Received July 30, 2020, accepted August 23, 2020, date of publication August 31, 2020, date of current version September 16, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3020265

Scheduling of a Robot's Tasks With the TaskER Framework

WOJCIECH DUDEK, (Student Member, IEEE), AND TOMASZ WINIARSKI, (Member, IEEE)
Warsaw University of Technology, Institute of Control and Computation Engineering, 00-665 Warsaw, Poland

Corresponding author: Wojciech Dudek (wojciech.dudek@pw.edu.pl)

This work was supported by the AAL JP (Active and Assisted Living Joint Programme) funded by the European Union's Horizon2020 Research and Innovation Programme Project "INCARE—Integrated Solution for Innovative Elderly Care," under Project AAL-2017-059-INCARE.

ABSTRACT Robots, in contrast to typical computational systems, affect the physical environment directly. Therefore, other assumptions must be considered for task management procedures in these system types. Robots coexist with humans in the environment and act upon potentially dangerous objects (e.g., a cooker); hence, extra safety procedures in robot task harmonisation must be ensured. Additionally, an algorithm that schedules tasks for a robot and optimises the robot's operation needs to consider the robot motion time, dynamics of the physical processes, changes in the robot user preferences and changes in the environment made by other habitants. In this article, we investigate the problem of switching between various independent tasks safely and state requirements for a control system resolving it. The tasks are uploaded to a store, launched on a robot at the user's request (similar to smartphone applications) and scheduled following a configurable algorithm. Furthermore, we design a model of systems satisfying the requirements. The systems are structured with agents of different classes. We propose a task-switching procedure and dedicated states of the finite-state machines describing the operation of the agents. Finally, we present a TaskER framework implementing the model, and we verify the model through the execution of an exemplary system in scenarios showing the benefits of the model implementation. As a result of our approach application, the robot tasks can be safely interrupted, postponed, resumed, and potential danger (e.g., leaving a cooker on for a long time) can be minimised.

INDEX TERMS Intelligent robots, cyber-physical systems, agent-based modeling, multitasking, robot programming.

I. INTRODUCTION

The development of cutting-edge technology increases the applicability, universality and popularity of robots. In the robot classification according to application field that was published in [1], there are 2 main branches (industrial and service robots), which are compositions of lower-level robot classes. The inspection of the service robot branch reveals the versatility of the applications that assist humans in, e.g., the areas of medicine (home care and telehealthcare [2]), home (automated vacuum cleaners [3]), education (multi-agent platform for teaching mobile robotics [4], a survey of robotics for education [5]) and defence (a security robot for human-robot interaction [6]). The applications differ in terms of their constraints and requirements for robotic systems. For example, robots for medicine are typically targeted to maximise human safety and personnel convenience, whereas industrial robots optimise production quality and quantity. In each application, robots conduct various tasks (e.g., personnel assistance [7], object transportation [8], or therapy [9] in medicine). Some tasks demand computational power and storage that exceed those that are delivered with a robot. Therefore, robotic systems are being supported by cloud computing [10]. In multi-robot systems, the cloud delivers common system-wide services for various robots or enables cooperation of the robots [11]. One example of such a service is a task store [12], which enables application domain experts to compose and deploy new tasks to the system.

The recent COVID-19 pandemic highlights a need for easy and convenient robot task configuration, especially in unexpected scenarios. Using the task store, robots that are available in galleries or at universities can be easily configured to help medical personnel at hospitals [13]. The tasks can be requested via multiple human-machine interfaces (Internet

The associate editor coordinating the review of this manuscript and approving it for publication was Fabrizio Messina.

of Things devices [14] or human-robot interfaces [15], [16]) by various operators (e.g., medical personnel or patients) at any time during the operation of the robot. Moreover, not all task requests are consulted between users, and it may be necessary to interrupt a task with another if the scenario changes. Therefore, the robots must manage their tasks and be able to suspend and resume them.

The above problems concern various applications of robots; hence, robotic systems are complicated and require a straightforward development method. The model-driven engineering approach, which is based on the model concept, facilitates the design procedure of a system and prevents the influence of cognitive biases [17] on the design of the system architecture at the initial phase [18].

In our analysis of various robotic systems, a set of typical use cases was identified, while the robots switch from one task to another. In the following part of this section, we define the problem of our work by description of the use cases and specification of the requirements for a robot system with modular tasks. The system is enhanced with task scheduling and cloud computing. At the end of the introduction, we discuss the contributions, applicability and concept of our work. In the subsequent Section II, a formal model of a robot control system that satisfies the requirements and follows the concept is described. Next, in Section III, we evaluate our approach via proposition of a verification scenario, implementing the model in the TaskER (Task schedulER) framework, and configuration and execution of an exemplary system. Furthermore, we describe how the system satisfied the requirements during the scenario execution. Finally, we discuss related works in the areas of robot system architectures, task models and task switching approaches for robots (Section IV), and we present the conclusions of our study in Section V. In Appendix A we show cooperation of the system parts in the proposed task switching procedure and in Appendix B we describe the analysis of the conducted verification proving that the exemplary system has the functionality required by the initial problem definition.

A. USE CASES AND REQUIREMENTS

The use cases of a robot system that are considered in this study are presented in Tab. 1. There are two classes of actors who interact with the robot system: developers and users. Based on an analysis of the use cases (which are described in the following subsections), the requirements for the system management of the use cases are formulated.

1) USE CASE – TASKS AS MODULAR EXTENSIONS

There are many possible duties that a robot can perform for its users. For example, in the area of helping elderly people, the study [19] identified multiple activities that threaten independent living in mobility, self-care and social interaction. Therefore, robots should be able to manage multiple tasks and extend the task set even after the system deployment according to the user's demands. A similar problem occurs in the smartphone market, and the solution is the use of an application store, which collects independent programs that can be downloaded and launched upon the request of a user. The application store of the service robot market must contain various tasks. Furthermore, as multiple tasks are available and are uploadable to the robot system on the fly, they must be independent.

TABLE 1. Use cases of the desired system.

Use case name	Tasks as modular extensions
Use case actor	Developer
User action	**System action**
Upload files of a new task to the system	Store the files in the cloud such that they are ready for use by any robot in the system

Use case name	Ease of configuration of the scheduling algorithm
Use case actor	Developer and user
User action	**System action**
Users state the requirements for the algorithm	—
Developers compose the algorithm and inject it as a module	Schedule tasks following the algorithm

Use case name	Coexistence in a shared environment
Use case actor	User
User action	**System action**
—	Expose an interface
Use the interface to request a new task	Initialise and add the task to a queue
Move around, conduct its duties	Update the schedule parameters of tasks by following the rules that are defined in the tasks
	Interrupt tasks gently, with consideration of the safety of users, objects and robots
	Reschedule the tasks: • repetitively, • if any schedule parameter of the queued tasks changes

2) USE CASE – EASE OF CONFIGURATION OF THE SCHEDULING ALGORITHM

Robot systems resolve various problems in many sectors, such as industry [20], healthcare [21] and entertainment [22]. These sectors optimise robot usage to realise various objectives, e.g., in industry, to maximise production quality and quantity, and in healthcare, to minimise the time and effort of nurses and medical personnel. In contrast, robots at the homes of elderly people should optimise the comfort and safety of these individuals. To realise various objectives, the robot controller must utilise a configurable scheduling algorithm that computes scheduling decisions that optimise the cumulative cost during the robot operation. Furthermore, the cost function can change with time and can depend on the specified task (e.g., as the delay in a task of water transportation to an elderly person increases, his/her discomfort and danger increases). Schedule parameters that are used to compute the costs and scheduling decisions can have various forms. Typically, priorities and temporary constraints are used as schedule parameters; however, they can be of any form that reflects the main objective of the robot system (e.g., a scheduling algorithm of a system for helping elderly people can maximise their comfort as a schedule parameter).

3) USE CASE – COEXISTENCE IN A SHARED ENVIRONMENT

Robots interact with the physical environment and affect it by their actions, which compose their tasks. Although the effects of their actions are immediate (turning on a cooker), the objective of the task will be completed with a delay (e.g., boiling water on the cooker) [23]. Therefore, the robot controller must be aware of the potential damage, injury or loss that can be caused by an interruption of an ongoing task (e.g., while a new task is initiated, a cooker remains on during realisation of the new task). Thus, the robot controller must not only consider the current state of the environment but also foresee the future effects of the current actions. Unfortunately, it is difficult to design a comprehensive model of an environment for estimating its state in the future based on the robot actions. Models of tasks are available that describe how the environment changes while the task proceeds [24]. However, even though the effects of the robot's actions are estimable, the environment can also be changed by other actors (such as humans, robots, and animals).

Cooperation with humans is a difficult problem, as humans differ in many aspects, and it is difficult to model their behaviours and needs. These problems also impact the task switching for human-robot collaboration. For example, a task deadline depends on high-level abstract conditions,[1] a user may change his mind regarding the priorities of the requested tasks, or a user can perform a part of a queued task.

4) SYSTEM REQUIREMENTS

We state the following requirements for a robot controller to enable the task harmonisation feature and address the problems that are identified in the above use cases:

- **R1** – the robot controller must maintain additional activities to enable advised task switch. It constantly listens to new task requests, even as the robot proceeds with a task, and potential switching between the ongoing task and the new task is managed as soon as possible;
- **R2** – the values of the schedule parameters that are used to compute scheduling decisions (e.g., priorities) are dynamically changing even if a task awaits execution, and if one of the parameters changes, then the scheduling algorithm is initiated. The parameters can be either task context-dependent or system-wide;
- **R3** – both the algorithm and the parameters that are used in the scheduling procedure depend on a system application and must be configured based on the system requirements;
- **R4** – the tasks that are available in the system are created independently and differ in terms of knowledgebase and context;
- **R5** – developed architecture must raise awareness of the task developer to foresee possible dangerous situations caused by a task switch. Additionally, the tasks execution method needs to enable independent tasks to oversee changes in the environment and in the system in order to change task-dependent schedule parameters. Furthermore, the ongoing task needs to be safely suspended before the controller switches to another task. As a result of this, a possible robot/environment damage or other loss due to a task interruption must be limited; and
- **R6** – a task plan and the initial conditions of its primitive actions may change while the task awaits execution or resumption.

[1]The control system of a robot must know if the user who requested task "B" can pre-empt his/her duties that are related to the task so that the robot could finish the ongoing task, namely, task "A", or if postponing requested task "B" is not acceptable.

B. CONTRIBUTIONS AND APPLICABILITY

In our study, we introduce TaskER model of a robot control system that extends the RAPP (Robotic Applications for Delivering Smart User Empowering Applications) architecture [12] with the task scheduling feature. Furthermore, we describe the TaskER framework, which implements this model. The model is a novel approach to schedule robot's tasks and is an answer to the problem of harmonisation the robot's tasks, i.e. it shapes a multi-robot system to flexibly and with reinforced safety, coordinate, organise and combine tasks assigned to a robot.

1) WORKS UPON WHICH WE RELY

Our system is derived from the RAPP platform and the robot task harmonisation concept that was published in [25]. The specification method of the system follows the embodied agent approach [26], [27], according to which agents are the main units of a system decomposition. Furthermore, the agents are composed of subsystems that carry out a functional part of the system. The embodied agent approach was used in the description of various robot systems, e.g., a dual-robot manipulation station [28], a mobile robot with various modes of locomotion [29], a dual-arm service robot [30] and a service humanoid robot [31]. The RAPP platform consists of agents that are distributed between the robot and the cloud according to the algorithm that was published in [32], [33].

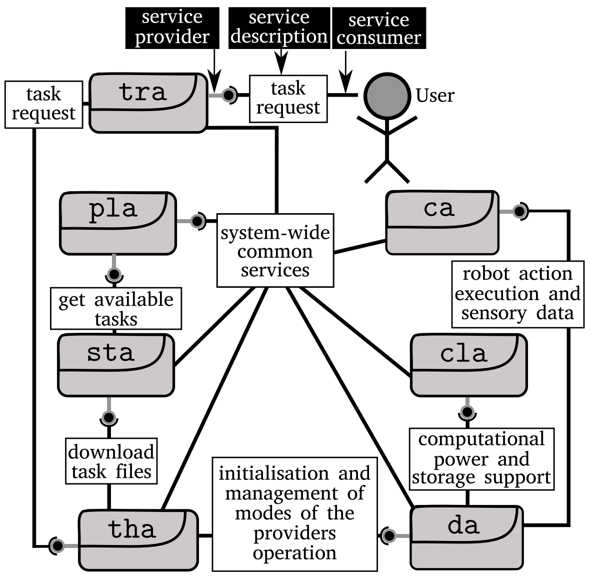

service provider
service description
service consumer
task request
tra
task request
User
pla
system-wide common services
ca
get available tasks
robot action execution and sensory data
sta
cla
computational power and storage support
download task files
initialisation and management of modes of the providers operation
tha
da

FIGURE 1. Cooperation of the agent classes introduced in RAPP (pla, sta, ca, cla, da) and in our approach (tra, tha) – services consumed and provided.

There are five classes of agents. The agents of a class have the same role in the system and have similar functionalities for fulfilling the role. A graphical visualisation of the services that are provided and consumed by classes from RAPP and those introduced in this work is shown in Fig. 1:

1) *Core Agent* (`ca` class) – manages the robot hardware, acts as a low-level controller of the robot and provides common, fundamental behaviours of the robot to the task-level controller—`da` class. Among others, a `ca` class commonly delivers motion planning and execution algorithms (e.g., [34]). Its control software operates on the robot computer. There is one `ca` class for each robot in the system.
2) *Platform Agent* (`pla` class) – a computational agent that operates in the cloud and provides system-wide services that require high computational power or huge storage space. There is one `pla` class agent in the system.
3) *Dynamic Agent* (`da` class) – a computational agent that manages a specified task. Such a task is composed of various actions, such as `ca` class behaviours and platform agent service requests. For example, a `da` class implements a human guiding task that is composed of robot motion action (`ca` class agent behaviour), and detection of a human in the pictures sent to `pla` class agent. In the RAPP project, only one `da` class operates on a robot that completes a task that is requested by a user. When the task is finished, the robot waits for further requests.
4) *Cloud Agent* (`cla` class) – a computational agent that may be spawned in the cloud by a `da` class to delegate complex task-related computation operations from the robot and to store large files in the cloud. It is strictly connected to the task and the `da` class that spawned it. There are at most as many `cla` class as `da` class agents because it is not obligatory for a `da` class to spawn a `cla` class agent in the cloud.
5) *RAPP Store* (`sta` class) – a computational agent that operates in the cloud and stores task files. The files of a specific task are spawned upon user request on the user's robot, and when initiated, become a `da` class agent.

RAPP agents can be assigned to layers via the three-layer architecture approach [35]. Hence, the `pla` class belongs to the deliberation layer, the `da` class constitute the sequencer layer and the `ca` class is the controller layer of a robot. The role and design of a `cla` class depend on a `da` class that is supported by it; however, the most intuitive strategy is to use `cla` class as a task-context-dependent deliberation layer.

2) CONTRIBUTION TO THE SATISFACTION OF THE STATED REQUIREMENTS

Two approaches are available for managing task harmonisation: regarding the tasks as constant and uninterruptible (as in the RAPP approach) or allowing for the system to interrupt the ongoing tasks. Thus, we extend the classification of tasks, robots, and task allocation introduced in [36] with an additional axis—task harmonisation: constant-task harmonisation (CT) vs interruptible-task harmonisation (IT). CT denotes that in case of a new task request during realisation of the current task, the decision on task switching will be postponed until the moment when the ongoing task finishes, while IT denotes that the decision on task switching is taken on the reception of a new task request and the robot can suspend the ongoing task, conduct a set of operations of the new task and restore the previous task from suspension.

Our work enables systems based on variable architecture (RAPP) to use an algorithm and versatile parameters to decide upon task switch. The parameters can be defined in the context of the whole system and can be task context-dependent. The harmonisation procedure proposed in this article is formally and precisely defined enabling its adaption to other architectures.

None of the models or example systems that are presented in the related work section (Section IV) satisfy all requirements that are stated in Section I-A4 and describe the behaviours and structure of a system that can harmonise tasks via the interruptible-task approach. In this article, we introduce a model of a robot control system that has the following features:

1) safe suspension and resumption of independent tasks,
2) facile reconfiguration of the scheduling algorithm and the parameters that are used to compute scheduling decisions,
3) computation of the schedule parameters that depend on a specified task context and abstract knowledge (e.g., an estimated time for completing the task and an estimated time for suspending the current task) and reappraisal of the parameters in reaction to changes in the environment,

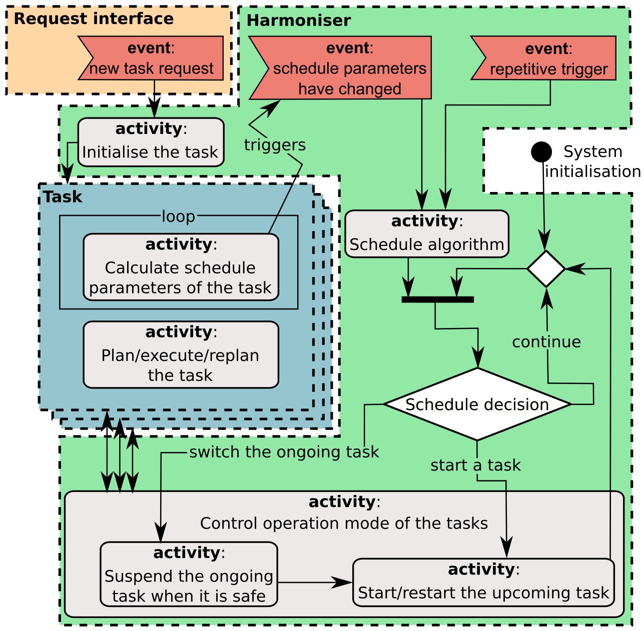

Request interface
event: new task request
Harmoniser
event: schedule parameters have changed
event: repetitive trigger
activity: Initialise the task
triggers
System initialisation
Task
loop
activity: Calculate schedule parameters of the task
activity: Plan/execute/replan the task
activity: Schedule algorithm
continue
Schedule decision
switch the ongoing task
start a task
activity: Control operation mode of the tasks
activity: Suspend the ongoing task when it is safe
activity: Start/restart the upcoming task

FIGURE 2. The procedure for handling task requests and for evaluating dynamically changing schedule parameters. It results in the replacement or continuation of the ongoing task. The harmoniser part was not considered in RAPP and the request interface was a part of a `ca` class agent, therefore, in our solution we define two additional agent classes that cover roles of these parts. The presented concept is formalised in our model and visualised in Fig. 8.

