WARSAW UNIVERSITY
OF TECHNOLOGY

Faculty of Electronics and Information Technology

Ph.D. THESIS

mgr inż. Wojciech Dudek

Prudent management of interruptible tasks executed by a service robot

Supervisor
dr hab. inż. Wojciech Szynkiewicz

WARSAW 2021

Acknowledgments

*First, I want to express my gratitude to my supervisor **Dr hab. Wojciech Szynkiewicz** for his scientific support and inspiration. His patience and forbearance shown in priceless comments made our cooperation fruitful in this dissertation. I cannot omit the effort of **Prof. Cezary Zieliński**. Numerous inspirational, scientific discussions with him were guides for my work that resulted in this dissertation.*

*Besides the professors' mentoring and guidance, the day-by-day support from my master's thesis supervisor **Dr Tomasz Winiarski** had another essential impact on my work. I am thankful for his time, scientific discussions, work reviews and jokes that helped me traverse problems. I want to thank **my scientific team members** for suggestions, fruitful cooperation in projects, and open discussions on not only scientific topics.*

*I want to thank my whole **family** and **friends** for the constant and unconditional motivation and support. In particular, I want to express my gratitude to my wife for relaxing trips and talks that preserved me from becoming another robot. Thank you for taking care of the things and duties that were not visible from above the papers and the computer's screen. I want to thank my siblings for being a shining and motivating example of a well-organised, ambitious, successful and dreams-achieving researcher and leader. Finally, I express my gratitude to my parents for teaching me priorities, critical thinking and factual discussion that are the most excellent tools for a researcher.*

Without you all, this work would be neither satisfactory nor possible to conduct.

Dziękuję całej rodzinie i przyjaciołom za bezustanne i bezwarunkowe wsparcie oraz motywację. W szczególności dziękuję mojej żonie za odprężające wycieczki i pogawędki, które uchroniły mnie przed pracocholizmem i przekształceniem się w kolejnego robota z laboratorium. Dziękuję Ci, za zajęcie się sprawami, które nie były dla mnie widoczne znad dokumentów, artykułów i ekranu komputera. Dziękuję mojemu rodzeństwu za motywujący wzór dobrze zorganizowanego, ambitnego i odnoszącego sukcesy naukowca, inżyniera i lidera. Dziękuję rodzicom za przekazane wartości oraz naukę zarówno krytycznego myślenia, jak i rzeczowej, opartej na faktach dyskusji, które są najlepszymi narzędziami naukowca.

Bez was wszystkich ta praca nie byłaby ani satysfakcjonująca, ani możliwa do wykonania.

Dziękuję!

I acknowledge the support of the INCARE AAL-2017-4-059 project "Integrated Solution for Innovative Elderly Care" founded by the AAL JP and co-funded by the AAL JP countries.

I acknowledge the support of the "Look & learn: Skill acquisition by a companion robot based on task demonstration" project founded by Warsaw University of Technology.

I acknowledge the support of the the FP7 Collaborative Project RAPP (Grant Agreement No. 610947), funded by the European Commission. The work is supported by the Polish Ministry for Science and Higher Education scientific research funds for the years 2014–2016 granted for the realization of co-financed international project.

Abstract

Task management is a core ability of living and artificial autonomous entities. In robotics, it is especially crucial for versatile service robots that are tailored to help humans in various duties. In some applications, robots are shared by multiple users that don't agree their requests. Versatile service robots often work in dynamic environments, and their users' needs and preferences change in time. Thus, the robots need an advanced task management ability that includes i.a. dynamic priority assignment, repetitive check of tasks' feasibility and task plans update. Additionally, robots need to foresee the consequences of their task interruption (e.g. leaving a cooker on). Therefore, the problem of prudent task management respecting the danger of interrupting robot's tasks arises. Prudent task management considered in this dissertation is constituted of i.a. safe suspension and resumption of independent tasks, schedule parameters reappraisal invoked by changes in the environment and termination of a queued tasks that are no longer feasible.

The research this dissertation describes was targeted to design a model of robot control systems with prudent task management. The model specifies a robot system with an agent-based approach and consists of multiple agents of various classes. A Harmoniser class Agent that manages task requests, initialises tasks and switches their modes of operation following a configurable task management algorithm. The model specifies Dynamic class Agents that execute single tasks described with Hierarchical Finite State Machines (HFSM). States of the HFSM have assigned goals, e.g., a safe suspension management of the current task before a task switch. Thanks to that, the model enables mitigation of the problems in predicting the consequences of the robot's activity interruption. The requirements specified for the conducted study forced convenient configuration of the model to enable its application to various robots executing diverse tasks in different environments.

The robot systems are complex, and their overall behaviour is troublesome to describe precisely. Therefore, this dissertation uses a formal method—Embodied Agent Meta-model to specify the robot control system model. Thanks to that, the experiments and the model implementation can be accurately reproduced. What is more, the model can be fitted for other requirements and reused.

The model resulted from the research is used to implement two example control systems of different robots (TIAGo mobile robot and Velma mobile manipulator). The robots work in different environments, and various algorithms manage their tasks. The example systems are launched in typical scenarios. The robots' behaviours and their tasks management are confronted with the requirements and constraints of the research.

Keywords: Service robots, System engineering, Finite State Machines, Scheduling, Task analysis

Rozważne zarządzanie przerywalnymi zadaniami wykonywanymi przez robota usługowego

Streszczenie

Zarządzanie zadaniami jest podstawową umiejętnością autonomicznych organizmów żywych i urządzeń. Problem zarządzania zadaniami jest kluczowy dla wszechstronnych robotów usługowych, które pomagają ludziom w wypełnianiu różnych obowiązków. Niekiedy roboty te są współdzielone przez wielu użytkowników, którzy nie uzgadniają ze sobą poleceń dla robota. Wszechstronne roboty usługowe często pracują w zmiennych środowiskach, a do tego ich użytkownicy zmieniają swoje żądania i preferencje. A zatem, takie roboty potrzebują odpowiedniego zarządzania zadaniami. Dzięki niemu robot ustala i aktualizuje priorytety, powtarzalnie sprawdza możliwość wykonania zadań oraz aktualizuje plany wykonania zadań. Ponadto, roboty przerywając obecnie realizowane zadanie, muszą przewidywać konsekwencje wynikające z tego przerwania (np. pozostawienie włączonej kuchenki). System sterowania robota, który posiada wyżej wymienione cechy oraz wstrzymuje/wznawia zadania, odświeża estymaty parametrów szeregowania w reakcji na zmiany w środowisku oraz zamyka zadania oczekujące, które stały się niewykonalne, nazywany jest w tej rozprawie rozważnym.

Problem podejmowany w niniejszej rozprawie to określenie modelu dla systemu sterowania robota, który jest rozważny podczas przełączania zadań. Model ten bazuje na podejściu agentowym i określa wiele agentów, które należą do różnych klas. Agent harmonogramujący zarządza żądaniami zadań, uruchamia zadania oraz zmienia ich tryby pracy według konfigurowalnego algorytmu zarządzania zadaniami. Opracowany model definiuje agenty dynamiczne, które wykonują pojedyncze zadania. Działanie tych agentów jest oparte o hierarchiczne automaty skończone, których stany mają przypisane cele. Przykładowo, wyróżnione są stany zarządzające bezpiecznym wstrzymaniem aktualnego zadania przed wykonaniem przełączenia na inne zadanie. Dzięki temu opracowany model upraszcza problem przewidywania konsekwencji wynikających z przerwania czynności robota oraz umożliwia przeciwdziałanie tym nieporządanym. Jednym z isotnych wymagań dla opisywanych prac jest łatwa i wielokierunkowa konfiguracja modelu. Dlatego możliwe jest wykorzystanie tego modelu w systemach różnych robotów wykonujących rozmaite zadania w zróżnicowanych środowiskach. Model bazuje na rozwiązaniu integrującym wiele robotów, przez co opracowane rozwiązanie może być łatwo zaaplikowane do systemów wielorobotowych. System sterowania wszechstronnego robota usługowego często jest skomplikowany, a opis jego działania jest zawiły i niejasny. W tej rozprawie zastosowano formalną notację agenta upostaciowionego do definicji proponowanego modelu.

Dzięki temu model jest szczegółowo przeanalizowany, a efekt pracy, przeprowadzone eksperymenty i implementacja mogą być dokładnie odtworzone.

Wynik opisanych w tej rozprawie badań został wykorzystany do opracowania przykładowych systemów sterowania dwóch robotów: mobilnego robota TIAGo oraz mobilnego manipulatora Velma. Roboty te pracują w różnych środowiskach, a ich zadania są szeregowane przez dedykowane algorytmy. W ramach weryfikacji przeanalizowano opracowany model w kierunku spełnienia założeń postawionych na początku badań. Ponadto skonfrontowano działanie przykładowych systemów sterowania w typowych scenariuszach dla problemu przełączania zadań.

Słowa kluczowe: Roboty usługowe, Inżynieria systemów, Automaty skończone, Szeregowanie, Analiza zadania

Table of Contents

Glossary

Term	Description
Action	An abstract, atomic step of an activity carried out by an entity.
Activity	An abstract complex operation of an entity defined by a flow of multiple actions.
Agent	In agent-based approach to systems specification, agents are the primary parts of a system. The detailed description of the Embodied Agent concept used in this dissertation is provided in Sec. 1.4.2 and Sec. 2.4.
Basic behaviour	A formally defined operation of an Embodied Agent while it stays in a given state of the FSM governing the agent's operation. The model of the basic behaviour is shown in Fig. 2.6.
Environment	It is a set of entities that the system act on and percept.
FSM	It is a Finite State Machine governing the operation of a subsystem of an Embodied Agent.
Initial condition	A logic function associated with an edge of an FSM graph and defines the condition for the edge. To preserve the FSM's uniqueness, the initial conditions associated with all edges directed from a state must be mutually exclusive, and their composition must return $True$ to preserve completeness of the FSM.

Schedule parameters	Arguments that are taken into account by the scheduling algorithm. They are used to compute costs that influence schedule decisions. They can have various complexity and interpretation depending on the requirements of the system. Typically, priorities and temporary constraints are used as schedule parameters; however, they can be of any form that reflects the system's main objective (e.g., a scheduling algorithm of a system for helping elderly people can maximise their comfort as a schedule parameter). The complete classification of the parameters considered in this study is shown in Fig. 2.2.
Scheduling algorithm	An algorithm that is defined in a Task Harmoniser Agent. The algorithm defines the strategy for choosing a Dynamic Agent that should execute its task at the time. It uses schedule parameters to compute the result that is called schedule decision.
Scheduling decision	It is a result of a scheduling algorithm. It can have one of the three values: 'continue', 'switch', 'start'. The 'continue' value means that the ${}^{r}\mathrm{a}_{\mathrm{exeDA}}$ should continue its task, or there are no Dynamic Agents awaiting execution of their tasks. The 'switch' value indicates the need to replace the ${}^{r}\mathrm{a}_{\mathrm{exeDA}}$ with an agent chosen by the scheduling algorithm (${}^{r}\mathrm{a}_{\mathrm{irrDA}}$). The 'start' value indicates the need to start the task of ${}^{r}\mathrm{a}_{\mathrm{irrDA}}$ and at the time, there is no ${}^{r}\mathrm{a}_{\mathrm{exeDA}}$.
Scope of an agent or agent set	It is specified in the left upper index of the agent/agent set symbol (α). If $\alpha = s$, the agent is system-wide and operates even if none of the robots is connected to the system. If $\alpha = \mathtt{R}$, the agent is associated with a robot from the system. If $\alpha = r$, where $r \in \mathtt{R}$, the agent is associated with the robot named r. If α is not defined, the agent set aggregates agents of different scopes.
Stage	A part of a task. It is represented as a state of the FSM that specifies the task.
State	A state of an FSM. Operation of an agent in a given state is defined by the basic behaviour associated with the state.

Subsystem	A part of an Embodied Agent. There are various types of subsystems and they are classified to control subsystems, virtual receptors, virtual effectors, real effectors, real receptors.
Super state	A state that is given by an FSM. If an FSM consists of a super state, the FSM is hierarchical.
System	A composition of all agents defined by the system developer. The system operates in its environment and can percept and affect it.
Task	An activity that needs to be executed by the robot to satisfy the request received from its user. Each Dynamic Agent is responsible for handling one request. Tasks have different types, and each type implement a parametrised activity.
Terminal condition	A logic function that defines a condition for basic behaviour to terminate. The model of a basic behaviour utilising the concept of the terminal condition is shown in Fig. 2.6.
Transition function	A function that for defined arguments (a subsystem's memory and input buffers) returns values to the output buffers and the memory of the subsystem that executes the function.

