt{cmd\_h}_{\text{stop}}$ is issued by the robot (for example, if the item is particularly delicate or bulky) is modeled by the self-loop (i.e., the robot instructs the human to conclude the phase, but they ignore it and prolong the action).

4.2. Robotic System Model

In the following, we briefly recap the main features of SHA $\mathcal{A}_\mathrm{r}$ modeling the robotic platform and present a new SHA $\mathcal{A}_\mathrm{b}$ modeling the robot's battery enhanced with the sensor readings' notification mechanism and a more precise charge/discharge dynamics.

4.2.1. Mobile Robot Model

The robot-modeling SHA is *agnostic* with respect to the specific manufacturer and model since it captures the high-level behavior of a generic mobile robotic platform. The SHA does not capture any aspects related to the hardware and electronic components of the robot. Therefore, the analysis carried out with our framework does not cover the possibility of mechanical failures. Referring to the *levels* identified by Lutz et al. to separate concerns in robotic systems' architectures [35], our model targets the *service* level, i.e., the layer serving as access point to the internal components of the robot. Under these premises, the two main actions a mobile wheeled robotic platform can perform are *moving* (forward or backward) and *rotating*, which is the behavior captured by the developed SHA $\mathcal{A}_\mathrm{r}$. Furthermore, we assume that, while the robot is moving, the linear velocity evolves according to a trapezoidal velocity profile.

$\mathcal{A}_\mathrm{r}$ has four ordinary non-committed locations, capturing the robot's behavior while: 1. idle (location r_idle, also corresponding to the initial location); 2. accelerating (location r_start); 3. moving at maximum speed (location r_mov); 4. turning (location r_turn); 5. decelerating (location r_stop). In $\mathcal{A}_\mathrm{r}$, the robot periodically shares the updated position values while in r_start, r_mov, and r_stop. The robot's coordinates within the floor layout are modeled by two dense-counter variables $\mathrm{r}_{\mathrm{pos}_\mathrm{x}}$ and $\mathrm{r}_{\mathrm{pos}_\mathrm{y}}$, which are periodically updated every T_poll time instants. Interested readers find the graphical representation and detailed description of $\mathcal{A}_\mathrm{r}$ in [15, Section II.C].

4.2.2. Battery Model

Mobile robots are typically powered by a lithium battery, which undergoes *charging* and *discharging* cycles. Therefore, SHA $\mathcal{A}_\mathrm{b}$ modeling the robot's battery, which is presented in this section and shown in Fig. 10, features two ordinary non-committed locations b_dis and b_rech corresponding to the discharge and recharge cycles, respectively, plus a deadlock location b_dead capturing the case in which the battery is fully discharged.

The main physical attribute for a battery is its voltage (representing the charge level), which is modeled by real-valued variable Q. Variable Q is initialized with the initial voltage value $\mathsf{C}_0 \in K$, i.e., an attribute of class Battery introduced in Section 3. Similarly to fatigue F in SHA modeling humans, the temporal dynamics of Q is determined by flow conditions in $\mathcal{F}(b_\mathrm{dis})$ and $\mathcal{F}(b_\mathrm{rech})$, whose integral is shown in Eq.8 and Eq.9, respectively. Compared to [13, 14], flow conditions have been refined to match the behavior of lithium batteries for real robotic devices. As a matter of fact, the entire discharge cycle (from 100% of the voltage capacity to 0%) can be approximated by an exponential curve [36]. Nevertheless, the real device is not operational when the voltage drops below a certain threshold (which can vary depending on the specific battery type and device it is powering), i.e., when the level of charge is not sufficient to power the wheel motors. Letting the battery pack discharge to very low levels (close to 0%) may actually permanently damage it [37]. Therefore, compared to [13, 14], we identified equations governing the evolution of variable Q (shown in Eq.8 and Eq.9) by fitting the discharge/charge curve when the robotic device is operative and can carry out the assigned mission. A cubic function showed a high fit to the real dynamics. Parameters $\mathsf{d}_{0,1,2,3}, \mathsf{r}_{0,1,2,3} \in K$ determining the discharge and recharge curves are fitted based on sensor measurements collected during charge/discharge cycles of the same robotic device. Parameter d_0 is always set to C_0 at the beginning of the scenario.

$$Q(t) = -\mathsf{d}_3 t^3 - 2\mathsf{d}_2 t^2 - \mathsf{d}_1 t + \mathsf{d}_0 \tag{8}$$

$$Q(t) = \mathsf{r}_3 t^3 + 2\mathsf{r}_2 t^2 + \mathsf{r}_1 t + \mathsf{r}_0 \tag{9}$$

The edges from b_dis to b_rech and viceversa both model controllable switches, triggered when the orchestrator fires an event through channels $\mathtt{cmd\_b}_\mathrm{start}$ and $\mathtt{cmd\_b}_\mathrm{stop}$ instructing the robot to start or stop recharging. On the other hand, the switch from b_dis to b_dead is uncontrollable as it occurs when $Q = \mathsf{C}_\mathsf{fail}$ holds, due to invariant $Q \geq \mathsf{C}_\mathsf{fail}$ on b_dis and condition $Q \leq \mathsf{C}_\mathsf{fail}$ on the edge to b_dead, where $\mathsf{C}_\mathsf{fail} \in K$ is an attribute of class Battery (see Section 3). The edge from b_dis to b_rech is also constrained by $Q > \mathsf{C}_\mathsf{fail}$, since the automaton must enter deadlock location b_dead when $Q = \mathsf{C}_\mathsf{fail}$ holds. The edges from b_dis to b_rech, and viceversa, are equipped with updates that initialize d_0 and r_0 from Eq.8 and Eq.9 with the residual charge value from the previous cycle (i.e., the value of dense counter b_chg) and reset clock t_phase.

The battery model features dense counter b_chg, representing the digital counterpart of Q, and a modeling pattern to periodically publish the latest charge measurement governed by clock t_upd (corresponding to the $\langle\mathsf{op}\rangle\_\mathsf{pub}_{\langle\mathsf{id}\rangle}$ pattern presented in [15, Section IV.2]). In $\mathcal{A}_\mathrm{b}$, this occurs while in b_dis and b_rech by switching to committed locations $b_{\mathrm{pub}_\mathrm{d}}$ and $b_{\mathrm{pub}_\mathrm{r}}$, respectively, when $t_\mathrm{upd} = \mathsf{T}_\mathsf{poll}$ holds. Upon these switches, clock t_upd is reset, to begin a new sensor refresh, and dense counter b_chg is updated through updates ξ_dis and ξ_rec. The new value of the battery charge depends on how many cycles lasting T_poll time units have been executed so far, hence how many measurements have been collected. For this reason, automaton $\mathcal{A}_\mathrm{b}$ features a dense counter $\mathrm{k} \in V_\mathrm{dc}$ that keeps track of the number of readings that have been done since the beginning of the scenario. Updates ξ_dis and ξ_rech compute the battery charge at the k-th refresh cycle $Q(\mathrm{k}\mathsf{T}_\mathsf{poll})$. They are obtained by expanding Eq.9 and Eq.8 when t is equal to $(\mathrm{k}-1)\mathsf{T}_\mathsf{poll} + \mathsf{T}_\mathsf{poll}$. Unlike the updates in automata modeling humans, dependency on index k cannot be removed in the equations defining b'_chg. At every sensor

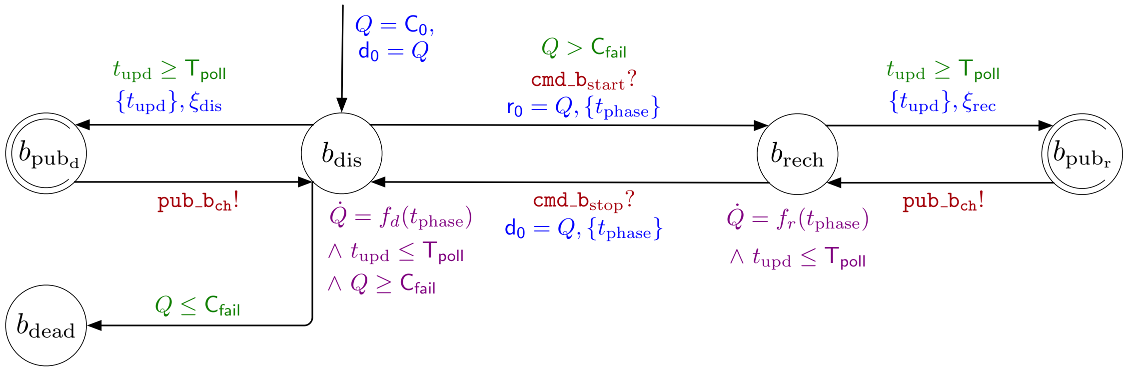


Figure 10: SHA $\mathcal{A}_b$ modeling the robot's battery behavior. Color-coding is the same as in Fig. 3.

Table 2: Updates for the $\mathcal{A}_b$ SHA modeling the robot's battery.

Symbol	Updates	Description
ξ_{dis}	$b'_{chg} = b_{chg} - T_{poll}((d_3 + 6d_3k){T_{poll}}^2 + (d_2 + 2d_2k)T_{poll} + d_1); k = k + 1;$	Discharge phase
ξ_{rec}	$b'_{chg} = b_{chg} + T_{poll}((r_3 + 6r_3k){T_{poll}}^2 + (r_2 + 2r_2k)T_{poll} + r_1); k = k + 1;$	Recharge phase

refresh, dense counter k is incremented. Updates are shown in Table 2 (where b'_{chg} is the new value of b_{chg} after the update). The updated value of b_{chg} is then published by firing an event through channel pub_b_{ch}.

4.3. Orchestrator Model

The orchestrator controls the robot's behavior based on the current state of the system to drive the mission to success. As described in previous sections, the humans, the robot, and its battery share sensor readings with the orchestrator, which checks these values against given policies to determine whether a certain event has to be fired. Specifically, the orchestrator is fully in control of the mobile robot's behavior (i.e., it issues every instruction to start or stop moving), while it issues *suggestions* for the human, e.g., to stop moving when they reach an alarming value of fatigue, which might be dismissed due to human free will. The degree of intrusiveness of how such suggestions are issued to the subjects must be tailored to the specific scenario and the involved subjects' demands; however, given the high-level perspective of the framework, this aspect is currently out-of scope. An abstract representation of the orchestrator SHA is shown in Fig. 11. The orchestrator operational states (the dashed boxes in Fig. 11) are modeled as submachines. All the edges connecting them are defined for events of the form c!, where c is a channel of the network, as the orchestrator proactively triggers suitable actions to govern the evolution of the entire scenario. The orchestrator operational states, i.e., the submachines in it, are described in detail in the following:

1. the orchestrator is in r_{idle} when, given the system's state, no action can start and, thus, the robot is waiting;
2. r_{rech} orchestrates the robot's behavior when it has to move to the recharge station and recharge;
3. r_{lead} controls the start and the end of the movement when, based on the interaction pattern characterizing the service underway, the robot *leads* the action (i.e., for the HumanFollower and HumanRecipient patterns);
4. h_{lead} controls the dual case, in which the movement is initiated by the human (i.e., the HumanLeader pattern);
5. r_{sync} controls the robot during the HumanCompetitor pattern [16];

Table 3: Orchestrator start and stop conditions (γ_{start} and γ_{stop}, respectively) for each submachine (sub.m.) of the orchestrator. Edge condition γ characterizing a submachine x is indicated with notation $\mathsf{x}.\gamma$ (e.g., $\mathsf{r_{rech}}.\gamma_{\text{start}}$).

Sub.m.	**Start condition** (γ_{start})	**Stop condition** (γ_{stop})
$\mathsf{r_{lead}}$	$\mathrm{h_{pattern}} \in \{\mathsf{follower}, \mathsf{recipient}\} \wedge \neg \mathrm{h_{served}} \wedge \mathrm{f} \leq \mathsf{F_{restart}} \wedge \mathrm{b_{chg}} \geq \mathsf{C_{rech}} \wedge \mathrm{dist}(\mathrm{r_{pos}}, \mathrm{h_{pos}}) \leq \mathsf{D_{restart}}$	$\mathrm{f} \geq \mathsf{F_{stop}} \vee \mathrm{b_{chg}} \leq \mathsf{C_{rech}} \vee \left(\mathrm{h_{served}} \wedge \bigvee_{i=1}^{\mathsf{N_h}} \neg \mathrm{h}_{i,\text{served}} \right) \vee \mathrm{dist}(\mathrm{r_{pos}}, \mathrm{h_{pos}}) \geq \mathsf{D_{stop}}$
$\mathsf{h_{lead}}$	$\mathrm{h_{pattern}} \in \{\mathsf{leader}\} \wedge \neg \mathrm{h_{served}} \wedge \mathrm{b_{chg}} \geq \mathsf{C_{rech}} \wedge \mathrm{h}'_{\text{pos}} \neq \mathrm{h_{pos}}$	$\mathrm{f} \geq \mathsf{F_{stop}} \vee \mathrm{b_{chg}} \leq \mathsf{C_{low}} \vee \left(\mathrm{h_{served}} \wedge \bigvee_{i=1}^{\mathsf{N_h}} \neg \mathrm{h}_{i,\text{served}} \right) \vee \mathrm{h}'_{\text{pos}} = \mathrm{h_{pos}}$

6. $\mathsf{hr_{int}}$ controls the robot's behavior while providing a service that requires precise or close-distance synchronization with the human (i.e., the $\mathsf{HumanApplicant}$ or $\mathsf{HumanRescuer}$ patterns) [16].

Submachines in Fig. 11 are endowed with *ports*, intended as in the $\langle \mathsf{op} \rangle\_\mathrm{pub}_h$ pattern. The orchestrator *enters* a submachine through the port marked by symbol "▶", and may *exit* through the ports marked by symbols "■", "×", and "✓", respectively, indicating whether the action has ended (or is momentarily suspended), the mission has ended with failure or with success. Ports highlight the transitions entering and leaving each submachine, constrained by conditions γ_{start}, γ_{stop}, γ_{fail}, and γ_{scs}, each associated with a component-specific formula. The orchestrator enters a submachine when the corresponding γ_{start} condition is true. If either of γ_{stop}, γ_{fail}, or γ_{scs} holds, the orchestrator exits the submachine. Locations o_{fail} and o_{scs} of Fig. 11 correspond to the end of the mission with failure or success, respectively, and are reached when either γ_{fail} (see Formula 10) or γ_{scs} (see Formula 11) holds. Failure occurs if, for at least one of the $\mathsf{N_h} \in K$ subjects in the scenario, human fatigue exceeds 1 (i.e., $\mathrm{f}_i \geq 1$ holds for some i) or the robot's charge drops to a neighborhood of $\mathsf{C_{fail}} \in K$ (i.e., $|\mathrm{b_{chg}} - \mathsf{C_{fail}}| \leq \epsilon$ holds). The latter condition accounts for small fluctuations of the estimated discharge curve.

$$\left(\bigvee_{i=1}^{\mathsf{N_h}} \mathrm{f}_i \geq 1 \right) \vee |\mathrm{b_{chg}} - \mathsf{C_{fail}}| \leq \epsilon \tag{10}$$

Location o_{scs} is reached when the mission has been successfully completed—i.e., when all humans in the scenario have been served: when a human in the scenario with $\mathsf{id} = i$ is served, the Boolean dense counter $\mathrm{h}_{i,\text{served}} \in V_{\text{dc}}$ is set to true.

$$\bigwedge_{i=1}^{\mathsf{N_h}} \mathrm{h}_{i,\text{served}} \tag{11}$$

We recall that the main expression whose value we calculate through SMC is $\mathbb{P}_{\text{M}}(\diamond_{\leq \tau}\ \mathsf{scs})$, where Boolean dense counter scs is set to true upon entering location o_{scs} (thus, when the condition in Formula 11 holds). As per Fig. 11, failure is possible for all submachines. On the other hand, only $\mathsf{r_{lead}}$, $\mathsf{h_{lead}}$, $\mathsf{r_{sync}}$, and $\mathsf{hr_{int}}$ have outgoing transitions towards o_{scs}, since recharging the robot does not impact service provision (thus, progress towards mission completion).