4) facile extension of available tasks,
5) update of a plan of an individual task prior to execution,
6) termination of a queued task if it is no longer beneficial,
7) rescheduling of tasks in reaction to the reappraisal of the schedule parameters, and
8) online task harmonisation reconfiguration in either the constant-task or interruptible-task scenario.

Furthermore, we define a formal notation for describing the model and the systems that are inherited from it. The notation helps the systems' developers in various aspects, for example, in relating analogous entities defined in different approaches, in diagnosing the system state more efficiently, or in making the model definition more precise than if it were defined by ambiguous relations and entities.

Finally, we have implemented the TaskER framework, which implements the model and supports the development of various tasks and scheduling algorithms. The general concept of the harmonisation procedure that is described formally in the model is illustrated in Fig. 2.

We distinguish three main parts in the harmonisation procedure: the request interface, the tasks and the harmoniser. The request interface defines a structure of task requests. Each task calculates its schedule parameters used to compute schedule by the harmoniser part, is responsible for its plan management and execution and share an interface to manage its mode of operation. The harmoniser part basing on the schedule parameters received from the tasks manages their modes of operation as it is defined in the schedule algorithm. The proposed harmonisation procedure prevents multiple tasks to execute their actions at once, what would result in a chaotic behaviour of the robot.

3) WORK CONSTRAINTS AND APPLICABILITY

Our model is based on a component structure and describes the required components and recommended interactions between them for handling task suspension and resumption. Additionally, tasks must be divided into stages, classified as suspendable, which can be interrupted, and blocking, which cannot be interrupted. A possible interruption of an ongoing task that is in a blocking stage is handled in a subsequent stage.

Our approach harmonises tasks that are conducted by a robot and not tasks of a whole system that consists of multiple robots. However, such a system also benefits from our model because the model and its description enable the harmonisation of tasks that are delegated to each robot of the multi-robot system.

Referencing the extended classification of robots, tasks and task allocations [36], our study considers the following:

1) robots that are single-task, namely, can execute at most one task at a time,
2) tasks that are single-robot, namely, require exactly one robot for realisation, and
3) task allocation that is instantaneous-assignment, namely, corresponds to the system that does not possess any information that is suitable for planning future task allocations.

Such a classified system consists of single-task robots that conduct single-robot tasks. However, the system may contain multiple robots that can complete numerous types of single-robot tasks, e.g., object transportation, hazard detection, and object search.

The TaskER model is implemented as a framework built upon ROS and the model extends the RAPP architecture. However, the structure and system behaviours that enable task harmonisation, can be applied to any other system which task model is definable by FSM at least at the top level. Exemplary integration of TaskER with tasks modelled with Petri Nets is presented in this article. The model can be used in a system with knowledge representation in the PDDL [37] to handle harmonisation of the sequences that are deployed by the planning component. Our model allows for `da` class agents to compute and send task-related parameters to the scheduling algorithm to influence the scheduling decision (e.g., the estimated time for completing a task and the estimated time for suspending the current task). The above assumptions are not restrictive; they allow for the development of a multi-tasking robot with interruptible-task harmonisation.

II. MODEL OF A ROBOT SYSTEM WITH A TASK HARMONISATION FEATURE

The model defines both the structure and behaviour of a multi-robot system that harmonises the robots' tasks (separately for each robot). The model is formally specified with embodied agent approach used in a variety of robotic systems (industrial [38], social [31], service [30]). Recently, the approach was expressed in SysML [39] with the use of diagrams visualising core terms and their relations. The problem of system decomposition into layers and modelling communication between the agents composing the system is resolved in [40]. In this section, the notation that is used to describe the proposed architecture is introduced. Then, the overall structure of the system and structures of the classes of agents that are used in it are defined. Finally, the operation and cooperation of the agents are described with finite-state machines (FSMs), UML activity and sequence diagrams.

A. NOTATION AND SYMBOLS

The agents in the embodied agent approach are abstract parts of a system, which communicate with each other and have the imperative to use their resources to realise their own objectives. Thus, the decomposition of a system into agents is based on the division of the system resources and responsibilities. It should be noted that the symbols in indexes used in the notation are divided by commas, so multi-letter symbols should not be confused with multiple single-letter symbols.

We assume that the system controls a set of robots, and the set is designated as $\mathtt{R}$. The set of all agents of the system (denoted as $\mathtt{A}$) is decomposed into two sets of agents. The first set (denoted as ${}^{s}\mathtt{A}$) consists of agents with system-wide responsibility (e.g., general services for all robots in the system, interfaces to other systems, and big data storage and services). The second set (denoted as ${}^{\mathtt{R}}\mathtt{A}$) consists of agents that manage robots' hardware and control the robots to perform tasks. A set of agents that are associated with a specified robot is denoted as ${}^{r}\mathtt{A}$, where r is the robot name ($r \in \mathtt{R}$). To denote a subset of the above sets (${}^{s}\mathtt{A}$, ${}^{\mathtt{R}}\mathtt{A}$, ${}^{r}\mathtt{A}$) that contains agents of a specified class, we use ${}^{h}\mathtt{A}_{u}$, where $h \in \{s, \mathtt{R}, r\}$ specifies the symbol of the set, and $u \in \{\mathtt{ca}, \mathtt{da}, \mathtt{cla}, \mathtt{tha}^{2}, \mathtt{tra}^{2}, \mathtt{pla}, \mathtt{sta}\}$ specifies the class of the agents. The sets and subsets are defined by (1)-(4).

$$\mathtt{A} = {}^{s}\mathtt{A} \cup {}^{\mathtt{R}}\mathtt{A}, \tag{1}$$

$${}^{s}\mathtt{A} = {}^{s}\mathtt{A}_{\mathtt{tra}} \cup {}^{s}\mathtt{A}_{\mathtt{pla}} \cup {}^{s}\mathtt{A}_{\mathtt{sta}}, \tag{2}$$

$${}^{\mathtt{R}}\mathtt{A} = {}^{\mathtt{R}}\mathtt{A}_{\mathtt{ca}} \cup {}^{\mathtt{R}}\mathtt{A}_{\mathtt{da}} \cup {}^{\mathtt{R}}\mathtt{A}_{\mathtt{tha}} \cup {}^{\mathtt{R}}\mathtt{A}_{\mathtt{cla}} \cup {}^{\mathtt{R}}\mathtt{A}_{\mathtt{tra}} = \bigcup_{r \in \mathtt{R}} {}^{r}\mathtt{A}, \tag{3}$$

$${}^{r}\mathtt{A} = {}^{r}\mathtt{A}_{\mathtt{ca}} \cup {}^{r}\mathtt{A}_{\mathtt{da}} \cup {}^{r}\mathtt{A}_{\mathtt{tha}} \cup {}^{r}\mathtt{A}_{\mathtt{cla}} \cup {}^{r}\mathtt{A}_{\mathtt{tra}}. \tag{4}$$

An agent is designated as ${}^{w}a_{j}$, where $w \in (\mathtt{R} \cup \{s\})$, and j is a unique identifier of the agent (e.g., ${}^{r1}a_{12}$). If $w \in \mathtt{R}$, then the agent is associated with a robot, whereas if $w = s$, then the agent is a system-wide agent. An agent consists of subsystems of various classes, however, in this work we omit the subsystem identifier in our notation as each of the agents described in this article consists of one subsystem. Management of behaviours of such an agent is defined by a **finite-state machine**, which is denoted as FSM_{j}. Finite-state machines are composed of states, which

[2] new classes of agents that were introduced in this article in Section II-B

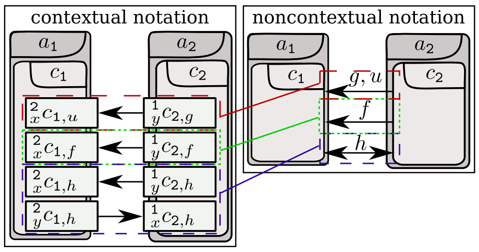


FIGURE 3. Contextual and corresponding non-contextual notation of an inter-agent communication, where c_{α} is a control subsystem of a_{α}.

for a specified FSM_{j}, we denote as $\mathtt{S}_{j}^{s}$, where s is an identifier of the state. A hierarchical FSM is an FSM that includes at least one super state that is defined by another FSM. The FSM that defines a super state $\mathtt{S}_{j}^{s}$ is denoted as $FSM_{j,s}$. A **basic behaviour** $\mathscr{B}_{j}^{b}$ describes an activity of a single-subsystem agent j, where b is an identifier of the basic behaviour. A basic behaviour $\mathscr{B}_{j}^{b}$ is assigned to a state $\mathtt{S}_{j}^{s}$ if $b = s$. **Termination conditions** are logic functions that are assigned to basic behaviours and define an event that terminates a basic behaviour. They are denoted as $tc_{j,b}$. **A transition function** f_{j}^{k} processes data from input buffers and saves it to output buffers, where k is an identifier of the function. A transition function f_{j}^{k} is computed in a basic behaviour $\mathscr{B}_{j}^{b}$ if $k = b$. Transition functions may be divided into **primitive transition functions**; such a function is denoted as pf_{j}^{p}, where p is an identifier of the primitive transition function. Transitions between states that compose an FSM are triggered by logic functions called **initial conditions**, which are denoted as $ic_{j,o}$, where o is an identifier of an initial condition. Subsystems communicate with each other via communication buffers. We distinguish **input buffers**—${}^{g}_{x}c_{j,u}$—and **output buffers**—${}^{g}_{y}c_{j,u}$—where u is an identifier of the buffer, and g is an identifier of the agent to which the buffer is connected. Links between buffers are specified on a structure diagram with either contextual or non-contextual notation, as presented in Fig. 3. Subsystems can store data in their **internal memory**. The internal memory of a control subsystem is denoted as ${}^{c}c_{j}$.

B. STRUCTURES AND BEHAVIOURS OF THE AGENTS

According to the requirements **R1**-**R6**, a robot controller may be requested to begin a task while it proceeds another task ($\mathtt{da}$ class). In contrast to the RAPP approach, our model allows for multiple $\mathtt{da}$ class agents to operate on one robot, but the model of $\mathtt{da}$ class differs from that in the RAPP project. Therefore, we define a set of $\mathtt{da}$ class agents that operate on a specified robot as ${}^{r}\mathtt{A}_{\mathtt{da}}$, where r is an identifier of the robot.

To satisfy the requirements of this study, we introduce additional agent classes that were not considered in the RAPP project—**`task harmoniser`** ($\mathtt{tha}$ class) and **`task requester`** ($\mathtt{tra}$ class). There is one $\mathtt{tha}$ class agent per robot ($|{}^{r}\mathtt{A}_{\mathtt{tha}}| = 1$) in the system, and the agent schedules ${}^{r}\mathtt{A}_{\mathtt{da}}$ agents based on a **scheduling algorithm** that is defined in a transition function of the $\mathtt{tha}$ class agent. The algorithm can use either **task-dependent schedule parameters**, which are computed by ${}^{r}\mathtt{A}_{\mathtt{da}}$ agents, or **task-independent schedule parameters**, which are computed by the algorithm itself. A tha class agent can be requested at any time by a tra class agent to launch a da class agent on the robot with which the tha class agent is associated. In Fig. 1, we present agent classes and services they provide and consume.

Agents from the da, tha, cla, pla, and sta classes consist of a control subsystem only (communicate with other agents and processes data). However, agents of the ca and tra classes can collect data from the environment (using real receptors) and/or affect it (using real effectors). Each of these abilities requires a suitable, additional set of subsystems; hence, agents of the ca and tra classes may consist of multiple subsystems of various types that depend on the agents' abilities. The method of composition of an agent from subsystems of different types was considered in [41].

Implementations of our model may consist of multiple tra class agents that are built with different subsystems and play the roles of various interfaces (e.g., a home automation system, a smartphone, or a human-robot interface). From this perspective, the structure and behaviours of a tra class agent are not important. However, the interfaces of tra class agents are. Hence, they are described in the tha class agent section (Section II-B2) and the tra class agent section (Section II-B3).

Based on the RAPP architecture and descriptions of tha class and tra class agents, we state the cardinalities of the sets of system agents:

$$|{}^{s}\mathtt{A}_{\mathtt{ca}}|=|{}^{s}\mathtt{A}_{\mathtt{tha}}|=|{}^{s}\mathtt{A}_{\mathtt{da}}|=|{}^{\mathtt{R}}\mathtt{A}_{\mathtt{pla}}|=|{}^{\mathtt{R}}\mathtt{A}_{\mathtt{sta}}|=0, \tag{5}$$

$$|{}^{s}\mathtt{A}_{\mathtt{pla}}| = |{}^{s}\mathtt{A}_{\mathtt{sta}}| = 1, \tag{6}$$

$$\forall r \in \mathtt{R} : |{}^{r}\mathtt{A}_{\mathtt{ca}}| = |{}^{r}\mathtt{A}_{\mathtt{tha}}| = 1 \land |{}^{r}\mathtt{A}_{\mathtt{cla}}| \leq |{}^{r}\mathtt{A}_{\mathtt{da}}|, \tag{7}$$

$$\forall r \in \mathtt{R} : |{}^{r}\mathtt{A}_{\mathtt{tra}}| = 0 \lor |{}^{r}\mathtt{A}_{\mathtt{tra}}| = 1. \tag{8}$$

A tra class agent can be associated with either a specified robot or a whole system. Following (8), each robot can have one tra class agent associated with it (we denote it as ${}^{r}a_{rr}$). Table 2 describes additional sets that are used in our work.

In Fig. 4, we present the general structure of a multi-robot system that utilises our model. The figure shows a cross-section in a layer of robot r and presents agent sets, agents of robot r, buffers of all agents and connections between those buffers. The general structure showing the agents associated with the example robot $r \in \mathtt{R}$ consists: ${}^{r}\mathtt{A}_{\mathtt{da}}$ set (including example ${}^{r}a_{dy}$); ${}^{r}a_{ha}$, which is a tha class agent of robot r; ${}^{r}a_{co}$, which is of ca class; ${}^{r}a_{rr}$, which is of tra class; and ${}^{r}\mathtt{A}_{\mathtt{cla}}$, which is a set of cla class agents that are associated with the agents of set ${}^{r}\mathtt{A}_{\mathtt{da}}$. The model consists of system-wide agents that communicate with all robots in the system: ${}^{s}a_{pl}$ of pla class, ${}^{s}a_{st}$ of sta class, and agents of set ${}^{s}\mathtt{A}_{\mathtt{tra}}$.

One of the model constraints is the independent management of tasks that are delegated to a specified robot.

TABLE 2. List of sets that are used in the model and exemplary system, where r is a robot of the system ($r \in \mathtt{R}$), and dy and ha are identifiers of a da class and a tha class agents respectively.

Set	Description
$\mathtt{R}$	set of names of the system robots
$\mathtt{D}_r$	a subset of ${}^{r}\mathtt{A}_{\mathtt{da}}$ that is defined by (28)
$\mathtt{F}_r$	a subset of ${}^{r}\mathtt{A}_{\mathtt{da}}$ that is defined by (41)
$\mathtt{G}_r$	a subset of ${}^{r}\mathtt{A}_{\mathtt{da}}$ that is defined by (42)
$\mathtt{B}_{ha,cmd}$	a set of buffers of ${}^{r}a_{ha}$ that is defined by (30)
$\mathtt{B}_{ha,task}$	a set of buffer of ${}^{r}a_{ha}$ that is defined by (32)
$\mathtt{B}_{ha,report}$	a set of buffers of ${}^{r}a_{ha}$ that is defined by (33)
$\mathtt{B}_{ha,reqSP}$	a set of buffers of ${}^{r}a_{ha}$ that is defined by (34)
$\mathtt{U}_{dy}$	a set of suspendable stages that compose the task that is realised by the ${}^{r}a_{dy}$ agent. A suspendable stage is expressed by (25)
$\mathtt{L}_{dy}$	a set of blocking stages that compose the task that is realised by the ${}^{r}a_{dy}$ agent. A blocking stage is expressed by (26)

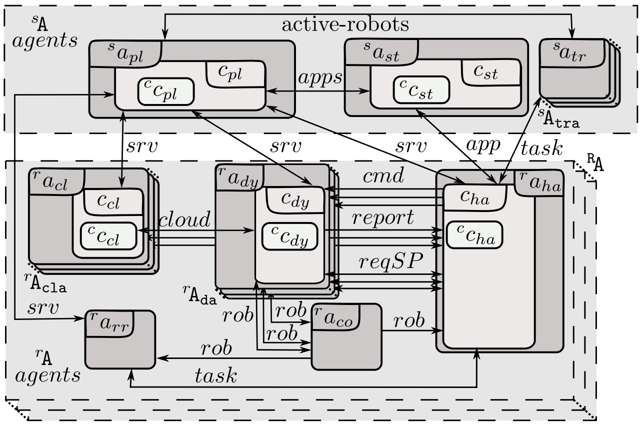


FIGURE 4. Decomposition of our architecture is derived from RAPP architecture, but new tra and tha class agents are introduced in our work (managing request interface and harmoniser roles visualised in Fig. 2), communication buffers are presented in the non-contextual notation shown in Fig. 3 and the roles of the inter-agent connections are presented in Fig. 1.