Symbol	Description
${}^{\alpha}\mathrm{a}_{\mathrm{name}}$	An agent that operates in the scope defined by α and is identified by the $name$ parameter.
${}^{r}\mathrm{a}_{\mathrm{exeDA}}$	Refers to a Dynamic Agent of the robot r that is currently executing the task it carries.
${}^{r}\mathrm{a}_{\mathrm{irrDA}}$	Refers to a Dynamic Agent of the robot r that is selected by the scheduling algorithm to replace ${}^{r}\mathrm{a}_{\mathrm{exeDA}}$.
`da`, `tha`, `tra`, `cla`, `exa`, `sta`, `pla`	Identifiers of the agent classes: Dynamic Agent (`da`), Task Harmoniser Agent (`tha`), Task Requester Agent (`tra`), Cloud Agent (`cla`), Executor Agent (`exa`), Store Agent (`sta`), Platform Agent (`pla`)
${}^{scope}\mathtt{A}_{class}$	An agent group that aggregates agents of a given $class$ and a given $scope$. If $scope$ is not defined, the agent set aggregates agents of a given class and various scopes.
`R`	A set of robots' names operating in the system.

c_{owner}	A control subsystem of an agent identified as $owner$.
${}^{\mathrm{c}}\mathrm{c}_{owner}$	Memory of a control subsystem of an agent identified as $owner$.
${}^{\mathrm{c}}\mathrm{c}_{owner}[field]$	A memory field of an $owner$ agent's control subsystem.
${}^{T}_{x}\mathrm{c}_{owner,interlocutor}[field]$	A field named $field$ of an input buffer of the agent $owner$ connected to the agent $interlocutor$.
${}^{T}_{y}\mathrm{c}_{owner,interlocutor}[field]$	A field named $field$ of an output buffer of the agent $owner$ connected to the agent $interlocutor$.
${}_{\alpha}\mathrm{bs}_{owner,interlocutors}[field]$	A set of buffer fields named $field$ of the agent $owner$ connected to the agent(s) $interlocutors$. If $interlocutors$ parameter is a set of agents, the symbol refers to the specified buffer fields of the $owner$ connected with all the agents aggregated in the agent set (e.g., $\mathrm{bs}_{ha,\mathtt{A}_{\mathtt{tra}}}[task]$ refers to a 'task' fields of 'ha' agent's buffers connected to all the agents of `tra` class). The α parameter is the direction of the buffers aggregated in the set, e.g. α ='x' for input, α ='y' for output, or α =' ' for both.
FSM_{owner}	A Finite State Machine describing the operation of the control subsystem of the $owner$ agent.
$\mathrm{FSM}_{owner,superState}$	A Finite State Machine describing the operation of the control subsystem in the super state named $superState$.
$\mathrm{S}^{name}_{owner}$	A state named by $name$ parameter of the control subsystem of the agent $owner$.
$\mathrm{ic}_{owner,name}$	An initial condition named by $name$ parameter of the agent $owner$.
U_{owner}	A set of all suspendable stages of the Dynamic Agent named $owner$. A suspendable stage is expressed by (3.24).
L_{owner}	A set of all blocking stages of the Dynamic Agent named $owner$. A blocking stage is expressed by (3.25)

$\mathcal{B}^{name}_{owner}$	A basic behaviour of the $owner$ agent's control subsystem. The basic behaviour is identified by the $name$ parameter.
$\mathrm{tc}_{owner,name}$	A terminal condition of the behaviour $name$ of the $owner$ agent's control subsystem.
$\mathrm{f}^{name}_{owner}$	A transition function of the behaviour $name$ of the $owner$ agent's control subsystem.
$\mathrm{pf}^{name}_{owner}$	A primitive transition function of the transition function $name$ of the $owner$ agent's control subsystem.
$\mathtt{D}_r$	A set of Dynamic Agents of the robot r awaiting for execution of the tasks they carry. $\mathtt{D}_r = {}^r\mathtt{A}_{\mathtt{da}} / \{{}^r\mathrm{a}_{\mathrm{exeDA}}, {}^r\mathrm{a}_{\mathrm{irrDA}}\}$.
$\mathtt{F}_r$	Is a subset of ${}^r\mathtt{A}_{\mathtt{da}}$ that are of `humanFell` type and the subset is defined by (4.3).
$\mathtt{G}_r$	Is a subset of ${}^r\mathtt{A}_{\mathtt{da}}$ that are of `guideHuman` type and the subset is defined by (4.4).
$newData(\alpha)$	A logic function that returns $True$ if the buffer given by α contains newer data than it was in the function's previous call. If α is a set of buffers, the function returns $True$ if any of the buffers' value changes.
$timer(\alpha)$	A logic function that returns $True$ repetitively with the frequency specified by α.

Chapter 1

Introduction

1.1 Motivation

Humans develop technology to solve the problems they encounter and to make life convenient. A significant part of the problems may seem new to us; however, studies show that various living organisms resolved them during evolution. Therefore, researchers often mirror the solutions developed and verified in real life by nature. This approach applies well in service robotics, where robots are used to replace humans/animals in tasks/duties that require skills similar to human's/animal's. Every living organism has desires (e.g. to endure) and receives stimulus that pushes it to carry out actions hoping to receive a short- or long-term reword. The more desires the organism has, the more complicated its task management is. An ability to order its activities constitutes the organism's autonomy. Humans would like to treat service robots like autonomous and smart servants helping them in everyday duties and fulfilling their whims. Therefore, the robots need to manage their activities accordingly to the wish of their users. However, on the one hand, users demand robots to be intelligent, and on the other hand, they do not want to specify an exact execution time of their requests. This ability is especially crucial in robot systems cooperating with multiple users who do not agree their requests between themselves. In such cases, the problem of multiple supervisors arises, and smart request management is obligatory. It should be noted that in contrast to cybernetic systems, robots affect the real world. Therefore, some robot tasks require suspending actions before they will be switched with another task. Visualisation of the problem concept is shown in Fig. 1.1. Multiple users request different tasks with various priorities at different times. Therefore, the robot needs to interrupt one task with another prudentially, e.g. turn off the cooker if the interrupted task switched it on. This dissertation addresses the need for prudent management of the task requests for a service robot and proposes a solution for specified constraints. The solution is named ***TaskER*** (Task har-

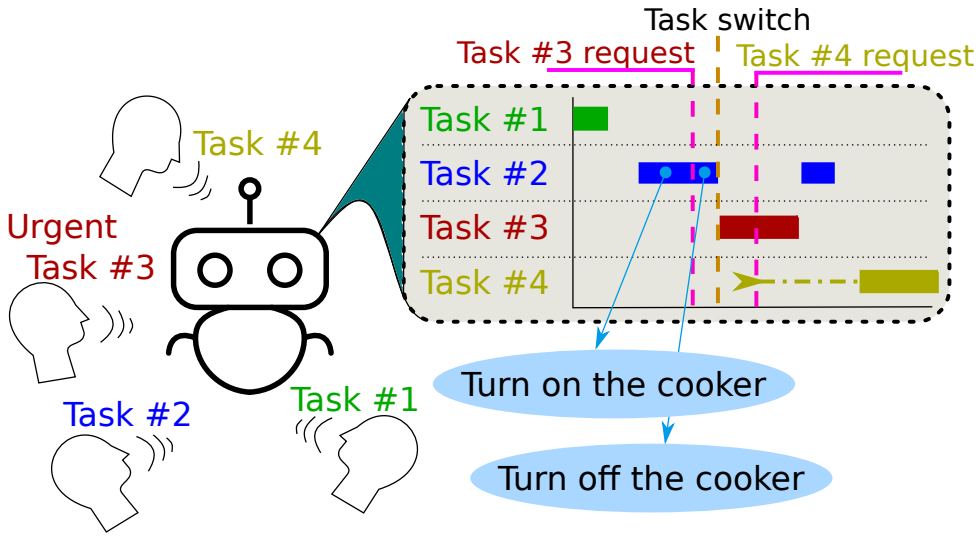


Figure 1.1: The concept of the problem considered in this dissertation.

monisER) and constitutes a set of constraints for the robot system development. Following the multi-application character of robot systems, TaskER applies to various applications involving robots of different types.

1.2 Background of the research

In everyday life, we humans have many duties. We want to stay productive and not waste any time. The authors of [1] report that humans tent to juggle multiple tasks simultaneously to be productive even at the expense of performance. The work results indicate that some multitasking improves productivity, but there is a point where multitasking has a negative effect. Additionally, the study confirms that more multitasking has a deleterious effect on performance effectiveness. Therefore, juggling multiple tasks is much more complex, where performance loss can have serious consequences. The authors of [2] test the students' reading performance while being distracted by instant messaging. The test result shows an extra time (excluding the time for instant messaging) is required to carry out an academic task at the same quality level while multitasking. There is also a study on humans' distraction by a cell phone use while driving [3]. Besides confirming the study thesis, the work reveals an interesting fact of inattention blindness of the driver while he uses a phone. The phenomenon causes the drivers to overlook information directly in their line of sight. The study reports that conversations with a passenger do not induce inattention blindness. There is a small group of "supertaskers" who can carry out this dual-task combination without impairment. Additionally, the authors discuss the neural regions that support this ability. The problem of multitasking effectiveness appears

in robotics as well; however, its source is different. Following the referenced papers, humans lose effectiveness while multitasking mostly because of distraction. Simultaneously, it seems to be a minor problem in the face of robots' artificial memory. However, for robots, multitasking effectiveness can be impaired by non-optimal task switch strategies or inaccurate definition of inter-task relations.

Tasks carried out by an entity need to be ordered by an algorithm that defines the entity's imperative. The algorithm ordering tasks usually minimises or maximises a cost function, but the function may be contingent and depends on various measurable and abstract parameters. An example here can be a human simply minimising travel distance while shopping products placed in different shop areas. However, when some products are limited, he/she will assign priorities to them, and he/she will switch the goods collecting strategy to approach a limited product first. Therefore, the environment changes and requests for new tasks change task plans, strategies, and parameters (e.g. priorities).

The robots are demanded to optimise a tremendous amount of parameters. In mobile robots, one of the most important tasks requiring optimisation is trajectory tracking [4] and obstacle avoidance [5]. Solutions differ in the considered constraints and robot kinematics (e.g. unicycle-like [6], [7], n-trailers [8]). However, robots do not only execute motion actions but also communicate with users and acquire information. Therefore, the overall robot's behaviour composes multiple motions, communication and computation actions that need to be managed and optimised.

Safety, fault-tolerance and error handling are other crucial problems in robotics [9], [10]. Robots interact with humans, operate in dynamic [11], unknown [12] and inaccessible by the service staff environments like space. In some applications, they can injure users [13], and generally, the expected response to an error/fault is complicated [14]. Hazards can be caused not only by a robot executing actions or its hardware faults, but by interrupting one task by another as well. Interruptions are non-standard behaviours in a system and need to be managed carefully, and any dangerous consequence of an interruption should be identified and prevented.

The development of cutting edge technology increases the applicability, universality and popularity of robots. Therefore, the management of robots' tasks is necessary and expected. Following the robot classification by application field (published in [15]), there are two main branches (industrial and service robots), which are compositions of lower level robots' classes. Inspection of the service robot branch reveals the versatility of the applications helping humans in, e.g. medicine [16], [17], home [18], education [19]–[21], agriculture [22] and defence [23] areas. Each of the applications has different constraints and requirements for robotic systems. In health care, usually, robots are targeted to maximise human safety and personnel convenience; on the other hand, industrial robots optimise production quality and quantity. It is possible be-

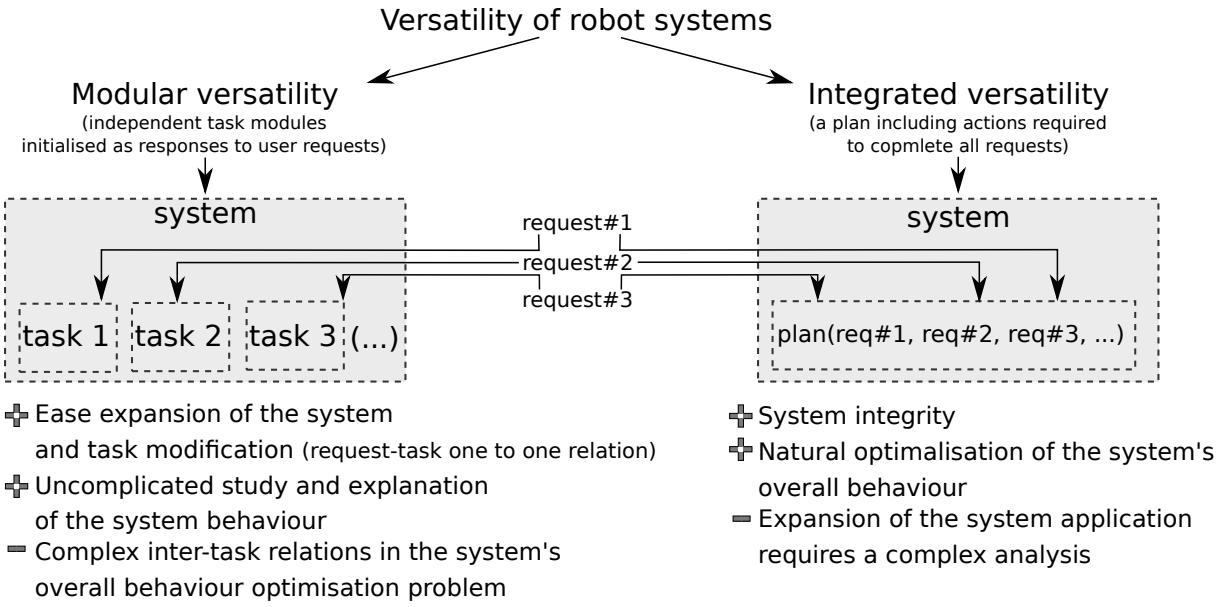


Figure 1.2: Classification of robot systems considering versatility implementation – modular and integrated versatility

cause human safety is preserved by safety devices and the environment structure (e.g., robot cages). In each of the applications, robots realize various tasks (e.g. in hospitals, there are nurse assistance [24], object transportation [25], or therapy [26]–[28] applications). Therefore, some state-of-the-art robotic systems designate a separate module, which manages the system according to the user request [29], [30]. Moreover, these works enable an extension of the requests available at the system deployment phase. For example, the authors of [30] utilise a task store concept, which enables application-domain experts to deploy new tasks to the system, and each of the tasks resolves one type of user's request. The tasks are available in the store, and an appropriate one is deployed on a robot on the robot's user request. The other approach to managing various user requests is to plan a multi-goal task that integrates all received requests. Tasks in this approach are decomposed into actions combined to complete all requests. Visualisation of the classification in the versatility implementation criterion is shown in Fig. 1.2.

Analysis of the classification in application extension direction shows persuasive arguments for modular versatility systems. The first is that each task includes its expert knowledge defined by concepts in a dedicated domain. Same concepts in various task contexts may have different meanings and may direct reasoning to different conclusions. An example is a state of a door (opened/closed) in contexts of a navigation task and a hazard detection task. In the navigation case, an ajar door is interpreted as closed because the robot can not traverse it; however, in the hazard detection case, such a state of the door (e.g., a passage to a confidential room) is interpreted as opened and results with an alarm.

The second advantage of modular versatility relates to the topic of this thesis – task management. Independence of the tasks allows developers to reason in the task's context and not in the whole system's context including the set of all possible tasks. Moreover, some tasks must be developed with a domain expert's support (e.g., a baking task with a baker). Thus, the analysis of the task interruption needs to be based on expert knowledge. Based on the analysis, the robot can compose a plan to minimise the cost of the current task interruption (e.g., by lowering the oven's temperature during baking to give the robot additional time to complete the interrupting task).

Thus, the development of multiple tasks is more convenient in modular versatility systems. Moreover, their specialisation and modification are more straightforward than modification of one knowledge base integrating contexts of all requests, as it is integrated versatility systems. It is challenging to analyse possible undesirable inferences and effects after adding a new concept to the knowledge base while extending the system's functionality.