Submachines $\mathsf{r_{idle}}$ and $\mathsf{r_{rech}}$ are presented in detail in [13], while $\mathsf{r_{sync}}$ and $\mathsf{hr_{int}}$ are introduced in [16]. Submachines $\mathsf{r_{lead}}$ and $\mathsf{h_{lead}}$ are briefly recapped in the following, as they handle interaction patterns covered in this paper and subject to the experimental validation process in Section 6. Table 3 contains the formulae for the start (γ_{start}) and stop (γ_{stop}) conditions of these submachines. Since the mission is a sequence of services involving a human agent, the orchestrator uses a dense counter $\mathrm{curr} \in [1, \mathsf{N_h}]$ to store the id of the currently served human. Its value is updated by the orchestrator every time a service ends, and the next one can start. The dense counters f, $\mathrm{h_{pattern}}$, $\mathrm{h_{served}} \in V_{\text{dc}}$ and $\mathrm{h_{pos}} \in V_{\text{dc}}$ keep track of the fatigue of the currently served human, the required interaction pattern, the completion of the service and the position of the human, respectively ($\mathrm{h_{pos}}$ is a shorthand representing a pair of coordinates); e.g., $\mathrm{f} = \mathrm{f}_i$ holds if i is equal to curr. Fig. 11 highlights the channels through which the orchestrator fires instructions when entering or

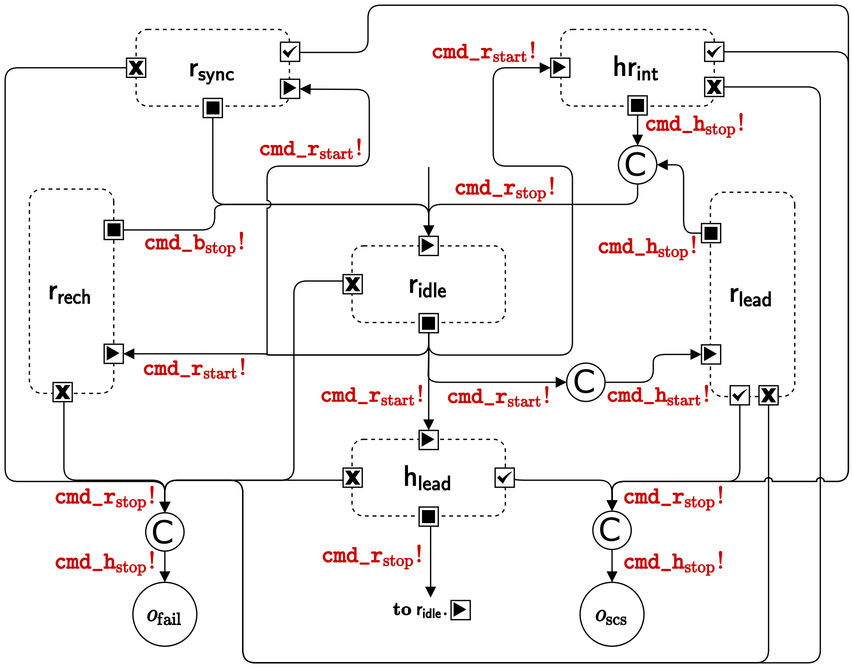


Figure 11: Orchestrator SHA, as seen in [16]. Submachines are represented as dashed boxes, with ports marked by symbols "▶", "■", "✓", and "×".

leaving a submachine. For the sake of clarity, if a is a submachine, e.g., $\mathsf{r_{lead}}$, and g is the condition on an edge through a port, e.g., γ_{start}, then we refer to g by writing $a.g$.

The orchestrator enters submachine $\mathsf{r_{lead}}$ to initiate the robot movement when the robot *leads* the action. As per Table 3, the $\mathsf{r_{lead}}.\gamma_{\text{start}}$ condition holds for the follower and recipient patterns. The robot begins assisting the currently served human if they are sufficiently close and the service has yet to be completed. Furthermore, for safety purposes, the action can start only if human fatigue is sufficiently low (less than $\mathsf{F_{restart}} \in K$) and the robot has sufficient charge (greater than $\mathsf{C_{rech}} \in K$). Upon entering $\mathsf{r_{lead}}$, the orchestrator fires $\texttt{cmd\_r}_{\text{start}}$ and $\texttt{cmd\_h}_{\text{start}}$ for the robot to start moving and the human to follow. The robot stops moving (events $\texttt{cmd\_r}_{\text{stop}}$ and $\texttt{cmd\_h}_{\text{stop}}$ fire) if either one of the following conditions holds: (a) human fatigue f exceeds a maximum tolerable value $\mathsf{F_{stop}} \in K$ (we recall that if the human faints, i.e., their fatigue is 1, the mission fails; hence, $\mathsf{F_{stop}}$ should be lower than 1 to allow the orchestrator to prevent failures caused by fainting); (b) battery charge drops below a value $\mathsf{C_{rech}}$ that calls for recharging; (c) the human has been served, but they were not the last one (if they were the last one, the mission would be complete); (d) the distance between the robot and the human is too large (greater than $\mathsf{D_{stop}} \in K$), indicating that the human has stayed behind and needs to get closer to the robot to proceed with the service.

The $\mathsf{h_{lead}}$ submachine controls the robot's behavior when the human leads the action (i.e., with the *leader* interaction pattern). As per Table 3, the orchestrator enters $\mathsf{h_{lead}}$ (i.e., $\mathsf{h_{lead}}.\gamma_{\text{start}}$ holds) if : (a) the currently assisted human is a leader; (b) the service has not been completed; (c) the robot is sufficiently charged; (d) the human is moving (their current position h'_{pos} is different from the previous sensor reading h_{pos}). Upon entering $\mathsf{h_{lead}}$, $\texttt{cmd\_r}_{\text{start}}$ is triggered and the robot starts following the human. The orchestrator exits $\mathsf{h_{lead}}$ (i.e., $\mathsf{h_{lead}}.\gamma_{\text{stop}}$ holds) when: (a) the currently served human has reached an excessive fatigue level ($\mathrm{f} \geq \mathsf{F_{stop}}$); (b) the robot's battery charge has dropped below the recharge threshold ($\mathrm{b}_{\text{chg}} \leq \mathsf{C_{rech}}$); (c) the human has set themselves as served, but there are other humans to serve in the scenario; (d) the human has stopped moving (their current position h'_{pos} is the same as the previous sensor reading and the service is yet to be concluded, hence the robots waits for the human to walk again). When γ_{stop} holds, the orchestrator stops the robot through channel $\texttt{cmd\_r}_{\text{stop}}$ and instructs the human to stop walking only if they are excessively fatigued. As explained in Section 4.1.2, the human may ignore the orchestrator instruction out of free will.

5. Scenario Deployment and Reconfiguration

In the following, we summarize the main features of the deployment and reconfiguration phases of the framework (phases **2** and **3** in Fig. 2) to keep the presentation of the validation process in Section 6 self-contained. We refer the interested reader to [15] for the complete description of the deployment approach.

5.1. Scenario Deployment

The goal of the deployment phase is to run the robotic application in a real environment or simulate it with a realistic physics engine. Also, deployment helps the application designer extract valuable information about the missions and, possibly, drive a reconfiguration of the scenario, since real executions are available from the scene. In both cases, executable software is built to run the application. The approach allows for a *hybrid* deployment environment adhering to the digital-twin paradigm [40] with a real robotic device in the physical environment interacting with human avatars in the virtual environment. When simulation is performed, the virtual agents can be controlled by means of specific software components that the simulator executes to manifest the agents' behavior in the virtual scene. Advanced simulator environments also offer rich control dashboards that render graphically the virtual 3D scene and allow the user of the simulator to interact with it through input devices while the scene develops. In our framework, to ensure that the deployed orchestrator enforces the same policies as in the formal model and that, in case of simulation, the virtual agents behave correspondingly to their respective SHA, a model-to-code mapping principle converts every SHA into an executable deployment unit. The latter consists of the executable orchestrator script or the scripts governing the agents' behavior—either the humans or the robot—within the virtual scene. As the presence of humans is not always guaranteed and, when human agents are patients in distress, even discouraged, the application designer performing the simulation directly controls human avatars within the virtual scene to make them act like real humans in real-world scenarios. The framework allows the designer to issue commands to the avatars by means of input devices, such as the keyboard, through the scripts which control them. The human actions modeled with the automata are mapped to keys; keystrokes performed by the application designer are interpreted by the scripts and then rendered in the scene. To replicate physical fatigue sensors in the simulated environment and the stochastic behavior of fatigue rates, scripts extract a random sample from a pool of publicly available electromyography signals and estimate fatigue curves using the technique described in [38, 39].

The deployed orchestrator and the agents communicate over a network of ROS publisher and subscriber nodes [41] (the "Middleware Layer" in Fig. 2). Each automaton described in Section 4 corresponds to a deployment unit (e.g., $\mathcal{A}_{\mathrm{O}}$ maps to the executable orchestrator and $\mathcal{A}_{\mathrm{R}}$ to the robot). The firing of an event through channels in set C in the formal model (of which Fig. 11 shows an overview) corresponds to the publication of a message on a ROS topic. More specifically, the deployment unit corresponding to the "*sender*" SHA (i.e., the one with the edge labeled with $\mathtt{c}!$ with $\mathtt{c} \in C$) is the publisher node, whereas the "*receiver*" SHA (i.e., with the edge labeled with $\mathtt{c}?$) is the subscriber node.

For each run or simulation of the application, system logs are collected and processed to be examined by the designer for the reconfiguration phase. Specifically, all data that sensors (either real or simulated) publish through ROS nodes (i.e., the robot's battery and position values, and all humans' fatigue and position values) are stored to be examined. The robot carries out the mission by providing services in sequence. The orchestrator logs relevant events concerning the advancement of each service: when it begins, when it is completed, when it has to be interrupted and why (either the human is too tired or the battery is too low), whether the entire mission ends in failure and the source of the failure. Data logged by the orchestrator are necessary to assess whether the deployed mission has ended with success.

5.2. Scenario Reconfiguration

The reconfiguration phase begins by processing data collected during the deployment to extract metrics comparable with those calculated at design-time. An example is the success rate observed during the deployment phase to be compared with the value of $\mathbb{P}_M(\psi)$, with ψ expressing the success of the mission. The designer performs this comparative analysis to assess whether unexpected contingencies emerged at runtime. If deploying the application highlights unforeseen critical situations, the designer may decide to **reconfigure** the scenario and iterate all the phases of the workflow in Fig. 2 with the new configuration.

Data analysis from deployment and reconfiguration may be necessary as SHA modeling human behavior have stochastic features that are necessarily an *approximation* of the behavior observable in reality. On the other hand, the framework targets the *service* level of robotic systems' architectures [35], as it focuses on the workflow of the mission rather than on aspects related to hardware or structural components: automata modeling the robot, its battery, and the orchestrator do not thus have stochastic features (i.e., in $\mathcal{A}_{\mathrm{r}}$, $\mathcal{A}_{\mathrm{b}}$, and $\mathcal{A}_{\mathrm{o}}$, function $\mathcal{D}(l)$ is undefined for all $l \in L$ and all edges have probability weight 1). Possible reconfiguration measures include:

RM1. Assigning a different **robot** to the task, if the facility has more than one available device. It may not be feasible for a human subject participating in a service to change their starting position due to facility policies (e.g., patients necessarily start in the waiting room). On the other hand, two robotic devices in a fleet may differ either because of their starting position or initial battery charge. In both cases, this reconfiguration measure can cut down the overall duration of the mission. In the first case, the robot may require less time to reach the first human to serve. In the second case, the robot may take less time to recharge, or skip recharging entirely while carrying out the mission.

RM2. Changing the **order** in which humans are to be served. Note that this is only possible if there are no logical *dependencies* between the services being swapped. This measure can reduce the overall duration of the mission if the robot has to cover a smaller distance between services and the maximum level of fatigue reached by human subjects (thus, impacting their wellbeing), for example if a patient has more time to rest between two services.

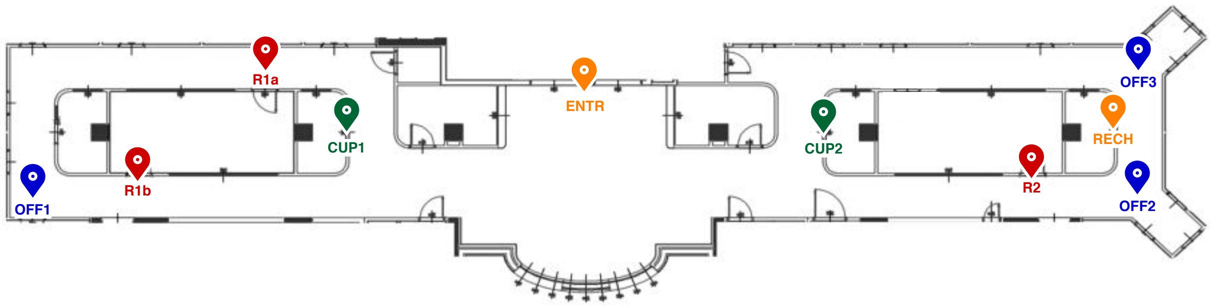
RM3. Changing the **target** of a pattern, if feasible and compliant with the facility policies. An example is a patient following the robot directly to the doctor's office without going through the waiting room first. This reconfiguration measure can reduce both the duration of the mission and the level of fatigue reached by the human subjects. As a matter of fact, reducing the active time leads to a decrease in the fatigue endured, since, during a fatigue cycle, time is the only variable in Eq.2.

RM4. Modifying the orchestrator's **thresholds**. For example, it is possible to reduce the charge threshold at which the robot is instructed to move to the recharge station or the fatigue level at which the orchestrator instructs the human to stop walking. The designer must handle the trade-off between the decrease in mission duration and the increase in probability of failure (for example, due to battery degradation).

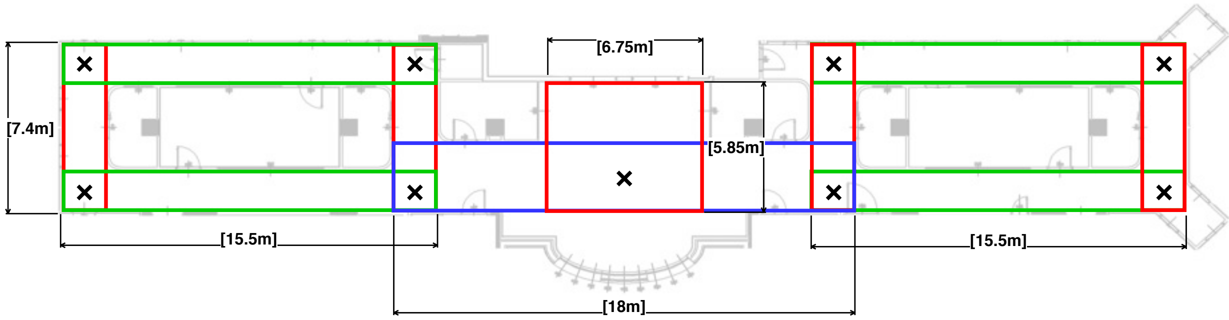
The cyclical nature of the framework allows the analyst to modify the scenario and perform multiple iterations of the analysis until verification results are deemed acceptable. The framework also supports the designer in terms of which and how many parameters of the scenario require manual specification. Parameters concerning the robot (i.e., speed and acceleration) and battery behavior (i.e., charge and discharge rates) are provided by the framework to decrease the manual effort required on the designer's side. Designers manually specify parameters concerning the specific robotic mission (i.e., how many humans are involved and the service they request) whose value cannot be known a-priori by the framework. Should the tuning of such parameters result overly cumbersome for the practitioner, splitting the mission into smaller sequences to be individually analised is feasible.

6. Experimental Validation

This section presents the results obtained while validating the presented approach on case studies inspired by the healthcare setting. The validation process addresses the following questions:



(a) Points of interest within the experimental setting: the entrance and recharge station are in orange, cupboards are in green (CUP1 and CUP2), examination/waiting room entrances are in red (R1a, R1b, and R2), and doctors' offices in blue (OFF1, OFF2, and OFF3).



(b) Representation of the layout abstraction as a set of rectangular areas, with wall sizes ([m]) and intersection points between areas marked by × symbols.

Figure 12: Layout used for the experimental validation.

G1. Is the formal model presented in Section 4 adherent to the physical robotic system and how accurate are SMC results?