Therefore, in the next sections, we assume the following instances of the agent classes and cardinalities of the agent sets:

$$|\mathtt{R}| = 1 \iff |{}^{\mathtt{R}}\mathtt{A}_{\mathtt{ca}}| = 1 \land |{}^{\mathtt{R}}\mathtt{A}_{\mathtt{tha}}| = 1, \tag{9}$$

$$\mathtt{R} = \{r\},\ {}^{r}\mathtt{A}_{\mathtt{ca}} = \{{}^{r}a_{co}\},\ {}^{r}\mathtt{A}_{\mathtt{tha}} = \{{}^{r}a_{ha}\},\ {}^{r}\mathtt{A}_{\mathtt{tra}} = \emptyset \tag{10}$$

$${}^{s}\mathtt{A}_{\mathtt{tra}} = \{{}^{s}a_{tr}\},\ {}^{s}\mathtt{A}_{\mathtt{pla}} = \{{}^{s}a_{pl}\},\ {}^{s}\mathtt{A}_{\mathtt{sta}} = \{{}^{s}a_{st}\}. \tag{11}$$

1) DYNAMIC AGENT CLASS

A da class plays a task bearer role and conducts a set of basic behaviours that are required for performing the task, along with a set of additional behaviours that are defined in our model to satisfy requirements of our study. In addition to the basic behaviours, we define a structure of $\mathtt{da}$ class agents, their FSM and communication buffers. We demonstrate the model of a $\mathtt{da}$ class on an example ${}^{r}a_{dy}$, which is associated with robot r (${}^{r}a_{dy} \in {}^{r}\mathbb{A}_{\mathtt{da}}$). We assume that conditions (9)-(11) are satisfied and that ${}^{r}a_{cl}$ supports ${}^{r}a_{dy}$ (12):

$$ {}^{r}\mathbb{A}_{\mathtt{da}} = \{{}^{r}a_{dy}\}, \; {}^{r}\mathbb{A}_{\mathtt{cla}} = \{{}^{r}a_{cl}\}, \; {}^{r}a_{cl} \sim {}^{r}a_{dy}, \tag{12} $$

where '$\sim$' denotes the support relation between a $\mathtt{cla}$ class agent and a $\mathtt{da}$ class agent.

a: BUFFERS OF da *CLASS AGENTS*

Each agent of $\mathtt{da}$ class has 10 buffers. The first pair of buffers, namely, ${}^{co}_{x}c_{dy,rob}$ and ${}^{co}_{y}c_{dy,rob}$, is used to communicate with ${}^{r}a_{co}$ to

1) command the robot (${}^{co}_{y}c_{dy,rob}$),
2) receive information about the robot and its environment (${}^{co}_{x}c_{dy,rob}$).

The second pair, namely, ${}^{cl}_{x}c_{dy,cloud}$ and ${}^{cl}_{y}c_{dy,cloud}$, is used to communicate with ${}^{r}a_{cl}$, e.g., to request complex computation services or algorithms that require user data that are stored in the cloud.

Transitions of FSM_{dy} (Fig. 5a) are triggered based on the value of ${}^{ha}_{x}c_{dy,cmd}$ buffer (13), where *data* consists of any constraints or arguments that are required by ${}^{r}a_{dy}$ for handling the command. However, the data field of the buffer is not obligatory for systems that utilise our model:

$$ {}^{ha}_{x}c_{dy,cmd} = [triggerFlag, data] \tag{13} $$
$$ triggerFlag \in \{start, susp, term\} \tag{14} $$

If the *triggerFlag* value consists of an identifier of an initial condition that is defined in FSM_{dy}, then the initial condition is triggered (15)-(17).

$$ {}^{ha}_{x}c_{dy,cmd} = [start, data] \Rightarrow ic_{dy,start} = true \tag{15} $$
$$ {}^{ha}_{x}c_{dy,cmd} = [susp, data] \Rightarrow ic_{dy,susp} = true \tag{16} $$
$$ {}^{ha}_{x}c_{dy,cmd} = [term, data] \Rightarrow ic_{dy,term} = true \tag{17} $$

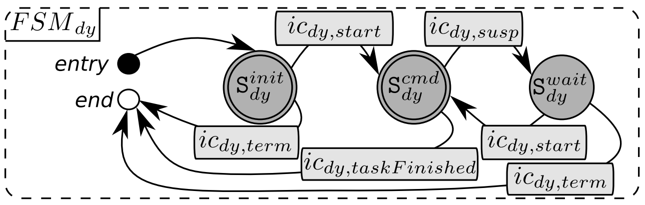
Buffer ${}^{ha}_{y}c_{dy,report}$ transmits the report of agent ${}^{r}a_{dy}$, which contains at least an identifier of the current state of FSM_{dy} (Fig. 5):

$$ \begin{aligned} {}^{ha}_{y}c_{dy,report} &= \{fsmState\} \\ fsmState &\in \{initComm, compSP, upTsk, exeTsk, \\ &\qquad susp, wait, end\} \end{aligned} \tag{18} $$

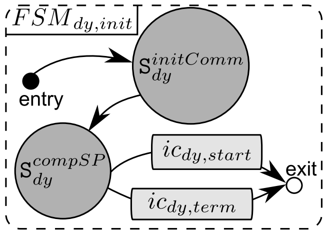
Additionally, this buffer may be extended with schedule parameters that are task-dependent, which must be calculated by ${}^{r}a_{dy}$:

$$ {}^{ha}_{y}c_{dy,report} = \{fsmState, scheduleParams\} \tag{19} $$

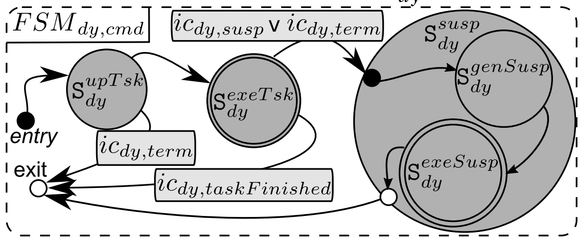
The structure of the *scheduleParams* field depends on the scheduling algorithm of the system. Therefore, it is configurable and not expressed in our model.



(a) Graph of the top level of the hierarchical FSM that governs the behaviours of a $\mathtt{da}$ class agent, where super states S^{init}_{dy} and S^{cmd}_{dy} are represented by $FSM_{dy,init}$ (Fig. 5b) and $FSM_{dy,cmd}$ (Fig. 5c), respectively.



(b) Internal FSM of the S^{init}_{dy}.



(c) Internal FSM of the S^{cmd}_{dy}.

FIGURE 5. **Hierarchical finite-state machine governing the operation of a** $\mathtt{da}$ **class presented on an exemplary agent** ${}^{r}a_{dy}$.

A pair of buffers (${}^{ha}_{x}c_{dy,reqSP}$ and ${}^{ha}_{y}c_{dy,reqSP}$) are used to deliver argument-dependent schedule parameters that are calculated by ${}^{r}a_{dy}$ on request from ${}^{r}a_{ha}$. The input buffer contains an identifier of a schedule parameter (*spID*) and the arguments that are required for calculating the parameter (*args*), and the output buffer stores the value of the requested parameter (*scheduleParam*). The buffers have the following structures:

$$ {}^{ha}_{x}c_{dy,reqSP} = [spID, \; args] \tag{20} $$
$$ {}^{ha}_{y}c_{dy,reqSP} = [scheduleParam] \tag{21} $$

An example schedule parameter that is argument-dependent is the estimated time for completing a task under specified conditions (e.g., a starting robot pose). The ${}^{ha}_{x}c_{dy,reqSP}$ and ${}^{ha}_{y}c_{dy,reqSP}$ buffers are used only in systems with harmonisation that is based on argument-dependent schedule parameters.

Finally, a set of buffers, namely, ${}^{pl}_{x}c_{dy,srv}$ and ${}^{pl}_{y}c_{dy,srv}$, are used to communicate with the ${}^{r}a_{pl}$ to call system-wide services that are supplied by ${}^{r}a_{pl}$, such as text-to-speech or speech-to-text.

b: FSM OF da *CLASS AGENTS*

The lifecycle of a $\mathtt{da}$ class is managed by a hierarchical FSM with three states at the top level—FSM_{dy} (Fig. 5a):

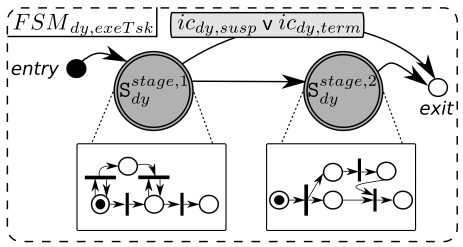

$FSM_{dy,exeTsk}$
$ic_{dy,susp} \vee ic_{dy,term}$
entry
$\mathsf{S}_{dy}^{stage,1}$
$\mathsf{S}_{dy}^{stage,2}$
exit

FIGURE 6. Integration of the TaskER model with a task modelled with Petri Nets.

1) S_{dy}^{init} – the process of ${}^{r}a_{dy}$ is initiated in the robot's computer. The internal FSM of this state is presented in Fig. 5b. Agent ${}^{r}a_{dy}$ is allowed to pull data from ${}^{r}a_{co}$ to compute schedule parameters and obtain the addresses of the remote services. However, ${}^{r}a_{dy}$ does not command ${}^{r}a_{co}$ to perform the task that is implemented in ${}^{r}a_{dy}$. This state consists of two internal states:
   1.1. $\mathsf{S}_{dy}^{initComm}$ – ${}^{r}a_{dy}$ creates communication interfaces and inititalises the values of its internal memory ${}^{c}c_{dy}$;
   1.2. S_{dy}^{compSP} – ${}^{r}a_{dy}$ sets the value of ${}^{ha}_{y}c_{dy,report}$ (18)-(19), and if the system uses task-dependent schedule parameters, then ${}^{r}a_{dy}$ computes these task-dependent schedule parameters, which are used by ${}^{r}a_{ha}$ to manage the operation of the agents from set ${}^{r}\mathbb{A}_{\mathtt{da}}$.
2) S_{dy}^{cmd} – ${}^{r}a_{dy}$ commands ${}^{r}a_{co}$ to complete its task. This state is represented by a hierarchical FSM (Fig. 5c), where
   2.1. S_{dy}^{exeTsk} is specified by an FSM ($FSM_{dy,exeTsk}$) that describes the task of ${}^{r}a_{dy}$, and each state of this FSM represents a **stage of the task**. The states of the FSM are denoted as $\mathsf{S}_{dy}^{stage,k}$, where k is an identifier of the stage. The stages can be defined by any task model (Petri Net, Behaviour tree, DSL, FSM). In case of a task based on Petri Nets, an exemplary $FSM_{dy,exeTsk}$ has the form presented in Fig. 6.
   2.2. S_{dy}^{upTsk} modifies or re-plans task defined by $FSM_{dy,exeTsk}$ to adapt it to changes in the environment that occurred when ${}^{r}a_{dy}$ was waiting for the execution of its task.
   2.3. S_{dy}^{susp} consists of an internal FSM that is composed of two states: $\mathsf{S}_{dy}^{genSusp}$ and $\mathsf{S}_{dy}^{exeSusp}$. The first creates $FSM_{dy,exeSusp}$ for governing the behaviours of the robot, which is required for suspending the ongoing task safely. The second state, namely, $\mathsf{S}_{dy}^{exeSusp}$, is a super state that is defined by $FSM_{dy,exeSusp}$. State S_{dy}^{susp} is responsible for conducting a set of actions to
      i) set the robot and the environment in safe conditions at the time of the task switching,
      ii) enable the resumption of the task that will be conducted by ${}^{r}a_{dy}$.
3) S_{dy}^{wait} – ${}^{r}a_{dy}$ awaits resumption, sets the value of ${}^{ha}_{y}c_{dy,report}$ and calculates only the schedule parameters.

If $|{}^{r}\mathbb{A}_{\mathtt{da}}| \geq 1$, then only one agent from set ${}^{r}\mathbb{A}_{\mathtt{da}}$ can be in state S_{dy}^{cmd} (other agents can be in other states) and execute its task. The initial conditions of FSM_{dy}, namely, $ic_{dy,start}$, $ic_{dy,term}$, and $ic_{dy,susp}$, are set by ${}^{r}a_{ha}$ via inter-agent connection; hence, ${}^{r}a_{ha}$ must not allow for more than one agent from ${}^{r}\mathbb{A}_{\mathtt{da}}$ to be in state S_{dy}^{cmd}. Only two initial conditions—$ic_{dy,taskFinished}$ and $ic_{dy,term}$—can be set by ${}^{r}a_{dy}$:
1) $ic_{dy,taskFinished}$ – the task that is implemented in S_{dy}^{exeTsk} is completed and reaches the final stage (the end state of $FSM_{dy,exeTsk}$ sets $ic_{dy,taskFinished}$),
2) $ic_{dy,term}$ – the task of ${}^{r}a_{dy}$ is aborted by a user or is no longer feasible, while the agent is in state S_{dy}^{exeTsk}, S_{dy}^{compSP}, or S_{dy}^{wait}.

c: BASIC BEHAVIOURS OF da CLASS AGENTS

The transition function that is calculated during the $\mathscr{B}_{dy}^{compSP}$ basic behaviour (f_{dy}^{compSP}) consists of one primitive transition function (pf_{dy}^{compSP}), which is composed of multiple primitive transition functions, each of which computes a distinct schedule parameter (e.g., the priority and the estimated time for completing the task that is implemented in ${}^{r}a_{dy}$). Equation (22) defines a composition of primitive transition functions in f_{dy}^{compSP} and triggering events of each of them.

$$f_{dy}^{compSP} = pf_{dy}^{compSP} = \begin{cases} pf_{dy}^{report} & timer(\text{repRate}) \\ pf_{dy}^{sp,1} & I^{1} \\ pf_{dy}^{sp,2} & I^{2} \\ \ldots & \ldots \\ pf_{dy}^{sp,p} & I^{p}, \end{cases} \tag{22}$$

where pf_{dy}^{report}, with the frequency defined in the *repRate* field of ${}^{c}c_{dy}$, sets the *fsmState* field of the ${}^{ha}_{y}c_{dy,report}$ buffer (given by (18) or (19)) and p is number of schedule parameters that f_{dy}^{compSP} can compute. If the system schedules tasks using task-dependent schedule parameters, then $p > 0$; if it does not, then $p = 0$. Each of the primitive transition functions $pf_{dy}^{sp,1}$-$pf_{dy}^{sp,p}$ fills part of the *scheduleParams* field (19), which the function calculates. By I^{id} we denote a logic sentence that activates the $pf_{dy}^{sp,id}$ primitive transition function. I^{id} can be related to the ${}^{ha}_{x}c_{dy,reqSP}$ buffer; hence, $pf_{dy}^{sp,id}$ will be triggered upon request from ${}^{r}a_{ha}$ in consideration of (20):

$$I^{id} : {}^{ha}_{x}c_{dy,reqSP} = [id,\ args] \tag{23}$$

Moreover, I^{id} can consider the present state of ${}^{r}a_{dy}$ (S_{dy}^{cmd}, S_{dy}^{wait}, or S_{dy}^{init}); hence, the computed schedule parameters will differ among the states of ${}^{r}a_{dy}$.

A transition function of $\mathscr{B}_{dy}^{upTsk}$ (namely, f_{dy}^{upTsk}) creates $FSM_{dy,exeTsk}$. Example approaches are the configuration of a statically defined template of an FSM and the creation of an entirely new FSM via planning algorithms.

A basic behaviour of a task stage ($\mathscr{B}_{dy}^{stage,k}$) consists of a transition function $f_{dy}^{stage,k}$ and a termination condition $tc_{dy,stage,k}$. The transition function is divided into two primitive transition functions:

1) $pf_{dy}^{stage,k}$ – an algorithm that leads to the objective of the k^{th} stage (e.g., reaching a specified destination),
2) pf_{dy}^{compSP} – computes schedule parameters and updates ${}^{ha}_{y}\mathcal{C}_{dy,report}$ during $\mathtt{S}_{dy}^{exeTsk}$ (22). The parameters are used by ${}^{r}a_{ha}$ to schedule ${}^{r}\mathtt{A}_{\mathtt{da}}$ agents.

The trigger events of the primitive transition functions are as follows:

$$f_{dy}^{stage,k} = \begin{cases} pf_{dy}^{stage,k} & timer(taskExeRate) \\ pf_{dy}^{compSP} & timer(SPupRate), \end{cases} \tag{24}$$

where k is an identifier of the task stage and $timer(X)$ is a function that returns true with frequency X. Both $taskExeRate$ and $SPupRate$ are fields of ${}^{c}c_{dy}$. Some stages of a task may involve operations that may not be interrupted, e.g., due to an unstable state of the object that is being manipulated or to an unknown suspension or resumption transition function for this stage. To prevent such a threat, we distinguish two types of task stages in our model:

1) **suspendable** – the task may be suspended in this stage, and the system can switch to another task with the ability to resume the current task in the future;
2) **blocking** – the task may not be suspended in this stage e.g. due to performing an unstable process or resume of the task would be impossible.

The suspendable stages of a task that is conducted by ${}^{r}a_{dy}$ compose the set $\mathtt{U}_{dy}$, and the blocking stages compose the set $\mathtt{L}_{dy}$. Each of the task stage types has an individual definition of a termination condition for terminating $\mathscr{B}_{dy}^{stage,k}$. The termination condition of a blocking-type stage is the satisfaction of the objective condition of the stage, e.g., the robot is at its destination for a navigation stage (25). A termination condition of a suspendable stage is the satisfaction of the objective condition of the stage with the alternative $ic_{dy,susp}$ condition (26). Assume the objective conditions $tc_{dy,stage,k,goal}$ and $tc_{dy,stage,j,goal}$ of a blocking stage $\mathtt{S}_{dy}^{stage,k}$ and a suspendable stage $\mathtt{S}_{dy}^{stage,j}$:

$$tc_{dy,stage,k} = tc_{dy,stage,k,goal} \tag{25}$$

$$tc_{dy,stage,j} = tc_{dy,stage,j,goal} \vee ic_{dy,susp} \tag{26}$$

Thus, if ${}^{r}a_{ha}$ triggers condition $ic_{dy,susp}$ during the operation of $\mathscr{B}_{dy}^{stage,j}$, then the basic behaviour is terminated and $\mathtt{S}_{dy}^{stage,j}$ is switched to $\mathtt{S}_{dy}^{susp}$.

The transition function $f_{dy}^{genSusp}$ of $\mathscr{B}_{dy}^{genSusp}$ creates an FSM (namely, $FSM_{dy,exeSusp}$) that utilises behaviours of the agent and inter-agent communication to suspend the ongoing task safely. The transition function $f_{dy}^{genSusp}$ may be defined by the task developer in the form of a decision tree, which leads to statically defined suspension strategies (given by FSMs). The second approach for defining $f_{dy}^{genSusp}$ is to provide a planner that considers the present state of the world and creates $FSM_{dy,exeSusp}$.

The operation of ${}^{r}a_{dy}$ in state $\mathtt{S}_{dy}^{exeSusp}$ is realised by the internal FSM of the state, namely, $FSM_{dy,exeSusp}$, which was defined in the previous state.