The recent COVID-19 pandemic shows the need for easy and convenient robot tasks configuration, especially in unexpected situations. Thanks to the store aggregating independent tasks, robots available in galleries or at universities could be easily configured to help medical personnel at hospitals [31].

The tasks can be requested via multiple human-machine interfaces (Internet of Things devices [32] or a human-robot interface [33]–[35]) by different operators (e.g. medical personnel or patients). Moreover, not all task requests have to be agreed between the operators, and there may be a necessity to interrupt one task with another in a circumstance change. Therefore, the robots have to manage their tasks and be able to suspend and resume them.

The above problems concern various applications of robots and make robotic systems complicated. Therefore, studies propose varied specification methods and models, which ease understanding and facilitate implementation of the system. Usually, the works are targeted to resolve this problem for a constrained set of systems. One of the approaches is Model-Driven Engineering (MDE). It defines the model concept to ease the design procedure of a particular system and prevent some cognitive biases [36] influence the system design at the initial phase [37]. MDE can be expressed with a formal notation based on mathematical equations and automata theory (e.g. Embodied Agent Meta-model [38]) or with graphical domain definition languages (e.g. SysML [39]). There is also a study combining the two approaches to express MDE in robotics [40].

1.3 Thesis of the dissertation

The research contributes to service robotics, addresses a problem of a responsible switch of robot's tasks and resolves it within the stated constraints. The following theses are formulated as a result of the conducted research:

- **A service robot can compromise the convenience and safety of humans and the safety of robots/objects in their environment while it switches independent tasks,**
- **In various robot applications, different algorithms and parameters for task scheduling should be used,**
- **The proposed model of a service robot control system (named *TaskER*) fosters safety and user's convenience while the robot manages independent, suspendable tasks in a dynamic environment, and the model is configurable in the aspects of:**
  - **the task scheduling algorithm,**
  - **the parameters used to compute schedule,**
  - **the interfaces for task requests,**
  - **the set of tasks available for robots in the system.**

1.4 Works that the study is based on

1.4.1 Model-Driven Engineering

There are various development procedures for Cyber-Physical Systems (CPSs). They are classified into three groups: conventional, hybrid and agile. The difference appears on the timeline while switching between the typical phases: plan, conceptualise, design, develop. The work [41] introduces a methodology for selecting the proper procedure for an individual problem. The CPSs may have much in common in the plan, conceptualise and design phases, although they will result in diverse systems in the end. This is because CPSs share similar equipment, environments, algorithms and have standard features to deal with different problems.

To facilitate the development of systems that resolve related problems or apply similar features, the Model-Driven Engineering (MDE) emerged [42]. It is a methodology to develop systems utilising the model concept. There are several popular definitions of the model:

1. the model is a set of declarations and rules describing the system under study [43],
2. the model is an impression of a system allowing predictions or reasoning to be made [44],

3. the model is a reduced depiction of some system that focuses interest on the system properties relevant for a given perspective [45],

4. the model is an abstract of a system created as a substitution of the system for investigation purposes [46].

Silva et al. [42] clue that cited authors agree that a model describes a system under study and vice-versa. This dissertation's definition of the model is adopted from his work: "*model is a system that helps to define and give answers of the system under study without the need to consider it directly*".

There are models defining constraints, general structure and operation of the systems resolving a given set of problems (like Smart Grid automation [47]) or applying similar features (like energy-aware scheduling [48]). In the MDE approach, there is a concept of meta-models defined as well. They are a model of models; therefore, a system resolving a stated problem can be developed by adjusting a meta-model to tailor the resulting system to the problem's requirements. There are several advantages of this methodology, and the most important from this thesis perspective are:

- facilitation of the system's creation deriving from a meta-model,
- shorten the system development time,
- increasing reliability of the systems, as meta-model are reusable and can be corrected based on the experience from multiple implementations.

In contrast to information systems, cyber-physical ones include hardware that percepts the system's environment and/or affects it. Therefore, there are dedicated models for this kind of systems.

1.4.2 Agent-based meta-model

An agent-based approach to systems specification defines autonomous entities called agents as primary parts of a system. Agents cooperate to solve a problem stated as a goal of the system. Agent-based systems offer flexibility and intuitive interpretation of the system's behaviour. The system parts' learning process can be expressed naturally, and the distribution of the systems' responsibilities is straightforward. Agents can communicate with other agents and the environment to compute decisions and learn elementary actions. Their knowledge can be expressed in various level of complexity and abstraction. Agents to affect/percept their environment and share their knowledge with other agents according to their and the system's goal. Agent-based

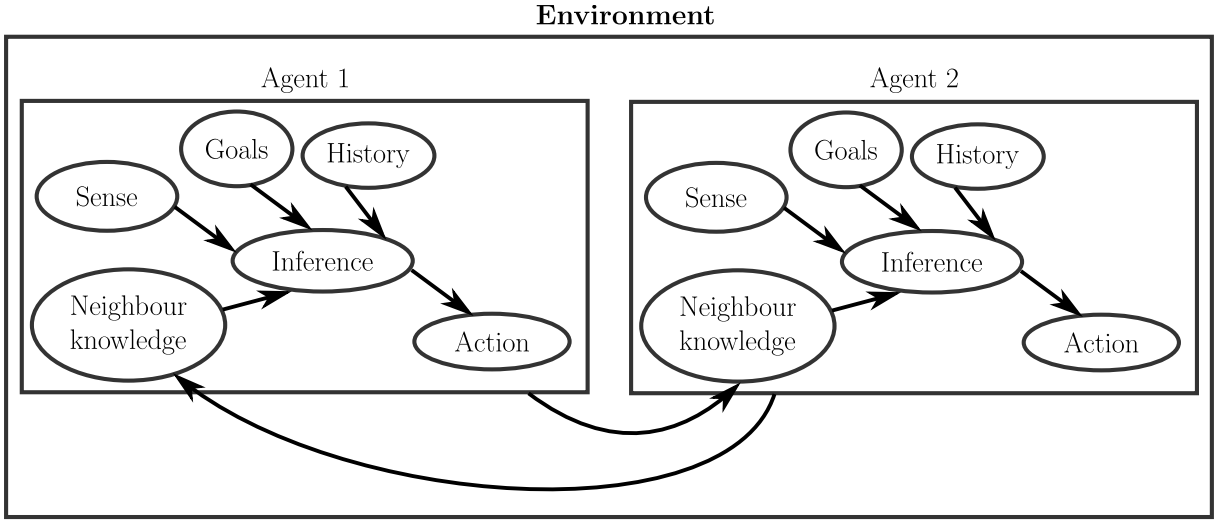


Figure 1.3: The concept of an agent-based system based on the survey [49].

systems solve problems in various areas like civil engineering, electrical engineering or computer science and robotics. A concept of agent-based meta-model proposed by Dorri et al. [49] is shown in Fig. 1.3.

Embodied Agent Meta-model

The concept to distribute systems to agents emerged and became popular in the area of cybernetic systems. Thanks to a natural representation of the intelligent system as cooperating agents, the concept was adapted in robotics as Embodied Agent Meta-model (**EAM**) [38], [40]. Robots are cyber-physical systems, so they sense and affect the environment; therefore, cybernetic agents evolved to Embodied Agents. Following the Embodied Agent Meta-model, agents are described with various structures contingent on their abilities to percept and affect the environment. There are four general activities of an agent:

- C – reasoning of the agent,
- E – influencing the environment,
- R – sensing the environment,
- T – communicating with the other agents.

The first one is obligatory; however, the other three are facultative. So EAM enumerates eight types of agents: C, CT, CE, CR, CET, CRT, CER, CERT. Description of the types and their typical assignment are described in [38], [40]. An agent's structure is decomposed into subsystems of various types that execute one of the above general activities. A control subsystem is aware of the agent's goal and manages the other subsystems to accomplish it. Besides the control subsystem, there are two branches— the *perception* branch and the *effect* branch. They

are decomposed into virtual and real subsystems; thus, there are virtual receptors and real receptors in the perception branch and virtual effectors and real effectors in the effect branch. The real subsystems interact with the environment (sense or affect it), and the virtual ones interface them to the control subsystem. Virtual receptors gather data and transform it into the form understandable by the control subsystem. Virtual effectors interpret commands from the control subsystem and transform them into the form understandable by the real effectors. Cardinality of the subsystems is a project-dependent decision and it is constrained by EAM.

The overall behaviour of a system modelled with EAM is a composition of all the agents' behaviours composing the system. An agent's behaviour is a composition of all the subsystems' behaviours comprising the agent. Finally, the behaviour of a subsystem is defined as a Finite State Machine (FSM). A basic behaviour gives the operation of a subsystem in a state of the FSM. Each basic behaviour executes a transition function calculating the subsystem's output, a terminal condition defining the basic behaviour's termination event, and an error condition defining an unexpected situation that needs to be managed. The FSM states are switched if a basic behaviour is finished (either the terminal or the error condition is satisfied). The next state is chosen based on evaluating the initial conditions, which are logic predicates labelling directed arcs of the FSM.

1.4.3 RAPP architecture

The RAPP[1] system [30] is specified in Embodied Agent Meta-model; thus, its structure comprises a set of cooperating agents. The system involves a cloud computing platform and multiple robots. A general use case of this system is to enable developers of robotic applications to implement robot type independent applications, upload them to the store, and enable robots to execute the applications on their users' requests. Since a robot downloads and launches the requested application, it governs the robot to achieve the application's goal. The application can also request the cloud platform to support the robot with the cloud resources (e.g. storage, computation power).

Agents and their roles in RAPP system

The RAPP system consists of agents distributed between the robot and the cloud according to the algorithm that was published in [50], [51]. There are five classes of agents. Agents of one class have the same role in the system and have similar functionalities allowing the role fulfilment. Graphical specification of the services that are provided and consumed by the classes used in RAPP is shown in Fig. 1.4:

[1] Robotic Applications for Delivering Smart User Empowering Applications

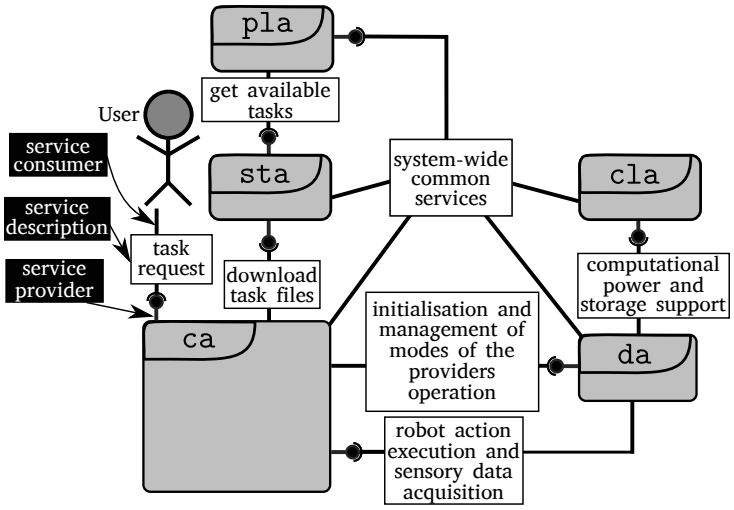


Figure 1.4: The cooperation of the agent classes introduced in RAPP (`pla`, `sta`, `ca`, `cla`, `da`).

1. *Core Agent* (`ca` class) – It is a CERT-type agent managing the robot hardware. It acts as a low-level controller of the robot and provides typical, fundamental robot behaviours to the task-level controller—`da` class. Among others, the `ca` class commonly delivers motion planning and execution algorithms (e.g., [52]–[54]). As it manages all sensors of a robot, it also processes requests of the user, downloads the requested task files (implementing `da`-class agents) and launches the task. If a new request is received, the current task is aborted straight away. Its control software operates on the robot's computer. There is one `ca` class agent for each robot in the system.
2. *Platform Agent* (`pla` class) – a CT-type agent that operates in the cloud and provides system-wide services that require high computational power or massive storage space. There is one `pla` class agent in the system.
3. *Dynamic Agent* (`da` class) – a CT-type agent that manages a specified task. Such a task is composed of various actions, such as requests of `ca` class behaviours and platform agent services. For example, a `da` class implements a human guide task composed of a robot motion action (one of the `ca` class agent's basic behaviour) and detection of a human in pictures sent to the `pla` class agent. In the RAPP project, only one `da` class agent operates on a robot at the time and executes its task. If the task is finished, the robot waits for further requests.
4. *Cloud Agent* (`cla` class) – a CT-type agent that may be spawned in the cloud by a `da` class to delegate complex task-related computation operations from the robot and store large files in the cloud. A `cla`-class agent is strictly connected to the task and the `da`-class agent that spawned it. There are at most as many `cla`- as `da`-class agents because it is not obligatory for a `da`-class agent to spawn a `cla`-class agent in the cloud.

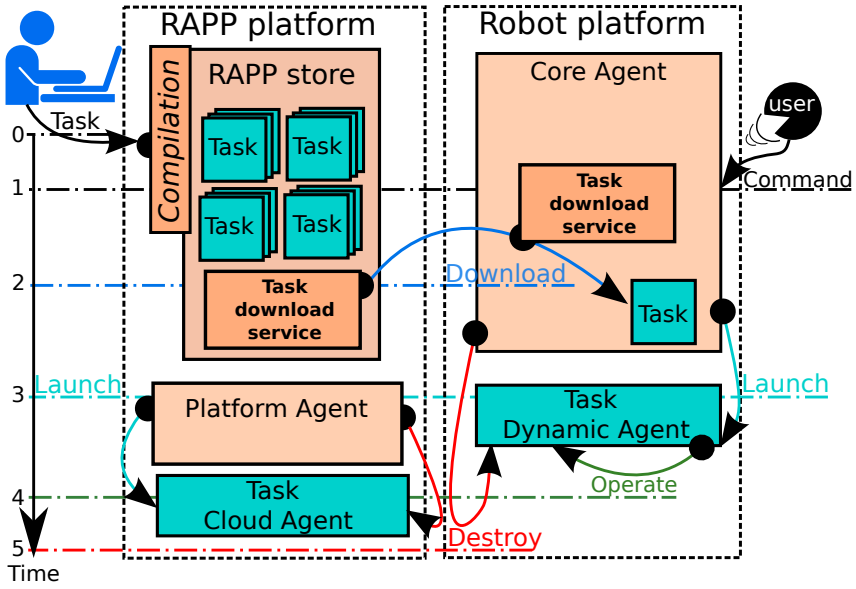


Figure 1.5: The life-cycle of a task in the RAPP system [56].