G2. Is the model-driven framework described in Section 3 and Section 5 practical and useful while developing interactive scenarios with multiple subjects and services? More specifically, we evaluate:

(a) how the DSL supports designers in configuring complex scenarios;

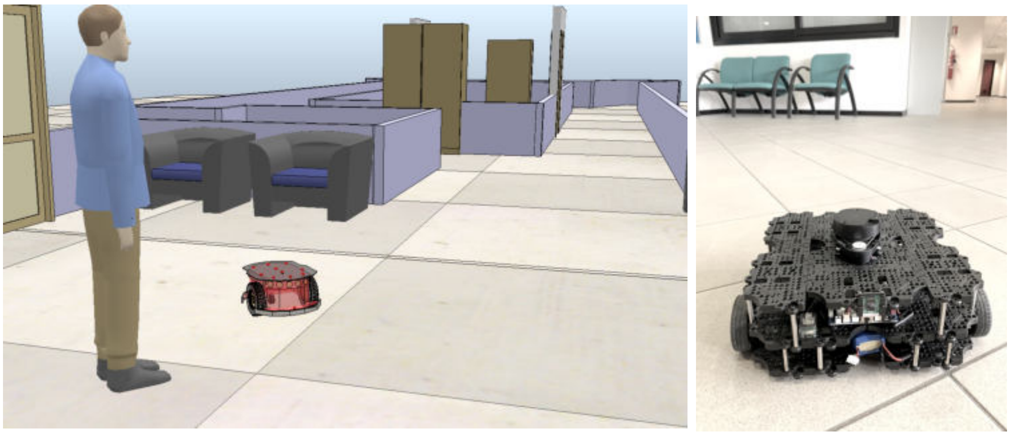
(b) how the design-time analysis phase provides reliable and valuable insights into the modeled scenario;

(c) how scenarios can be reconfigured to improve key indicators (the probability of success and estimated fatigue level of human subjects).

Both validation phases have been carried out following the framework workflow in Fig. 2. Firstly, we model the scenarios through the DSL described in Section 3. The case studies feature one mobile robot providing services (in compliance with the patterns presented in Section 3.1) to one or multiple humans. Agents operate within the floor layout represented in Fig. 12, corresponding to the third floor of Building 22 of Politecnico di Milano. Specifically, Fig. 12a highlights the POIs, while Fig. 12b shows how the real layout is abstracted as a set of rectangular areas as described in Section 3.2.1. While the physical layout where the robotic device moves is a university building, its areas are repurposed in the simulation environment to reproduce a healthcare setting. The layout features a main entrance ENTR where the robot meets patients to assist them and two side aisles, each with cupboards containing medical kits (CUP1 and CUP2), two rooms serving either as waiting rooms for the patients or examination rooms where doctors administer medications (R1a, R1b, and R2), and doctors' offices (OFF1, OFF2, and OFF3).

DSL models[5] are automatically converted into a JSON file and finally into an Uppaal model implementing

[5]The DSL sources are available at:https://github.com/LesLivia/hri_dsl.



(a) Portion of the simulation scene with the simulated human and the *phantom* robot replicating the real robot's behavior.

(b) Real TurtleBot3 operating in the physical environment and reacting to the human in the simulation scene.

Figure 13: Hybrid real/simulated deployment environment.

the SHA network described in Section 4 modeling the specific scenario.[6] The framework also automatically sets up and runs the SMC experiment. For the case studies discussed in this section, we perform SMC through Uppaal v.4.1.26 on a machine running macOS v.10.15.7 with 4 cores and 8GB of RAM.

All case studies are subsequently deployed as described in Section 5.1 [15].[7] We have adopted the digital-twin deployment pattern [42] (see Fig. 13) with a real mobile robot operating in the physical environment (shown in Fig. 13b) and reacting to virtual human subjects in the simulation scene (of which a portion is shown in Fig. 13a). The hybrid deployment environment allows us to to verify the adherence of the robotic system's model and the orchestrator's efficacy with a real device while also performing several runs with (virtual) subjects exhibiting critical fatigue profiles. Electromyography signals serving as dataset to simulate fatigue curves in the simulation environment are provided by [43]. The mobile device is a TurtleBot3 Waffle Pi.[8] Scenarios are deployed using V-REP v.3.6.2 for the simulation scene, Python v.3.6.9 for the orchestrator script, and ROS Melodic to communicate with virtual agents and the TurtleBot3 [41]. The deployment software tools run on a single machine running Ubuntu v.18.04 with 2 cores and 4GB of RAM.

6.1. Formal Model Validation

The purpose of the hereby presented experiments is to assess the accuracy of the formal model presented in Section 4 and of the SMC results (validation goal **G1**). To this end, we perform the design-time analysis as presented in Section 3 on three scenarios, referred to as HF ("Human Follower"), HL ("Human Leader") and LB ("Low Battery"), all taking place in the experimental setup in Fig. 12 and described in detail in Table 4. The three scenarios are structured to validate the main features of the formal model: human-robot dyadic leader/follower dynamics, human fatigue model, robot's battery model, and orchestrator policies. Therefore, the three experiments test the formal model with both critical (low battery in LB and high fatigue in HL) and non-critical elements (high battery in HF/HL and low fatigue in HF/LB).

We perform SMC with decreasing values of time bound τ to estimate the success probability of the three scenarios (i.e., query with QueryType P_SCS). This corresponds to the value of expression $\mathbb{P}_M(\Diamond_{\leq\tau}\mathsf{scs})$, where M is the SHA network modeling each mission. SMC experiments are performed in Uppaal with

[6]The software tool to automatically generate the Uppaal model is available at: https://github.com/LesLivia/hri_designtime.

[7]The software tool implementing the deployment approach is available at: https://github.com/LesLivia/hri_deployment

[8]Full documentation and technical specifications available at: https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/.

Table 4: Scenarios used for the formal model validation phase (abbreviation and detailed description). For each service, we indicate the starting location of the human and the target location as START $\rightarrow$ TARGET.

SCENARIO	DESCRIPTION	MISSION
HF	The robot (Tbot) *leads* the human (H1) from OFF1 to CUP1. The robot is sufficiently charged to complete the mission, and the human exhibits a non-critical fatigue profile (Young/Healthy).	H1 Follower OFF1 $\rightarrow$ CUP1
HL	The robot (Tbot) *follows* the human (H1) from OFF1 to CUP1. The robot is sufficiently charged to complete the mission, and the human exhibits a critical fatigue profile (Elderly/Sick).	H1 Leader OFF1 $\rightarrow$ CUP1
LB	The robot (Tbot) *leads* the human (H1) from OFF1 to CUP1. The robot gets fully discharged during the mission, while the human exhibits a non-critical fatigue profile (Young/Healthy).	H1 Follower OFF1 $\rightarrow$ CUP1

Algorithm 1 Estimation of the the success probability CI for a set of deployment traces $\mathcal{DT}$.

Input: $\mathcal{DT}$, τ, $\mathsf{N_h}$, $\mathsf{T_{int}}$, α

1: $\mathcal{DT}_{\text{scs}} \leftarrow \emptyset$
2: **for** $dt \in \mathcal{DT}$ **do**
3: $\quad \mathcal{SVD}_{dt} \leftarrow \{t | t \in \mathbb{N} \wedge i \in [1, \mathsf{N_h}] \wedge \mathrm{h}_{i,\text{svd}} \in dt(t)\}$ ▷ Timestamps corresponding to the end of a service.
4: $\quad$ **if** $|\mathcal{SVD}| = \mathsf{N_h} \wedge \mathsf{max}(\mathcal{SVD}) \leq \tau - \mathsf{T_{int}}$ **then**
5: $\quad\quad \mathcal{DT}_{\text{scs}} \leftarrow \mathcal{DT}_{\text{scs}} \cup \{dt\}$ ▷ All humans have been served within $\tau - \mathsf{T_{int}}$.
6: $\quad$ **end if**
7: **end for**
8: $p_l \leftarrow \mathsf{ppf}(\alpha/2, |\mathcal{DT}_{\text{scs}}|, |\mathcal{DT}| - |\mathcal{DT}_{\text{scs}}| + 1)$
9: $p_u \leftarrow \mathsf{ppf}(1 - \alpha/2, |\mathcal{DT}_{\text{scs}}| + 1, |\mathcal{DT}| - |\mathcal{DT}_{\text{scs}}|)$
10: $\epsilon \leftarrow (p_u - p_l)/2$
11: $p \leftarrow p_l + \epsilon$

Output: p, ϵ

default statistical parameters, thus with $\epsilon = \alpha = 0.05$. We also estimate the maximum human fatigue value (expression $\mathrm{E}_{M,\tau}[\mathsf{max}(\mathrm{f})]$), and the robot's residual charge at the end of the mission (expression $\mathrm{E}_{M,\tau}[\mathsf{min}(\mathrm{b}_{\text{chg},\%})]$), i.e., queries with QueryType E_FTG and E_CHG, respectively.

Subsequently, we deploy the three scenarios in the digital-twin setting (see Fig. 13) to collect runtime observations, compute the same metrics, and compare the results with those obtained at design-time. To this end, we apply a partial replication of SMC (summarized by Algorithm 1) to the traces collected through deployment to estimate the success probability range observed at runtime. We refer to the simulation log and sensor logs collected during a single run (described in Section 5.1) as **deployment trace**. Given deployment trace dt, we indicate as $dt(t)$ the set of data (sensor readings and milestones recorded by the orchestrator, if any) logged at time $t \in \mathbb{N}$. Since the orchestrator records the timestamp at which each human is served, it is possible to infer from a deployment trace dt whether the mission ended successfully. If human $i \in [1, \mathsf{N_h}]$ has been served in trace dt, there exists $t \in \mathbb{N}$ such that $\mathrm{h}_{i,\text{svd}} \in dt(t)$ holds. We indicate as $\mathcal{DT}$ the set of all deployment traces collected for a given scenario. Set $\mathcal{SVD}_{dt} = \{t | t \in \mathbb{N} \wedge i \in [1, \mathsf{N_h}] \wedge \mathrm{h}_{i,\text{svd}} \in dt(t)\}$ at Line 3 in Algorithm 1 contains the timestamps corresponding to the completion of a service in a specific trace dt. Similar to SMC, given a time-bound τ and the set of collected deployment traces $\mathcal{DT}$, for each deployment trace $dt \in \mathcal{DT}$ we check whether the mission has ended with success within τ (i.e., whether $\diamond_{\leq \tau}$ scs holds for dt). Algorithm 1 checks through the condition on Line 4 whether set $\mathcal{SVD}_{dt}$ has $\mathsf{N_h}$ elements (i.e., *all* humans have been served) and the maximum of $\mathcal{SVD}_{dt}$ is smaller than $\tau - \mathsf{T_{int}}$ (i.e., the *last* human to be served has been served within the time bound minus the time required by the orchestrator to process the information). If condition on Line 4 is verified, trace dt constitutes a success and is added to set $\mathcal{DT}_{\text{scs}}$ by

Table 5: Comparison between the results obtained through SMC at design time (DT) and the results obtained by deploying the three model validation scenarios (DEPL). For decreasing values of time bound τ ([s]), the table contains the verification time ([min]), the probability CI estimated through Uppaal, the CI observed at runtime, and the runs necessary to compute such estimations. The table also contains the estimated maximum human fatigue values ([$0-1$]) and minimum charge levels ([%]). For each metric, configurations leading to the least accurate results are highlighted in grey.

SC.	τ	**Ver.Time**	**Success Probability**		**Runs**		**Max. Fatigue**		**Min. Charge**	
		[min]	DT	DEPL	DT	DEPL	DT	DEPL	DT	DEPL
HF	75	0.41	0.952 ± 0.05	0.950 ± 0.05	29	110	0.0208 ± 0.004	0.0211 ± 0.007	95.54%	95.75%
	53	3.42	0.859 ± 0.05	0.865 ± 0.06	199	110	0.0177 ± 0.005	0.0179 ± 0.008	96.86%	96.27%
	50	3.23	0.812 ± 0.05	0.800 ± 0.07	250	110	0.0167 ± 0.003	0.0168 ± 0.008	97.05%	96.51%
	40	4.25	0.433 ± 0.05	0.500 ± 0.10	395	110	0.0125 ± 0.004	0.0125 ± 0.008	97.65%	97.11%
	34	2.19	0.239 ± 0.05	0.252 ± 0.08	296	110	0.0112 ± 0.003	0.0114 ± 0.003	98.01%	98.10%
HL	50	0.31	0.952 ± 0.05	0.950 ± 0.05	29	120	0.2015 ± 0.037	0.1939 ± 0.050	73.71%	73.32%
	42	2.72	0.826 ± 0.05	0.840 ± 0.07	236	120	0.1763 ± 0.038	0.1623 ± 0.051	74.19%	74.17%
	38	2.46	0.676 ± 0.05	0.664 ± 0.09	354	120	0.1599 ± 0.014	0.1577 ± 0.025	74.43%	74.38%
	35	3.94	0.409 ± 0.05	0.395 ± 0.09	389	120	0.1545 ± 0.042	0.1561 ± 0.027	74.61%	74.55%
	33	2.13	0.216 ± 0.05	0.208 ± 0.07	277	120	0.1451 ± 0.045	0.1434 ± 0.025	74.74%	74.91%
LB	150	1.02	0.000 ± 0.05	0.000 ± 0.05	29	107	–	–	0.001%	−0.001%

the instruction on Line 5.

Algorithm 1 computes $\mathbb{P}_{\mathcal{DT}}(\diamond_{\leq \tau}\ \mathsf{scs})$ in terms of a Bayesian Confidence Interval (CI) of the form $p \pm \epsilon$ with confidence level $1-\alpha$. We adopt the Clopper-Pearson approach for binomial distributions to compute the CI as it is also exploited by the Uppaal tool. Specifically, $p_l = p - \epsilon$ can be calculated as the α-quantile of a Beta distribution with parameters $\mathsf{successes}$ and $\mathsf{trials} - \mathsf{successes} + 1$ (Line 8), while $p_u = p + \epsilon$ can be calculated as the γ-quantile with $\gamma = 1 - \alpha$ and parameters $\mathsf{successes} + 1$ and $\mathsf{trials} - \mathsf{successes}$ (Line 9) [45].[9] Unlike point estimator $\mathsf{successes}/\mathsf{trials}$, this procedure also provides an insight into the variability of the success rate (i.e., the value of ϵ) and how its value changes as more runs are performed.

The results of the SMC experiments, the time and runs necessary to complete it with the required level of confidence, and the fatigue and charge estimations are reported in Table 5 (marked as DT, "Design Time"). The success probability ranges estimated for scenarios HF, HL, and LB through Algorithm 1 are reported in Table 5 (marked as DEPL, "Deployment").

Results in Table 5 corroborate the intuition that, for decreasing values of τ, the probability of success decreases both at design time and during deployment. Experimental results with the largest difference between design-time and deployment estimations are highlighted in grey. We select values of parameter τ to be displayed in Table 5 corresponding to probability ranges in three macro-intervals: *high* success probability (i.e., with $p > 80\%$), *average* success probability (i.e., with $40\% < p < 70\%$), and *low* success probability (i.e., with $p < 25\%$). Scenarios HF and HL require 75s and 50s, respectively, to end successfully with probability approximately equal to 1, while it drops to approximately 20% when the analysis is bounded to 34s and 33s. The variability of the success probability between runs within 34s (for HF) or 33s (for HL) and those requiring up to 75s is due to the human stopping haphazardly during the mission as described in Section 4.1, causing a delay in the completion of the mission. As shown in Table 5, the configurations with the largest difference between design-time and runtime estimations of the success probability (highlighted in grey) are also the ones requiring the largest number of traces for the SMC experiment (395 and 389 compared to the 110 and 120 performed in the physical setting). This is due to how $\mathbb{P}_M(\diamond_{\leq \tau}\ \mathsf{scs})$ and $\mathbb{P}_{\mathcal{DT}}(\diamond_{\leq \tau}\ \mathsf{scs})$ are calculated: if traces that have been generated or collected are consistent with each other (e.g., they are all successful), it takes a smaller set of traces to obtain a certain value of ϵ than when the number of successes fluctuates. Indeed, the estimated success probabilities resulting from the configurations requiring the largest number of runs (395 for HF and 389 for HL) are also the closest to 50%. In the worst case, the values of p estimated at

[9]The Python implementation exploits the scipy.stats.distributions.beta.ppf function (referred to as ppf in Algorithm 1) from the SciPy library to calculate the required quantiles. Full documentation available at: https://docs.scipy.org.