The transition function of the $\mathscr{B}_{dy}^{wait}$ basic behaviour (denoted as f_{dy}^{wait}) equals f_{dy}^{compSP}, as presented in (22):

$$f_{dy}^{wait} = f_{dy}^{compSP} \tag{27}$$

2) TASK HARMONISER AGENT CLASS

Here, we describe the structure and behaviours of `tha` class agents (which correspond to schedulers in operating systems) on an exemplary agent ${}^{r}a_{ha}$. The agent of this class receives and processes requests for new tasks from `tra` class agents and schedule tasks of the robot r. Agent ${}^{r}a_{ha}$ switches the states of the agents from set ${}^{r}\mathtt{A}_{\mathtt{da}}$ to

1) optimise a specified scheduling criterion (e.g., highest priority first, shortest job first, or shortest remaining time first);
2) enable the controlled suspension and resumption of the tasks that are implemented in `da` class.

a: TASK HARMONISER BUFFERS AND MEMORY

The memory of ${}^{r}a_{ha}$ (namely, ${}^{c}c_{ha}$) consists of multiple named fields and they are required to manage dynamic agents. The scheduling of ${}^{r}\mathtt{A}_{\mathtt{da}}$ agents is realised by ${}^{r}a_{ha}$ and is based on modification of the values of ${}^{c}c_{ha}$ fields ${}^{c}c_{ha,exeDA}$, ${}^{c}c_{ha,irrDA}$ and ${}^{c}c_{ha,idleDA}$ by assigning identifiers of ${}^{r}\mathtt{A}_{\mathtt{da}}$ agents to them. The first field stores an identifier of an agent ${}^{r}a_{dy} \in {}^{r}\mathtt{A}_{\mathtt{da}}$ that is currently in state $\mathtt{S}_{dy}^{cmd}$ and conducts the task that is implemented in it. We refer to an agent that is assigned to this field as ${}^{r}a_{exeDA}$. The ${}^{c}c_{ha,irrDA}$ field stores an identifier of an ${}^{r}a_{dy} \in {}^{r}\mathtt{A}_{\mathtt{da}}$ that was chosen by $pf_{ha}^{schedule}$ to replace ${}^{r}a_{exeDA}$. We refer to an agent that is assigned to the ${}^{c}c_{ha,irrDA}$ field as ${}^{r}a_{irrDA}$. ${}^{c}c_{ha,idleDA}$ is a one-dimensional array of size $\mathtt{N}$ that is filled with identifiers of the agents that belong to set $\mathtt{D}_{r}$:

$$\mathtt{D}_{r} = {}^{r}\mathtt{A}_{\mathtt{da}} \setminus \left\{ {}^{r}a_{exeDA}, {}^{r}a_{irrDA} \right\} \tag{28}$$

$$|\mathtt{D}_{r}| = \mathtt{N} \tag{29}$$

The management of the operation of an example ${}^{r}a_{dy} \in {}^{r}\mathtt{A}_{\mathtt{da}}$ is realised by sending state switch requests via buffer ${}^{dy}_{y}c_{ha,cmd}$ to ${}^{ha}_{x}\mathcal{C}_{dy,cmd}$. The state switch request that is sent to ${}^{dy}_{y}c_{ha,cmd}$ has a structure that is similar to that of ${}^{ha}_{x}\mathcal{C}_{dy,cmd}$ and may consist of one of the values that is defined in (14). As the cardinality of ${}^{r}\mathtt{A}_{\mathtt{da}}$ may exceed 1, there is a set $\mathtt{B}_{ha,cmd}$ that contains buffers for managing the states of ${}^{r}\mathtt{A}_{\mathtt{da}}$ agents:

$$\begin{aligned} {}^{dy}_{y}c_{ha,cmd} \in \mathtt{B}_{ha,cmd} &\iff {}^{r}a_{dy} \in {}^{r}\mathtt{A}_{\mathtt{da}} \\ |\mathtt{B}_{ha,cmd}| &= |{}^{r}\mathtt{A}_{\mathtt{da}}| \end{aligned} \tag{30}$$

The ${}^{tr}_{x}c_{ha,task}$ and ${}^{tr}_{y}c_{ha,task}$ buffers are an interface (denoted as $taskIF_{ha,tr}$) for agent ${}^{s}a_{tr}$ of `tra` class. As the cardinality of the sum ${}^{r}\mathtt{A}_{\mathtt{tra}} \cup {}^{s}\mathtt{A}_{\mathtt{tra}}$ may exceed 1, there is a set of input

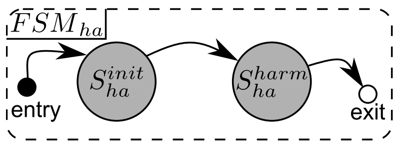


FIGURE 7. Finite-state machine that governs the behaviour of ${}^{r}a_{ha}$.

and output buffers that connect ${}^{r}a_{ha}$ with all $\mathtt{tra}$ class agents in the system:

$$\{{}^{tr}_{x}c_{ha,task}, {}^{tr}_{y}c_{ha,task}\} = taskIF_{ha,tr}, \quad (31)$$

$$taskIF_{ha,tr} \in \mathrm{B}_{ha,task} \Leftrightarrow a_{tr} \in ({}^{r}\mathrm{A}_{\mathtt{tra}} \cup {}^{s}\mathrm{A}_{\mathtt{tra}}), \quad (32)$$

where $\mathrm{B}_{ha,task}$ is a set of task buffers of agent ${}^{r}a_{ha}$ for receiving new task requests and responding to them. Moreover, our model supplies agent ${}^{r}a_{ha}$ with two sets of buffers, namely, $\mathrm{B}_{ha,report}$ and $\mathrm{B}_{ha,reqSP}$, for collecting reports and task-dependent schedule parameters from ${}^{r}\mathrm{A}_{\mathtt{da}}$ agents:

$${}^{dy}_{x}c_{ha,report} \in \mathrm{B}_{ha,report} \Leftrightarrow {}^{r}a_{dy} \in {}^{r}\mathrm{A}_{\mathtt{da}}, \quad (33)$$

$$\left\{{}^{dy}_{x}c_{ha,reqSP}, {}^{dy}_{y}c_{ha,reqSP}\right\} \in \mathrm{B}_{ha,reqSP} \Leftrightarrow {}^{r}a_{dy} \in {}^{r}\mathrm{A}_{\mathtt{da}} \quad (34)$$

The buffers of these sets are used to

1) $\mathrm{B}_{ha,report}$ – collect the current states of ${}^{r}\mathrm{A}_{\mathtt{da}}$ agents and their schedule parameters (18),(19);
2) $\mathrm{B}_{ha,reqSP}$ – request and receive from ${}^{r}a_{dy}$ argument-dependent data for the scheduling algorithm.

There is a pair of buffers, namely, ${}^{st}_{y}c_{ha,app}$ and ${}^{st}_{x}c_{ha,app}$, for collecting files of tasks that are stored in ${}^{s}a_{st}$. Additionally, ${}^{r}a_{ha}$ consists of ${}^{pl}_{x}c_{ha,srv}$ and ${}^{pl}_{y}c_{ha,srv}$ buffers, which are used to communicate with ${}^{s}a_{pl}$ to call system-wide services that are supplied by ${}^{s}a_{pl}$ (e.g., the current list of the tasks that are available in ${}^{s}a_{st}$).

Finally, there is an input buffer ${}^{co}_{x}c_{ha,rob}$, which can be used by the scheduling algorithm that is implemented in ${}^{r}a_{ha}$ to obtain data regarding the current state of the robot and its environment.

b: BASIC BEHAVIOURS OF $\mathtt{tha}$ CLASS AGENTS

Management of ${}^{r}a_{ha}$ behaviours is specified by the FSM with two states (Fig. 7) that point to the basic behaviours: S^{init}_{ha} to $\mathscr{B}^{init}_{ha}$ and S^{harm}_{ha} to $\mathscr{B}^{harm}_{ha}$. The former basic behaviour depends on the implementation of ${}^{r}a_{ha}$ as it is responsible for initialising ${}^{c}c_{ha}$ and establishing connections of the buffers of ${}^{r}a_{ha}$. The latter ($\mathscr{B}^{harm}_{ha}$) consists of a composite transition function, which is decomposed into the following:

1) pf^{initDA}_{ha} – which reads data from buffer ${}^{tr}_{x}c_{ha,task}$, assigns an identifier to the received task (e.g., dy), sets up ${}^{r}a_{dy}$ and adds it to set D_r. Finally, FSM_{dy} managing operation of ${}^{r}a_{dy}$ is initiated, and the initial arguments are passed from the requester (${}^{s}a_{tr}$) to ${}^{r}a_{dy}$,
2) $pf^{schedule}_{ha}$ – which defines the scheduling algorithm, which, with a configurable frequency (defined in the ${}^{c}c_{ha,sFreq}$ field) and based on the schedule parameters that are received from buffers $\mathrm{B}_{ha,report}$ and $\mathrm{B}_{ha,reqSP}$, assigns identifiers of the agents from set D_r to the ${}^{c}c_{ha,irrDA}$ field. The algorithm of this primitive transition function is configurable because it depends on a specified system and its objectives; hence, it is left to be defined by the designer of the robot system. The function is defined in (35):

$$pf^{schedule}_{ha} : \left\{\mathrm{B}_{ha,report}, \mathrm{B}_{ha,reqSP}\right\} \rightarrow {}^{c}c_{ha,irrDA} \quad (35)$$

3) $pf^{switchDA}_{ha}$ – which reads the value of the ${}^{c}c_{ha,irrDA}$ field and sets the values of $\mathrm{B}_{ha,cmd}$ buffers to either suspend, start, or terminate agents from ${}^{r}\mathrm{A}_{\mathtt{da}}$. This primitive transition function sets the ${}^{c}c_{ha,exeDA}$ field. Its algorithm is presented as Alg. 1.

Algorithm 1 Algorithm of $pf^{switchDA}_{ha}$

```
Result: Suspension of the task of ${}^{r}a_{exeDA}$ and
        starting/resumption of the task of ${}^{r}a_{irrDA}$
1  //check if ${}^{r}a_{exeDA}$ executes its task
2  if ${}^{exeDA}_{x}c_{ha,report}$ == [cmd, scheduleParams] then
3      ${}^{exeDA}_{y}c_{ha,cmd}$ = [susp, data];
4  //check if ${}^{r}a_{exeDA}$ has suspended its task
5  else if ${}^{exeDA}_{x}c_{ha,report}$ == [wait, scheduleParams] then
6      $\mathrm{D}_r = \mathrm{D}_r \cup \{{}^{c}c_{ha,exeDA}\}$;
7      ${}^{c}c_{ha,exeDA} = \{\}$;
8  //check if there is no agent assigned to ${}^{r}a_{exeDA}$ and if there is ${}^{r}a_{irrDA}$
9  if ${}^{c}c_{ha,exeDA}$ == {} $\wedge$ ${}^{c}c_{ha,irrDA} \neq$ {} then
10     ${}^{c}c_{ha,exeDA} = {}^{c}c_{ha,irrDA}$;
11     ${}^{c}c_{ha,irrDA} = \{\}$;
12     ${}^{irrDA}_{y}c_{ha,cmd}$ = [start, data];
13 end
14 //check if a buffer of $\mathrm{B}_{ha,report}$ consists of 'end' flag and if so, remove the appropriate agent from ${}^{r}\mathrm{A}_{\mathtt{da}}$
15 if $\exists {}^{\alpha}_{x}c_{ha,report} \in \mathrm{B}_{ha,report} \ni {}^{\alpha}_{x}c_{ha,report} = \{end\}$ then
16     ${}^{r}\mathrm{A}_{\mathtt{da}} = {}^{r}\mathrm{A}_{\mathtt{da}} \setminus {}^{r}a_{\alpha}$;
17 end
```

Each of the primitive transition functions is triggered by a specified event, and the primitive transition functions are composed to construct f^{harm}_{ha}, which is presented in (36):

$$f^{harm}_{ha} = \begin{cases} pf^{initDA}_{ha} & \exists x \in \mathrm{B}_{ha,task} : newData(x), \\ pf^{schedule}_{ha} & timer({}^{c}c_{ha,sFreq}) \vee shdl_{event}, \\ pf^{switchDA}_{ha} & {}^{c}c_{ha,irrDA} \neq \emptyset \vee term_{event}, \end{cases} \quad (36)$$

$$\begin{aligned} shdl_{event} &= ({}^{c}c_{ha,exeDA} = \emptyset \wedge {}^{c}c_{ha,irrDA} = \emptyset \wedge \mathrm{D}_r \neq \emptyset) \\ &\quad \vee newData(\mathrm{B}_{ha,report}), \quad (37) \\ term_{event} &= \exists {}^{\alpha}_{x}c_{ha,report} \in \mathrm{B}_{ha,report} \ni {}^{\alpha}_{x}c_{ha,report} = \{end\}, \quad (38) \end{aligned}$$

where $newData(x)$ is true on data changes in buffer x and $timer(x)$ is triggered with a frequency given by x.

3) TASK REQUESTER CLASS AGENTS

Agents of tra class are entities that can request new tasks. There may be many tra class agents in the system, which differ in structure. Some may be only processing agents with a control subsystem only and request new tasks based on a timer, and others may be equipped with a physical device or a sensor that initiates the request for a task. As there may be many robots in the system, agents of $\mathtt{A}_{\mathtt{tra}}$ (in our example, ${}^{s}a_{tr}$) must obtain a list of the robots and their addresses from the pla class (in our example, ${}^{s}a_{pl}$) by the *active-robots* interface (in our example, buffers ${}^{pl}_{y}c_{tr,active-robots}$ and ${}^{pl}_{x}c_{tr,active-robots}$).

Each of the tra class and cla class agents has a pair of buffers for communicating with agent ${}^{s}a_{pl}$ to call system-wide services that are supplied by agent ${}^{s}a_{pl}$. In our example, the buffers are ${}^{pl}_{x}c_{tr,srv}$, ${}^{pl}_{y}c_{tr,srv}$ and ${}^{pl}_{x}c_{cl,srv}$, ${}^{pl}_{y}c_{cl,srv}$.

C. THE HARMONISATION PROCEDURE

Robot systems that utilise our model can schedule their tasks and are composed of agents of various classes. Models of the classes were presented in the previous sections; however, for a comprehensive description of the system behaviour, a depiction of the inter-agent interactions is required. Thus, here, we present the workflow of agents in the face of a typical task harmonisation problem. The system that is considered in the scenarios consists of one robot ($\mathtt{R} = \{r\}$), one tha class agent (${}^{r}\mathtt{A}_{\mathtt{tha}} = \{{}^{r}a_{ha}\}$), one ca class (${}^{r}\mathtt{A}_{\mathtt{ca}} = \{{}^{r}a_{co}\}$) and one tra class (${}^{s}\mathtt{A}_{\mathtt{tra}} = \{{}^{s}a_{tr}\}$). As the objective of this study is to describe the robot task harmonisation model, this section focuses on the workflow of ${}^{s}a_{tr}$, ${}^{r}a_{ha}$ and da class agents and not the other agents, which are described in [12].

An outline of the task harmonisation procedure in a system that utilises our model is presented as an activity diagram Fig. 8. The complex activities that are illustrated in the rounded rectangles and the interaction among the system agents in each of them is presented in the form of a sequence diagram in the Appendix A. The harmonisation procedure is initiated by one of the following events, as marked in the activity diagram:

1) **event 1** – agent ${}^{r}a_{ha}$ receives a new task request from agent ${}^{s}a_{tr}$ – the initial event of pf^{initDA}_{ha} is satisfied,
2) **event 2** – a trigger repetitively initiates $pf^{schedule}_{ha}$ (36).
3) **event 3** – ${}^{c}c_{ha,exeDA}$ and ${}^{c}c_{ha,irrDA}$ are empty, but there is an idle daclass agent or ${}^{r}a_{ha}$ receives new report with schedule parameters (37).

The procedure is terminated if either *activity 1*, *activity 5*, *activity 6*, or *activity 7* is completed. The procedure consists of two decision nodes: *D1* and *D2*. The former is realised by the initial event of $pf^{switchDA}_{ha}$, which checks the output of $pf^{schedule}_{ha}$. The latter decision node, namely, *D2*, is checked in the conditions of Alg. 1.

III. VERIFICATION

The verification of the model is based on the design, development and testing of an exemplary system that is based on the

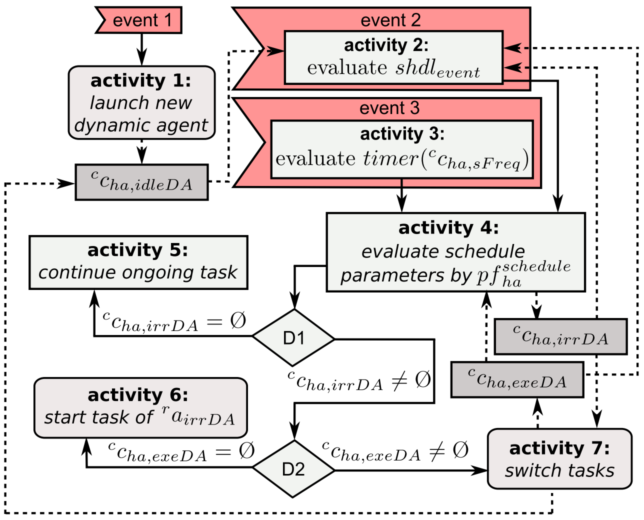

event 1
event 2
activity 2:
evaluate $shdl_{event}$
activity 1:
launch new dynamic agent
event 3
activity 3:
evaluate $timer({}^{c}c_{ha,sFreq})$
${}^{c}c_{ha,idleDA}$
activity 4:
evaluate schedule parameters by $pf^{schedule}_{ha}$
activity 5:
continue ongoing task
${}^{c}c_{ha,irrDA} = \emptyset$
D1
${}^{c}c_{ha,irrDA}$
${}^{c}c_{ha,irrDA} \neq \emptyset$
${}^{c}c_{ha,exeDA}$
activity 6:
start task of ${}^{r}a_{irrDA}$
${}^{c}c_{ha,exeDA} = \emptyset$
D2
${}^{c}c_{ha,exeDA} \neq \emptyset$
activity 7:
switch tasks

FIGURE 8. **Task harmonisation procedure in the TaskER model based on the concept visualised in Fig. 2.**

model. Therefore, we define the application of the exemplary system (section III-A), describe the TaskER framework that implements the agent classes using ROS (section III-B) and design the system via model configuration (section III-C). Next, the operation and tests of the exemplary system are presented (section III-D). This section ends with a discussion of the system actions and the requirements satisfaction.