5. *RAPP Store* (`sta` class) – a CT-type agent that operates in the cloud and stores task files. Files of a specific task are spawned on a robot upon its user request, and when launched, become a `da`-class agent.

RAPP agents can be assigned to layers defined by the three-layer architecture approach [55]. 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.

Task execution in RAPP system

The life-cycle of a typical task—Dynamic Agent in RAPP system is presented in Fig. 1.5. Implemented and built tasks for all supported robots are stored in the RAPP Store (step = 0). On a user request to complete a task (step = 1), the Core Agent downloads appropriate files from the RAPP store (step = 2). If the task is composed of two agents— the Cloud Agent and Dynamic Agent, the former is launched in the cloud with the Platform Agent's help and the latter in the Robot platform (step = 3). Next, the Dynamic Agent takes control over the Robot platform and using the Platform Agent's and Cloud Agent's services manages the user request (step = 4). Finally, when the task is finished, the Dynamic Agent and the corresponding Cloud Agent are terminated, and the main control of the Robot platform returns to the Core Agent (step = 5). The detailed specification of the general structure of the RAPP system is presented in [57].

1.5 Organisation of the research

The workflow of the research described in this dissertation can be presented in a waterfall graph flowing from the precise definition of the problem through the proposition and evaluation of a robot controller model ending with the analysis of its implementation (Fig. 1.6).

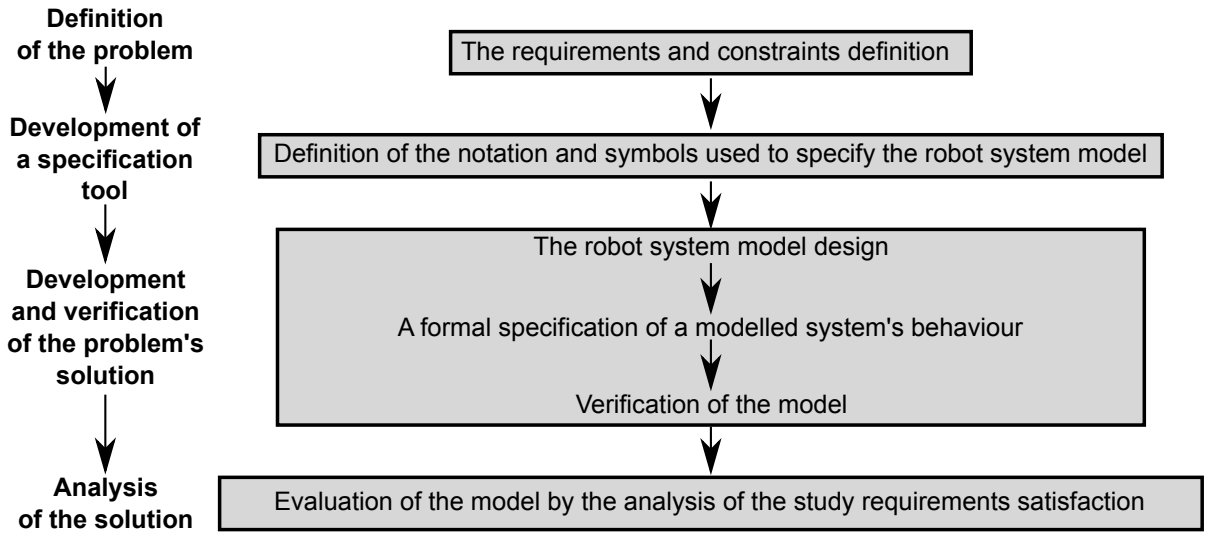


Figure 1.6: The workflow of the research, and organisation of its presentation.

The organisation of the main parts of this dissertation reflects the research workflow and is as follows:

- First, the precise definition of the problem is provided:
  - a set of use cases is defined to focus the research on real situations and problems faced by a robot control system (Sec. 2.1),
  - a set of requirements and constraints is defined to shape the model to manage the use cases (Sec. 2.2),
  - the contribution and applicability of the model are declared (Sec. 2.3).
- Next, the notation and symbols used in the model's definition are described (Sec. 2.4).
- Subsequently, the structure and behaviour of a system based on the model is formulated (Sec. 3.1-3.5). This part is an extended description of the work conducted for this dissertation and published in [58]. It consists of an in-depth description of the model configuration for various applications.
- In the following Sections 4.2 and 4.3, constraints and configurations of the model for the example robot systems are presented. The former section is an extended description of the

system and its verification involving the TIAGo robot [59] (Fig. 4.1a). It was originally brought in [58]. The latter section is a novel description of the Velma mobile-manipulator verification system [60], [61] (Fig. 4.1b).

- Confrontation of the proposed solution with the study requirements is presented in Sec. 4.2.4 and with related works in Sec. 5.1,
- The dissertation ends with a discussion of the conducted research results, and argumentation of the thesis stated in the introduction (Sec. 5.2). The study revealed some additional problems to be resolved in future work and they are gathered in Sec. 5.3.

Chapter 2

Explanation of the problem and the formal notation

In the following sections, the assumptions and constraints bounding the conducted research are declared by the use cases in Sec. 2.1 and the requirements resulting from them in Sec. 2.2. The initial constraints of the proposed model, specified with the use cases and the requirements, are chosen to satisfy INCARE project [62] goals, e.g. support the elderly in retirement homes. Additionally, the system needs to allow an easy and snappy extension of the available task set. In Sec. 2.3, the concept of the solution, the applicability of the model and the contribution of this work is stated. Finally, in Sec. 2.4, the formal notation used to define the model is introduced.

2.1 Considered use cases

Good practices of engineering advice to investigate the stated problem, specify use cases and requirements first and then design the system which manages the use cases and fulfils the requirements. Therefore, from the vast area of robotic research, this thesis regards systems facing the use cases presented in Fig. 2.1 and Tab. 2.1.

There are two classes of actors who interact with the robot system: a developer and a user. Based on the analysis of the use cases (described in the following subsections), the requirements for the system are formulated.

2.1.1 Use case – Tasks as modular extensions

There are many possible duties that a robot can carry out for its users. For example, in the area of helping elderly people, the study [63] identified multiple activities that threaten independent living in mobility, self-care and social interaction. Therefore, robots should be able to manage

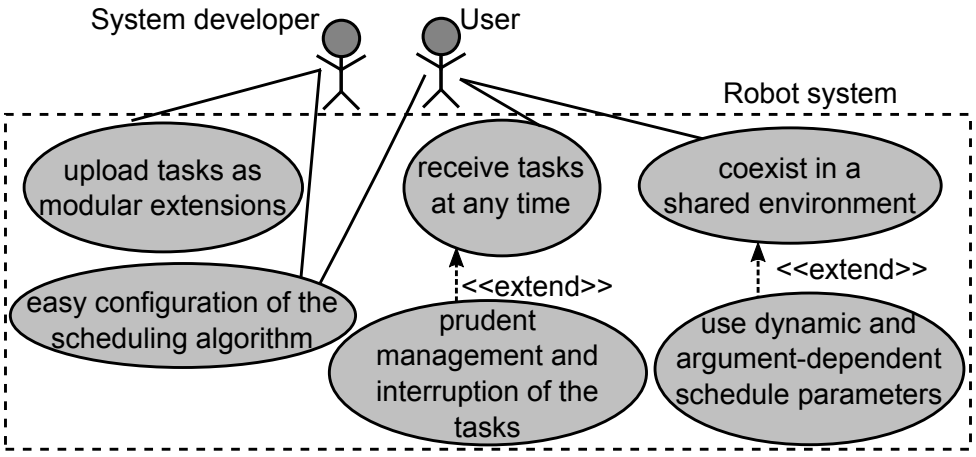


Figure 2.1: The use case diagram for the systems considered in this study.

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 to this problem is an application store, which collects independent programs that can be downloaded and launched upon a user's request.

2.1.2 Use case – Easy configuration of the scheduling algorithm

Robot systems resolve various problems in many sectors, such as industry [64], healthcare [65] and entertainment [66]. Robots in these sectors use different criteria to manage objectives, e.g., in industry, they maximise production quality and quantity, and in healthcare, they minimise nurses' and medical personnel's time consumption and effort. In contrast, robots in the homes of elderly people should optimise these individuals' comfort and safety. To complete various objectives, the robot controller must utilise a configurable scheduling algorithm that computes schedule decisions that minimise the cumulative cost during the robot operation. Furthermore, the cost function can change with time. It can depend on a specified task (e.g., as the delay in water transportation to an elderly person increases, his/her discomfort and danger increases). Schedule parameters that are used to compute the costs and schedule 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).

Table 2.1: 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 available for the robots in the system

Use case name	Easy 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 tasks at any time	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

2.1.3 Use case – Coexistence in a shared environment

Robots interact with the physical environment and affect it with actions that compose their tasks. Although effects of the 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) [67]. Therefore, the robot controller must be aware of the potential damage, injury, or loss caused by an interruption of an ongoing task (e.g., while a new task is initiated, a cooker remains on during the new task execution). Thus, the robot controller must consider the current state of the environment and foresee the future effects of the current and future actions. Unfortunately, it is challenging to design a comprehensive model of an environment for estimating its state in the future based on

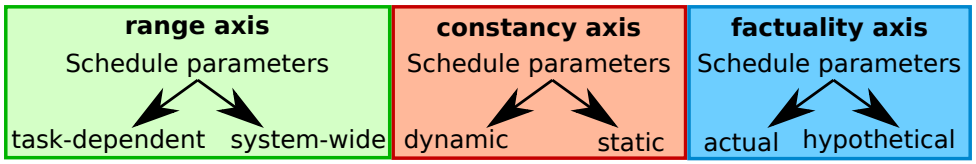


Figure 2.2: The classification of schedule parameters.

the robot's actions. The known task models describe how the environment changes while the task proceeds [68]. 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 complex problem, as humans differ in many aspects, and it is not easy to model their behaviours and needs. These problems also impact the task switching for human-robot collaboration. For example, a task's deadline depends on high-level abstract conditions[1], users can change priorities of the requested tasks and can complete a part of a queued task. Therefore, the parameters used to schedule the tasks should be dynamic and repetitively updated. Furthermore, the scheduling algorithm should consider various sequences of the tasks awaiting execution and choose the best one. For this reason, the system should be able to calculate hypothetical schedule parameters for tasks. They are used to reflect a task execution cost assuming a hypothetical state of the environment and the system. The classification of schedule parameters used to manage the tasks is shown in Fig. 2.2. The classification is conducted along three axes: range, constancy and factuality. The range axis divides schedule parameters into $system\text{-}wide$ that regard all available tasks and $task\text{-}dependent$ that are compared between tasks of the same type. The constancy axis divides schedule parameters into those set once ($static$) and those that can change during the system operation ($dynamic$). The factuality axis divides schedule parameters into $actual$ ones calculated for the current state of the system and the environment and $hypothetical$ ones calculated for a specified hypothetical situation.

2.2 The system requirements

The following requirements for a robot controller are stated to enable the task harmonisation feature and address the problems that are specified in the above use cases:

`R1` – the robot controller maintains additional activities to enable an advised task switch. It constantly listens to task requests and executes a schedule decision and switches tasks, even if the robot executes another task;

[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 the requested task "B" is not acceptable.

R2 – values of the schedule parameters (e.g., priorities) used to compute schedule decisions are dynamically changing even if a task awaits execution. If one of the parameters changes, then the scheduling algorithm is initiated. All classes of the schedule parameters shown in Fig. 2.2 can be used in the system;

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 individual system's requirements;

R4 – the tasks that are available in the system are created independently and differ in terms of knowledge base and contexts;

R5 – the developed model must raise awareness of the task developer to foresee possible dangerous situations caused by a task switch. Additionally, the proposed task execution method enables independent tasks to oversee changes in the environment and set various schedule parameters. Furthermore, the ongoing task is carefully suspended before the controller switches to another task. As a result of this, a possible robot/environment damage or other loss due to the task interruption is limited; and

R6 – a plan of a task that awaits execution is updated before the task execution.

2.3 Contribution and applicability

In the study, the TaskER model is introduced. It extends the RAPP architecture [30] to enable the task scheduling feature. The TaskER model is a novel approach to schedule robot tasks and is an answer to robot tasks harmonisation problem, i.e. it shapes a robot system to flexibly coordinate, organise and combine tasks assigned to the robot.

2.3.1 Contribution

Two approaches are available for managing task harmonisation: considering the tasks as constant and uninterruptible (as in the RAPP system) or allowing the system to interrupt the ongoing tasks. Thus, the classification of tasks, robots, and task allocation introduced in [69] needs to be extended with an additional criterion—task harmonisation. The extended classification is shown in Fig. 2.3. The constant-task (CT) harmonisation manifests with postponing task switch execution until the ongoing task completes. Thus, while the robot executes a task, all received requests will be analysed only after the current task is complete. In contrast, the interruptible-task (IT) harmonisation enables the system to decide on the task switch as soon as the new task

request is received. In this case, the robot can suspend the ongoing task, conduct a set of actions of the new task and restore the previous task from suspension.

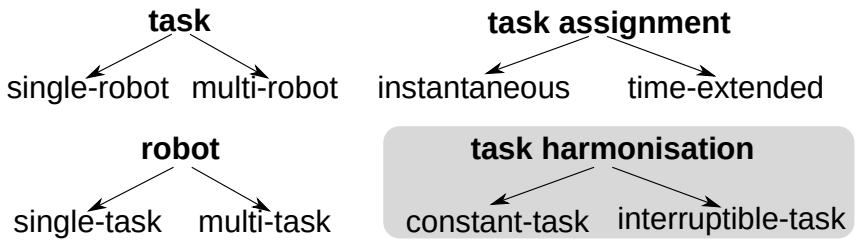


Figure 2.3: The taxonomy of tasks, robots, and task allocation described in [69] extended with the task harmonisation criterion.

The work presented in this dissertation enables systems based on the RAPP architecture to use an algorithm and versatile parameters to manage task switches. Task harmonisation can be either of constant-task or interruptible task type. The parameters can be any of the previously described classes.