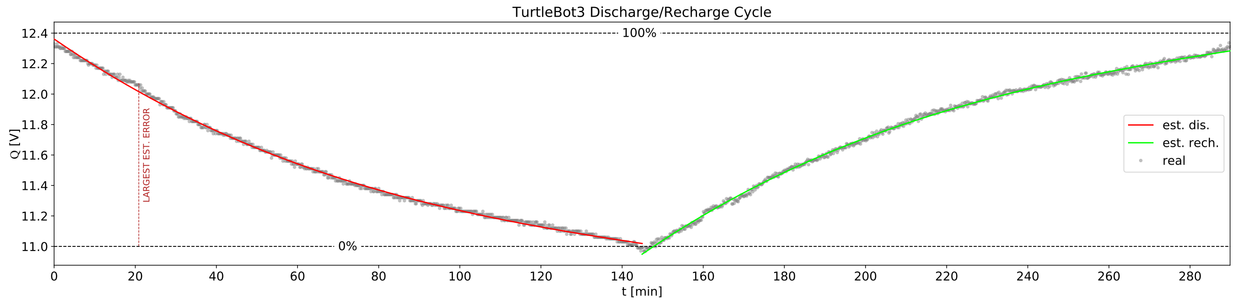


Figure 14: Graph representing the battery voltage (Q [V]) evolution during a complete TurtleBot3 discharge/recharge cycle (approximately 140min each). Dots represent real voltage sensor readings. The red and green lines are the fitted discharge and recharge curve, respectively. Black dashed lines mark the voltage values corresponding to 100% (about 12.4V) of the charge and 0% (11.0V). The red dashed line marks the time instant where the design-time estimation is the least accurate.

design-time and runtime differ by 6.7% (for HF) and 1.4% (for HL) while this drops to 0.2% in the best case.

Data collected through the three scenarios are also necessary to assess whether the formal model accurately captures the robot's battery voltage drop (i.e., the battery discharging while the robot is operative). To estimate the expected value of the minimum charge for the same decreasing values of τ used for the success probability ranges, we calculate the value of expression $\mathrm{E}_{M,\tau}[\mathsf{min}(\mathrm{b}_{\mathrm{chg},\%})]$ in Uppaal (column DT) and the average of minimum values logged for each deployment trace $\mathrm{E}_{\mathcal{DT},\tau}[\mathsf{min}(\mathrm{b}_{\mathrm{chg},\%})]$ (column DEPL). Table 5 reports the battery charge percentage estimations at time τ for the three scenarios, highlighting, as in previous cases, the configurations leading to the largest estimation error. We recall that the differential equations obtained by deriving Eq.8 and Eq.9 constrain the battery voltage ([V]), whose value in the real system is directly measured through a sensor. The percentages shown in Table 5 are calculated according to Eq.12, where $\mathrm{b}_{\mathrm{chg}}$ represents the dense counter presented in Section 4.2.2 (for column DT) or the value shared by the real sensor (for column DEPL), $\mathsf{C}_{\mathsf{fail}}$ is the lowest voltage value allowed by the device (as presented in Section 4.2.2), and $\mathsf{C}_{\mathsf{full}}$ is the (approximate) voltage value when the battery is fully charged.

$$\mathrm{b}_{\mathrm{chg},\%} = \frac{\mathrm{b}_{\mathrm{chg}} - \mathsf{C}_{\mathsf{fail}}}{\mathsf{C}_{\mathsf{full}} - \mathsf{C}_{\mathsf{fail}}} \cdot 100 \tag{12}$$

For the three scenarios, we estimate the residual battery charge, assuming that C_0 is set to 99%, 75%, 0.8% for HF, HL, and LB, respectively. The largest estimation error is 0.61% for HF and 0.53% for HL. Unsurprisingly, charge-related estimations are more accurate than for human fatigue, which has a higher degree of variability and is subject to the human's unforeseeable choices. Nevertheless, the time span required for the scenarios is orders of magnitude shorter than a full battery discharge cycle (approximately 2.5h). Therefore, while these estimations provide insights into how accurate the model is in the range of seconds, we have also assessed its accuracy in the longer run. We have recorded the real battery sensor readings over the course of three full discharge/recharge cycles. This set of data has been used to fit parameters in Eq.8 and Eq.9 governing the evolution of real-valued variable Q in the SHA. Fig. 14 shows the voltage curves (modeled in SHA through real-valued variable Q) resulting from the fitting (red and green lines), compared against the actual sensors readings of a fourth complete discharge/recharge cycle (grey dots). Sensor readings used to fit the curve parameters and those shown in Fig. 14 are different data sets. The largest estimation difference (also highlighted in Fig. 14) is 0.54%, which is comparable to the previously described differences obtained with the three scenarios.

Scenario LB requires a separate analysis. The purpose of this experiment is to assess whether the formal model accurately captures reality in a boundary condition where the robot's charge is insufficient to complete the mission (as previously mentioned, in this case C_0 is 0.8%). When the mission starts, since the charge level is too low ($c \leq \mathsf{C}_{\mathsf{low}}$ holds), the orchestrator immediately instructs the robot to start moving towards the recharge station, which, however, requires about 3.5min to be reached while the robot has only 2.5min of battery life left. The design-time analysis correctly predicts this outcome as the mission has 0% success probability within 150s (see Table 5), and all the collected deployment traces end in failure. No fatigue estimation is provided in this case since the human never starts moving. As a matter of fact, the robot needs to recharge as soon as the mission starts, thus the orchestrator immediately enters submachine $\mathsf{r_{rech}}$ (see Fig. 11) without sending any instructions to the human. As per Formula 10, failure occurs when voltage drops sufficiently close to 0%, which is why the estimation reported in Table 5 is not exacly 0% but the estimated probability of failure is still 1. On the other hand, the negative percentage estimated from deployment traces is due to how the real device works. As soon as the detected battery voltage equals 11V, the device will start emitting an acoustic signal to notify the need to recharge, it will beep for a few seconds and then stop sending power to the motors (thus, no motion is possible). From the moment it starts beeping to the moment it stops moving, the voltage drops slightly *below* 11V, leading to the negative percentage (see Eq.12).

Concerning the estimation of the human fatigue, Table 5 reports the maximum value estimated through Uppaal (column DT) and from deployment traces (column DEPL). To estimate the maximum fatigue expected value at design time, we compute the value of expression $\mathrm{E}_{M,\tau}[\mathsf{max}(\mathrm{f})]$ in Uppaal (Table 5, column DT), whose result is a 95% confidence interval of the requested value [17]. The same procedure is applied to the set of deployment traces $\mathcal{DT}$. For each deployment trace, we calculate the maximum value of fatigue of the human subject within time bound τ. The so-obtained values constitute the set of independent samples, of which we subsequently calculate the 95% confidence level (results are reported in Table 5, column DEPL). In scenario HF, the human has the least critical fatigue profile (Young/Healthy), thus it only reaches a fatigue value of approximately 2%. For this scenario, in the worst case, the largest design-time estimation error (calculated as difference between the average value at design time and observed during deployment) is 1.75%. In HL, the human reaches higher fatigue values (up to approximately 20%). All intervals calculated from deployment traces fall within the range estimated at design-time. In the worst case, also highlighted in Table 5, the estimation error is 8.6%. Although design-time estimations pertaining to fatigue are promising, it is important to remark that the simulator scripts governing human behavior during deployment directly result from the model-to-code transformation (presented in [15]) of the SHA described in Section 4.1. These results, therefore, do not constitute a conclusive empirical proof that SHA modeling humans accurately capture reality. Nevertheless, since simulated sensors share their readings with the orchestrator over actual ROS topics, the limited design time-to-deployment errors indicate that modeling patterns dealing with readings update and publishing (i.e., the pattern in Fig. 6 and the RosPubNode pattern presented in [15]) are reliable.

Concerning performance and scalability, given the limited complexity of the scenarios analysed in this batch of SMC experiments, verification experiments performed through Uppaal last between 0.31 and 4.25min. For the deployment phase, we have performed a total of 337 real runs (110 for HF, 120 for HL, 107 for LB). By factoring in the time required to perform each run, reset the layout to its starting configuration at the end of each run, and the time required to recharge the robot, this corresponds to approximately 64h of non-stop deployment and runtime data collection. Considering that, in a real healthcare facility, employees have multiple tasks to deal with, robots are not actively deployed 100% of the time, and there are time breaks between shifts, the data collection phase would take a longer time.

6.2. Model-Driven Framework Validation

After the analysis on the accuracy of the formal model, we focus on the overall efficacy of the model-driven framework (goal **G2**). To this end, we have analysed 41 real-world scenarios describing service robotic applications extracted from [46, 47, 48, 49, 50, 51]. We remark that, given that service robot deployment is not widespread at the time of writing, there is no structured repository collecting natural language specifications of such scenarios; to the best of the authors' knowledge, the RoboMAX repository (providing 14 of the 41 identified scenarios) is the first attempt in this direction [47]. Therefore, the scenario collection phase has been performed manually by surveying related works in the literature and commercial service robots' documentation. Within the set of eligible scenarios, we consider 14 scenarios to fall outside of the scope of our framework. As our work targets service robot applications featuring mobile robots that *interact* with humans, we consider out-of-scope all scenarios featuring robots that operate autonomously (e.g., performing patrolling or automated room cleaning), are teleoperated, or provide information without expecting any reaction on the human side (e.g., periodical medication reminders). As for the 27 scenarios that are within the scope of the framework, we assess that 24 can be modeled through our framework, leading to a coverage percentage of

Table 6: Scenarios used for the framework validation phase (abbreviation, detailed description, and sequence of services). For each service, we indicate the starting location of the human and the target location as START → TARGET.

SCENARIO	DESCRIPTION	MISSION
DPa	The robot (Tbot) serves a patient-doctor pair (P1/D1, respectively). The robot meets the patient by the entrance (ENTR) and *leads* them to the waiting room (R1b) to wait for the doctor to visit them. The robot *follows* the doctor to CUP1 where they fetch required tools, and *follows* them back (carrying the tools) to the examination room (R2) where the patient will receive the treatment. Finally, the robot returns to R1b and *escorts* the patient to R2 where the doctor is waiting.	P1 Follower ENTR → R1b, D1 Leader R2 → CUP1, D1 Leader CUP1 → R2, P1 Follower Rb1 → R2
DPb	The robot (Tbot) serves a patient-doctor pair (P1/D1, respectively). The robot meets the patient by the entrance (ENTR) and *leads* them to the waiting room (R1a) to wait for the doctor to visit them. The robot approaches CUP2 to retrieve a required medical kit, and then *delivers* it to D1 at OFF2. The robot *follows* the doctor to the examination room (R2) where the patient will receive the treatment. Finally, the robot returns to R1a and *escorts* the patient to R2 to be treated.	P1 Follower ENTR → R1a, D1 Recipient OFF2 ↔ CUP2, D1 Leader OFF2 → R2, P1 Follower R1a → R2
DPc	The robot (Tbot) serves two patient-doctor pairs (P1 and P2 are patients, D1 and D2 are doctors). The robot meets P1 by the entrance (ENTR) and *leads* them to the waiting room (R1a), then it performs the same task for P2 *leading* them from the entrance to R1b. The robot fetches the first required medical kit from CUP1 and *delivers* it to D1 at OFF1. The robot then serves D2 by *following* them to CUP2 and back to their office (OFF3) while carrying the kit. Finally, the robot *leads* P1 to OFF1 and P2 to OFF3 as both doctors are ready to visit them.	P1 Follower ENTR → R1a, P2 Follower ENTR → R1b, D1 Recipient OFF1 ↔ CUP1, D2 Leader OFF3 → CUP2, D2 Leader CUP2 → OFF3, P1 Follower R1a → OFF1, P2 Follower R1b → OFF3

88.8%. The three scenarios that our framework does not cover feature: a) *exoskeletons*, which are considered service robots by standard ISO 13482 [52] (thus, they are in-scope with respect to our framework) but require different software development practices than mobile robots that our framework targets; b) *cognitive* interaction (e.g., comforting children or rehabilitating cognitive skills of patients recovering from strokes), whereas our framework targets *physical coordination* between humans and robots.

To address goal **G2**, we have developed three more sophisticated scenarios, referred to as DPa ("Doctor-Patient"), DPb, and DPc, collecting the most frequent elements of the 24 real-world scenarios found in the literature (fetch-and-delivery tasks, doctor-patients dynamics, patient greeting, and transporting items). The three scenarios are described in detail in Table 6. In all three scenarios, the robot has to serve one (in DPa and DPb) or two (DPc) pairs of human subjects representing a doctor and a patient. The robot always accompanies the patient to the waiting room (R1a, R1b, or R2) first (adhering, thus, to the Follower pattern), and then supports the doctor in retrieving the instrumentation needed to treat the patient. Doctors are either Leaders (D1 in DPa and DPb, and D2 in DPc) or Recipients (D1 in DPc) depending on whether they personally lead the robot to destination or it moves independently and then delivers the resource. For this experimental phase, the robot's charge is always sufficient (the opposite boundary condition has already

Table 7: Results of the DSL2SHA calculation, of the design-time analysis (DT) and of the deployment phase (DEPL) for scenarios DPa, DPb, and DPc. For decreasing values of τ ([s]), the table contains the verification time ([min]), the success probability CI estimated through Uppaal, the mean success rate observed at runtime, the estimated maximum fatigue value for all humans, and the estimated minimum charge value for the robot. Fatigue and charge level are estimated for the maximum value of τ for each scenario. For each metric, configurations leading to the least accurate results are highlighted in grey.

SC.	DSL2SHA	τ	Ver.Time [min]	Success Probability DT	Success Probability DEPL ($\overline{p}$)	HUM.	Max. Fatigue DT	Max. Fatigue DEPL	ROB.	Min. Charge DT	Min. Charge DEPL
DPa	235/863 (27.2%)	400	16.36	0.933 ± 0.05	1.00	P1	0.2664 ± 0.014	0.2385	Tbot	82.4%	82.77%
		350	54.67	0.706 ± 0.05	0.60	D1	0.0372 ± 0.004	0.0332			
		300	59.09	0.419 ± 0.05	0.40						
DPb	235/876 (26.8%)	520	22.69	0.909 ± 0.05	1.00	P1	0.2469 ± 0.009	0.2191	Tbot	80.4%	79.99%
		450	64.29	0.597 ± 0.05	0.50	D1	0.0248 ± 0.007	0.0235			
		400	26.88	0.227 ± 0.05	0.20						
DPc	283/1161 (15.7%)	1500	54.54	0.920 ± 0.05	1.00	P1	0.2860 ± 0.063	0.3070	Tbot	64.3%	67.85%
		1400	123.52	0.792 ± 0.05	0.80	D1	0.0064 ± 0.002	0.0067			
		1300	175.68	0.421 ± 0.05	0.40	P2	0.6028 ± 0.044	0.6610			
						D2	0.0218 ± 0.002	0.0261			

been investigated with scenario LB), and patients exhibit more critical fatigue profiles than doctors.

As per Fig. 2, the entry point to the design-time phase analysis is the specification of the scenarios through the DSL presented in Section 3. The complete DSL file for the three scenarios is reported in Appendix B. All scenarios are set in the same layout (shown in Fig. 12), thus, there is only one floor definition. Agents participating in the three scenarios are fixed; specifically, the DSL features one robot definition (identified as Tbot) and eight human definitions (P1 and D1 for DPa, P1 and D1 for DPb, and P1, D1, P2, and D2 for DPc). We recall that the maximum velocity and acceleration for the robot are directly derived from its type parameter, which, in this case, is turtlebot3_wafflepi. Each scenario in Table 6 corresponds to a robotic *mission*, thus, there are three mission definition blocks defining the sequence of services that the robot must provide to complete the mission with success. Finally, queries are defined to compute the metrics required to carry out this design-time analysis, i.e., the probability of success for decreasing values of τ, estimated fatigue for all human subjects, and residual battery charge. Parameter R (the bound on runs) is set to auto, to indicate that Uppaal should generate as many runs as necessary to compute estimations with the requested confidence level. As per Fig. 5, for each mission (thus, in our case, DPa, DPb, DPc), a JSON file is automatically generated and converted into a pair of Uppaal model/query files to perform verification.

We assess the "efficiency" of the DSL in terms of effort saved compared to manually drafting the SHA network modeling each scenario. To this end, we calculate the ratio (indicated as DSL2SHA in Table 7) between the size of a DSL instance and the size of the corresponding SHA network. We compute the size of a DSL model as the number of words needed to configure the scenario. Counting words rather than abstract elements captured by the DSL gives us a more accurate indication of the DSL's verbosity: note that, since the declaration of each element in Section 3.2 requires at least one word, counting abstract elements would result in more favorable ratios. Given a SHA $\mathcal{A}$, we compute its size, indicated as $|\mathcal{A}|$ according to Eq.13:

$$|\mathcal{A}| = |\mathcal{E}| + |\Gamma(W)| + |C_{!?}| + |\Xi(W)| + |L| + |\mathcal{D}| + |\mathcal{F}| + |W| \tag{13}$$

The size of a network of SHA equals the sum of the sizes of all the SHA that compose it. Table 7 reports the resulting ratios.