A. REQUIREMENTS OF THE EXEMPLARY SYSTEM

The verification of the model and framework is conducted by implementing the procedure of task harmonisation (Fig. 8) in a simulated environment using a TIAGo robot. The configuration of the verification system is as follows:

1) task harmonisation is of interruptible-task type,
2) there is one robot in the system ($\mathtt{R} = \{r\}$), one tha class agent (${}^{\mathtt{R}}\mathtt{A}_{\mathtt{th}} = \{{}^{r}a_{ha}\}$) and one ca class (${}^{\mathtt{R}}\mathtt{A}_{\mathtt{ca}} = \{{}^{r}a_{co}\}$),
3) there is one tra class agent—${}^{s}a_{tr}$,
4) agents of set ${}^{r}\mathtt{A}_{\mathtt{da}}$ provide to agent ${}^{r}a_{ha}$ task-dependent schedule parameters that are defined by the following structure:

$$scheduleParams = [tskType,\ cost,\ cps,\ cTime,\ endPose]. \quad (39)$$

The *tskType* field consists of an identifier of the task type that the da class manages. In this verification, ${}^{s}a_{st}$ provides two types of tasks (40).

$$tskType \in \{guideHuman,\ humanFell\} \quad (40)$$

The *guideHuman* task objective is to approach a specified human, introduce the task, guide him to a specified location and say goodbye. The *humanFell* task is an emergency call for a robot to approach a specified human who likely fell and to check his consciousness.

We denote by $tskType_{dy}$ a type of task that is conducted by agent ${}^{r}a_{dy}$. Agents of set ${}^{r}\mathtt{A}_{\mathtt{da}}$ that initialised their report buffers (${}^{ha}_{y}c_{dy,report} \neq \emptyset$ for ${}^{r}a_{dy}$) with the first report (by pf^{compSP}_{dy} for ${}^{r}a_{dy}$) and have a task of type *guideHuman* belong to set $\mathtt{G}_r$, and those that initialised their report buffers and their tasks are of type *humanFell* and belong to set $\mathtt{H}_r$:

$$e = {}^{r}a_{dy} \in {}^{r}\mathtt{A}_{\mathtt{da}} \wedge {}^{ha}_{y}c_{dy,report} \neq \emptyset$$

$${}^{r}a_{dy} \in \mathtt{F}_r : e \wedge tskType_{dy} = humanFell \quad (41)$$

$${}^{r}a_{dy} \in \mathtt{G}_r : e \wedge tskType_{dy} = guideHuman \quad (42)$$

$cost_{dy}$ is a task-dependent schedule parameter of ${}^{r}a_{dy}$ that depends on the current state of the robot and its environment:

4.1. For $\mathtt{G}_r$ agents, it represents the comfort of a person who is cooperating with the robot during the task. The value of the *cost* parameter changes with time and is calculated via the following equation:

$$cost = \frac{w_{stand}}{t_{stand}} + \frac{w_{sit}}{t_{sit}} \quad (43)$$

i) t_{stand} is an estimate of the time for which the guided person stands, which is measured from the receipt of the task request until the task objective is realised;

ii) w_{stand} is the scaling factor for standing posture that is parametrised according to the person's health condition;

iii) t_{sit} is an estimate of the time for which the guided person sits, which is measured from the receipt of the task request until the task objective is realised; and

iv) w_{sit} is the scaling factor for sitting posture that is parametrised according to the person's health condition.

4.2. For $\mathtt{F}_r$ agents, the *cost* parameter represents the distance to the human that fell in consideration of the current pose of the robot.

The larger the value of the $cost_{dy}$ parameter, the less urgent the task of ${}^{r}a_{dy}$ is. *cps* is an estimate of the first derivative of the task cost in consideration of the current state of the robot and its environment. *cTime* is an estimate of the task duration if it were initialised at the estimation time (considering the current state of the robot and its environment). *endPose* is the pose of the robot when the task is completed.

5) ${}^{r}a_{ha}$ schedules ${}^{r}\mathtt{A}_{\mathtt{da}}$ agents (using $pf^{schedule}_{ha}$) based on algorithm 2. Hence, tasks of the same type compete with one another based on the *cost* parameter and the parameters that are requested by $pf^{schedule}_{ha}$ (via $\mathtt{B}_{ha,reqSP}$ buffers). The algorithm always causes an interruption of a guideHuman task by a humanFell task. In line with 23, the algorithm requests two argument-dependent parameters: *cc* and *ccps*. The former is the estimated cost of the recipient's task completion if the task were started from the pose that is specified in the argument. The *ccps* parameter is the cost per second while waiting for the recipient's task to begin.

B. IMPLEMENTATION OF THE MODEL AND EXEMPLARY SYSTEM

The TaskER framework delivers two Python classes, namely, `TaskHarmoniser` and `DynamicAgent`, which implement the `tha` and `da` classes, respectively. The implementation of the introduced framework and the agents, which are specific to the exemplary system (${}^{r}a_{co}$, ${}^{s}a_{pla}$, ${}^{s}a_{tr}$, and ${}^{s}a_{st}$), is based on the ROS framework.

Agents ${}^{r}a_{ha}$ and ${}^{s}a_{tr}$ are implemented as separate ROS nodes and are launched upon system startup. ${}^{r}a_{co}$ is robot-specific and is implemented as ROS nodes that serve common TIAGo robot actions (e.g., navigation and manipulation) in the Gazebo simulator.

Upon `TaskHarmoniser` class initialisation, a ROS node is launched with a ROS service interface to request new tasks (implementing pf^{initDA}_{ha}) and with a subscriber for ROS topic messages, which receives the reports from ${}^{r}\mathtt{A}_{\mathtt{da}}$ agents. Upon each task request (via the ROS service), agent ${}^{r}a_{ha}$ launches a new ROS node that implements a suitable `da` class agent. Henceforth, the node of the `da` class agent will publish its report messages to the ROS topic that is subscribed by the `tha` class agent that is discussed above. In addition to the above interfaces, the `TaskHarmoniser` class implements exemplary primitive transition functions namely, $pf^{switchDA}_{ha}$ and $pf^{schedule}_{ha}$, respectively, as presented in this article (Alg. 1 and Alg. 2).

For each type of tasks in the system, a dedicated script is implemented with the `DynamicAgent` Python class. The task scripts are started by a `tha` class agent upon task request from a `tra` class agent. Upon initialisation of the `DynamicAgent` Python class, considering *dy* as an identifier of the `da` class agent that is being initialised:

1) a ROS node is launched;
2) a ROS service of `pla` class is requested to launch a ROS node of ${}^{r}a_{cl} \in {}^{r}\mathtt{A}_{\mathtt{cla}}$ (if ${}^{r}a_{dy}$ requires support from `cla` class);
3) interfaces of ${}^{r}a_{dy}$ are created:
   3.1. ${}^{ha}_{x}c_{dy,cmd}$ as a ROS topic subscriber, which sets initial conditions $ic_{dy,start}$, $ic_{dy,susp}$, and $ic_{dy,term}$;
   3.2. ${}^{ha}_{y}c_{dy,report}$ as a ROS topic publisher (which is associated with ${}^{dy}_{x}c_{ha,report}$),
   3.3. ${}^{ha}_{x}c_{dy,reqSP}$ and ${}^{ha}_{y}c_{dy,reqSP}$ as a ROS service (which are associated with ${}^{dy}_{y}c_{ha,reqSP}$ and ${}^{dy}_{x}c_{ha,reqSP}$, respectively);
   3.4. ${}^{co}_{x}c_{dy,rob}$ and ${}^{co}_{y}c_{dy,rob}$ by the robot API class initialisation; and
   3.5. ${}^{cl}_{x}c_{dy,cloud}$ and ${}^{cl}_{y}c_{dy,cloud}$ as a ROS service client that is connected to the ROS service that is shared by ${}^{r}a_{cl}$ supporting ${}^{r}a_{dy}$.

Algorithm 2 Algorithm of an Example $pf_{ha}^{schedule}$ That Is Implemented in the Verification System

```
Input: $\{\mathtt{B}_{ha,report}, \mathtt{B}_{ha,reqSP}, \mathtt{G}_r, \mathtt{F}_r, {}^{c}c_{ha}\}$
Result: $\{{}^{c}c_{ha,irrDA}\}$
$dac = \emptyset$;
//get an identifier of an agent from $\mathtt{F}_r$ set with the lowest cost
$cHF = \arg\min_{dy:{}^{r}a_{dy} \in \mathtt{F}_r} cost_{dy}$ ;
$cGH = \arg\min_{dy:{}^{r}a_{dy} \in \mathtt{G}_r} cost_{dy}$;
if $cHF == \emptyset \wedge cGH == \emptyset$ then
    return;
else
    if ${}^{c}c_{ha,irrDA} == \emptyset$ then
        if $cHF \neq \emptyset$ then
            if ${}^{r}a_{exeDA} \notin \mathtt{F}_r$ then
                ${}^{c}c_{ha,irrDA} = \{cHF\}$ ;
                return;
            end
            //set an identifier of a candidate for a task switch
            $dac = cHF$;
        else
            if $cGH \neq \emptyset \wedge {}^{r}a_{exeDA} \notin \mathtt{F}_r$ then
                $dac = cGH$;
            end
        end
        if $dac \neq \emptyset$ then
            //request the candidate for argument dependent schedule parameters, where 1 is the identifier of a primitive transition function of $f_{dac}^{compSP}$
            ${}^{dac}_{y}c_{ha,reqSP} = \{1,\ endPose_{exeDA}\}$;
            $\{cc_{dac}, ccps_{dac}\} = {}^{dac}_{x}c_{ha,reqSP}$;
            //request of an estimated cost of ${}^{c}c_{ha,exeDA}$ task suspendion and start after completion of ${}^{r}a_{dac}$ task ($cc_{exeDA}$) and cost per second while waiting for resumption ($ccps_{exeDA}$)
            ${}^{exeDA}_{y}c_{ha,reqSP} = \{1,\ endPose_{dac}\}$;
            $\{cc_{exeDA}, ccps_{exeDA}\} = {}^{exeDA}_{x}c_{ha,reqSP}$;
            $c_{switch} = cost_{dac} + cc_{exeDA} + cps_{exeDA} * cTime_{dac}$;
            $c_{wait} = cost_{exeDA} + cc_{dac} + cps_{dac} * cTime_{exeDA}$;
            if $c_{switch} < c_{wait}$ then
                ${}^{c}c_{ha,irrDA} = \{dac\}$
            end
        end
    end
end
```

The above actions are managed in state $\mathtt{S}_{dy}^{initComm}$. Next, agent ${}^{r}a_{dy}$ follows FSM_{dy} and the basic behaviours that are described in Section II-B1.c.

C. CONFIGURATION OF THE DYNAMIC AGENT MODEL

The model of $\mathtt{da}$ class (section II-B1) must be configured for a specified task to satisfy the requirements of the task. The configuration procedure is presented for an example $\mathtt{da}$ class—${}^{r}a_{dy} \in {}^{r}\mathtt{A}_{\mathtt{da}}$ managing a task of type *guideHuman*. The procedure is as follows:

1) task objective definition:
   1.1. definition of $FSM_{dy,exeTsk}$, including the task stages and the initial conditions of the stages;
   1.2. classification of the task stages into suspendable and blocking; and
   1.3. specification of basic behaviours that are conducted in the stages and the termination conditions of the basic behaviours;
2) task update (f_{dy}^{upTsk}), suspension $(f_{dy}^{genSusp})$ and schedule parameters calculation (f_{dy}^{compSP}) definition.

Configuration of the model for the tasks of *guideHuman* type resulted in the following: $FSM_{dy,exeTsk}$ as illustrated in Fig. 9, f_{dy}^{upTsk} as presented in Alg. 3, and f_{dy}^{compSP} and $f_{dy}^{genSusp}$ as follows:

$$f_{dy}^{compSP} = \begin{cases} pf_{dy}^{report} & timer({}^{c}c_{dy,repRate}) \\ pf_{dy}^{sp,1} & I^1, \end{cases} \tag{44}$$

$$I^1 = newData({}^{ha}_{x}c_{dy,reqSP}), \tag{45}$$

$$pf_{dy}^{sp,1} : \{{}^{ha}_{x}c_{dy,reqSP}, {}^{co}_{x}c_{dy,rob}, {}^{c}c_{dy}\} \rightarrow \{cc_{dy}, ccps_{dy}\}, \tag{46}$$

$$f_{dy}^{genSusp} = \begin{cases} \mathscr{B}_{dy}^{exeSusp} = \mathscr{B}_{dy}^{save} & \neg greeted \\ \mathscr{B}_{dy}^{exeSusp} = \mathscr{B}_{dy}^{apologize} & greeted, \end{cases} \tag{47}$$

where *greeted* is true if the person being guided was already greeted by the robot and false if was not. In the first case of (47), $\mathscr{B}_{dy}^{save}$ is assigned to $\mathscr{B}_{dy}^{exeSusp}$. The basic behaviour saves already computed results, which are useful after the task resumption (e.g., the person's pose, if it was determined, can be used as an initial value for the person search algorithm). The termination condition of $\mathscr{B}_{dy}^{save}$ always holds ($tc_{dy,save} = true$); hence, after one iteration of $\mathscr{B}_{dy}^{exeSusp}$, the transition to $\mathtt{S}_{dy}^{wait}$ is triggered. In the second case, $\mathscr{B}_{dy}^{apologise}$ is assigned to $\mathtt{S}_{dy}^{exeSusp}$. This basic behaviour stops the robot, saves important data (as in the $\mathscr{B}_{dy}^{save}$ case), and asks the person to stay and not follow the robot because it received another important task.

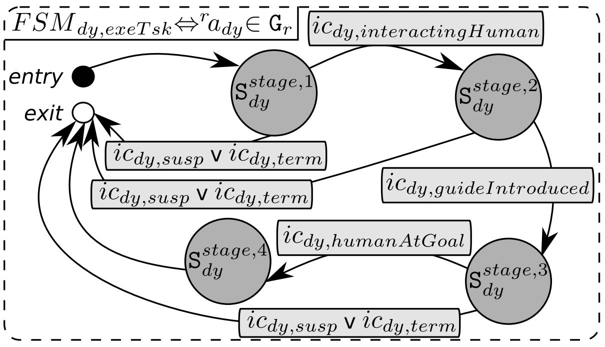
D. EXECUTION OF THE EXEMPLARY SYSTEM

We tested the system in a scenario that involved the execution of the task harmonisation procedure (section II-C) over 80 times in a changing environment. The scenario for evaluation of the exemplary system involve:

- suspension of the ongoing task in various stages,
- dynamic changes of the environment state (independent from the robot actions),

Algorithm 3 Algorithm of f_{dy}^{upTsk}, Where *suspended* Is *true* If the Task Was Suspended and *false* If It Was Not

```
Input: $FSM_{dy,exeTsk}$
Result: Modification of $FSM_{dy,exeTsk}$
1 if $suspended == true$ then
2     $\mathscr{B}_{dy}^{stage,2} = \mathscr{B}_{dy}^{greet\_and\_apologise}$;
3 else
4     $\mathscr{B}_{dy}^{stage,2} = \mathscr{B}_{dy}^{greet}$;
5 end
```



$FSM_{dy,exeTsk} \Leftrightarrow {}^{r}a_{dy} \in \mathtt{G}_r$
$ic_{dy,interactingHuman}$
entry
exit
$s_{dy}^{stage,1}$
$s_{dy}^{stage,2}$
$ic_{dy,susp} \vee ic_{dy,term}$
$ic_{dy,susp} \vee ic_{dy,term}$
$ic_{dy,guideIntroduced}$
$ic_{dy,humanAtGoal}$
$s_{dy}^{stage,4}$
$s_{dy}^{stage,3}$
$ic_{dy,susp} \vee ic_{dy,term}$

FIGURE 9. The FSM of a human guidance task, where the basic behaviour of $s_{dy}^{stage,1}$ moves the robot to a requested human, the basic behaviour of $s_{dy}^{stage,2}$ interacts with the human to introduce the task and ask him to follow the robot, the basic behaviour of $s_{dy}^{stage,3}$ moves the robot to the destination and checks if the human follows the robot, and the basic behaviour of $s_{dy}^{stage,4}$ finishes the interaction with the human.

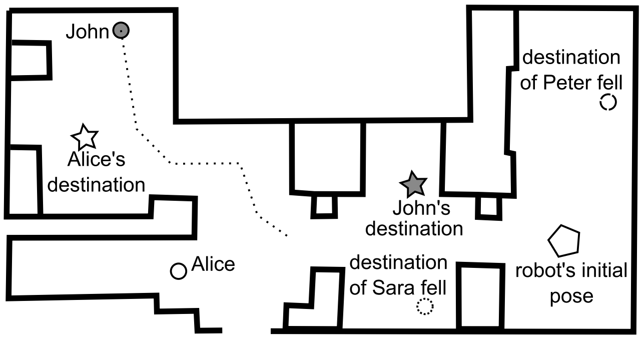


FIGURE 10. Verification environment setup.

- resumption of a suspended task in the case of the environment state change,
- abortion of a suspended task on its request,
- unpredictable task requests,

The initial setup of the environment is shown in Fig. 10. Four people, namely, John, Alice, Peter and Sara, were in the environment. In this article we describe a period of our tests in which four tasks were conducted:

1) guide John – a *guideHuman*-type task that was managed by ${}^{r}a_{d1}$,
2) guide Alice – a *guideHuman*-type task that was managed by ${}^{r}a_{d2}$,
3) Peter fell – a *humanFell*-type task that was managed by ${}^{r}a_{d3}$,
4) Sara fell – a *humanFell*-type task that was managed by ${}^{r}a_{d4}$.

Various tasks (integrated as dynamic agents) can be applied to our architecture, however, we propose the two types in order to keep the scenario simpler and clearer, as the model and the harmonisation procedure is complex itself. The new tasks can be applied as it was described in Sec. III-C.

In Fig. 10, we represented the initial pose of the robot as a pentagon and John and Alice as filled and empty circles, respectively. The path that was traversed by John while he was waiting for the robot is represented as a dotted line. The destinations of John and Alice are represented by filled and empty stars. The poses in which Sara and Peter fell are represented as dotted and dashed circles. The verification scenario was recorded, and the video is available [42].