None of the models nor example systems presented in the related work (Section 5.1) satisfy all the requirements stated in Section 2.2 and describe a system that can harmonise tasks following the interruptible-task approach. In this dissertation, the model of a robot control system that has the following features is introduced:

1. safe suspension and resumption of independent tasks,
2. convenient reconfiguration of a scheduling algorithm and parameters that it requires to compute schedule decisions,
3. computation of 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,
4. convenient extension of available tasks,
5. a task plan update before the task execution,
6. termination of a queued task if it is no longer beneficial/feasible,
7. task rescheduling in reaction to reappraisal of schedule parameters, and
8. online reconfiguration of task harmonisation to either constant-task or interruptible-task model.

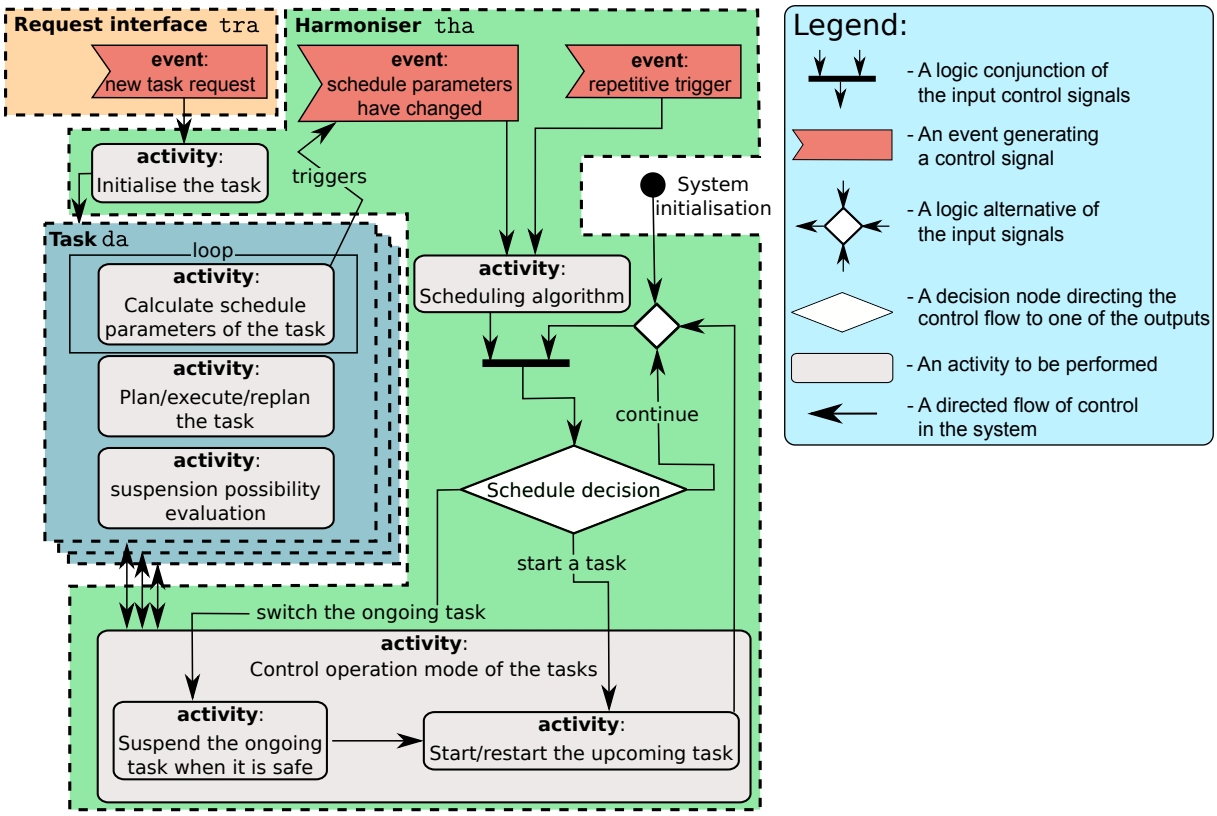


Figure 2.4: The procedure for handling new task requests and evaluating dynamically changing schedule parameters results in replacing or continuing the ongoing task. The arrows show the control flow between the abstract activities of the abstract parts of the system. The logic conjunction used in the diagram holds handling a new schedule decision while the previous one is being managed. The decisions are not queued; therefore, the last decision is managed when it is possible. The events of Harmoniser trigger the Scheduling algorithm in both event and timely manner. The activity 'initialise the task' invokes a new Dynamic Agent handling the requested task.

Furthermore, the formal notation for describing the model and the systems inherited from it is proposed. The notation helps the systems' developers in various aspects, for example, in relating analogous entities defined in different approaches, in the efficient diagnosis of the system state, or in making the model definition more precise than if it was defined by ambiguous relations and entities.

Finally, the implementation of the TaskER framework is described. It follows the model and supports the development of various tasks and scheduling algorithms application. The general concept of the harmonisation procedure described formally in the model is illustrated in Fig. 2.4. The harmoniser part was not considered in RAPP, and the request interface was a part of a `ca` class agent; therefore, in the TaskER model, additional agent classes cover the roles of these parts. The concept shown in Fig. 2.4 is formalised in the model and visualised in Fig. 3.8.

There are three main parts in the harmonisation procedure: the request interface, the tasks and the harmoniser. The request interface defines the structure of the task requests. Each task calculates its schedule parameters (used to compute the schedule by the harmoniser part), is responsible for its plan management and execution and shares an interface to manage its mode of operation. Based on the schedule parameters received from the tasks, the harmoniser part manages their operation modes as it is defined in the scheduling algorithm. The proposed harmonisation procedure prevents multiple tasks from executing their actions at once, thus, prevents the robot to behave chaotically.

2.3.2 Constraints and applicability

The model is based on a component structure and describes the required components and recommended interactions between them to handle task suspension and resumption. Additionally, tasks must be divided into stages that are classified as suspendable or blocking. The former ones can be interrupted, and the other cannot. A task executing a blocking stage holds the interruption until the task reaches a first suspendable stage.

The proposed approach harmonises tasks conducted by a robot and not tasks of a whole system that consists of multiple robots. However, a multi-robot system also benefits from the proposed model because the model is applicable to each robot of the multi-robot system.

Referencing the extended classification of robots, tasks, and task allocations [69], this study considers:

1. single-task robots that can execute at most one task at a time,
2. single-robot tasks that require exactly one robot for completion, and
3. instantaneous-assignment task allocation that corresponds to the system that does not possess any information 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. The TaskER model can be applied to a system that uses FSM as the task model, at least at the top level. An exemplary integration of TaskER with tasks modelled with Petri Nets is presented in this dissertation. The model can be used in a system with knowledge representation in the PDDL [70] to harmonise the sequences deployed by the planning component. The TaskER model allows `da`-class agents to compute and send task-related parameters to

the scheduling algorithm to influence the schedule decision (e.g., the estimated time for completing the task and the estimated time for suspending the current task). The above assumptions are not restrictive; however, they allow for developing a multi-tasking robot with interruptible-task harmonisation.

2.4 Notation of the model specification

Following the Embodied Agent Meta-model, the agents are abstract parts building a system. They communicate with each other and have the imperative to use their resources to complete their objectives. A system's responsibilities are divided into objectives that are distributed among the agents. 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.

It is assumed that the system can involve a set of robots; however, the tasks of each of them are harmonised independently. The set of the robots 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 execute tasks. A set of agents associated with a specified robot is denoted as ${}^{r}\mathtt{A}$, where r is the robot's 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 ${}^{h}\mathtt{A}_{u}$ is used, where $h \in \{s, \mathtt{R}, r\}$ specifies the symbol of the set, and $u \in \{\mathtt{exa}, \mathtt{da}, \mathtt{cla}, \mathtt{tha}^{1}, \mathtt{tra}^{1}, \mathtt{pla}, \mathtt{sta}\}$ specifies the class of the agents. The sets and subsets are defined formally by (2.1)-(2.4).

$$\mathtt{A} = {}^{s}\mathtt{A} \cup {}^{\mathtt{R}}\mathtt{A}, \tag{2.1}$$

$${}^{s}\mathtt{A} = {}^{s}\mathtt{A}_{\mathtt{tra}} \cup {}^{s}\mathtt{A}_{\mathtt{pla}} \cup {}^{s}\mathtt{A}_{\mathtt{sta}}, \tag{2.2}$$

$${}^{\mathtt{R}}\mathtt{A} = {}^{\mathtt{R}}\mathtt{A}_{\mathtt{exa}} \cup {}^{\mathtt{R}}\mathtt{A}_{\mathtt{da}} \cup {}^{\mathtt{R}}\mathtt{A}_{\mathtt{tha}} \cup {}^{\mathtt{R}}\mathtt{A}_{\mathtt{cla}} \cup {}^{\mathtt{R}}\mathtt{A}_{\mathtt{tra}} = \bigcup_{r \in \mathtt{R}} {}^{r}\mathtt{A}, \tag{2.3}$$

$${}^{r}\mathtt{A} = {}^{r}\mathtt{A}_{\mathtt{exa}} \cup {}^{r}\mathtt{A}_{\mathtt{da}} \cup {}^{r}\mathtt{A}_{\mathtt{tha}} \cup {}^{r}\mathtt{A}_{\mathtt{cla}} \cup {}^{r}\mathtt{A}_{\mathtt{tra}}. \tag{2.4}$$

An agent is designated as ${}^{w}\mathrm{a}_{\mathrm{j}}$, where $w \in (\mathtt{R} \cup \{s\})$, and j is a unique identifier of the agent (e.g., ${}^{r1}\mathrm{a}_{12}$). 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. Agents consist of subsystems of various classes (control subsystem, virtual and real receptors and effectors). This dissertation describes thoroughly only the two agent classes ($\mathtt{da}$, $\mathtt{tha}$) that are CT type, and they consist of a single control subsystem only. Therefore,

[1] new agent classes that are introduced in this dissertation in Section 3

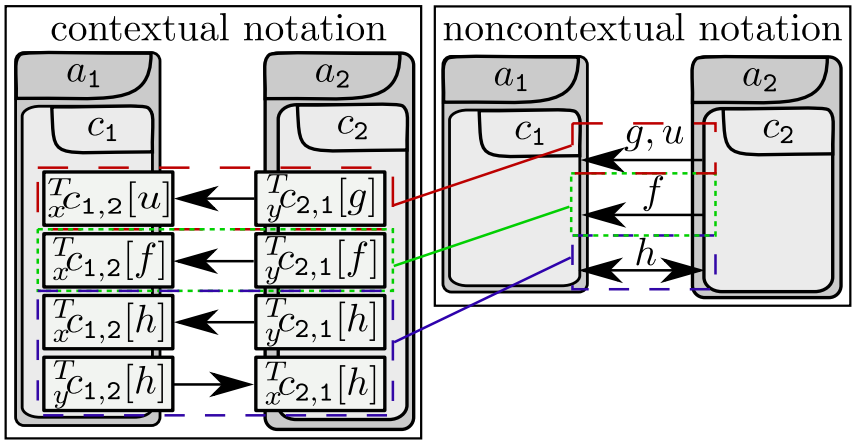


Figure 2.5: The contextual and corresponding non-contextual notation of an inter-agent communication, where c_α is a control subsystem of a_α.

in this work, the subsystem identifiers are omitted in the notation. The other agent classes appear only in the model decomposition in the inter-agent communication specification of `da` and `tha` agent classes. Subsystems communicate via communication buffers, and the buffers have at least one field for data. In this work, only the inter-agent communication is specified, and the inter-subsystem is out of its scope. Therefore, there are **input buffers**—$\overset{T}{x}\mathrm{c}_{j,g}[u]$ and **output buffers**—$\overset{T}{y}\mathrm{c}_{j,g}[u]$, where u is an identifier of the buffer field, 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. 2.5. Subsystems can store data in their **internal memory**. The internal memory of a control subsystem is denoted as $\mathfrak{c}_j$.

Management of an agent behaviour is defined by a **Finite-State Machine**, which is denoted as FSM_j. The finite-state machine is composed of states, which for a specified FSM_j, are denoted as 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 the super state S_j^s is denoted as $\mathrm{FSM}_{j,s}$. A **basic behaviour** (denoted as $\mathcal{B}_j^b$) defines the operation of a single-subsystem agent j, where b is an identifier of the basic behaviour. The basic behaviour $\mathcal{B}_j^b$ is assigned to S_j^s if $b = s$. The model of the basic behaviour is derived from the universal architectural pattern and specification method [38] and is shown in Fig. 2.6. **Terminal conditions** are logic functions assigned to the basic behaviours and define an event that terminates the basic behaviour. They are denoted as $\mathrm{tc}_{j,b}$. The **transition function** f_j^k processes data from input buffers and saves it to output buffers, where k is an identifier of the function. The transition function f_j^k is computed in the basic behaviour $\mathcal{B}_j^b$ if $k = b$. Transition functions can be divided into **primitive transition functions**; such a function is denoted as pf_j^p, where p is its identifier. Transitions between states that compose an FSM are triggered by logic functions called

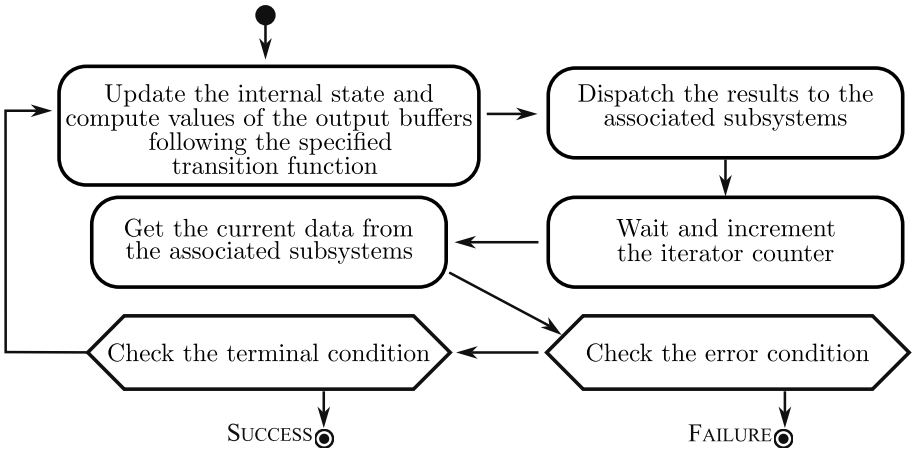


Figure 2.6: The model of a basic behaviour derived from [38].