SMC results are reported in Table 7 and discussed in the following. For this validation phase, the duration of verification experiments performed through Uppaal ranges from 16.36min to 175.68min in the worst case (i.e., scenario DPc, which has the most complex robotic mission and highest τ values). Unlike the previous phase, the goal in this case is to test framework's efficacy when developing realistic scenarios. Therefore, we do not collect a large batch of deployment traces to keep the duration of the deployment phase more practical (i.e., shorter than 1h). Given the smaller number of deployment traces that have been collected, the probability of success of the deployed system is not calculated through Algorithm 1, as it would yield scarcely significant CIs. In this case, we adopt *point estimator* $\overline{p}$ given by the percentage of successful runs as specified by Eq.14, where $\mathcal{DT}$ is the set of deployment traces and set $\mathcal{SVD}_{dt}$ is calculated from a deployment trace $dt \in \mathcal{DT}$ as in Algorithm 1, Line 3.

$$\overline{p} = \frac{|\{dt|dt \in \mathcal{DT} \wedge |\mathcal{SVD}_{dt}| = \mathsf{N_h} \wedge \mathsf{max}(\mathcal{SVD}_{dt}) \leq \tau - \mathsf{T_{int}}\}|}{|\mathcal{DT}|} \cdot 100 \tag{14}$$

Metrics related to fatigue (for each human subject) and battery charge are computed as in the previous validation phase.

Through the design-time analysis we estimate that the three scenarios require a τ of approximately 7min, 9min, and 25min, respectively, to end in success with probability greater than 90%. As previously mentioned, the robot's charge is not critical for any of the scenarios: although DPc is the most demanding in terms of robot's power, since the initial charge $\mathsf{C_0}$ is 99% the estimated residual charge at the end of the mission is greater than 60%. Doctors (i.e., agents D1 and D2) all adhere to the Elderly/Healthy fatigue profile, thus they do not constitute a criticality to the mission. As per Table 7, they reach an estimated maximum fatigue level between 2.18% and 3.7%, in particular D1 in DPc (who participates in the Recipient pattern) reaches the lowest fatigue value (0.6%) as they only move haphazardly out of free will while waiting for the robot to deliver the resource (see Section 4.1.3). On the other hand, as expected, patients reach more critical values. We remark that, although they walk for longer, patient P1 in all three scenarios reaches fatigue levels compatible with those estimated for HL because they have time to rest while the robot is assisting the doctor.

The design-time analysis highlights that the most concerning aspect among the three scenarios is the fatigue level reached by patient P2 in DPc, as they also adhere to a critical fatigue profile (Elderly/Sick) and have to cover a significant distance from R1b to OFF3. For all experiments, threshold $\mathsf{F_{high}}$ (see Table 3) is set to 0.6. Therefore, some traces of the formal model feature the orchestrator instructing P2 to stop and rest (when $\mathsf{f} \geq \mathsf{F_{high}}$ holds) causing a delay in the mission. We remark that this safety measure embedded in the orchestrator is necessary to prevent the patient from reaching the maximum value of fatigue, but it is not sufficient to prevent them from reaching a significant (average) fatigue level (i.e., approximately 60%).

Results observed by deploying the scenarios in the hybrid setting corroborate the outcomes predicted at design-time. As in the first validation phase, Table 7 highlights the results corresponding to larger estimation errors. Specifically, success probability ranges estimated at design time and reported in column DT (we recall that rates in column DEPL are point estimators and, thus, not reported as ranges) are the least accurate when the average success rate is closer to 50% or 60% (DPa with τ = 350s and DPb with τ = 450s, also highlighted in grey). On the other hand, estimations of the fatigue level have design time-to-deployment differences range from approximately 5% in the best case (D1 in DPc) to 16% in the worst case (D2 in DPc, also highlighted in grey). Nevertheless, we recall that, although errors are larger than those obtained with the first three scenarios, these are not an indication of inaccuracies within the formal model as only 5 deployment traces are performed for DPa, DPb, and DPc (compared to more than 100 for the previous validation phase).

Given the results of the first design-time analysis round and the data collected during deployment, the designer in charge of developing and maintaining these scenarios may choose to apply reconfiguration measures and refine the three robotic missions as described in Section 5. The reconfiguration measures applied to the three scenarios (hereinafter referred to as R-DPa, R-DPb, and R-DPc) and the updated sequences of services are described in Table 8. Since the robot's battery was not a critical element in the first round of analysis, replacing the robot with a different one or recharging it would not impact the updated results. For scenarios DPa and DPb, the sequence of services (i.e., the robot's mission) is modified to reduce the time required to complete the mission or, in other words, to obtain a high probability of success for smaller values of τ. In these two cases, the patient is led straight to the examination room rather than to the waiting room first.[10] As for the third scenario, the goal is to lighten the strain on the patient in the most delicate condition. Therefore, the robot serves P2 first and leads them to the doctor's office last to allow them a longer recovery time while in the waiting room. Table 9 reports the DSL2SHA ratio for the reconfigured scenarios. Note that

[10]Note that, in a real healthcare facility, this may not be feasible in all cases: the examination room must either be empty when P1 is served or equipped to host more than one patient simultaneously.

Table 8: Reconfiguration measures applied to scenarios DPa, DPb, and DPc, and updated sequence of services.

SCENARIO	RECONFIGURATION MEASURES	MISSION
R-DPa	The robot (Tbot) *leads* P1 directly to R2, then it serves D1 by *following* them to CUP1 and back to R2.	P1 Follower ENTR $\rightarrow$ R2, D1 Leader R2 $\rightarrow$ CUP1, D1 Leader CUP1 $\rightarrow$ R2
R-DPb	The robot (Tbot) serves D1 first by *fetching* the resource from CUP2 and *follows* them to R2, then it serves P1 and *leads* them to R2.	D1 Recipient OFF2 $\leftrightarrow$ CUP2, D1 Leader OFF2 $\rightarrow$ R2, P1 Follower ENTR $\rightarrow$ R2
R-DPc	The robot (Tbot) *leads* P2 to R1b first and then provides the same sequence of services as scenario DPc.	P2 Follower ENTR $\rightarrow$ R1b, P1 Follower ENTR $\rightarrow$ R1a, D1 Recipient OFF1 $\leftrightarrow$ CUP1, D2 Leader OFF3 $\rightarrow$ CUP2, D2 Leader CUP2 $\rightarrow$ OFF3, P1 Follower R1a $\rightarrow$ OFF1, P2 Follower R1b $\rightarrow$ OFF3

Table 9: Results of DSL2SHA calculation and of the design-time analysis of scenarios R-DPa, R-DPb, and R-DPc. For decreasing values of τ ([s]), the table contains the verification time ([min]), the success probability CI estimated through Uppaal, the estimated maximum fatigue value for all humans, and the estimated minimum charge value for the robot. Fatigue and charge level are only estimated for the maximum value of τ for each scenario.

SC. (M)	DSL2SHA	τ	**Ver.Time** [min]	**Success Probability** ($\mathbb{P}_M(\diamond_{\leq\tau}\mathsf{scs})$)	HUM.	**Max. Fatigue** ($\mathrm{E}_{M,\max(\tau)}[\max(F_i)]$)	ROB.	**Min. Charge** ($\mathrm{E}_{M,\max(\tau)}[\min(C)]$)
R-DPa	227/768 (29.5%)	300 250 200	6.86 32.56 30.10	0.950 ± 0.05 0.498 ± 0.05 0.258 ± 0.05	P1 D1	0.1042 ± 0.014 0.0367 ± 0.004	Tbot	86.51%
R-DPb	227/781 (29.0%)	350 320 300	10.54 55.46 40.69	0.950 ± 0.05 0.741 ± 0.05 0.206 ± 0.05	P1 D1	0.1387 ± 0.008 0.0235 ± 0.001	Tbot	85.45%
R-DPc	283/1161 (15.7%)	1500 1400 1300	20.60 121.59 149.31	0.950 ± 0.05 0.854 ± 0.05 0.556 ± 0.05	P1 D1 P2 D2	0.3034 ± 0.061 0.0032 ± 0.001 0.3761 ± 0.029 0.0245 ± 0.016	Tbot	64.28%

for R-DPc the ratio is unvaried since only the order in which humans are served is changed. On the other hand, for R-DPa and R-DPb removing one human declaration (13 words) reduces the SHA network size by 11% and 10.8%, respectively.

The results of the second round of design-time analysis are reported in Table 9. Quality metrics report a slight improvement compared to the first round of analysis since the robotic mission is shorter for R-DPa and R-DPb. In this case, verification time ranges from 10.54min to approximately 149.31min in the worst case, as scenario R-DPc is substantially unvaried in terms of performance. Estimations inform us that for R-DPa and R-DPb the updated mission can be completed successfully in less time as we obtain a success probability $> 90\%$ for τ equal to 300s and 350s, respectively (compared to 400s and 520s for the initial configuration). Since the patient only walks from the entrance to R2, their estimated maximum level of fatigue is also approximately 60% (in R-DPa) and 43% (in R-DPb) lower than fatigue estimations obtained with DPa and DPb, respectively, whereas the value remains essentially unchanged for D1 as they perform the same actions as in the original scenario. For R-DPc, we observe that allowing P2 more time to rest in the waiting room reduces their maximum fatigue level by 37%. Furthermore, as they do not reach the critical threshold $\mathsf{C}_{\mathsf{high}} = 60\%$ anymore, the orchestrator does not instruct them to stop mid-service, leading to a slight reduction in the duration of the mission.

6.3. Discussion

We can summarize how we have addressed the validation goals as follows:

G1. We have performed more than 300 runs of three experimental scenarios in a digital-twin environment involving simulated humans and a real robotic device communicating via ROS. Collected deployment traces have been exploited to assess the accuracy of the formal model and SMC results.

G2. We have assessed the coverage of our development framework with respect to existing real-world scenarios in the service robotics domain. We have then collected the most recurring tasks within the collected set of real applications into three scenarios to be analysed and developed through our framework. In this regard:

(a) We have assessed the efficiency of the presented DSL by calculating the number of words necessary to configure the whole SHA network (i.e., the DSL2SHA metric).

(b) We have analysed the three scenarios at design-time and the results of such analysis are reflected by the deployment traces.

(c) We have reconfigured the three scenarios in light of the collected deployment traces and iterated the design-time analysis.

Concerning the analysis of the formal model accuracy (goal **G1**), as discussed in Section 6.1, we obtain relative estimation errors for the probability of success and charge level smaller than 10% also in boundary conditions, e.g., involving subjects with a critical fatigue profile or a robot close to full discharge. Success probability and minimum battery charge ranges provide empirical evidence of the reliability of the SHA modeling the robotic system. Since we have only performed experiments with virtual human agents whose model derives from literature analysis, the validation of the formal model of human behavior needs further investigation. As future work, the validation process is to be completed by performing experiments with real human subjects to assess the accuracy of SHA modeling human behavior.

Coverage analysis (enabling the pursuit of goal **G2**) yields that more than 80% of the collected real-world scenarios within the scope of this work can be designed and deployed through the presented framework. The analysis carried out on scenarios DPa, DPb, and DPc shows how the framework supports practitioners throughout the entire development process by automating the generation of the formal model and the deployment of the resulting application.

The analysis of the DSL2SHA ratio (smaller than 30% in all cases) shows that the DSL requires less effort than manually drafting the formal model (goal **G2**a). DSL2SHA values show that the DSL grows more efficient than the manual creation of the formal model as the size of the SHA network in question increases. This is due to the fact that the portion of DSL configuring the geometrical layout (which is the same for all scenarios in Section 6.2, regardless of the complexity of the mission) is the most verbose element. This issue is to be addressed in the future by automatically acquiring the information regarding the layout from planimetries to significantly boost the efficiency of the DSL.

Indicators estimated through the design-time analysis phase of the three scenarios are corroborated by the observations collected during deployment (goal **G2**b). With a small number of deployment traces (i.e., 5), relative estimation errors do not exceed 16%. As for goal **G2**c, reconfiguration measures applied to scenarios DPa and DPb (through minor modifications to the DSL specification) improve the estimated success probabilities with a time bound smaller by 25% (300s compared to 400s) and 33% (350s compared to 570s), respectively. As previously discussed, reconfiguring DPc reduces the physical effort imposed on subject P2.

7. Related Work

Introducing formal analysis into the robot software development process is a long-standing issue in the research community. In a survey from 2019, Luckcuck et al. [53] examine more than 60 papers focused on specification and verification of autonomous robotic systems, emphasizing both the community's interest in the topic and the challenges to face as we move forward. In the following, we report on works existing in the literature proposing:

1. formal modeling and verification techniques for the analysis of robotic applications, especially dealing with human-robot interaction;
2. user-friendly DSLs for the specification of robotic missions.

7.1. Formal Analysis for Robotic Applications Development

Developing software for the robotic domain is an elaborate process given the complexity and unstructured nature of the system itself [7]. Therefore, it usually requires a combination of different software development techniques to achieve a satisfactory result. Several works focus on tasks such as testing and simulation [54] or implementation [55], which are substantial to the development process but out of the scope of this review. In the following, we focus on the early design phase and report on works exploiting formal methods to this end.

Existing works can be classified based on the *formalism* used to model the environment and the agents' behavior and the *verification* technique applied to check properties.

7.1.1. Temporal Logic-based Robotic Applications Modeling

As for the first criterion, temporal logic notations are often adopted to model the robotic task. Gainer et al. [56] present the CRutoN tool to analyse a personal robot's behavior in a domestic setting. The work models the robot's behavior as a set of logic constraints, which are automatically parsed and converted into a NuSMV model [57]. The generated model is put through model checking to verify relevant properties about the system, e.g., that the robot never fails to alert the user about an event that requires their attention. Webster et al. [58] had previously exploited the *BrahmsToPromela* tool [59] for the same case study. The human users and the robot are modeled as *agents* using Brahms. Brahms models are then automatically translated into Promela and verified through the SPIN model-checker. Both works treat human behavior as a black-box whose actions are selected non-deterministically out of a pre-determined set. The work by Vicentini et al. [60] introduces an innovative risk assessment procedure for collaborative industrial tasks based on the TRIO temporal logic language [61]. Similarly to previous examples, the authors model the agents and the task through a set of logic formulae to find safety hazards and assess their severity. As previously mentioned, human-robot interaction introduces uncertainties into the model, thus, the work has been subsequently extended to include manifestations of erroneous human behavior [62].

7.1.2. State-based Robotic Applications Modeling

State-based formalisms are also a popular choice to model the behavior of robotic systems. Most works pair the state-based model of the system with a set of logic properties to perform verification. Ding et al. [63] exploit Finite State Machines to model collaborative industrial tasks, later extended to cover multi-robot multi-human tasks [64], where unexpected events due to the presence of humans are modeled as exceptions and paired with a recovery strategy. Porfirio et al. [65] explore how formal verification can be used to ensure that robots adhere to *social* norms while interacting with humans. Norms, expressed as LTL formulae, constitute the properties to be verified, whereas interaction sequences are modeled as a composition of Labelled Transition Systems (LTSs). The work by Adam et al. [66] also targets the social robotics field, as the authors propose the CAIO framework. The authors exploit the Belief Desire Intention (BDI) architecture and models of human cognition to develop a perception and deliberation process that drives the robot towards making decisions in a human-like fashion and making human-robot interaction feel more natural. Araiza-Illan et al. [67] exploit the AgentSpeak language [68] to implement BDI agents and automatically generate test cases for interactive robotic applications. The framework is tested on a cooperative table assembly case study, where the robot's BDI agent infers the human's state based on three sensors and reacts accordingly as encoded by the AgentSpeak model. Quottrup et al. [69] model multi-robot systems as a network of Timed Automata and verify whether collisions potentially occur or some robots are not able to complete their goal, which are all expressed as CTL properties and verified through Uppaal. Zhou et al. [70] propose a similar approach based on Timed Automata and MITL properties focused on motion planning to synthesize optimal trajectories based on verification results. Some works have also exploited Hybrid Automata to incorporate physical laws into the verification process. Molnar et al. [71] introduce the concept of Model Composition Agents (MCA), which encapsulate a Hybrid Automaton modeling either an agent or the environment and its interaction with other automata in the system. The resulting network of MCA is abstracted as an LTS and model checked to diagnose faults in the original system.