The detailed description of the system behaviour during the evaluation is described in Appendix B. The Fig. 11 visualises value changes of the key parameters of schedule algorithm (Alg. 2) and assignments of ${}^{r}\mathtt{A}_{\mathtt{da}}$ agents to ${}^{r}a_{irrDA}, {}^{r}a_{exeDA}$, $\mathtt{D}_r$ roles.

E. SATISFACTION OF THE REQUIREMENTS IN THE EXEMPLARY SYSTEM

The ${}^{s}a_{tr}$ can request multiple tasks at any time, which are initialised and ready to be conducted by the robot. The requests are processed, suitable `da` class agents are created, and in each iteration of $pf_{ha}^{schedule}$, a `da` class can be assigned to ${}^{r}a_{irrDA}$; thus, **R1** is satisfied.

Schedule parameters can be computed repetitively (22) by pf_{dy}^{report}, which fills the ${}_{y}^{ha}\mathfrak{c}_{dy,report}$ buffer (19). Additionally, the scheduling algorithm can request argument-dependent schedule parameters, which are computed by dedicated primitive transition functions of f_{dy}^{compSP}; thus, requirement **R2** is satisfied.

The procedure scheduling `da` class agents is divided into multiple primitive transition functions, which are distributed among `da` class agents and ${}^{r}a_{ha}$, which, via function $pf_{ha}^{schedule}$, computes a decision regarding switching an ongoing task with a new task or continuing the current task. Therefore, the exemplary system may be easily configured to use other schedule parameters or scheduling algorithms that are implemented in $pf_{ha}^{schedule}$. Thus, **R3** is satisfied.

Tasks are created independently and stored in the cloud. A `tha` class agent of a robot downloads them upon the request of the robot's user. The initiated tasks become `da` class agents and compute schedule parameters in consideration of expert knowledge and the context of the task. For example, a general parameter, such as the estimated time for completing the task, can be calculated with consideration of the expert knowledge in the `da` class (such as a speed limitation due to transportation of a delicate object). Thus, requirement **R4** is satisfied.

Our model divides tasks into stages that can be suspended and stages that cannot. It also defines the FSM of `da` class

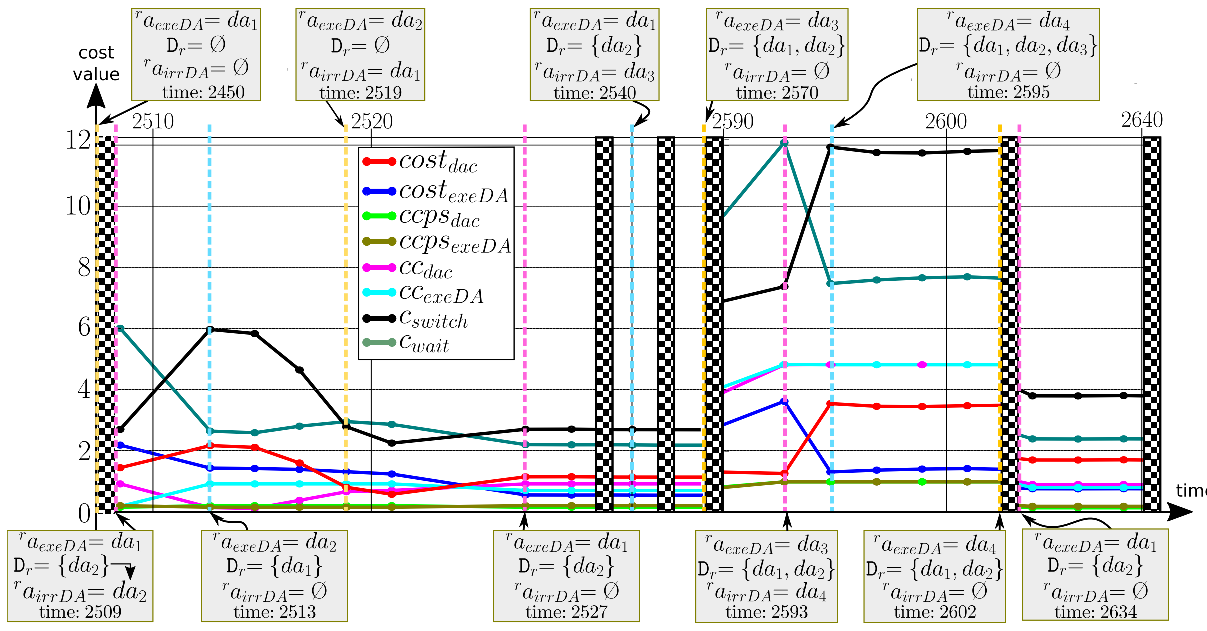


FIGURE 11. Data that were calculated by $pf_{th}^{schedule}$ during the verification. The analysis of these data is presented in Appendix B.

with dedicated suspension states, which are triggered if the agent receives a suspension signal and the current stage of the agent's task is suspendable. Hence, interruption by a set of actions and the prevention of a possible inconvenient behaviour or damage to the system and its environment are managed by the agent, which possesses expert knowledge regarding the current task and its progress. For example, suppose the robot in the verification scenario did not warn John that the robot switched tasks. In such a case John would follow the robot unnecessarily. Hence, the suspension state avoided an inconvenient walk for John while the robot conducted other tasks. Another example of a suspension action is turning off a cooker if it was turned on during the execution of the task and the estimated interruption time exceeds the cooking time of the dish that is being prepared. Hence, requirement **R5** is satisfied.

Due to the frequently triggered pf_{dy}^{report} function, $^r a_{dy}$ can react to changes in a dynamic environment and update its schedule parameters. The changes in the environment also affect the plans of `da` class agents. Therefore, our model foresees the need to update the plan (i.a. in state S_{dy}^{upTsk}) before the agent executes its task. The most straightforward example result of $\mathscr{B}_{dy}^{upTsk}$ can be an extension or reduction of the plan due to actions by an actor that is co-working with the robot in the environment. Thus, requirement **R6** is satisfied.

IV. RELATED WORK

Our study considers two aspects: structure and behaviour modelling for robotic systems and the management of multiple tasks in these systems. Therefore, we compare our approach to other available approaches for these two aspects.

1) STRUCTURES AND BEHAVIOUR MODELS FOR ROBOT SYSTEMS

One of the most popular approaches for modelling robot control systems is the use of a layered architecture, e.g., the 3T architecture [35], [43]. It is broadly utilised and compared with the other available approaches, e.g., in SmartMDSD Toolchain [44], [45]. The authors of [37] designed the middle layer of the 3T architecture, which was realised with a finite-state machine (FSM), for the integration of the symbolic plan that was provided by the knowledge representation with the geometric planner that instructs the robot. Each description layer of a task (symbolic planning, behaviour sequencing, or command execution) uses a different context and is designed to satisfy its objectives and handle exceptions that are specified in the abstraction-layer dictionary, e.g., a symbolic planner that uses logic variables and functions or a behaviour sequencer that uses primitive behaviours and transition functions.

Our study considers all three layers that are defined in the above architectures. `ca` class is the command execution layer, the `da` class and `tha` class agents constitute the sequencer layer, and the symbolic planners that are utilised by the sequencer layer (implemented in either `cla` class or `pla` class) constitute the deliberation layer.

Coordination between the layers is a complex problem, especially considering the difference in their contexts. Stenmark *et al.* proposed the integration of high-level task description with action execution [46]. However, coordination of the tasks was not a goal of this work. Our approach enables the harmonisation of a robot's tasks by the sequencer layer, which receives error messages from the commanding layer and can request new plans from the deliberative layer.

2) MULTI-TASK ROBOT SYSTEMS

A task planning problem for human-robot interaction schemes that are scheduled by a symbolic planner is described in [47]. The authors of the article focus on social constraints that should be considered in a task and motion planning of a human-aware robot. However, the problem of the suspension and resumption of the tasks is not resolved. Studies have also been conducted on sequencing robot behaviours at the controller level (e.g., [48], [49]). Robot-human communication is yet another problem that is related to the management of robot tasks. The authors of [50] describe a model of a system that is capable of interruption and switching of a conversation domain. However, these approaches preclude the execution and scheduling of various independent tasks.

The overall behaviour of the robot in our exemplary system can be modelled with a hierarchical FSM (HFSM), e.g., SMACH library [51] for task-level robot control. However, understanding the system performance, configuring the scheduling algorithm and tailoring the suspension/replanning actions for a specified application would be complicated. The decomposition of the robot control system into separate agents of various classes renders the system specification process more convenient, more transparent and easier. Additionally, SMACH does not support dynamic HFSMs natively, which are proposed in our study (behaviour of one state defines the subsequent one).

The processing of multiple tasks in state-of-the-art robot systems is realised via high-level deliberation. Reasoning algorithms compute a multi-objective plan that satisfies temporal constraints [52] and uncertainties in conditional planning [53]. This approach utilises a semantic planner, which requires a complex knowledgebase that integrates predicates of all possible requests from the system user and requires that the predicates be compatible among the system tasks. Therefore, the extension of the system with additional tasks is complicated and interferes with the well-tested tasks. Additionally, if the priority of a task changes, a task is cancelled or a task's parameters change, then this approach requires re-planning of the actions. The available frameworks that utilise semantic planning ROSPlan [54], SmartTCL [45] do not provide a mechanism for re-initialising planning in the face of a task modification.

In state-of-the-art architectures, robot tasks are separated from the rest components of robot control system. Various models are used to express the task and they can be divided to task definition models and task execution models. The models of former type are commonly used with semantic planners, which reason basing on the task definition and the world state. As a result, planners return specification of the robot behaviour, which is required to complete the task. Both the task and the behaviour are expressed in the terms defined in the model, which commonly is a Domain Specification Language (DSL). The example task definition approaches are PDDL [55], Golog [56]. The behaviour computed by a semantic planner can be transformed to a model, which abstracts from semantic symbols and DSL terms i.e. to a task execution model. For example, the behaviour can be decomposed into states which need to be managed to complete the task, as it is in Hierarchical Finite-State Machines (HFSM) approach [57]. Additionally, there are Petri Nets and Behaviour Trees (BTs) [58]. The latter approach was initially used to define behaviour of Non-Playable Characters (NPC) in game industry. Proponents of BTs approach state that it is a very efficient way of creating complex systems that are both modular and reactive. Some of the task models can be transformed to other models, e.g. HFSM to BT and vice versa [58]. We use HFSM approach, because it is a core model of agent behaviour in the formal specification method we use (Embodied Agent Approach). However, tasks modelled in other approaches can be integrated with our architecture, as it is done in Petri Net case in Fig. 6.

The authors of [59] propose a method to coordinate tasks specified in Golog language. The approach is compatible with a formalised method to reason about task plans and composition of task suspension sequences. The work introduces the concept of `promises`, which denote asserted or required conditions used to determine conflicts during task switching. In contrast, we define formally in our work an architecture which can be used more broadly, as it is not restricted to a specific task modelling language. We present our work with the use of FSMs, as it is a popular behaviour modelling approach and there are methods to transform it to other approaches. Therefore, the TaskER architecture can be assumed as independent from a task model, planning method and language used for the definition of the algorithm managing tasks. It integrates concepts of i.a. scheduling algorithm, task update, schedule parameters calculation and specifies an architecture which satisfies the requirements stated in this article introduction. Therefore, the concept of `promises` can be used to specify the scheduling algorithm defined in the TaskER and different planning methods can be used in the task update concept. Furthermore, as the contexts of tasks are separated, tasks can be specified with dedicated models,they can use different planning methods to update their plans and use various schedule parameters. The latter can be task-context-dependent in TaskER, so other priority can be assigned to a door opening task in case of a visitor and other in the case of an emergency.

Robot systems are part of a larger group: cyber-physical systems (CPSs). Various articles consider the task harmonisation problem in this group [60]–[62]. However, they are focused mostly on algorithms for optimisation of the task switching moment or coordination of the tasks that are delegated to multiple devices to realise a specified objective. The authors of [60] present a methodology for describing, managing and realising objectives of a device community. This study shows how to describe the objective of the whole community of devices with simple roles of each device. The second paper [61] presents an algorithm for minimising a deadline meeting ratio. The algorithm considers

TABLE 3. The comparison of scheduling in typical operating systems and robot systems that use TaskER.

Feature	OS process scheduling	TaskER harmonisation
Reject tasks/jobs mechanism	True (sporadic scheduling)	All tasks are accepted, TaskER has an ability to terminate unfeasible tasks
Schedule entities	Processes (Threads)	`da` class agents
States of the schedule entities	New, ready, running, terminated, and waiting	Given by the hierarchical FSM Fig. 5
Execution unit	CPU	Robot (`ca` class)
The utilisation of a sophisticated calculation of priorities and deadlines, which are dependent on job/task context	False	True
Effect of an atomic instruction	Memory state change, or an elementary action on i/o devices	Robot configuration or its environment state change
Long-term scheduling entity	Job scheduler	`tra` class agents as interfaces for users, or external systems
Short-term scheduling entity	CPU scheduler	Distributed between a `tha` class and `da` class agents, which know the contexts and constraints of the physical environment for each of the tasks
Medium-term scheduling entity	Swapping mechanism	———"———
Instant execution of the scheduler decision	True	When it is safe, otherwise suspends the ongoing task first
Procedure of job/task suspension and resumption	Save/load the process memory (relatively a quick action)	Suspension and resumption procedures are aware of the physical environment and are described in this article (relatively can be a time-consuming action)
Re-scheduling jobs/tasks on their requests	False	True, as `da` class agents calculate schedule parameters and any change in the parameters triggers the scheduling algorithm
Abstract parameters are used in the scheduling process	False	True (e.g., maximisation of human convenience, or for a restaurant runner task, the time at which a new client approaches a table serves as a deadline for cleaning the table)
The types of jobs/tasks can have dedicated schedule parameters	False, processes have priorities which can be directly compared	True, various `da` class agents can compute different schedule parameters which can be transformed and compared with each other by `tha` class agent (e.g., object transportation uses the shortest travel distance, and human guidance maximises human convenience)
Hard restriction of deadlines	Available	Depends on the robot application: for social and service robots, the deadlines can be fuzzy

the time that is required by the servicing node for moving from one place to another. However, the authors do not demonstrate how to perform the task switch (in a scenario that involves a robot, not all tasks may be interrupted at any time). The final study [62] considers a dynamic allocation of computational tasks among distributed CPS devices.

The authors of [63] investigate the problem of integrating time- and event-triggered systems in a mixCPS architecture. However, they focus on computation task assignment and packet transmission optimisation to minimise the application-level delays. The architecture considers delays of sensor-controller and controller-actuator communications and the controller processing time.

In contrast to a typical CPS (where many tasks are processed rhythmically [64] or task activation is statically defined by rules as in home automation applications), robots are being unpredictably requested for tasks, and the user of a robot would like to change his/her requests and preferences while the robot performs the requested task.

Task queueing and management is an old topic in cyber research [65], [66] and a hot topic in cyber-physical systems [67], [68]. The robot task harmonisation problem has similarities with process scheduling in operating systems (e.g., sporadic scheduling in real-time systems [69]). Both consider unit work management: in robotics, a robot performs multiple tasks, and in an operating system, a CPU conducts multiple processes. The correlations between scheduling operating system processes and the TaskER model are presented in Tab. 3.

V. CONCLUSION

The article considers the problem of switching tasks that are conducted by robots. The structure of the robot controller is variable, and it is composed of multiple agents that are derived from various agents' classes. The activities of the robots that are working in the physical environment differ, and the assumptions and conditions that should be considered differ from those in standard computational systems.

The flexibility of a robot system is crucial in the case of unexpected difficulties and depends on the system application assumptions, e.g., a market crash, a factory modernisation, or a pandemic. Therefore, robot systems must be easily configurable especially at the task level.

One of the challenges that must be overcome for the realisation of flexibility is the development of multi-task robot controllers that can harmonise robots tasks. In this study, we present sample use cases that describe the desired behaviours of a robot control system that is faced with the task harmonisation problem. Then, based on the use cases, we formulate six requirements for a control system that features task harmonisation. Next, we describe the model of such a system. The model adapts and extends the known variable structure for robot control systems, namely, the RAPP architecture, to satisfy the previously specified requirements. Finally, we implemented the TaskER framework, which enables the efficient creation of the robot tasks and injects them as modules into the systems that utilise our model. The algorithm that defines the strategy for task scheduling differs among robots, applications and environments; hence, the TaskER framework applies the algorithm as an interchangeable function.

The model that is described in this article is presented in mathematical and UML diagram formulations. Thus, the overall behaviour of a system that follows the model is strictly defined. Requested tasks have their contexts and are implemented as separate processes, which facilitates understanding of the current state of the system and the progress of each task.

Model of a dynamic agent, being an encapsulation of a task in this architecture, defines constraints and specifies the tasks to enable harmonisation feature. A huge part of the task is not constrained by the model proposed in this work. This is because it is not important from the perspective of this work goal. The method to define actions realised by the task is another important problem in robotics and any which results with a plan representable in FSM can be used. In the architecture, we define when and by which agent a planning method will be executed. Various planning methods require different world models. As our harmonisation method does not depend on any planning method or world model, any can be used. Furthermore, our work can be applied even in systems without deliberation and semantic planning. Each task (dynamic agent) can use different planning method to update its actions and constraints or do not use any and have its actions statically defined.

Moreover, tasks that are managed by `da` class agents can

1) compute task-dependent parameters that are used for task scheduling,
2) suspend their operation safely in the case of a task switch,
3) update their plans prior to the task execution, and
4) block a task switch if it would be dangerous according to the context and knowledge of the agent.

The conducted tests show that the exemplary system satisfies the specified requirements. Moreover, they demonstrate the benefits for a system that follows the model.