Table 2.2: Symbols and elements derived from Embodied Agent Meta-model reduced to CT type agent specification. The indexes designate: α – the agent's scope (name of a robot associated with an agent or s for system-wide agents), ag – an agent's name, $interlocutor$ – a name of an agent connected to a buffer of the agent ag, $field$ – a name of a field of a buffer or memory, $superState$ – an identifier of a super state, id – an identifier of a specification element.

Agent	Input Buffer	Memory	Initial Condition
${}^{\alpha}\mathtt{a}_{ag}$	${}^{T}_{x}c_{ag,interlocutor}[field]$	${}^{c}c_{ag}$	$ic_{ag,id}$
	Output Buffer	Memory Field	Terminal Condition
	${}^{T}_{y}c_{ag,interlocutor}[field]$	${}^{c}c_{ag}[field]$	$tc_{ag,id}$
	Transition Function	FSM	State
	f^{id}_{ag}	FSM_{ag}	$\mathtt{S}^{id}_{ag}$
	Primitive Transition Function	FSM of Super State	Basic behaviour
	pf^{id}_{ag}	$FSM_{ag,superState}$	$\mathcal{B}^{id}_{ag}$

initial conditions denoted as $\mathrm{ic}_{j,o}$, where o is an identifier of the initial condition. The model of basic behaviours and their composition in the FSM results with the following control flow. The subsequent basic behaviour (e.g. $\mathcal{B}^2_j$) is executed if the terminal condition of the current basic behaviour (e.g. $\mathcal{B}^1_j$) is satisfied and if the initial condition between the states $\mathtt{S}^1_j$ and $\mathtt{S}^2_j$ is $True$. The notation elements' and their symbols are shown collectively in Tab. 2.2.

Chapter 3

The robot system model enabling prudent task management

In this chapter, the model of the system satisfying the stated requirements is presented. It is defined with the use of the formal notation and follows the Embodied Agent Meta-model. The proposed model is holistically specified on the agent layer; however, the three agent classes introduced in this work are thoroughly described according to their internal structures and behaviours. The general structure of the model is described roughly in Sec. 3.1 and the agents composing it are introduced in Sec. 3.2, Sec. 3.3 and Sec. 3.4. Subsequently, in Sec. 3.5 an analysis of the system's behaviour is presented.

3.1 The system structure

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

To satisfy the requirements of this study, there are additional agent classes introduced, which were not considered in the RAPP system—**`task harmoniser`** (`tha` class) and **`executor agent`** (`exa` class) and **`task requester`** (`tra` class). There is one `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 `tha`-class agent. The algorithm uses either **task-context dependent** or **system-wide** schedule parameters. The former ones are computed by ${}^{r}\mathtt{A}_{\mathtt{da}}$ agents, and the other are computed by the scheduling algorithm. 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. Additionally, each robot in the system is controlled by exa-class agent. It is responsible for the robot hardware management. It serves robot's core functionality; it shares data from the robot's sensors and exposes an interface to call the robot actions. In Fig. 3.1, the agent classes and services they provide and consume are presented.

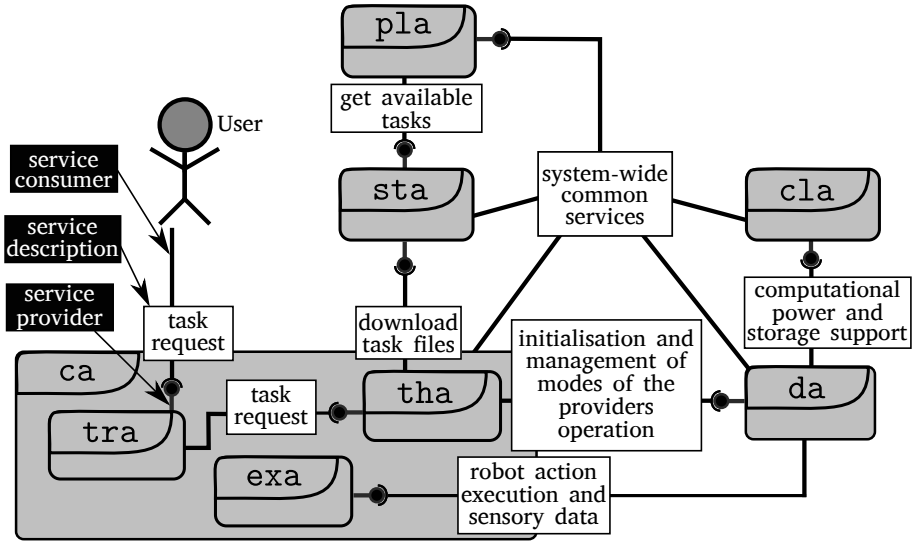


Figure 3.1: Cooperation of the agent classes introduced in RAPP: pla, sta, ca, cla, da (Fig. 1.4) modified by the proposed decomposition of ca to tra, tha and exa classes.

Implementations of the TaskER model may involve multiple tra-class agents composed of different subsystems that play the roles of various interfaces (e.g., a home automation system, a smartphone, or a human-robot interface). From this study 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 3.3) and the tra-class agent section (Section 3.4).

Based on the RAPP architecture and descriptions of tha-class and tra-class agents, the cardinalities of the agent sets[1] composing the system are stated:

$$|{}^{s}\mathtt{A}_{\mathtt{exa}}| = |{}^{s}\mathtt{A}_{\mathtt{tha}}| = |{}^{s}\mathtt{A}_{\mathtt{da}}| = \emptyset, \tag{3.1}$$

$$|{}^{s}\mathtt{A}_{\mathtt{pla}}| = |{}^{s}\mathtt{A}_{\mathtt{sta}}| = 1, \tag{3.2}$$

$${}^{\mathtt{R}}\mathtt{A}_{\mathtt{pla}} = {}^{\mathtt{R}}\mathtt{A}_{\mathtt{sta}} = \emptyset, \tag{3.3}$$

$$\forall r \in \mathtt{R} : |{}^{r}\mathtt{A}_{\mathtt{exa}}| = |{}^{r}\mathtt{A}_{\mathtt{tha}}| = 1 \wedge |{}^{r}\mathtt{A}_{\mathtt{cla}}| \leq |{}^{r}\mathtt{A}_{\mathtt{da}}|, \tag{3.4}$$

$$\forall r \in \mathtt{R} : |{}^{r}\mathtt{A}_{\mathtt{tra}}| = 0 \vee |{}^{r}\mathtt{A}_{\mathtt{tra}}| = 1. \tag{3.5}$$

[1]The glossary at the beginning of the dissertation introduces the symbols used to denote the agent sets.

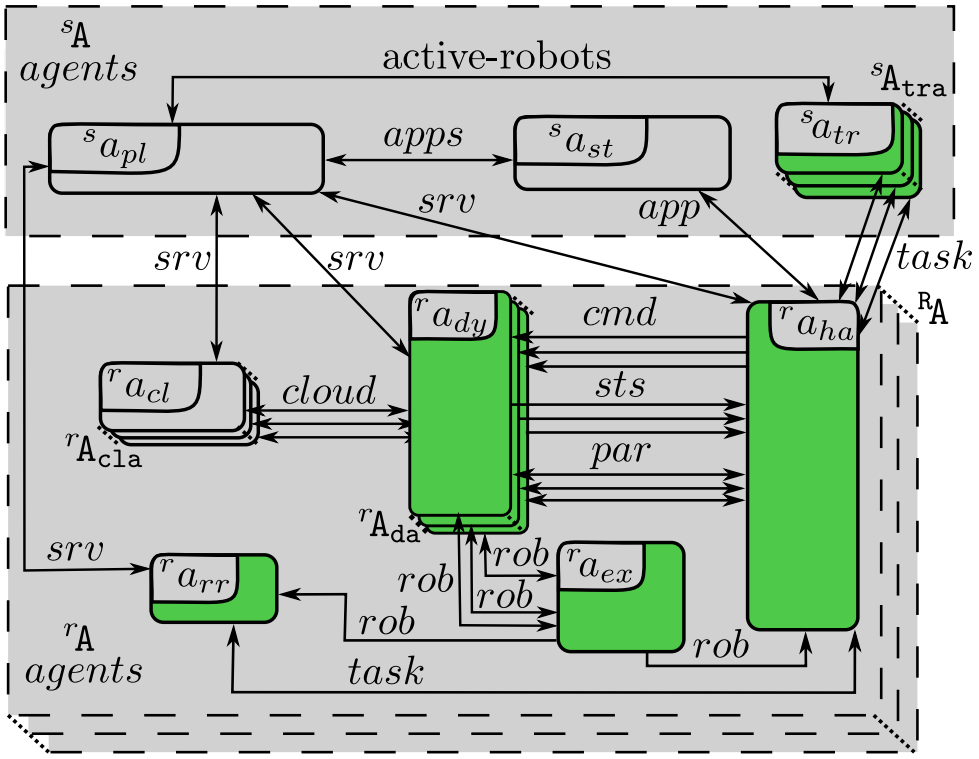


Figure 3.2: The model of a system that is composed of R set of robots. The model contains the following agents that are associated with the example robot $r \in \mathtt{R}$: ${}^{r}\mathtt{a}_{\mathtt{dy}} \in {}^{r}\mathtt{A}_{\mathtt{da}}$; ${}^{r}\mathtt{a}_{\mathtt{ha}}$ that is a tha-class agent of the robot r; ${}^{r}\mathtt{a}_{\mathtt{ex}}$ that is of exa class; ${}^{r}\mathtt{a}_{\mathtt{rr}} \in {}^{r}\mathtt{A}_{\mathtt{tra}}$ that is of tra-class agent associated with the robot r; and ${}^{r}\mathtt{A}_{\mathtt{cla}}$ that is a set of cla-class agents that are associated with the agents of ${}^{r}\mathtt{A}_{\mathtt{da}}$. The model consists of system-wide agents that communicate with all the robots in the system: ${}^{s}\mathtt{a}_{\mathtt{pl}}$ of pla class, ${}^{s}\mathtt{a}_{\mathtt{st}}$ of sta class, and agents of ${}^{s}\mathtt{A}_{\mathtt{tra}}$ set.

A tra-class agent can be associated with either a specified robot or be system-wide. Following (3.5), each robot can have one tra-class agent associated with it or none. If there is one, it is denoted as ${}^{\alpha}\mathtt{a}_{\mathtt{rr}}$, where $\alpha \in \mathtt{R}$. Such an agent is used in systems that enable robots to collect user requests with their receptors.

In Fig. 3.2, the general structure of the TaskER model is presented. It shows a cross-section of a multi-robot system revealing in details structure in the layer of the robot named r. The agent sets compose agents shared between robots and agents controlling the robot. Communication buffers are presented in the non-contextual notation that is presented in Fig. 2.5. The four green-marked agents are the primary concern of this dissertation as their responsibility is associated with the task harmonisation. The other agents support the marked ones with cloud computing, storage and system-common services.

One of the model's crucial constraints is the independent management of the tasks delegated to a specified robot. Therefore, in the next sections, the following instances of the agent classes and cardinalities of the agent sets are assumed:

$$|\mathtt{R}| = 1 \Longleftrightarrow |{}^{\mathtt{R}}\mathtt{A}_{\mathtt{exa}}| = 1 \wedge |{}^{\mathtt{R}}\mathtt{A}_{\mathtt{tha}}| = 1, \tag{3.6}$$

$$\mathtt{R} = \{\mathtt{r}\},\ {}^{r}\mathtt{A}_{\mathtt{exa}} = \{{}^{r}\mathtt{a}_{\mathrm{ex}}\},\ {}^{r}\mathtt{A}_{\mathtt{tha}} = \{{}^{r}\mathtt{a}_{\mathrm{ha}}\},\ {}^{r}\mathtt{A}_{\mathtt{tra}} = \emptyset, \tag{3.7}$$

$${}^{s}\mathtt{A}_{\mathtt{tra}} = \{{}^{s}\mathtt{a}_{\mathrm{tr}}\},\ {}^{s}\mathtt{A}_{\mathtt{pla}} = \{{}^{s}\mathtt{a}_{\mathrm{pl}}\},\ {}^{s}\mathtt{A}_{\mathtt{sta}} = \{{}^{s}\mathtt{a}_{\mathrm{st}}\}. \tag{3.8}$$

3.2 The Dynamic Agent class

A `da` class plays a task bearer role and conducts basic behaviours required for the task completion, along with additional basic behaviours that are defined in TaskER to satisfy the requirements of the study. Moreover, TaskER specifies the structure of `da`-class agents, their FSMs and communication buffers. The model of `da`-class agents is demonstrated with an example agent—${}^{r}\mathtt{a}_{\mathrm{dy}}$ that is associated with the robot r (${}^{r}\mathtt{a}_{\mathrm{dy}} \in {}^{r}\mathtt{A}_{\mathtt{da}}$). In the following considerations, the conditions (3.6)-(3.8) are satisfied, and ${}^{r}\mathtt{a}_{\mathrm{cl}}$ supports ${}^{r}\mathtt{a}_{\mathrm{dy}}$ (3.9):

$${}^{r}\mathtt{a}_{\mathrm{dy}} \in {}^{r}\mathtt{A}_{\mathtt{da}},\ {}^{r}\mathtt{A}_{\mathtt{cla}} = \{{}^{r}\mathtt{a}_{\mathrm{cl}}\},\ {}^{r}\mathtt{a}_{\mathrm{cl}} \sim {}^{r}\mathtt{a}_{\mathrm{dy}}, \tag{3.9}$$

where '$\sim$' denotes the support relation between a `cla`-class agent and a `da`-class agent.

3.2.1 The buffers of `da`-class agents

Each agent of `da` class has 10 buffers. Visualisation of the buffers in the non-contextual notation, which is used in the following description, is shown in Fig. 3.3. The first pair of buffers, namely, ${}^{T}_{x}\mathtt{c}_{dy,ex}[rob]$ and ${}^{T}_{y}\mathtt{c}_{dy,ex}[rob]$, is used to communicate with ${}^{r}\mathtt{a}_{\mathrm{ex}}$ to:

1. command the robot (${}^{T}_{y}\mathtt{c}_{dy,ex}[rob]$),
2. receive information describing the robot and its environment states (${}^{T}_{x}\mathtt{c}_{dy,ex}[rob]$).