7.1.3. Formalizations of Human Behavior

As human-robot interaction becomes a key element in modern robotic systems, particular attention has been given to how the unpredictability due to the presence of humans can be formally modeled. In this aspect, two main research directions emerge from the literature: game-based approaches and probabilistic models. The possibility to model the interaction between a robotic agent and the environment as a *game* to synthesize a robot controller strategy (if it exists) is investigated in [72]. Kress et al. emphasize the challenge to find a proper abstraction of the environment model that allows for significant verification results without leading to state space explosion. Chen et al. [73] apply the approach based on Timed Game Automata (TGA) and LTL to surveillance, monitoring, and delivery tasks in partially unknown environments. The work by Bersani et al. [74] addresses applications involving robots and humans working in a shared environment, modeled as TGA networks. Humans are modeled as *uncontrollable* agents, to capture the uncertainty of their behavior. A robot controller that also accounts for unpredictable human moves is then automatically synthesized through the Uppaal-TIGA tool.

On the other hand, probabilistic models of human behavior and decision making (e.g., the *Boltzmann policy* [75]) are well-established in the literature and have been successfully applied to the robotic domain. Mason et al. [76] exploit Markov Decision Processes (MDPs) to model an assistive-living scenario and verify probabilistic properties (expressed in PCTL logic) through the PRISM model checker [77]. The work by Junges et al. [78] combines the two approaches since it models the robot as a stochastic *controllable* agent and the human as stochastic and *uncontrollable*, which, when combined, produce a stochastic two-player game. In this case, optimal robot policies are also synthesized through PRISM-Games [79]. Vibekananda et al. [80] exploit Probabilistic State Machines to perform human pose estimation and predict their intention while interacting with a robot. Galin et al. [81] build upon a previous study on how Cellular Automata with probabilistic transitions can be used to model human motion in partially unknown environments [82]. The authors exploit these theoretical results to develop the model of a shared workspace where human and robot work simultaneously to compute the area where their trajectories are more likely to overlap.

7.1.4. Verification Techniques and Tools

Since state-based formalisms and temporal logics are the most popular choices when it comes to modeling the robotic system, it follows that model-checking is the natural choice in terms of verification technique [53], given the availability of powerful model checkers such as Uppaal [27] and SPIN [83]. Models based on MDPs, such as the one developed by Ye et al. [84], can be verified through Probabilistic Model Checking, which is most often performed through PRISM [77]. Statistical Model Checking (SMC), which is the verification technique used in our framework, has also gained momentum over the last few years. The most common motivation pertains to the reduced verification times, which lead to more practical approaches. Paigwar et al. [85] exploit SMC to estimate the probability of collisions in automated driving systems. Foughali et al. [86] apply SMC to formally verify real-time properties, like schedulability and readiness, of robotic software. Herd et al. [87] focus on multi-agent systems, and on swarm robotics in particular: in this case, SMC dampens issues related to the size of the problem, which cannot be handled by traditional model checking techniques.

7.2. Specification Languages for the Robotic Domain

In 2014, Nordmann et al. [88] surveyed 137 papers presenting robotic DSLs. At the time of writing, Scopus indexes more than 90 papers published since 2014 with keywords robot* and domain-specific language. These numbers show that DSL development is a cornerstone of the robotic software engineering process since it automates the generation of code or complex models and makes development frameworks accessible to a wider audience. Referring to the classification in [88], in the following we report on the subset of works on this topic dealing with the *scenario building* phase, i.e., DSLs to specify high-level environment features and the robot's task, as these are the closest to our work.

7.2.1. DSLs for Scenario Building.

Noreils and Chatila [89] present a high-level notation to specify reactive robotic mission plans. The language envisages the specification of modules, which are further structured into three architectural layers: the *functional* layer to specify the lower-level robot's capabilities, the *planning* layer to specify task sequences, and the *control* layer that translates plans into requests to the functional modules. Knoop et al. [90] present an approach to automatically generate robotic tasks starting from representations of tasks in the human operational space, adhereing to the Programming by Demonstration paradigm. Finucane et al. [91] present the LTLMoP framework to automatically synthesize and deploy robot controllers. The framework converts Structured English specifications describing the robotic task into equivalent LTL formulae, which are then synthesized into an automaton (the *discrete* controller). The work has been subsequently extended by Raman et al. [92] with implicit memory strategies to model robotic tasks depending on events that occurred in the past (e.g., "every time you sense *order*, visit the *kitchen*"). Kunze et al. [93] present SRDL, a framework extending the KnowRob knowledge base [94] with notions about *robots*, hardware *components*, *actions*, and *capabilities* (of performing a certain action). Miyazawa et al. [95] introduce RoboChart, a DSL to model and verify real-time concurrent robotic tasks with budgets and deadlines (i.e., cost and time constraints). RoboChart semantics, which is based on Timed Automata and Timed Communicating Sequential Processes (CSP) [96], makes the notation amenable to formal verification, specifically model-checking. Ciccozzi et al. [97] propose a family of three languages to specify missions for multi-robot systems: the Monitoring Mission Language to specify task sequences, the Robot Language to configure the individual robots, and the Behavior Language to specify the atomic movements of robots.

7.2.2. DSLs for Human-Robot Interaction

The advent of human-robot interaction and collaborative robotics has also influenced DSL development. Gavran et al. [98] introduce the TOOL DSL to specify collaborative assembly tasks in the manufacturing sector through a textual notation that is accessible to non-experts. Detzner et al. [99] present LoTLan, a DSL to describe material flow processes in warehouses. The work consists of a procedure to map human vocal requests (e.g., "I need an item") to common semantics, identifying *who* has to perform *which action*, and finally LoTLan primitives, which are then converted into plans for AGVs. Forbig et al. [100] exploit their language CoTaL [101] to model interactive tasks between a humanoid robot and a stroke patient performing arm mobility recovery exercises. The resulting specification captures all phases needed for the exercise session, how the humanoid robot can detect whether the patient has completed an exercise or not and how to react accordingly.

7.3. Discussion

As this survey shows, numerous approaches exploit formal methods to analyze robotic applications. Specifically, several attempts have been made at formalizing the aspects of human behavior that are significant while interacting with a robot and should, thus, impact the results of the formal analysis. Most of these works present approaches that are either deterministic, game-based, or probabilistic, such as the hereby presented framework. Deterministic approaches—such as architectures based on BDI agents—potentially result in less complex models and more favorable verification times. However, assuming complete rationality and absence of fuzziness is reasonable for robotic agents (the orchestrator SHA indeed inherits most of its substructures from the BDI architecture) or for human agents performing small repetitive tasks in controlled environments [67, 66]. Human-robot interactions in the service sector feature virtually no constraint on human behavior, thus deterministic models are overly restrictive. Furthermore, service robots applications involve people from various age groups with different characteristics and performing a broad range of tasks. Therefore, while estimating the outcome of a scenario, the exploration of the state space of all possible behaviors should be guided by such features. Game-based approaches, although effective when exploited for controller synthesis [74], imply an exhaustive exploration of human actions (i.e., the *opponent*'s move) irrespective of their likelihood given the specific scenario configuration. For these reasons, probabilistic approaches are particularly suited for the purpose of this framework. Specifically, to the best of the authors' knowledge, this is the first attempt at combining probabilistic weights on human actions with a hybrid and stochastic characterization of physiological processes. Due to its complexity, the resulting model is more practically manageable through SMC rather than probabilistic model checking. Indeed, works exploiting exhaustive techniques such as [102] focus on smaller setups targeting a specific task (e.g., the handover of an item). Despite the loss in reliability introduced by SMC that only relies on a finite set of runs of the systems, the proposed framework is applicable to a broad range of scenarios (as shown by the coverage analysis results) while still providing results at design-time that accurately reflect runtime observations.

As per Section 7.2, the literature is rich with DSLs for the robotic domain, but proposals targeting interactive applications are lacking. Specifically, existing works target the manufacturing sector [98, 99] or very specific tasks from the healthcare setting [100], whereas the service sector calls for more general-purpose primitives to define how robots and humans interact. Other works propose a high-level specification of mission patterns for multi-robot teams in environments (possibly) populated by humans [46, 12], but this has not been attempted for applications where humans are actively involved as in the domain of this framework.

8. Conclusion and Future Work

We have presented a specification, modeling and analysis framework targeting interactive service robotic applications. The framework has been extended with respect to previous publications with the introduction of a custom DSL, refined formal models of the battery and the human behavior with a stochastic characterization of human fatigue, and extensive experimental validation results including real-life experiments, coverage analysis, and DSL evaluation.

The framework is open to extensions. The quality metrics analyzed in the paper are not the only measures of interest for a potential application designer using our modeling approach. With the current model, the analysis can be easily enriched with measures such as the percentage of time agents spend in a certain operational state, the frequency of a human agent reacting to a command issued by the orchestrator, the number of times a certain action is taken, etc. For the sake of clarity, we kept the presentation limited to the human fatigue and battery charge. In addition, a number of physiological or psychological indicators can be considered to enrich the model of humans, provided that they can be represented by means of (OD) equations or discrete/automata-based features. These indicators would allow the designer to model the sensitivity of humans to phenomena that affect their responsiveness when environmental conditions are not ideal. For instance, meaningful physiological values can be the heartbeat rate, the blood pressure, the breath frequency of the patients in critical health conditions, and psychological indicators include stress, patience or the level of engagement of the operators and doctors who take part in a scenario.

The availability of tools such as the one presented in this work implies the need for a criterion to establish which is the "right accuracy" to consider when judging the outcomes of the analysis. To the best of the authors' knowledge, such a criterion does not exist and its definition would require specific work, possibly conducted by healthcare specialists and medical engineers in real contexts. Nonetheless, the paper shows in Section 6 an approach to assess the accuracy of our tool that is based on the comparison of quantitative metrics obtained as a result of the formal analysis of the models and implemented real-world scenarios.

We plan on improving the level of support provided to the designer by—at least partially—automating the reconfiguration phase. The model of human behavior can be automatically refined based on observations collected during deployment. The manual effort required on the designer's side can be further reduced by automatically computing alternative mission plans leading to better key indicator values (success probability and human subjects' physical strain) than those resulting from the initial plan. In addition, we plan on assessing the DSL with the engagement of non-expert users of formal verification and of healthcare operators, and to add syntactical structures that allow operators to integrate their own interaction patterns in the framework without resorting to customized translations set out by formal method experts.

References

[1] C. B. Frey, M. A. Osborne, The future of employment: How susceptible are jobs to computerisation?, Technological forecasting and social change 114 (2017) 254–280.

[2] A. A. Morgan, J. Abdi, M. A. Syed, G. E. Kohen, P. Barlow, M. P. Vizcaychipi, Robots in healthcare: a scoping review, Current Robotics Reports (2022) 1–10.

[3] Robots and healthcare saving lives together (2022).
URL `https://www.automate.org/industry-insights/robots-and-healthcare-saving-lives-together`

[4] Global healthcare assistive robot market (2022).
URL `https://www.globenewswire.com/news-release/2021/09/24/2303007/0/en/Global-Healthcare-Assistive-Robot-Market-Expected-19-CAGR-Will-Reach-USD-1-2-Billion-by-2026-Facts-Factors.html`

[5] Robotics and the impact on nursing practice (2020).
URL `https://www.nursingworld.org/~494055/globalassets/innovation/robotics-and-the-impact-on-nursing-practice_print_12-2-2020-pdf-1.pdf`

[6] Fraunhofer Institute for Manufacturing Engineering and Automation, EFFIROB: Economic feasibility studies on innovative service robot applications (2010).
URL `https://www.ipa.fraunhofer.de/en/reference_projects/EFFIROB.html`

[7] S. García, D. Strüber, D. Brugali, T. Berger, P. Pelliccione, Robotics software engineering: A perspective from the service robotics domain, in: ESEC/FSE, ACM, USA, 2020, pp. 593–604.

[8] K. Ehrlenspiel, A. Kiewert, U. Lindemann, M. S. Hundal, Cost-efficient design, Vol. 544, Springer, 2007.

[9] D. Brugali, Software product line engineering for robotics, Software Engineering for Robotics (2021) 1–28.

[10] P. Payne, M. Lopetegui, S. Yu, A review of clinical workflow studies and methods, in: Cog. Inf., Springer, 2019, pp. 47–61.

[11] ISO/PAS 21448:2019, Road vehicles — Safety of the intended functionality, ISO/PAS, 2019.

[12] S. García, P. Pelliccione, C. Menghi, T. Berger, T. Bures, PROMISE: high-level mission specification for multiple robots, in: Intl. Conf. on Software Engineering, ACM, Seoul, South Korea, 2020, pp. 5–8.

[13] L. Lestingi, M. Askarpour, M. M. Bersani, M. Rossi, Formal verification of human-robot interaction in healthcare scenarios, in: SEFM, Springer, 2020, pp. 303–324.

[14] L. Lestingi, M. Askarpour, M. M. Bersani, M. Rossi, A model-driven approach for the formal analysis of human-robot interaction scenarios, in: IEEE SMC, 2020, pp. 1907–1914.

[15] L. Lestingi, M. Askarpour, M. M. Bersani, M. Rossi, A deployment framework for formally verified human-robot interactions, IEEE Access 9 (2021) 136616–136635.

[16] L. Lestingi, C. Sbrolli, P. Scarmozzino, G. Romeo, M. M. Bersani, M. Rossi, Formal modeling and verification of multi-robot interactive scenarios in service settings, in: Intl. Conf. on Formal Methods in Software Engineering, 2022, pp. 80–90.

[17] A. David, K. G. Larsen, A. Legay, M. Mikučionis, D. B. Poulsen, Uppaal SMC tutorial, STTT 17 (4) (2015) 397–415.

[18] C. J. Clopper, E. S. Pearson, The use of confidence or fiducial limits illustrated in the case of the binomial, Biometrika 26 (4) (1934) 404–413.

[19] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, S. Yovine, The algorithmic analysis of hybrid systems, TCS 138 (1) (1995) 3–34.

[20] S. F. Arenis, M. Vujinovic, B. Westphal, On implementable timed automata, in: Formal Techniques for Distributed Objects, Components, and Systems, Vol. 12136 of Lecture Notes in Computer Science, Springer, Valletta, Malta, 2020, pp. 78–95.

[21] A. David, K. G. Larsen, A. Legay, M. Mikucionis, D. B. Poulsen, J. van Vliet, Z. Wang, Statistical model checking for networks of priced timed automata, in: Formal Modeling and Analysis of Timed Systems, Vol. 6919 of Lecture Notes in Computer Science, Springer, Aalborg, Denmark, 2011, pp. 80–96.

[22] U. Grenander, Stochastic processes and statistical inference, Arkiv för matematik 1 (3) (1950) 195–277.

[23] G. Agha, K. Palmskog, A survey of statistical model checking, TOMACS 28 (1) (2018) 1–39.

[24] R. Alur, T. Feder, T. A. Henzinger, The benefits of relaxing punctuality, Journal of the ACM (JACM) 43 (1) (1996) 116–146.

[25] D. Tardioli, R. Parasuraman, P. Ögren, Pound: A multi-master ros node for reducing delay and jitter in wireless multi-robot networks, Robotics and Autonomous Systems 111 (2019) 73–87.

[26] G. Behrmann, A. Cougnard, A. David, E. Fleury, K. G. Larsen, D. Lime, Uppaal-tiga: Time for playing games!, in: International Conference on Computer Aided Verification, Springer, 2007, pp. 121–125.

[27] K. G. Larsen, P. Pettersson, W. Yi, UPPAAL in a nutshell, International Journal on Software Tools for Technology Transfer 1 (1-2) (1997) 134–152.

[28] J. Z. Liu, R. W. Brown, G. H. Yue, A dynamical model of muscle activation, fatigue, and recovery, Biophysical Journal 82 (5) (2002) 2344–2359.

[29] S. Konz, Work/rest: Part ii-the scientific basis (knowledge base) for the guide 1, EGPS 1 (401) (2000) 38.

[30] Z. Givi, M. Y. Jaber, W. P. Neumann, Modelling worker reliability with learning and fatigue, Applied Mathematical Modelling 39 (17) (2015) 5186–5199.

[31] B. Liu, L. Ma, C. Chen, Z. Zhang, Experimental validation of a subject-specific maximum endurance time model, Ergonomics 61 (6) (2018) 806–817.