We plan to integrate the TaskER framework with a known framework for semantic planning (such as ROSPlan [54]) and to call the planner in both $\mathtt{S}_{dy}^{upTsk}$ and $\mathtt{S}_{dy}^{genSusp}$ to generate $FSM_{dy,exeTsk}$ and $FSM_{dy,exeSusp}$, respectively. This feature enables tasks that are implemented in the TaskER framework to update their plans via deliberation. Furthermore, in future work, we plan to do the following:

1) propose and verify algorithms for functions $pf_{ha}^{schedule}$ and f_{dy}^{compSP} for calculating scheduling decisions and parameters for various tasks, applications and environment classes (e.g., using a objective optimisation algorithm such as [70]),
2) define a guideline and an algorithm for classification of the task stages into suspendable and blocking,
3) extend the PDDL standard to support our approach and enable semantic planners to consider the task scheduling problem while composing $FSM_{dy,exeTsk}$ and $FSM_{dy,exeSusp}$ FSMs,
4) evaluate metrics for task scheduling quality evaluation,
5) provide an adaptation mechanism for the scheduling algorithm to enable learning of the priorities of tasks based on the user's intentions [71], [72],
6) design an ontology for the robot task harmonisation problem that facilitates the configuration of the schedule parameters (according to robot skill reconfiguration [73]),
7) conduct tests with end users within the INCARE project [74], and
8) integrate our architecture with one of the formally defined task models featured with consistent language for scheduling algorithm specification and planning method [59].

APPENDIX A
KEY ACTIVITIES IN THE HARMONISATION PROCEDURE

1) LAUNCH NEW DYNAMIC AGENT

The interaction among the agents of the system is presented in Fig. 12. Agent ${}^{s}a_{tr}$ sends a request to ${}^{r}a_{ha}$ via buffer ${}^{ha}_{y}c_{tr,task}$, which is connected to buffer ${}^{tr}_{x}c_{ha,task}$. Consequently, pf_{ha}^{initDA} is triggered (36), which manages the initialisation of a new dynamic agent, starts an application that implements the `da` class model, passes initial data to it and extends

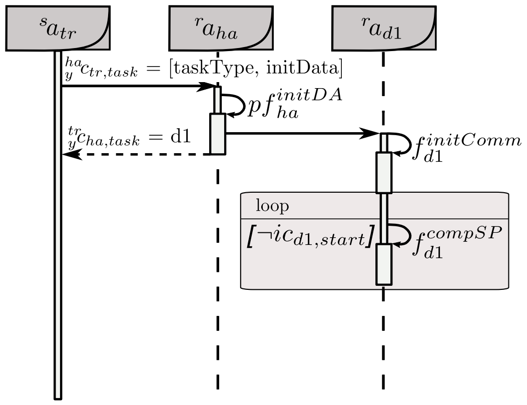


FIGURE 12. Sequence diagram of a new task request management and a da class initialisation.

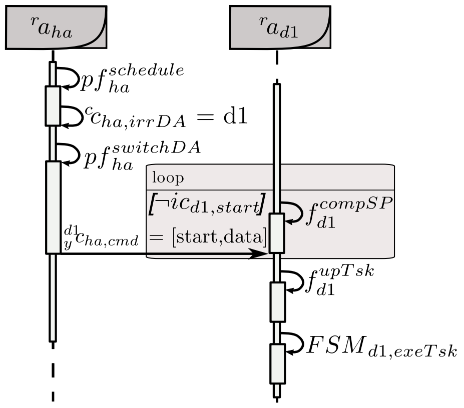


FIGURE 13. Sequence diagram of triggering a da class agent.

array $^{c}c_{ha,idleDA}$ with the name of the new dynamic agent (e.g., *d1*). Thus, $^{r}a_{d1}$ is created, and FSM_{d1} switches to S^{init}_{d1}, where the system-specific initialisation actions are conducted ($\mathrm{S}^{initComm}_{d1}$). Finally, $^{r}a_{d1}$ switches to state S^{compSP}_{d1}, in which it repetitively calculates the schedule parameters (given by (22)) and saves them in buffer $^{ha}_{y}c_{d1,report}$.

2) START TASK OF $^{r}a_{irrDA}$

In this case, none of the agents in $^{r}\mathrm{A}_{\mathrm{da}}$ plays the role of $^{r}a_{ha,exeDA}$, and $pf^{schedule}_{ha}$ assigns the role of $^{r}a_{irrDA}$ to an agent in set D_{r}. The interaction among the agents in this activity, with $^{r}a_{irrDA} = {}^{r}a_{d1}$, is illustrated in Fig. 13.

3) SWITCH TASKS

This activity begins only when $pf^{schedule}_{ha}$, knowing that $^{r}a_{exeDA}$ conducts its task, decides to assign the role of $^{r}a_{irrDA}$ to an agent in set D_{r}. The sequence of the system agent interactions in this activity is presented in Fig. 14. According to (36), the initial event of function $pf^{switchDA}_{ha}$ is satisfied until it does not remove the identifier of $^{r}a_{irrDA}$ from the $^{c}c_{ha,irrDA}$ field in line 11 of Alg. 1. Depending on the state of agent $^{r}a_{exeDA}$, either the condition in line 2 or that in line 5 of Alg. 1 is satisfied. In the former case, $^{r}a_{ha}$ sends a suspension signal to $^{r}a_{exeDA}$, and in the latter, it removes

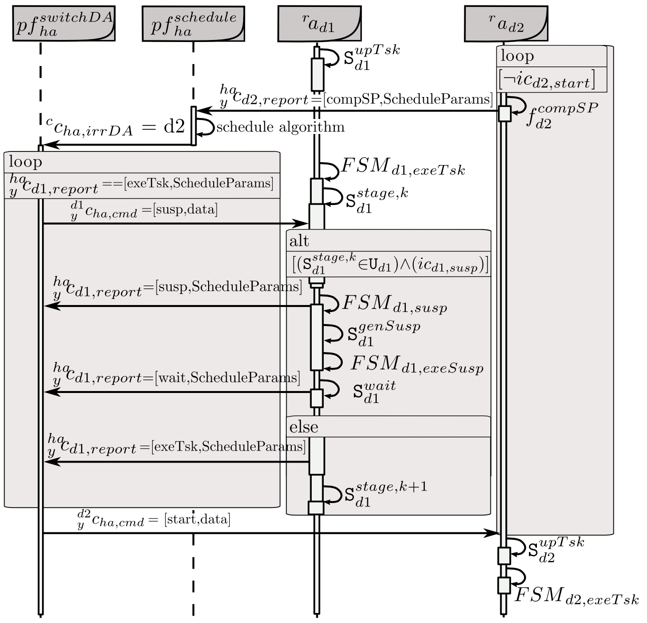

$pf^{switchDA}_{ha}$
$pf^{schedule}_{ha}$
$^{r}a_{d1}$
$^{r}a_{d2}$
S^{upTsk}_{d1}
loop
$[\neg ic_{d2,start}]$
$^{ha}_{y}c_{d2,report}$ = [compSP,ScheduleParams]
f^{compSP}_{d2}
$^{c}c_{ha,irrDA}$ = d2
schedule algorithm
loop
$FSM_{d1,exeTsk}$
$^{ha}_{y}c_{d1,report}$ == [exeTsk,ScheduleParams]
$^{d1}_{y}c_{ha,cmd}$ = [susp,data]
$\mathrm{S}^{stage,k}_{d1}$
alt
$[(\mathrm{S}^{stage,k}_{d1} \in \mathrm{U}_{d1}) \wedge (ic_{d1,susp})]$
$^{ha}_{y}c_{d1,report}$ = [susp,ScheduleParams]
$FSM_{d1,susp}$
$\mathrm{S}^{genSusp}_{d1}$
$FSM_{d1,exeSusp}$
$^{ha}_{y}c_{d1,report}$ = [wait,ScheduleParams]
S^{wait}_{d1}
else
$^{ha}_{y}c_{d1,report}$ = [exeTsk,ScheduleParams]
$\mathrm{S}^{stage,k+1}_{d1}$
$^{d2}_{y}c_{ha,cmd}$ = [start,data]
S^{upTsk}_{d2}
$FSM_{d2,exeTsk}$

FIGURE 14. Sequence diagram that presents the cooperation of the system agents while switching agents $^{r}a_{irrDA}$ and $^{r}a_{exeDA}$ (during *activity 7* shown in Fig. 8), where $\mathrm{S}^{stage,k}_{d1} \in FSM_{d1,exeTsk}$, $^{r}a_{d1}$ is initially $^{r}a_{exeDA}$ and $^{r}a_{d2}$ becomes $^{r}a_{irrDA}$ as a result of the $pf^{schedule}_{ha}$ output.

$^{r}a_{exeDA}$ from the $^{c}c_{ha,exeDA}$ field. As this activity is started only if $^{c}c_{ha,exeDA} \neq \emptyset$, the first iteration of $pf^{switchDA}_{ha}$ will access the condition in line 2 and send a suspension signal to $^{r}a_{exeDA}$ ($^{exeDA}_{y}c_{ha,cmd}$=[susp,data]). Henceforth, two cases are considered:

1) If the ongoing stage is suspendable ($\mathrm{S}^{stage,k}_{exeDA} \in \mathrm{U}_{exeDA}$), then $FSM_{exeDA,exeTsk}$ is terminated, as presented in Fig. 5c, and $^{r}a_{exeDA}$ is switched to $FSM_{exeDA,susp}$. Now, the agent prepares a plan for the task suspension (in $f^{genSusp}_{exeDA}$) and realises the plan in $FSM_{exeDA,exeSusp}$. Finally, when the task is suspended, the agent switches to $\mathrm{S}^{wait}_{exeDA}$ and saves a report message to $^{ha}_{y}c_{exeDA,report}$ that contains the name of the current state – *wait*. While $^{r}a_{exeDA}$ follows the above sequence, the algorithm of $pf^{switchDA}_{ha}$ always enters the condition in line 2 of Alg. 1. When $^{r}a_{exeDA}$ enters state $\mathrm{S}^{wait}_{exeDA}$, the condition in line 5 is satisfied, and $pf^{switchDA}_{ha}$ moves the name of $^{r}a_{exeDA}$ from the $^{c}c_{ha,exeDA}$ field to set D_{r}. Next, the algorithm enters the condition in line 9 as the $^{c}c_{ha,exeDA}$ field is empty. Consequently, $pf^{switchDA}_{ha}$ moves the name of $^{r}a_{irrDA}$ from $^{c}c_{ha,irrDA}$ to the $^{c}c_{ha,exeDA}$ field and sends the start signal to $^{r}a_{irrDA}$ ($^{irrDA}_{y}c_{ha,cmd}$ = [start,data]).
2) If the ongoing stage is blocking ($\mathrm{S}^{stage,k}_{exeDA} \in \mathrm{L}_{exeDA}$), then the suspension signal ($^{ha}_{x}c_{exeDA,cmd}$=[susp,data]) is ignored; hence, this stage and all subsequent blocking stages will be completed. When $FSM_{exeDA,exeTsk}$ finally enters a suspendable stage, the system activity will follow the sequence that is described in the first case.

APPENDIX B
VERIFICATION DATA ANALYSIS

In Fig. 11, we present values of the following schedule parameters, which were used in function $pf_{ha}^{schedule}$ during the verification period: $cost_{dac}$, $cost_{exeDA}$, cc_{dac}, cc_{exeDA}, $ccps_{dac}$, $ccps_{exeDA}$, c_{switch}, and c_{wait}. The rectangularly highlighted sections of the figure show the states of ${}^{c}c_{ha,idleDA}$, ${}^{c}c_{ha,exeDA}$ and ${}^{c}c_{ha,irrDA}$ in crucial moments of the test period. First, at $time = 2450$, ${}^{r}a_{d1}$ is initialised and executes its task. Then, between $time = 2450$ and $time = 2509$, ${}^{r}a_{d2}$ is initialised, which computes its schedule parameters, and eventually, at $time = 2509$, $pf_{ha}^{schedule}$ sets $dac = d2$ (line 18 of Alg. 2). After comparing c_{switch} and c_{wait} (line 29 of Alg. 2), it assigns the identifier of ${}^{r}a_{d2}$ to ${}^{c}c_{ha,irrDA}$. In response to this, $pf_{ha}^{switchDA}$ is triggered and sends a suspension signal to ${}^{r}a_{d1}$, which is in $\mathtt{S}_{d1}^{stage,1}$(the robot approaches John). The agent switches to state $\mathtt{S}_{d1}^{genSusp}$ (which, following (47), assigns $\mathscr{B}_{d1}^{save}$ to $\mathscr{B}_{d1}^{exeSusp}$) and subsequently switches to $\mathtt{S}_{d1}^{exeSusp}$ and $\mathtt{S}_{d1}^{wait}$. As ${}^{r}a_{d1}$ notifies $pf_{ha}^{switchDA}$ that the suspension strategy ($FSM_{d1,exeSusp}$) has been completed, $pf_{ha}^{switchDA}$ sends a start signal to ${}^{r}a_{d2}$. Finally, at $time = 2513$, ${}^{c}c_{ha,exeDA} = d2$ and ${}^{c}c_{ha,idleDA} = \mathtt{D}_r = \{d1\}$.

From $time = 2513$ to $time = 2519$, John moves closer to the destination of the guidance that is managed by ${}^{r}a_{d1}$; hence, the cost ($cost_{d1} = cost_{dac}$) decreases. At $time = 2519$, the cost of the task switch, namely, c_{switch}, is less than the opposite cost of not switching, namely, c_{wait}. This is because c_{switch} is the sum of the following costs: suspension of the task of ${}^{r}a_{d2}$, completion of the task of ${}^{r}a_{d1}$ (approaching John and guiding John to his destination) and completion of the task of ${}^{r}a_{d2}$ starting from the John's destination pose. c_{wait} is the sum of the following costs: completion the ongoing task of ${}^{r}a_{d2}$ and completion of the task of ${}^{r}a_{d1}$ starting from the Alice's destination pose. Therefore, in response to the environmental setup change (John's movement), the former is less than the latter. Until $time = 2527$, ${}^{r}a_{d2}$ was suspending its task; finally, at this time, ${}^{r}a_{d1}$ can resume its task. ${}^{r}a_{d1}$ completes stages $\mathtt{S}_{d1}^{stage,1}$ and $\mathtt{S}_{d1}^{stage,2}$.

At $time = 2540$, while it was in $\mathtt{S}_{d1}^{stage,3}$ (guiding John to his destination), the robot received a request for the Peter fell task. ${}^{r}a_{d3}$ is initialised, and as the Peter fell task is of humanFell type (${}^{r}a_{d3} \in \mathtt{F}_r$) and the guide John task is of guideHuman type (${}^{r}a_{d1} \in \mathtt{G}_r$), the condition in line 10 of Alg. 2 is satisfied, and ${}^{r}a_{d3}$ is immediately assigned to ${}^{c}c_{ha,irrDA}$. At this point, ${}^{r}a_{d1}$ switches to $\mathtt{S}_{d1}^{genSusp}$, which, according to (47), assigns $\mathscr{B}_{d1}^{apologise}$ to $\mathscr{B}_{d1}^{exeSusp}$. Then, ${}^{r}a_{d1}$ switches to $\mathtt{S}_{d1}^{exeSusp}$, and the robot apologises to John, asks him to stop following it and stores his current pose. Finally, ${}^{r}a_{d1}$ switches to $\mathtt{S}_{d1}^{wait}$, and $pf_{ha}^{switchDA}$ sends a start signal to ${}^{r}a_{d3}$ at $time = 2570$.

Next, while the robot is moving to the Peter fell destination (the dashed circle in Fig. 10), it receives a request for the Sara fell task (whose position is marked with a dotted circle). Comparison of c_{switch} and c_{wait} at $time = 2593$ resulted in a task switch; hence, ${}^{r}a_{d3}$, following FSM_{d3}, switches to state $\mathtt{S}_{d3}^{wait}$, and the robot starts the Sara fell task.

Next, at $time = 2602$, ${}^{r}a_{d3}$ is in $\mathtt{S}_{d3}^{wait}$ and is calculating f_{d3}^{wait} (including pf_{d3}^{report}). It receives information that Peter is fine and there is no need to check his health status. Consequently, f_{d3}^{wait} sends a report to ${}^{r}a_{ha}$ based on (19):

$$ {}_{y}^{ha}c_{d3,report} = [end, scheduleParams] \tag{48} $$

As a result, $term_{event}$ is triggered (36) and $pf_{ha}^{switchDA}$ removes ${}^{r}a_{d3}$ from set ${}^{r}\mathtt{A}_{da}$. During the verification, there was no candidate for ${}^{c}c_{ha,irrDA}$ in the following periods: between the initialisation of ${}^{r}a_{d1}$ ($time = 2450$) and ${}^{r}a_{d2}$ ($time = 2509$), between $time = 2523$ and $time = 2527$ (the following line 8 of Alg. 2), between $time = 2570$ and $time = 2593$ and between $time = 2602$ and $time = 2634$. Therefore, $dac = \emptyset$, and no values of the considered parameters were calculated by $pf_{th}^{schedule}$.

REFERENCES

[1] M. Ben-Ari and F. Mondada, "Robots and their applications," in *Elements of Robotics*. Cham, Switzerland: Springer, 2018, pp. 1–20, doi: 10.1007/978-3-319-62533-1_1.

[2] T. Dahl and M. Boulos, "Robots in health and social care: A complementary technology to home care and telehealthcare?" *Robotics*, vol. 3, no. 1, pp. 1–21, Dec. 2013.

[3] E. Ruiz, R. Acuna, N. Certad, A. Terrones, and M. E. Cabrera, "Development of a control platform for the mobile robot roomba using ROS and a kinect sensor," in *Proc. Latin Amer. Robot. Symp. Competition*, Oct. 2013, pp. 55–60.

[4] G. Farias, E. Fabregas, E. Peralta, H. Vargas, S. Dormido-Canto, and S. Dormido, "Development of an easy-to-use multi-agent platform for teaching mobile robotics," *IEEE Access*, vol. 7, pp. 55885–55897, 2019.

[5] J. Malec, "Some thoughts on robotics for education," in *Proc. AAAI Spring Symp. Robot. Edu.*, 2001, pp. 1–4.

[6] A. Lopez, R. Paredes, D. Quiroz, G. Trovato, and F. Cuellar, "Robotman: A security robot for human-robot interaction," in *Proc. 18th Int. Conf. Adv. Robot. (ICAR)*, Jul. 2017, pp. 7–12.

[7] T. L. Chen and C. C. Kemp, "A direct physical interface for navigation and positioning of a robotic nursing assistant," *Adv. Robot.*, vol. 25, no. 5, pp. 605–627, Jan. 2011.

[8] A. G. Ozkil, Z. Fan, S. Dawids, H. Aanes, J. K. Kristensen, and K. H. Christensen, "Service robots for hospitals: A case study of transportation tasks in a hospital," in *Proc. IEEE Int. Conf. Autom. Logistics*, Aug. 2009, pp. 289–294.