The second pair, namely, ${}^{T}_{x}\mathtt{c}_{dy,cl}[cloud]$ and ${}^{T}_{y}\mathtt{c}_{dy,cl}[cloud]$, is used to communicate with ${}^{r}\mathtt{a}_{\mathrm{cl}}$, e.g., to request complex computation services or algorithms that require user data that are stored in the cloud.

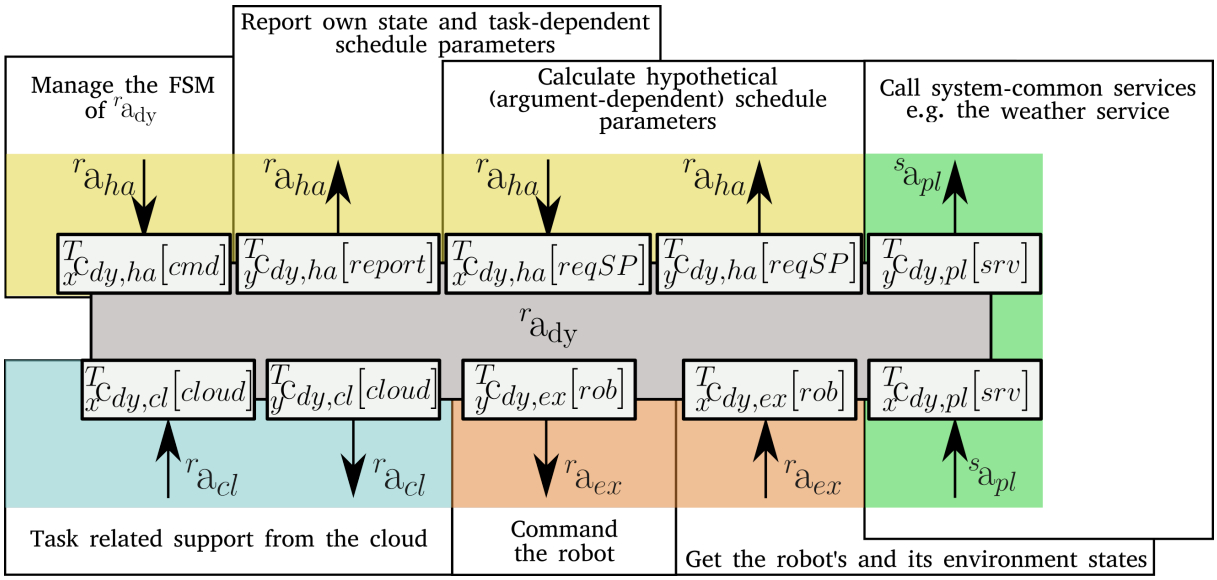


Figure 3.3: The buffers of da-class agents.

Transitions of FSM$_{dy}$ (Fig. 3.4a) are triggered based on the value of $^{T}_{x}c_{dy,ha}[cmd]$ buffer (3.10), 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 the proposed model:

$$^{T}_{x}c_{dy,ha}[cmd] = [triggerFlag, data], \tag{3.10}$$

$$triggerFlag \in \{start, susp, term\}. \tag{3.11}$$

If the $triggerFlag$ value consists of an identifier of an initial condition that is defined in FSM$_{dy}$, then the initial condition is triggered (3.12)-(3.14):

$$^{T}_{x}c_{dy,ha}[cmd] = [start, data] \Rightarrow ic_{dy,start} = true, \tag{3.12}$$

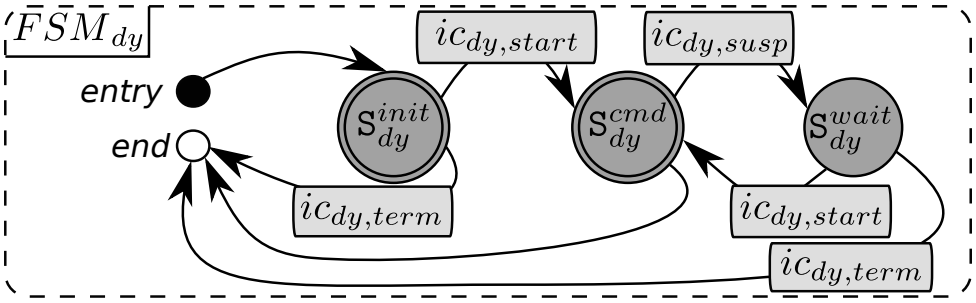
$$^{T}_{x}c_{dy,ha}[cmd] = [susp, data] \Rightarrow ic_{dy,susp} = true, \tag{3.13}$$

$$^{T}_{x}c_{dy,ha}[cmd] = [term, data] \Rightarrow ic_{dy,term} = true. \tag{3.14}$$

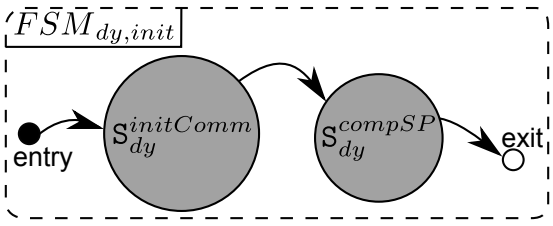
Buffer $^{T}_{y}c_{dy,ha}[report]$ transmits the $^{r}a_{dy}$ agent's report to $^{r}a_{ha}$. The report contains at least an identifier of the current state of FSM$_{dy}$:

$$^{T}_{y}c_{dy,ha}[report] = fsmState, \tag{3.15}$$

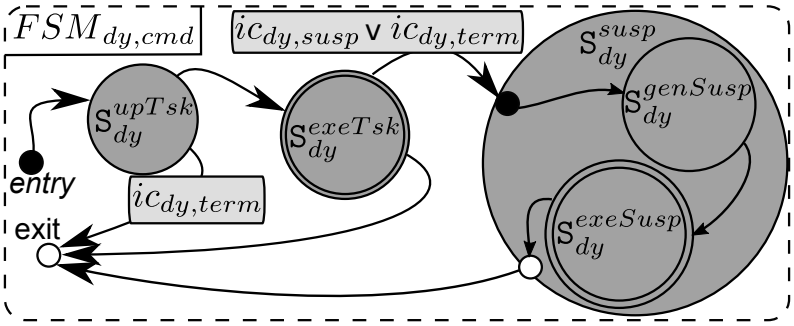
$$fsmState \in \{initComm, compSP, upTsk, exeTsk, susp, wait, end\}.$$



(a) The graph of the top level of a hierarchical FSM that governs behaviour of a da-class agent, where super states S^{init}_{dy} and S^{cmd}_{dy} are specified in Fig. 3.4b and in Fig. 3.4c, respectively.



(b) The internal FSM of S^{init}_{dy}.



(c) The internal FSM of S^{cmd}_{dy}.

Figure 3.4: The hierarchical finite-state machine governing the operation of a da class shown for an example agent ${}^{r}a_{dy}$.The initial conditions of the edges without labels are logical complements of the predicates defined by all labelled alternative edges originated from the same state.

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

$$ {}^{T}_{y}c_{dy,ha}[report] = [fsmState, scheduleParams]. \tag{3.16}$$

The structure of the $scheduleParams$ field depends on the scheduling algorithm defined for the robot (e.g., an estimated time for completing the task, time for suspending the current task, or travel distance can be used). Therefore, it is configurable and not expressed strictly in the model. They need to be defined in the process of the model implementation for given requirements.

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

$$ {}^{T}_{x}\mathtt{c}_{dy,ha}[reqSP] = [spID,\ args], \tag{3.17} $$

$$ {}^{T}_{y}\mathtt{c}_{dy,ha}[reqSP] = [scheduleParam]. \tag{3.18} $$

An example schedule parameter that is hypothetical is the estimated time for completing a task under specified conditions (e.g., an initial robot pose). The ${}^{T}_{x}\mathtt{c}_{dy,ha}[reqSP]$ and ${}^{T}_{y}\mathtt{c}_{dy,ha}[reqSP]$ buffers are used only in the systems using hypothetical schedule parameters.

Finally, a couple of buffers— ${}^{T}_{x}\mathtt{c}_{dy,pl}[srv]$ and ${}^{T}_{y}\mathtt{c}_{dy,pl}[srv]$ is used to communicate with ${}^{r}\mathtt{a}_{\mathrm{pl}}$ to call system-common services that are supplied by ${}^{r}\mathtt{a}_{\mathrm{pl}}$, such as text-to-speech or speech-to-text.

3.2.2 The FSM of `da`-class agents

A behaviour of a `da`-class agent is managed by a hierarchical FSM (HFSM). An example for ${}^{r}\mathtt{a}_{\mathrm{dy}}$—$\mathrm{FSM}_{dy}$ is shown in Fig. 3.4a. The HFSM consists of three states at the top-level:

1. $\mathtt{S}^{init}_{dy}$ – the process of ${}^{r}\mathtt{a}_{\mathrm{dy}}$ is initiated in the robot's computer. The internal FSM of this state is presented in Fig. 3.4b. ${}^{r}\mathtt{a}_{\mathrm{dy}}$ is allowed to pull data from ${}^{r}\mathtt{a}_{\mathrm{ex}}$ to compute schedule parameters and obtain addresses of remote services. However, ${}^{r}\mathtt{a}_{\mathrm{dy}}$ does not command ${}^{r}\mathtt{a}_{\mathrm{ex}}$ to execute the task that is implemented in ${}^{r}\mathtt{a}_{\mathrm{dy}}$. This state consists of two internal states:

   (a) $\mathtt{S}^{initComm}_{dy}$ – ${}^{r}\mathtt{a}_{\mathrm{dy}}$ creates communication interfaces and initialises the values of its internal memory ${}^{c}\mathtt{c}_{dy}$. The behaviour executed in this state ($\mathcal{B}^{initComm}_{dy}$) is executed once; therefore, its terminal condition equals *True*: $\mathrm{tc}_{dy,initComm} := True$;

   (b) $\mathtt{S}^{compSP}_{dy}$ – ${}^{r}\mathtt{a}_{\mathrm{dy}}$ sets the value of ${}^{T}_{y}\mathtt{c}_{dy,ha}[report]$ (3.15)-(3.16), and if the system uses task-dependent schedule parameters, then ${}^{r}\mathtt{a}_{\mathrm{dy}}$ computes these task-dependent schedule parameters. The parameters are further used by ${}^{r}\mathtt{a}_{\mathrm{ha}}$ to manage the operation of all agents from ${}^{r}\mathtt{A}_{\mathtt{da}}$. The behaviour executed in this state ($\mathcal{B}^{compSP}_{dy}$) is executed repetitively until the agent is started or terminated by ${}^{r}\mathtt{a}_{\mathrm{ha}}$; therefore, the terminal condition of $\mathcal{B}^{compSP}_{dy}$ is:

$$ \mathrm{tc}_{dy,compSP} := {}^{T}_{x}\mathtt{c}_{dy,ha}[cmd] = [start, data] \vee {}^{T}_{x}\mathtt{c}_{dy,ha}[cmd] = [term, data]. \tag{3.19} $$

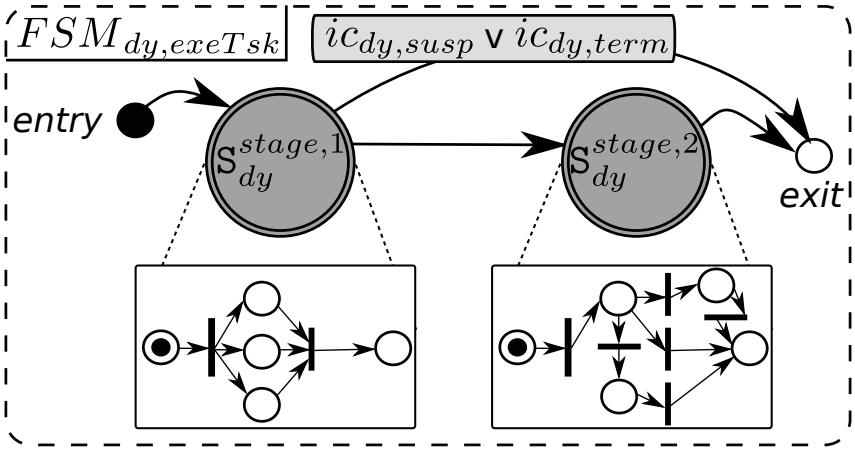


Figure 3.5: Integration of the TaskER model with example task stages given by Petri Nets.

2. S_{dy}^{cmd} – ${}^{r}a_{dy}$ commands ${}^{r}a_{ex}$ to complete its task. This state is represented by a hierarchical FSM (Fig. 3.4c), where

   (a) S_{dy}^{exeTsk} is specified by $FSM_{dy,exeTsk}$ 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 $S_{dy}^{stage,k}$, where k is an identifier of the stage. The stages can be defined by any task model (Petri Net, Behaviour tree, DSL, FSM). Only the top-level of the task must be specified as an FSM. An example $FSM_{dy,exeTsk}$ that integrates Petri Net stages with TaskER is presented in Fig. 3.5. The terminal conditions of the basic behaviours executed in this state depend on the type of a given stage and are specified in the next section.

   (b) S_{dy}^{upTsk} modifies or re-plans task defined in $FSM_{dy,exeTsk}$ to adapt it to changes in the environment that occurred when ${}^{r}a_{dy}$ was waiting for its task execution. The behaviour executed in this state ($\mathcal{B}_{dy}^{upTsk}$) is executed once; therefore, $tc_{dy,upTsk} := True$;

   (c) S_{dy}^{susp} consists of an internal FSM that is composed of two states: $S_{dy}^{genSusp}$ and $S_{dy}^{exeSusp}$. The first creates $FSM_{dy,exeSusp}$ for governing behaviours of the robot that are required for suspending the ongoing task safely. The second state—$S_{dy}^{exeSusp}$, is a super state that executes $FSM_{dy,exeSusp}$. S_{dy}^{susp} is responsible for conducting a set of actions to:

      i. set the robot and its environment in safe conditions to execute the interrupting task,

      ii. enable resumption of the current task.