[32] M. Roberto, D. Farina, M. Gazzoni, M. Schieroni, Effect of age on muscle functions investigated with surface electromyography, Muscle & nerve 25 (1) (2002) 65–76.

[33] M. Hadley, A deterministic model of the free will phenomenon, Journal of Consciousness Exploration & Research 9 (1) (2018).

[34] C. Calude, F. Kroon, N. Poznanovic, Free will is compatible with randomness, Philosophical Inquiries 4 (2) (2016) 37–52.
[35] M. Lutz, D. Stampfer, A. Lotz, C. Schlegel, Service robot control architectures for flexible and robust real-world task execution: Best practices and patterns, in: INFORMATIK 2014, Vol. P-232 of LNI, GI, Stuttgart, Germany, 2014, pp. 1295–1306.
[36] O. Tremblay, L.-A. Dessaint, A.-I. Dekkiche, A generic battery model for the dynamic simulation of hybrid electric vehicles, in: Vehicle Power and Propulsion Conference, IEEE, 2007, pp. 284–289.
[37] H. Chuangfeng, L. Pingan, J. Xueyan, Measurement and analysis for lithium battery of high-rate discharge performance, Procedia Engineering 15 (2011) 2619–2623.
[38] R. Merletti, L. L. Conte, C. Orizio, Indices of muscle fatigue, J. of Electromyography and Kinesiology 1 (1) (1991) 20–33.
[39] L. Lindstrom, R. Kadefors, I. Petersen, An electromyographic index for localized muscle fatigue, Journal of Applied Physiology 43 (4) (1977) 750–754.
[40] A. Rasheed, O. San, T. Kvamsdal, Digital twin: Values, challenges and enablers from a modeling perspective, IEEE Access 8 (2020) 21980–22012.
[41] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, ROS: an open-source robot operating system, in: ICRA Workshop on Open Source Software, Vol. 3, IEEE, Kobe, Japan, 2009, p. 5.
[42] T. Nguyen, M. Reynolds, R. Kandaswamy, et al., Emerging technologies and trends impact radar: 2021, Gartner Research Notes. Available at (2021).
[43] H. G. Kang, J. B. Dingwell, Differential changes with age in multiscale entropy of electromyography signals from leg muscles during treadmill walking, PloS one 11 (8) (2016) e0162034.
[44] E. T. Jaynes, O. Kempthorne, Confidence intervals vs bayesian intervals, in: Foundations of probability theory, statistical inference, and statistical theories of science, Springer, 1976, pp. 175–257.
[45] F. Scholz, Confidence bounds and intervals for parameters relating to the binomial, negative binomial, poisson and hypergeometric distributions with applications to rare events, 2008 (2008).
[46] C. Menghi, C. Tsigkanos, P. Pelliccione, C. Ghezzi, T. Berger, Specification patterns for robotic missions, IEEE Trans. Software Eng. 47 (10) (2021) 2208–2224.
[47] M. Askarpour, C. Tsigkanos, C. Menghi, R. Calinescu, P. Pelliccione, S. García, R. Caldas, T. J. von Oertzen, M. Wimmer, L. Berardinelli, et al., RoboMAX: Robotic Mission Adaptation eXemplars, in: 2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), IEEE, 2021, pp. 245–251.
[48] K. Baraka, M. M. Veloso, Mobile service robot state revealing through expressive lights: formalism, design, and evaluation, International Journal of Social Robotics 10 (1) (2018) 65–92.
[49] Case Studies - Service Robots, `https://ifr.org/case-studies/service-robots-case-studies` (2018).
[50] M. Bajones, D. Fischinger, A. Weiss, P. D. L. Puente, D. Wolf, M. Vincze, T. Körtner, M. Weninger, K. Papoutsakis, D. Michel, et al., Results of field trials with a mobile service robot for older adults in 16 private households, ACM Transactions on Human-Robot Interaction (THRI) 9 (2) (2019) 1–27.
[51] Care-O-bot 3 Application Scenarios, `https://www.care-o-bot.de/en/care-o-bot-3/application.html` (2012).
[52] ISO 13482, Robots and robotic devices - Safety requirements for personal care robots, ISO, 2014.
[53] M. Luckcuck, M. Farrell, L. A. Dennis, C. Dixon, M. Fisher, Formal specification and verification of autonomous robotic systems: A survey, ACM Comput. Surv. 52 (5) (2019) 100:1–100:41.
[54] S. Abbaspour Asadollah, R. Inam, H. Hansson, A survey on testing for cyber physical system, in: International Conference on Testing Software and Systems, Springer, Sharjah and Dubai, UAE, 2015, pp. 194–207.
[55] G. Ajaykumar, M. Steele, C.-M. Huang, A survey on end-user robot programming, ACM Computing Surveys 54 (8) (2021) 1–36.
[56] P. Gainer, C. Dixon, K. Dautenhahn, M. Fisher, U. Hustadt, J. Saunders, M. Webster, Cruton: Automatic verification of a robotic assistant's behaviours, in: Critical Systems: Formal Methods and Automated Verification, Springer, Turin, Italy, 2017, pp. 119–133.
[57] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, A. Tacchella, Nusmv 2: An opensource tool for symbolic model checking, in: International conference on computer aided verification, Springer, Springer, Copenhagen, Denmark, 2002, pp. 359–364.
[58] M. Webster, C. Dixon, M. Fisher, M. Salem, J. Saunders, K. L. Koay, K. Dautenhahn, Formal verification of an autonomous personal robotic assistant, in: AAAI Spring Symposia, AAAI Press, Palo Alto, California, USA, 2014, pp. 1–6.
[59] R. Stocker, L. A. Dennis, C. Dixon, M. Fisher, Verifying brahms human-robot teamwork models, in: Logics in Artificial Intelligence, Vol. 7519 of Lecture Notes in Computer Science, Springer, Toulouse, France, 2012, pp. 385–397.
[60] F. Vicentini, M. Askarpour, M. G. Rossi, D. Mandrioli, Safety assessment of collaborative robotics through automated formal verification, IEEE Transactions on Robotics 36 (1) (2019) 42–61.
[61] C. A. Furia, D. Mandrioli, A. Morzenti, M. Rossi, Modeling Time in Computing, Monographs in Theoretical Computer Science. An EATCS Series, Springer, Berlin, Heidelberg, 2012.
[62] M. Askarpour, D. Mandrioli, M. Rossi, F. Vicentini, Formal model of human erroneous behavior for safety analysis in collaborative robotics, Robotics and Computer-Integrated Manufacturing 57 (2019) 465–476.
[63] H. Ding, J. Heyn, B. Matthias, H. Staab, Structured collaborative behavior of industrial robots in mixed human-robot environments, in: Intl. Conf. on Automation Science and Engineering, IEEE, Madison, WI, USA, 2013, pp. 1101–1106.
[64] H. Ding, M. Schipper, B. Matthias, Collaborative behavior design of industrial robots for multiple human-robot collaboration, in: IEEE Internationel Symposium on Robotics, IEEE, Seoul, Korea (South), 2013, pp. 1–6.
[65] D. Porfirio, A. Sauppé, A. Albarghouthi, B. Mutlu, Authoring and verifying human-robot interactions, in: ACM Symposium on User Interface Software and Technology, ACM, Berlin, Germany, 2018, pp. 75–86.
[66] C. Adam, W. Johal, D. Pellier, H. Fiorino, S. Pesty, Social human-robot interaction: A new cognitive and affective

interaction-oriented architecture, in: Intl. Conf. on Social Robotics, Vol. 9979 of Lecture Notes in Computer Science, Springer, Kansas City, MO, USA, 2016, pp. 253–263.

[67] D. Araiza-Illan, A. G. Pipe, K. Eder, Intelligent agent-based stimulation for testing robotic software in human-robot interactions, in: Workshop on Model-Driven Robot Software Engineering, ACM, Leipzig, Germany, 2016, pp. 9–16.

[68] A. S. Rao, AgentSpeak(L): BDI agents speak out in a logical computable language, in: Workshop on Modelling Autonomous Agents in a Multi-Agent World, Vol. 1038 of Lecture Notes in Computer Science, Springer, Eindhoven, The Netherlands, 1996, pp. 42–55.

[69] M. M. Quottrup, T. Bak, R. Izadi-Zamanabadi, Multi-robot planning: a timed automata approach, in: IEEE International Conference on Robotics and Automation, IEEE, New Orleans, LA, USA, 2004, pp. 4417–4422.

[70] Y. Zhou, D. Maity, J. S. Baras, Timed automata approach for motion planning using metric interval temporal logic, in: European Control Conference, IEEE, Aalborg, Denmark, 2016, pp. 690–695.

[71] L. Molnar, S. M. Veres, Hybrid automata dicretising agents for formal modelling of robots, IFAC Proceedings Volumes 44 (1) (2011) 49–54.

[72] H. Kress-Gazit, T. Wongpiromsarn, U. Topcu, Correct, reactive, high-level robot control, IEEE Robotics & Automation Magazine 18 (3) (2011) 65–74.

[73] Y. Chen, J. Tumova, C. Belta, LTL robot motion control based on automata learning of environmental dynamics, in: IEEE International Conference on Robotics and Automation, IEEE, St. Paul, Minnesota, USA, 2012, pp. 5177–5182.

[74] M. M. Bersani, M. Soldo, C. Menghi, P. Pelliccione, M. Rossi, PuRSUE-from specification of robotic environments to synthesis of controllers, Formal Aspects of Computing 32 (2) (2020) 187–227.

[75] C. L. Baker, J. Tenenbaum, R. R. Saxe, Goal inference as inverse planning, in: Proceedings of the Annual Meeting of the Cognitive Science Society, 29 (29), 2007.

[76] G. Mason, R. Calinescu, D. Kudenko, A. Banks, Assurance in reinforcement learning using quantitative verification, Advances in hybridization of intelligent methods 85 (2017) 71.

[77] M. Z. Kwiatkowska, G. Norman, D. Parker, PRISM 4.0: Verification of probabilistic real-time systems, in: Computer Aided Verification, Vol. 6806 of Lecture Notes in Computer Science, Springer, Snowbird, UT, USA, 2011, pp. 585–591.

[78] S. Junges, N. Jansen, J.-P. Katoen, U. Topcu, Probabilistic model checking for complex cognitive tasks–a case study in human-robot interaction, arXiv preprint arXiv:1610.09409 (2016).

[79] T. Chen, V. Forejt, M. Kwiatkowska, D. Parker, A. Simaitis, PRISM-games: A model checker for stochastic multi-player games, in: Intl. Conf. on TOOLS and Algorithms for the Construction and Analysis of Systems, Springer, Rome, Italy, 2013, pp. 185–191.

[80] V. Dutta, T. Zielinska, Predicting the intention of human activities for real-time human-robot interaction (HRI), in: Int. Conf. on Social Robotics, Vol. 9979 of Lecture Notes in Computer Science, Springer, Kansas City, MO, USA, 2016, pp. 723–734.

[81] R. R. Galin, R. V. Meshcheryakov, M. V. Mamchenko, Analysis of intersection of working areas within the human-robot interaction in a shared workspace, in: Proceedings of the Computational Methods in Systems and Software, Springer, Czech Republic, 2021, pp. 749–759.

[82] R. Breukelaar, T. Bäck, Using a genetic algorithm to evolve behavior in multi dimensional cellular automata: emergence of behavior, in: Genetic and Evolutionary Computation Conference, ACM, Washington DC, USA, 2005, pp. 107–114.

[83] G. J. Holzmann, The model checker SPIN, IEEE Transactions on Software Engineering 23 (5) (1997) 279–295.

[84] K. Ye, A. Cavalcanti, S. Foster, A. Miyazawa, J. Woodcock, Probabilistic modelling and verification using RoboChart and PRISM, Software and Systems Modeling (2021) 1–50.

[85] A. Paigwar, E. Baranov, A. Renzaglia, C. Laugier, A. Legay, Probabilistic collision risk estimation for autonomous driving: Validation via statistical model checking, in: IEEE Intelligent Vehicles Symposium, IEEE, Las Vegas, NV, USA, 2020, pp. 737–743.

[86] M. Foughali, F. Ingrand, C. Seceleanu, Statistical model checking of complex robotic systems, in: International Symposium on Model Checking Software, Vol. 11636 of Lecture Notes in Computer Science, Springer, Beijing, China, 2019, pp. 114–134.

[87] B. Herd, S. Miles, P. McBurney, M. Luck, Quantitative analysis of multiagent systems through statistical model checking, in: Int. Workshop on Engineering Multi-Agent Systems, Springer, 2015, pp. 109–130.

[88] A. Nordmann, N. Hochgeschwender, S. Wrede, A survey on domain-specific languages in robotics, in: Simulation, Modeling, and Programming for Autonomous Robots, Vol. 8810 of Lecture Notes in Computer Science, Springer, Bergamo, Italy, 2014, pp. 195–206.

[89] F. R. Noreils, R. Chatila, Plan execution monitoring and control architecture for mobile robots, IEEE Trans. Robotics Autom. 11 (2) (1995) 255–266.

[90] S. Knoop, M. Pardowitz, R. Dillmann, Automatic robot programming from learned abstract task knowledge, in: Intl. Conf. on Intelligent Robots and Systems, IEEE, California, USA, 2007, pp. 1651–1657.

[91] C. Finucane, G. Jing, H. Kress-Gazit, LTLMoP: Experimenting with language, temporal logic and robot control, in: Intl. Conf. on Intelligent Robots and Systems, IEEE, Taipei, Taiwan, 2010, pp. 1988–1993.

[92] V. Raman, B. Xu, H. Kress-Gazit, Avoiding forgetfulness: Structured english specifications for high-level robot control with implicit memory, in: Intl. Conf. on Intelligent Robots and Systems, IEEE, Vilamoura, Algarve, Portugal, 2012, pp. 1233–1238.

[93] L. Kunze, T. Roehm, M. Beetz, Towards semantic robot description languages, in: Intl. Conf. on Robotics and Automation, IEEE, Shanghai, China, 2011, pp. 5589–5595.

[94] M. Tenorth, M. Beetz, KnowRob: A knowledge processing infrastructure for cognition-enabled robots, Int. J. Robotics Res. 32 (5) (2013) 566–590.

[95] A. Miyazawa, P. Ribeiro, W. Li, A. Cavalcanti, J. Timmis, J. Woodcock, RoboChart: modelling and verification of the functional behaviour of robotic applications, Software & Systems Modeling 18 (5) (2019) 3097–3149.
[96] S. Schneider, Concurrent and real-time systems: the CSP approach, John Wiley & Sons, 1999.
[97] F. Ciccozzi, D. D. Ruscio, I. Malavolta, P. Pelliccione, Adopting MDE for specifying and executing civilian missions of mobile multi-robot systems, IEEE Access 4 (2016) 6451–6466.
[98] I. Gavran, O. Mailahn, R. Müller, R. Peifer, D. Zufferey, Tool: accessible automated reasoning for human robot collaboration, in: Intl. Symp. on New Ideas, New Paradigms, and Reflections on Programming and Software, 2018, pp. 44–56.
[99] P. Detzner, T. Kirks, J. Jost, A novel task language for natural interaction in human-robot systems for warehouse logistics, in: Intl. Conf. on Computer Science & Education, IEEE, Toronto, ON, Canada, 2019, pp. 725–730.
[100] P. Forbrig, A. Bundea, Modelling the collaboration of a patient and an assisting humanoid robot during training tasks, in: Human-Computer Interaction, Vol. 12182 of Lecture Notes in Computer Science, Springer, Copenhagen, Denmark, 2020, pp. 592–602.
[101] P. Forbrig, A. Dittmar, M. Kühn, A textual domain specific language for task models: Generating code for CoTaL, CTTE, and HAMSTERS, in: Symposium on Engineering Interactive Computing Systems, ACM, Paris, France, 2018, pp. 5:1–5:6.
[102] M. Webster, D. Western, D. Araiza-Illan, C. Dixon, K. Eder, M. Fisher, A. G. Pipe, A corroborative approach to verification and validation of human–robot teams, Intl. Journ. of Robotics Research 39 (1) (2020) 73–99.

Appendix A. SHA Semantics

This appendix presents the semantics of SHA and shows, in Fig. A.15, an equivalent representation of the automaton of Fig. 3a that is fully compliant with the introduced syntax and semantics of SHA. The automaton of Fig. 3a must be understood as an equivalent, simplified representation of the automaton of Fig. A.15, where the latter differs from the former in the way the edge exiting location *cool* is connected to locations *high* and *low*.