[9] S. A. Frennert, A. Forsberg, and B. Östlund, "Elderly People's perceptions of a telehealthcare system: Relative advantage, compatibility, complexity and observability," *J. Technol. Hum. Services*, vol. 31, no. 3, pp. 218–237, Jul. 2013.

[10] J. Wan, S. Tang, H. Yan, D. Li, S. Wang, and A. V. Vasilakos, "Cloud robotics: Current status and open issues," *IEEE Access*, vol. 4, pp. 2797–2807, 2016.

[11] W. Chen, Y. Yaguchi, K. Naruse, Y. Watanobe, K. Nakamura, and J. Ogawa, "A study of robotic cooperation in cloud robotics: Architecture and challenges," *IEEE Access*, vol. 6, pp. 36662–36682, 2018.

[12] C. Zieliński *et al.*, "Variable structure robot control systems: The rapp approach," *Robot. Auto. Syst.*, vol. 94, pp. 226–244, 2017.

[13] G.-Z. Yang, B. J. Nelson, R. R. Murphy, H. Choset, H. Christensen, S. H. Collins, P. Dario, K. Goldberg, K. Ikuta, N. Jacobstein, D. Kragic, R. H. Taylor, and M. McNutt, "Combating COVID-19—The role of robotics in managing public health and infectious diseases," *Sci. Robot.*, vol. 5, no. 40, Mar. 2020, Art. no. eabb5589. [Online]. Available: https://robotics.sciencemag.org/content/5/40/eabb5589

[14] P. P. Ray, "Internet of robotic things: Concept, technologies, and challenges," *IEEE Access*, vol. 4, pp. 9489–9500, 2016.

[15] D. Perzanowski, A. C. Schultz, W. Adams, E. Marsh, and M. Bugajska, "Building a multimodal human-robot interface," *IEEE Intell. Syst.*, vol. 16, no. 1, pp. 16–21, Jan. 2001.

[16] I. A. Awada, I. Mocanu, S. Jecan, L. Rusu, A. M. Florea, O. Cramariuc, and B. Cramariuc, "Mobile@Old—An assistive platform for maintaining a healthy lifestyle for elderly people," in *Proc. E-Health Bioeng. Conf. (EHB)*, Jun. 2017, pp. 591–594.

[17] A. Zalewski, K. Borowa, and A. Ratkowski, "On cognitive biases in architecture decision making," in *Software Architecture*, A. Lopes and R. de Lemos, Eds. Cham, Switzerland: Springer, 2017, pp. 123–137.

[18] A. Zalewski and S. Kijas, "Beyond ATAM: Early architecture evaluation method for large-scale distributed systems," *J. Syst. Softw.*, vol. 86, no. 3, pp. 683–697, Mar. 2013.

[19] S. Bedaf, G. J. Gelderblom, D. S. Syrdal, H. Lehmann, H. Michel, D. Hewson, F. Amirabdollahian, K. Dautenhahn, and L. de Witte, "Which activities threaten independent living of elderly when becoming problematic: Inspiration for meaningful service robot functionality," *Disab. Rehabil., Assistive Technol.*, vol. 9, no. 6, pp. 445–452, Nov. 2014.

[20] J. F. Engelberger, *Robotics in Practice: Management and Applications of Industrial Robots*. Springer, 2012.

[21] K. M. Tsui and H. A. Yanco, "Assistive, rehabilitation, and surgical robots from the perspective of medical and healthcare professionals," in *Proc. AAAI Workshop Hum. Implications Hum.-Robot Interact.*, Jul. 2007, pp. 1–6.

[22] F. Hegel, M. Lohse, A. Swadzba, S. Wachsmuth, K. Rohlfing, and B. Wrede, "Classes of applications for social robots: A user study," in *Proc. RO-MAN 16th IEEE Int. Symp. Robot Hum. Interact. Commun.*, Aug. 2007, pp. 938–943.

[23] M. Vukobratovic, *Dynamics and Robust Control of Robot-Environment Interaction*, vol. 2. Singapore: World Scientific, 2009.

[24] V. Kumar, E. Todorov, and S. Levine, "Optimal control with learned local models: Application to dexterous manipulation," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2016, pp. 378–383.

[25] W. Dudek, M. Wegierek, J. Karwowski, W. Szynkiewicz, and T. Winiarski, "Task harmonisation for a single-task robot controller," in *Proc. 12th Int. Workshop Robot Motion Control (RoMoCo)*, Jul. 2019, pp. 86–91.

[26] C. Zieliñski and T. Winiarski, "Motion generation in the MRROC++ robot programming framework," *Int. J. Robot. Res.*, vol. 29, no. 4, pp. 386–413, Apr. 2010.

[27] T. Kornuta, C. Zieliński, and T. Winiarski, "A universal architectural pattern and specification method for robot control system design," *Bull. Polish Acad. Sci., Tech. Sci.*, vol. 68, no. 1, pp. 3–29, Feb. 2020.

[28] C. Zieliński, T. Kornuta, P. Trojanek, T. Winiarski, and M. Walęcki, "Specification of a multi-agent robot-based reconfigurable fixture control system," in *Robot Motion and Control*, K. Kozłowski, Ed. London, U.K.: Springer, 2012, pp. 171–182.

[29] D. Seredynski, M. Stefanczyk, K. Banachowicz, B. Swistak, V. Kutia, and T. Winiarski, "Control system design procedure of a mobile robot with various modes of locomotion," in *Proc. 21st Int. Conf. Methods Models Autom. Robot. (MMAR)*, Aug. 2016, pp. 490–495.

[30] D. Seredynski, T. Winiarski, and C. Zielinski, "FABRIC: Framework for agent-based robot control systems," in *Proc. 12th Int. Workshop Robot Motion Control (RoMoCo)*, Jul. 2019, pp. 215–222.

[31] W. Dudek, K. Banachowicz, W. Szynkiewicz, and T. Winiarski, "Distributed NAO robot navigation system in the hazard detection application," in *Proc. 21st Int. Conf. Methods Models Autom. Robot. (MMAR)*, Aug. 2016, pp. 942–947.

[32] W. Dudek, W. Szynkiewicz, and T. Winiarski, "Nao robot navigation system structure development in an agent-based architecture of the RAPP platform," in *Challenges in Automation, Robotics and Measurement Techniques*, R. Szewczyk, C. Zieliński, M. Kaliczyńska, Eds. Cham, Switzerland: Springer, 2016, pp. 623–633.

[33] W. Dudek, W. Szynkiewicz, and T. Winiarski, "Cloud computing support for the multi-agent robot navigation system," *J. Autom., Mobile Robot. Intell. Syst.*, vol. 11, no. 2, pp. 67–74, Jun. 2017.

[34] K. Tchoń, K. Arent, M. Janiak, and Ł. Juszkiewicz, "Motion planning for the mobile platform rex," in *Recent Advances in Automation, Robotics and Measuring Techniques*, R. Szewczyk, C. Zieliński, M. Kaliczyńska, Eds. Cham, Switzerland: Springer, 2014, pp. 497–506.

[35] E. Gat, R. P. Bonnasso, and R. Murphy, "On three-layer architectures," *Artif. Intell. mobile robots*, vol. 195, p. 210, Feb. 1998.

[36] B. P. Gerkey and M. J. Matarić, "A formal analysis and taxonomy of task allocation in multi-robot systems," *Int. J. Robot. Res.*, vol. 23, no. 9, pp. 939–954, Sep. 2004.

[37] Z. Kootbally, C. Schlenoff, C. Lawler, T. Kramer, and S. K. Gupta, "Towards robust assembly with knowledge representation for the planning domain definition language (PDDL)," *Robot. Comput.-Integr. Manuf.*, vol. 33, pp. 42–55, Jun. 2015.

[38] C. Zieliński, W. Kasprzak, T. Kornuta, W. Szynkiewicz, P. Trojanek, M. Walęcki, T. Winiarski, and T. Zielińska, "Control and programming of a multi-robot-based reconfigurable fixture," *Ind. Robot, Int. J.*, vol. 40, no. 4, pp. 329–336, Jun. 2013.

[39] T. Winiarski, M. Węgierek, D. Seredyński, W. Dudek, K. Banachowicz, and C. Zieliński, "EARL—Embodied agent-based robot control systems modelling language," *Electronics*, vol. 9, no. 2, p. 379, Feb. 2020.

[40] M. Figat and C. Zielinski, "Robotic system specification methodology based on hierarchical Petri nets," *IEEE Access*, vol. 8, pp. 71617–71627, 2020.

[41] T. Kornuta, T. Winiarski, and C. Zieliński, "Specification of abstract robot skills in terms of control system behaviours," in *Progress in Automation, Robotics and Measuring Techniques*, R. Szewczyk, C. Zieliński, and M. Kaliczyńska, Eds. Cham, Switzerland: Springer, 2015, pp. 139–152.

[42] W. Dudek. *A Video Showing Scheduling of a Robot's Tasks With the TaskER Framework*. WUT, Institute of Control and Computation Engineering. Accessed: Aug. 31, 2020. [Online]. Available: https://vimeo.com/403391725

[43] R. Volpe, I. Nesnas, T. Estlin, D. Mutz, R. Petras, and H. Das, "The CLARAty architecture for robotic autonomy," in *Proc. IEEE Aerosp. Conf.*, Mar. 2001, pp. 1–121.

[44] S. Dennis, L. Alex, L. Matthias, and S. Christian, "The smartmdsd toolchain: An integrated mdsd workflow and integrated development environment (ide) for robotics software," *J. Softw. Eng. Robot.*, vol. 7, pp. 3–19, 2016, doi: 10.6092/JOSER_2016_07_01_p3.

[45] A. Steck and C. Schlegel, "Smarttcl: An execution language for conditional reactive task execution in a three layer architecture for service robots," in *Proc. Int. Workshop Dyn. Lang. RObotic Sensors Syst. (DYROS/SIMPAR)*, 2010, pp. 274–277.

[46] M. Stenmark, J. Malec, and A. Stolt, "From high-level task descriptions to executable robot code," in *Intelligent Systems*, D. Filev, J. Jabłkowski, J. Kacprzyk, M. Krawczak, I. Popchev, L. Rutkowski, V. Sgurev, E. Sotirova, P. Szynkarczyk, and S. Zadrozny, Eds. Cham, Switzerland: Springer, 2015, pp. 189–202.

[47] R. Alami, A. Clodic, V. Montreuil, E. A. Sisbot, and R. Chatila, "Toward human-aware robot task planning," in *Proc. AAAI Spring Symp., Boldly go Where Hum.-Robot team Has Gone Before*, 2006, pp. 39–46.

[48] T. Winiarski, K. Banachowicz, M. Walecki, and J. Bohren, "Multibehavioral position-force manipulator controller," in *Proc. 21st Int. Conf. Methods Models Autom. Robot. (MMAR)*, Aug. 2016, pp. 651–656.

[49] M. Stilman, "Task constrained motion planning in robot joint space," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Oct. 2007, pp. 3074–3081.

[50] M. Nakano, Y. Hasegawa, K. Nakadai, T. Nakamura, J. Takeuchi, T. Torii, H. Tsujino, N. Kanda, and H. G. Okuno, "A two-layer model for behavior and dialogue planning in conversational service robots," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2005, pp. 3329–3335.

[51] J. Bohren and S. Cousins, "The SMACH high-level executive [ROS News]," *IEEE Robot. Autom. Mag.*, vol. 17, no. 4, pp. 18–20, Dec. 2010.

[52] S. Amador, S. Okamoto, and R. Zivan, "Dynamic multi-agent task allocation with spatial and temporal constraints," in *Proc. 28th AAAI Conf. Artif. Intell.*, 2014, pp. 1384–1390.

[53] A. Nouman, I. F. Yalciner, E. Erdem, and V. Patoglu, "Experimental evaluation of hybrid conditional planning for service robotics," in *Proc. Int. Symp. Experim. Robot.*, D. Kulić, Y. Nakamura, O. Khatib, and G. Venture, Eds. Cham, Switzerland: Springer, 2016, pp. 692–702.

[54] M. Cashmore, M. Fox, D. Long, D. Magazzeni, B. Ridder, A. Carrera, N. Palomeras, N. Hurtos, and M. Carreras, "Rosplan: Planning in the robot operating system," in *Proc. 21th Int. Conf. Automated Planning Scheduling*, 2015, pp. 333–341.

[55] S. Sievers, G. Röger, M. Wehrle, and M. Katz, "Theoretical foundations for structural symmetries of lifted PDDL tasks," in *Proc. Int. Conf. Automated Planning Scheduling*, vol. 29, no. 1, 2019, pp. 446–454.

[56] H. J. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. B. Scherl, "GOLOG: A logic programming language for dynamic domains," *J. Log. Program.*, vol. 31, nos. 1–3, pp. 59–83, Apr. 1997.

[57] C. Zieliński and T. Kornuta, "Specification of tasks in terms of object-level relations for a two-handed robot," in *Recent Advances in Automation, Robotics and Measuring Techniques*, R. Szewczyk, C. Zieliński, and M. Kaliczyńska, Eds. Cham, Switzerland: Springer, 2014, pp. 543–552.

[58] M. Colledanchise and P. Ögren, *Behavior Trees in Robotics and AI: An Introduction*. Boca Raton, FL, USA: CRC Press, 2018.
[59] G. Gierse, T. Niemueller, J. Claßen, and G. Lakemeyer, "Interruptible task execution with resumption in golog," in *Proc. 22nd Eur. Conf. Artif. Intell.*, 2016, pp. 1265–1273.
[60] M. Kim, H. Ahn, and K. P. Kim, "Process-aware Internet of Things: A conceptual extension of the Internet of Things framework and architecture," *TIIS*, vol. 10, no. 8, pp. 4008–4022, 2016.
[61] S. Park, J.-H. Kim, and G. Fox, "Effective real-time scheduling algorithm for cyber physical systems society," *Future Gener. Comput. Syst.*, vol. 32, pp. 253–259, Mar. 2014.
[62] H. Mora, J. F. Colom, D. Gil, and A. Jimeno-Morenilla, "Distributed computational model for shared processing on cyber-physical system environments," *Comput. Commun.*, vol. 111, pp. 68–83, Oct. 2017.
[63] J. Yao, X. Xu, and X. Liu, "MixCPS: Mixed time/event-triggered architecture of cyber–physical systems," *Proc. IEEE*, vol. 104, no. 5, pp. 923–937, May 2016.
[64] J. Kim, K. Lakshmanan, and R. Rajkumar, "Rhythmic tasks: A new task model with continually varying periods for cyber-physical systems," in *Proc. IEEE/ACM 3rd Int. Conf. Cyber-Phys. Syst.*, Apr. 2012, pp. 55–64.
[65] J. Xu and D. L. Parnas, "Scheduling processes with release times, deadlines, precedence and exclusion relations," *IEEE Trans. Softw. Eng.*, vol. 16, no. 3, pp. 360–369, Mar. 1990.
[66] E. D. Jensen, C. D. Locke, and H. Tokuda, "A time-driven scheduling model for real-time operating systems," in *Proc. RTSS*, vol. 85, 1985, pp. 112–122.
[67] F. Zhang, K. Szwaykowska, W. Wolf, and V. Mooney, "Task scheduling for control oriented requirements for cyber-physical systems," in *Proc. Real-Time Syst. Symp.*, Nov. 2008, pp. 47–56.
[68] M. Ghobaei-Arani, A. Souri, F. Safara, and M. Norouzi, "An efficient task scheduling approach using moth-flame optimization algorithm for cyber-physical system applications in fog computing," *Trans. Emerg. Telecommun. Technol.*, vol. 31, no. 2, Feb. 2020, Art. no. e3770.
[69] K. Jeffay, "Scheduling sporadic tasks with shared resources in hard-real-time systems," in *Proc. 13th IEEE Real-Time Syst. Symp.*, Phoenix, AZ, USA, Dec. 1992, pp. 89–99.
[70] J. Ota, "Goal state optimization algorithm considering computational resource constraints and uncertainty in task execution time," *Robot. Auto. Syst.*, vol. 57, no. 4, pp. 403–410, Apr. 2009.
[71] C. Sirithunge, A. G. B. P. Jayasekara, and D. P. Chandima, "Proactive robots with the perception of nonverbal human behavior: A review," *IEEE Access*, vol. 7, pp. 77308–77327, 2019.
[72] T. Winiarski, W. Dudek, M. Stefańczyk, Ł. Zieliński, D. Giełdowski, and D. Seredyński, "An intent-based approach for creating assistive robots' control systems," 2020, *arXiv:2005.12106*. [Online]. Available: http://arxiv.org/abs/2005.12106
[73] E. A. Topp, M. Stenmark, A. Ganslandt, A. Svensson, M. Haage, and J. Malec, "Ontology-based knowledge representation for increased skill reusability in industrial robots," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2018, pp. 5672–5678.
[74] INCARE Project Web Page. *WUT, Institute of Control and Computation Engineering*. [Online]. Available: https://www.robotyka.ia.pw.edu.pl/projects/incare/

WOJCIECH DUDEK (Student Member, IEEE) received the M.Sc./Eng. degree in control and robotics from the Warsaw University of Technology (WUT). He is currently a Research and a Didactics Assistant with the Faculty of Electronics and Information Technology (FEIT), Institute of Control and Computation Engineering (ICCE), WUT. He is also a contributor to international projects, e.g., RAPP (European Commission–FP 7) and INCARE (European Commission AAL Joint Programme). His research interests include modeling, design, and programming of robot controllers, especially in the areas of mobile robots and their localization, navigation, and harmonization of their tasks. He received the Young Author Award at the IEEE 12th International Workshop on Robot Motion and Control for the precursory article of this study.

TOMASZ WINIARSKI (Member, IEEE) received the M.Sc./Eng. and Ph.D. degrees in control and robotics from the Warsaw University of Technology (WUT), in 2002 and 2009, respectively. He is currently an Assistant Professor with WUT. He is also a member of the Robotics Group and the Head of the Robotics Laboratory, Faculty of Electronics and Information Technology (FEIT), Institute of Control and Computation Engineering (ICCE). He is also working on the modeling and design of robots and on programming methods for robot control systems. He is also the Head of the WUT Group in AAL–INCARE Project "Integrated Solution for Innovative Elderly Care." His research interests include service and social robots and didactic robotic platforms. His personal experience concerns the development and modeling of robotic frameworks, manipulator position–force and impedance control, and safety in robotic research.