Complex systems constituted by multiple entities can be modeled as a combination of SHA, which form a **network**. To make n automata $A_1, \ldots, A_n$ (each one defined as in Definition 1) form a network, the following properties must be satisfied [21]. Every automaton A_i must be deterministic, i.e., there are no two (or more) edges, outgoing from a location of A_i, defined by the same event and whose edge conditions can be satisfied by the same valuation. Moreover, they must guarantee the following two semantic properties, called input-enabledness and time divergence. Let $v'_{\text{var}} : W_i \to \mathbb{R}$ be a valuation for variables in W_i, $l_i \in L_i$ be a location of automaton A_i and let pair (l_i, v_{var}) be a configuration of A_i. For every automaton A_i, and for all configurations (l_i, v_{var}) and channel $\mathtt{c} \in C$, there exists an edge $\mathtt{c}?$ that can be taken, i.e., the edge is enabled as the associated edge condition in $\Gamma(W)$ is satisfied by v_{var}. Intuitively, this assumption ensures that every automaton can fire a transition in every possible configuration. Second, every automaton A_i always allows for executions such that if A_i is equipped with an extra clock which is never reset then, in all executions of A_i, this clock cannot be bounded by any arbitrary integer constant (i.e., no Zeno executions are feasible). Finally, every automaton A_i is defined by considering the same set of channels ($C_i = C_j$ when $i \neq j$) and no pair of transitions, each one belonging to two different automata in the network, are built by referring to the same event $\mathtt{c}!$. These properties are with no loss of generality. For example, disjointedness of channels can always be achieved by choosing properly defined symbol sets C_i. In addition, for simplicity, for the network $A_1, \ldots, A_n$ to be composable the sets of real-valued variables must be pairwise disjoint ($W_i \cap W_j = \emptyset$ when $i \neq j$). However, this constraint can be relaxed through a suitable extension of the semantics.

In the following, we outline the semantics of an SHA network including n automata $A_1, \ldots, A_n$, each defined as in Definition 1. The semantics of a composable network $A_1, \ldots, A_n$ is defined based on the configurations (of the network), each one being a tuple of the form $(s_1, \ldots, s_n)$ where every state s_i is a configuration of automaton A_i. There are two possible types of configuration changes realized, respectively, by *discrete transitions* and *time transitions*. A discrete transition occurs when one or more automata take an edge. In the latter case, at least two automata synchronize with each other. Synchronization among different automata inside a network occurs through the channels of set C [27]. Given a channel $\mathtt{c} \in C$ and two edges of two distinct automata, whose events are $\mathtt{c}!$ (the *sender*) and $\mathtt{c}?$ (the *receiver*), triggering an event through channel $\mathtt{c}$ causes both edges to fire simultaneously. Synchronization always requires at most one sender and possibly many receivers (even none). In Fig. 3b, the thermostat can trigger an event through channels $\mathtt{on}!$ and $\mathtt{off}!$ to start or stop heating the room. The triggered event is then received by the room automaton

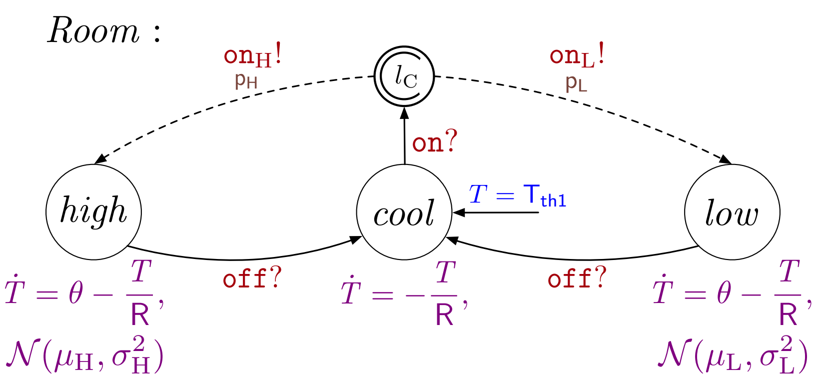


Figure A.15: SHA modeling the room from the running example in Section 2 with detailed representation of how probability weights on receiving edges are handled.

through labels `on`? and `off`?, which makes the corresponding edges fire. Taking an edge (l, c, γ, ξ, l') of automaton A_i with configuration (l, v_{var}) implies that the edge is enabled—i.e., all the conditions in $\Gamma(W)$ associated with the edge are satisfied by the values defined by v_{var}. Upon taking the edge, the location of A_i changes from l to l' and the associated update ξ is executed, resulting in configuration (l', v'_{var}). Since several automata may be involved in a synchronisation, and many updates can be executed simultaneously, specific rules are needed to regulate their execution. The value of a variable w in v'_{var} is determined based on the interpretation of w, i.e., whether w is a stochastic parameter or not. In the former case, upon entering a location $l' \in L_i$ such that $\mathcal{D}_i(l')$ is defined, a realization of distribution $\mathcal{D}_i(l')$ (e.g., $\mathcal{N}(\mu_{\text{H}}, \sigma^2_{\text{H}})$ in Fig. A.15) defines the value of w in v'_{var} (e.g., θ in Fig. A.15); otherwise, when $\mathcal{D}_i(l')$ is not defined, the value of w in v'_{var} and in v_{var} is the same [17]. In the latter case, w is not interpreted as a randomly distributed parameter and its value in v'_{var} is the value of the assignment associated with w' in ξ, that is obtained by evaluating every non-primed variables of the constraint with values from v_{var}. The configuration (l', v'_{var}) is such that the valuation v'_{var} satisfies the invariant $\mathcal{I}_i(l')$.

Besides randomly distributed variables, in SHA, probability measures can be associated with delays to model the elapsing of time in the network, hence the wait between the occurrence of two discrete transitions. According to [21], the adopted probabilistic semantics is based on the "principle of independence" among automata in the network. Upon the firing of an edge, for every automaton A_i in the network, a delay d_i models the time A_i waits before taking an edge for event $\mathsf{c}!$, for some $\mathsf{c} \in C$. If no edges for event $\mathsf{c}!$ originate from l', then d_i is ∞. Otherwise, let $d_{\min}(l', v'_{\text{var}})$ be the minimum delay that automaton A_i should wait before an edge whose event is $\mathsf{c}!$, and departing from l', is enabled; and let $d_{\max}(l', v'_{\text{var}})$ be the maximum delay that automaton A_i can wait before all edges, for events $\mathsf{c}!$, with $\mathsf{c} \in C$, exiting l' are disabled (note that both values are a function of the invariant $\mathcal{I}_i(l')$, of the edge conditions and of the current valuation v'_{var}). If $d_{\min}(l', v'_{\text{var}})$ is not defined, then d_i is ∞; otherwise, if $d_{\min}(l', v'_{\text{var}})$ is defined, d_i is a realization of the probability distribution $\mu_i(l', v'_{\text{var}})$. If $d_{\max}(l', v'_{\text{var}})$ is finite, then $\mu_i(l', v'_{\text{var}})$ is a uniform distribution over the interval $[d_{\min}(l', v'_{\text{var}}), d_{\max}(l', v'_{\text{var}})]$; otherwise, $\mu_i(l', v'_{\text{var}})$ is an exponential distribution over $[d_{\min}(l', v'_{\text{var}}), \infty)$. If d_i is ∞, by input-enabledness, then A_i can take an edge, whose event is $\mathsf{c}?$, for some $\mathsf{c} \in C$. Otherwise, by definition of d_i, after d_i time units from the current discrete transition, automaton A_i can surely take an edge, whose event is possibly $\mathsf{c}!$, for some $\mathsf{c} \in C$. Since the network consists of n automata, the minimum allowed progress d_{m} is selected among the n delays $d_1, \dots, d_n$. If d_{m} is finite, then d_{m} is the time the network waits before an automaton performs a new discrete transition.

The wait between the execution of two discrete transitions, lasting a generic $\delta > 0$ time units, is a timed transition, i.e., a configuration change such that no location of the automata in the network is modified but values of the variables evolve because of the elapsing of time. The configuration of the i-th automaton at the end of this wait is (l'', v''_{var}), with $l'' = l'$, where (l', v'_{var}) is the configuration whence the timed transition starts. All the variables of the set W_i evolve according to the flow conditions $\mathcal{F}_i(l')$. In the case of clocks $x \in X_i$, for instance, they are incremented by the value δ, hence, $v''_{\text{var}}(x) = v'_{\text{var}}(x) + \delta$ holds. The value of the other variables is determined based on the differential equation specified by $\mathcal{F}_i(l')$. With the adopted semantics, δ is the value d_{m} calculated at the occurrence of the last discrete transition.

At the end of the time interval lasting d_{m} units of time, the automaton A_i such that $d_i = d_{\text{m}}$ holds performs a discrete transition for some event $\mathsf{c}!$, with $\mathsf{c} \in C$. If several edges are enabled in (l', v'_{var}), probability distribution $\mathcal{P}(l')(\mathsf{c}!, \gamma', \xi', l'') \in [0, 1]$ with $l'' \in L_i$, $\gamma' \in \Gamma_i(W_i)$, and $\xi' \in \wp(\Xi_i(W_i))$ determines how likely the system is to evolve in one direction rather than the other. In Fig. A.15, $\mathsf{p_L}$ and $\mathsf{p_H}$ are the probability of the switching of the heating when the window is open or closed, respectively, which takes place after the synchronization between the two automata has been achieved through channel `on`. Channels $\texttt{on}_{\text{H}}$ and $\texttt{on}_{\text{L}}$ in Fig. A.15 model a probabilistic choice and are not intended for synchronizing the two automata.

We remark that some of the models presented in this work do not conform with the disjointness of the set of real-valued variables, hence two or more automata can use the same variable. This, however, is with no loss of generality in our work, because it is always possible to introduce suitable transitions and local copies of the shared variables and build a network such that all sets of real-valued variables are pairwise disjoint. An automaton $\mathcal{A}_1$ can always make an automaton $\mathcal{A}_2$ change a variable v in W_2 by means of two synchronizing edges with a dedicated event, possibly representing the operation to be carried out on v.

Appendix B. DSL for Framework Validation Scenarios

This Appendix contains the DSL configuration of scenarios DPa, DPb, and DPc. The complete .dsl file is constituted by the concatenation of Listings 5 through 9.

Listing 5: DSL section defining layout areas (i.e., the rectangles higlighted in Fig. 12b) and POIs: specifically, the entrances to the three offices, to the waiting room and emergency room, the two cupboards, main entrance, and robot's recharge station. As all scenarios are set in the same layout, the DSL features only one layout definition.

```
param measurement_unit cm
define layout:
  area a1 in (0.0,110.0) (1550.0,299.5)
  area a2 in (0.0,110.0) (185.0,850.0)
  area a3 in (0.0,672.5) (1550.0,850.0)
  area a4 in (1352.0,110.0) (1550.0,850.0)
  area a5 in (2970.0,110.0) (4512.5,299.5)
  area a6 in (2970.0,110.0) (3155.0,850.0)
  area a7 in (2970.0,672.5) (4512.5,850.0)
  area a8 in (4322.0,110.0) (4512.5,850.0)
  area a9 in (1945.0,0.0) (2670.0,695.0)
  area a10 in (1352.0,110.0) (3155.0,425.0)

  poi OFF1 in (200.0, 200.0)
  poi OFF2 in (4400.0, 200.0)
  poi OFF3 in (4400.0, 700.0)
  poi R1a in (1200.0, 680.0)
  poi R1b in (400.0, 270.0)
  poi R2 in (4000.0, 270.0)
  poi CUP1 in (1400.0, 450.0)
  poi CUP2 in (3000.0, 450.0)
  poi ENTR in (2300.0, 600.0)
  poi RECH in (4250.0, 450.0)
```

Listing 6: DSL section defining robot Tbot and its features. As illustrated in Section 6, it is a TurtleBot3 Waffle Pi starting with 90% of charge.

```
define robots:
  robot Tbot in (2300.0, 400.0) id 1 type turtlebot3_wafflepi charge 90
```

Listing 7: DSL section defining the human subjects and their features. Patients (P1a, P1b, P1c, and P2c) all have sick fatigue profiles, and only P2c belongs to the elderly age group. Doctors (D1a, D1b, D1c, and D2c) all have healthy fatigue profiles and belong to the elderly age group, except for D2c. Walking speeds are set to 40cm/s for patients, and 100cm/s for doctors.

```
define humans:
  human P1a in (2300.0, 600.0) id 1 speed 40.0 is young_sick freewill
normal
  human D1a in (4400.0, 700.0) id 2 speed 100.0 is elderly_healthy freewill
 low

  human P1b in (2300.0, 600.0) id 1 speed 40.0 is young_sick freewill
normal
  human D1b in (4400.0, 700.0) id 2 speed 100.0 is elderly_healthy freewill
 normal

  human P1c in (2290.0, 600.0) id 1 speed 40.0 is young_sick freewill high
```

```
    human P2c in (2400.0, 580.0) id 2 speed 40.0 is elderly_sick freewill normal
    human D1c in (200.0, 200.0) id 3 speed 100.0 is elderly_healthy freewill low
    human D2c in (4400.0, 700.0) id 4 speed 100.0 is young_healthy freewill normal
```

Listing 8: DSL section defining the service sequences (i.e., the robotic missions). As described in Section 6.2, each scenario corresponds to a mission declaration. Service sequences are defined as in Table 6 and Table 8.

```
define mission DPa:
  do robot_leader for P1a with target R1b
  do robot_follower for D1a with target CUP1
  do robot_follower for D1a with target R2
  do robot_leader for P1a with target R2

define mission DPb:
  do robot_leader for P1b with target R1a
  do robot_transporter for D1b with target CUP2
  do robot_follower for D1b with target R2
  do robot_leader for P1b with target R2

define mission DPc:
  do robot_leader for P1c with target R1a
  do robot_leader for P2c with target R2
  do robot_transporter for D1c with target CUP1
  do robot_follower for D2c with target CUP2
  do robot_follower for D2c with target OFF3
  do robot_leader for P1c with target OFF1
  do robot_leader for P2c with target OFF3

define mission R-DPa:
  do robot_leader for P1a with target R2
  do robot_follower for D1a with target CUP1
  do robot_follower for D1a with target R2

define mission R-DPb:
  do robot_transporter for D1b with target CUP2
  do robot_follower for D1b with target R2
  do robot_leader for P1b with target R2

define mission R-DPc:
  do robot_leader for P2c with target R1b
  do robot_leader for P1c with target R1a
  do robot_transporter for D1c with target CUP1
  do robot_follower for D2c with target CUP2
  do robot_follower for D2c with target OFF3
  do robot_leader for P1c with target OFF1
  do robot_leader for P2c with target OFF3
```

Listing 9: DSL section defining the queries to be performed for the design-time analysis. Queries defined in this Listing yield the results shown in Table 7 and Table 9.

```
define queries of mission DPa:
  compute probability_of_success with duration 400 runs auto
  compute probability_of_success with duration 350 runs auto
  compute probability_of_success with duration 300 runs auto
  compute expected_charge with duration 400 runs auto
  compute expected_fatigue with duration 400 runs auto

define queries of mission DPb:
  compute probability_of_success with duration 520 runs auto
  compute probability_of_success with duration 450 runs auto
  compute probability_of_success with duration 400 runs auto
  compute expected_charge with duration 520 runs auto
  compute expected_fatigue with duration 520 runs auto

define queries of mission DPc:
  compute probability_of_success with duration 1500 runs auto
  compute probability_of_success with duration 1400 runs auto
  compute probability_of_success with duration 1300 runs auto
  compute expected_charge with duration 1500 runs auto
  compute expected_fatigue with duration 1500 runs auto

define queries of mission R-DPa:
  compute probability_of_success with duration 300 runs auto
  compute probability_of_success with duration 250 runs auto
  compute probability_of_success with duration 200 runs auto
  compute expected_charge with duration 300 runs auto
  compute expected_fatigue with duration 300 runs auto

define queries of mission R-DPb:
  compute probability_of_success with duration 350 runs auto
  compute probability_of_success with duration 320 runs auto
  compute probability_of_success with duration 300 runs auto
  compute expected_charge with duration 350 runs auto
  compute expected_fatigue with duration 350 runs auto

define queries of mission R-DPc:
  compute probability_of_success with duration 1500 runs auto
  compute probability_of_success with duration 1400 runs auto
  compute probability_of_success with duration 1300 runs auto
  compute expected_charge with duration 1500 runs auto
  compute expected_fatigue with duration 1500 runs auto
```