3. $\mathtt{S}_{dy}^{wait}$ – ${}^{r}\mathrm{a}_{\mathrm{dy}}$ awaits resumption, sets a value of ${}_{y}^{T}\mathrm{c}_{dy,ha}[report]$ and calculates schedule parameters. The basic behaviour executed in this state ($\mathcal{B}_{dy}^{wait}$) is terminated if the agent's task is started or terminated by ${}^{r}\mathrm{a}_{\mathrm{ha}}$; therefore:

$$\mathrm{tc}_{dy,wait} := {}_{x}^{T}\mathrm{c}_{dy,ha}[cmd] = [start, data] \vee {}_{x}^{T}\mathrm{c}_{dy,ha}[cmd] = [term, data]. \tag{3.20}$$

If $|{}^{r}\mathtt{A}_{\mathtt{da}}| \geq 1$, then only one agent from the set ${}^{r}\mathtt{A}_{\mathtt{da}}$ can be in the state $\mathtt{S}_{dy}^{cmd}$ and execute its task. The other agents can be in the other states. The initial conditions of FSM_{dy}, namely, $\mathrm{ic}_{dy,start}$, $\mathrm{ic}_{dy,term}$, and $\mathrm{ic}_{dy,susp}$, are set by ${}^{r}\mathrm{a}_{\mathrm{ha}}$ via inter-agent connection; hence, ${}^{r}\mathrm{a}_{\mathrm{ha}}$ must not allow for more than one agent from ${}^{r}\mathtt{A}_{\mathtt{da}}$ to be in $\mathtt{S}_{dy}^{cmd}$. ${}^{r}\mathrm{a}_{\mathrm{dy}}$ sets only the $\mathrm{ic}_{dy,term}$ condition. It happens if the agent's task is aborted by a user or is no longer feasible.

3.2.3 The basic behaviours of `da`-class agents

The schedule parameters' computation – $\mathcal{B}_{dy}^{compSP}$ The transition function that is calculated in $\mathcal{B}_{dy}^{compSP}$, namely, f_{dy}^{compSP} consists of one primitive transition function ($\mathrm{pf}_{dy}^{compSP}$). The latter is further decomposed into multiple primitive transition functions. Each of them computes a distinct schedule parameter (e.g., the priority and the estimated time for completing the task that is implemented in ${}^{r}\mathrm{a}_{\mathrm{dy}}$). Equation (3.21) defines the composition of the primitive transition functions in f_{dy}^{compSP} and the events triggering each of them.

$$\mathrm{f}_{dy}^{compSP} = \mathrm{pf}_{dy}^{compSP} = \begin{cases} \mathrm{pf}_{dy}^{report}, & \text{if } timer({}^{c}\mathrm{c}_{dy}[stsRate]) \\ \mathrm{pf}_{dy}^{sp,1}, & \text{if } I^{1} \\ \mathrm{pf}_{dy}^{sp,2}, & \text{if } I^{2} \\ ..., & \text{if } ... \\ \mathrm{pf}_{dy}^{sp,p}, & \text{if } I^{p} \end{cases}, \tag{3.21}$$

where $\mathrm{pf}_{dy}^{report}$, with the frequency defined in the *stsRate* field of the agent's memory (${}^{c}\mathrm{c}_{dy}$), sets the $fsmState$ field of the ${}_{y}^{T}\mathrm{c}_{dy,ha}[report]$ buffer (given by (3.15) or (3.16)) and p is a number of schedule parameters that f_{dy}^{compSP} computes. If the system uses task-dependent schedule parameters, then $p > 0$; if it does not, then $p = 0$. Each of the primitive transition functions $\mathrm{pf}_{dy}^{sp,1}$-$\mathrm{pf}_{dy}^{sp,p}$ fills a part of the $scheduleParams$ field (3.16), which the function calculates. The symbol I^{id} denotes a logic sentence that activates the $\mathrm{pf}_{dy}^{sp,id}$ primitive transition function. I^{id} can be related to the ${}_{x}^{T}\mathrm{c}_{dy,ha}[reqSP]$ buffer; hence, $\mathrm{pf}_{dy}^{sp,id}$ will be triggered upon a request from ${}^{r}\mathrm{a}_{\mathrm{ha}}$ in consideration of (3.17):

$$I^{id} : {}_{x}^{T}\mathrm{c}_{dy,ha}[reqSP] = [id,\ args] \tag{3.22}$$

An example of hypothetical schedule parameters is the robot path distance in the task execution. An example argument for a hypothetical parameter can be a start position of a robot. It can be beneficial to establish the best path for the robot managing multiple transportation tasks. For this example $id = 1$, $args = robotPosition$.

The task update – $\mathcal{B}_{dy}^{upTsk}$ The transition function of $\mathcal{B}_{dy}^{upTsk}$ (namely, f_{dy}^{upTsk}) creates $\mathrm{FSM}_{dy,exeTsk}$. Example methods can configure of a statically defined template of an FSM or create an entirely new FSM using planning algorithms.

The task execution – $\mathrm{FSM}_{dy,exeTsk}$ The transition function $\mathrm{f}_{dy}^{stage,k}$ and the terminal condition $\mathrm{tc}_{dy,stage,k}$ define the basic behaviour $\mathcal{B}_{dy}^{stage,k}$ of the k^{th} task stage. Each $\mathrm{f}_{dy}^{stage,k}$ is divided into two primitive transition functions:

1. $\mathrm{pf}_{dy}^{stage,k}$ – commands ${}^{r}\mathrm{a}_{\mathrm{ex}}$ to achieve an objective of the k^{th} stage (e.g., reaching a specified destination),
2. $\mathrm{pf}_{dy}^{compSP}$ – computes schedule parameters and updates ${}_{y}^{T}\mathrm{c}_{dy,ha}[report]$ during $\mathtt{S}_{dy}^{exeTsk}$. It was formally defined in (3.21). The schedule parameters are used by ${}^{r}\mathrm{a}_{\mathrm{ha}}$ to schedule ${}^{r}\mathtt{A}_{\mathtt{da}}$ agents.

Therefore, the transition function is specified with the following composition:

$$\mathrm{f}_{dy}^{stage,k} = \begin{cases} \mathrm{pf}_{dy}^{stage,k} \\ \mathrm{pf}_{dy}^{compSP} \end{cases}, \tag{3.23}$$

where k is an identifier of the task stage. Some stages of a task may involve operations that may not be interrupted, e.g., due to an unstable state of the manipulated object or to an unknown suspension or resumption transition function for this stage. To prevent such a threat, the task stages are classified in the TaskER model to:

1. **suspendable** – the task may be suspended in this stage. 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 carrying out an unstable process or resume of the current task would be impossible.

The suspendable stages of a task that is conducted by ${}^{r}\mathrm{a}_{\mathrm{dy}}$ 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 terminal condition for terminating $\mathcal{B}_{dy}^{stage,k}$. The terminal 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 (3.24). A terminal condition of a suspendable stage is satisfaction of an objective condition of the stage with an alternative $\mathrm{ic}_{dy,susp}$ and $\mathrm{ic}_{dy,term}$ conditions (3.25). Assuming the objective conditions $\mathrm{tc}_{dy,stage,k,goal}$ and $\mathrm{tc}_{dy,stage,j,goal}$ of a blocking stage $\mathtt{S}_{dy}^{stage,k}$ and a suspendable stage $\mathtt{S}_{dy}^{stage,j}$:

$$\mathrm{tc}_{dy,stage,k} := \mathrm{tc}_{dy,stage,k,goal}, \tag{3.24}$$

$$\mathrm{tc}_{dy,stage,j} := \mathrm{tc}_{dy,stage,j,goal} \vee \left({}_{x}^{T}\mathrm{c}_{dy,ha}[cmd] = [susp, data]\right) \vee \left({}_{x}^{T}\mathrm{c}_{dy,ha}[cmd] = [term, data]\right). \tag{3.25}$$

Thus, if ${}^{r}\mathrm{a}_{\mathrm{ha}}$ triggers $\mathrm{ic}_{dy,susp}$ or $\mathrm{ic}_{dy,term}$ condition during the operation of $\mathcal{B}_{dy}^{stage,j}$, then the basic behaviour is terminated and $\mathtt{S}_{dy}^{stage,j}$ is switched to $\mathtt{S}_{dy}^{susp}$.

The task suspension generation – $\mathcal{B}_{dy}^{genSusp}$ The transition function $\mathrm{f}_{dy}^{genSusp}$ of $\mathcal{B}_{dy}^{genSusp}$ creates the FSM (namely, $\mathrm{FSM}_{dy,exeSusp}$) that utilises behaviours of the agent and inter-agent communication to suspend the ongoing task safely (e.g., commands for ${}^{r}\mathrm{a}_{\mathrm{ex}}$). The transition function $\mathrm{f}_{dy}^{genSusp}$ 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 to define $\mathrm{f}_{dy}^{genSusp}$ is to provide a planner that considers the present state of the world and creates $\mathrm{FSM}_{dy,exeSusp}$.

The task suspension execution – $\mathrm{FSM}_{dy,exeSusp}$ Operation of ${}^{r}\mathrm{a}_{\mathrm{dy}}$ in $\mathtt{S}_{dy}^{exeSusp}$ is specified with the internal FSM of the state, namely, $\mathrm{FSM}_{dy,exeSusp}$, which was defined in the previous state.

Waiting for the task execution – $\mathcal{B}_{dy}^{wait}$ The transition function of $\mathcal{B}_{dy}^{wait}$ (denoted as f_{dy}^{wait}) is identical with f_{dy}^{compSP} that is defined in (3.21):

$$\mathrm{f}_{dy}^{wait} := \mathrm{f}_{dy}^{compSP}. \tag{3.26}$$

3.3 The Task Harmoniser Agent class

Agents of `tha` class have a similar role in TaskER model to schedulers in operating systems. They assign tasks for execution and manage the tasks' modes of operation. In this analysis ${}^{r}\mathrm{a}_{\mathrm{ha}}$ is used as an example `tha`-class agent. Agents of this class receive and process requests for new tasks from `tra`-class agents and schedule tasks assigned to the robot associated with the `tha`-class agent. The ${}^{r}\mathrm{a}_{\mathrm{ha}}$ agent switches the states of the ${}^{r}\mathtt{A}_{\mathtt{da}}$ set to:

Table 3.1: Initial values of ${}^{c}\mathrm{c}_{ha}$ fields that are set by f_{ha}^{init}, where DAID is an identifier of a da-class agent, and x is a value that is set by an individual system's developer.

${}^{c}\mathrm{c}_{ha}$ **field**	**example initial value**	**type/unit**
${}^{c}\mathrm{c}_{ha}[sFreq]$	x	float/Hz
${}^{c}\mathrm{c}_{ha}[idleDA]$	empty	array[N]
${}^{c}\mathrm{c}_{ha}[exeDA]$	empty	DAID
${}^{c}\mathrm{c}_{ha}[irrDA]$	empty	DAID

1. optimise a specified scheduling criterion (e.g., the highest priority first, the shortest job first, or the shortest remaining time first);

2. enable a controlled suspension and resumption of the tasks that are implemented in da-class agents.

3.3.1 The buffers and memory of ${}^{r}\mathrm{a}_{\mathrm{ha}}$

The memory of ${}^{r}\mathrm{a}_{\mathrm{ha}}$ (namely, ${}^{c}\mathrm{c}_{ha}$) consists of multiple named fields; Tab. 3.1 presents a minimal set of them. ${}^{r}\mathtt{A}_{\mathtt{da}}$ agents are scheduled by ${}^{r}\mathrm{a}_{\mathrm{ha}}$ by assignment of ${}^{r}\mathtt{A}_{\mathtt{da}}$ agents' identifiers to ${}^{c}\mathrm{c}_{ha}[exeDA]$, ${}^{c}\mathrm{c}_{ha}[irrDA]$ and ${}^{c}\mathrm{c}_{ha}[idleDA]$. The first field stores an identifier of an agent of the set ${}^{r}\mathtt{A}_{\mathtt{da}}$ that is currently in $\mathtt{S}_{dy}^{cmd}$ and executes its task. Such an agent is denoted with the symbol ${}^{r}\mathrm{a}_{\mathrm{exeDA}}$. The field ${}^{c}\mathrm{c}_{ha}[irrDA]$ stores an identifier of an agent of the set ${}^{r}\mathtt{A}_{\mathtt{da}}$ that was chosen by a scheduling algorithm to replace ${}^{r}\mathrm{a}_{\mathrm{exeDA}}$. This agent is referred to by the symbol ${}^{r}\mathrm{a}_{\mathrm{irrDA}}$. ${}^{c}\mathrm{c}_{ha}[idleDA]$ is a one-dimensional array of size N that is filled with identifiers of the agents that belong to the set $\mathtt{D}_r$:

$$\mathtt{D}_r = {}^{r}\mathtt{A}_{\mathtt{da}} \setminus \left\{ {}^{r}\mathrm{a}_{\mathrm{exeDA}}, {}^{r}\mathrm{a}_{\mathrm{irrDA}} \right\}, \tag{3.27}$$

$$|\mathtt{D}_r| = \mathtt{N}. \tag{3.28}$$

In Fig. 3.6 the visualisation of all tha-class agent buffers is shown.

Management of the operation of an example ${}^{r}\mathrm{a}_{\mathrm{dy}} \in {}^{r}\mathtt{A}_{\mathtt{da}}$ is executed by sending state switch requests via the buffer ${}_{y}^{T}\mathrm{c}_{ha,dy}[cmd]$ to ${}_{x}^{T}\mathrm{c}_{dy,ha}[cmd]$. A state switch request consists of one of the values that is defined in (3.11). As the cardinality of ${}^{r}\mathtt{A}_{\mathtt{da}}$ may exceed 1, there is a set that contains buffers for managing the states of ${}^{r}\mathtt{A}_{\mathtt{da}}$ agents:

$$\begin{gathered} {}_{y}^{T}\mathrm{c}_{ha,dy}[cmd] \in {}_{y}\mathrm{bs}_{ha,{}^{r}\mathtt{A}_{\mathtt{da}}}[cmd] \iff {}^{r}\mathrm{a}_{\mathrm{dy}} \in {}^{r}\mathtt{A}_{\mathtt{da}}, \\ |{}_{y}\mathrm{bs}_{ha,{}^{r}\mathtt{A}_{\mathtt{da}}}[cmd]| = |{}^{r}\mathtt{A}_{\mathtt{da}}|. \end{gathered} \tag{3.29}$$

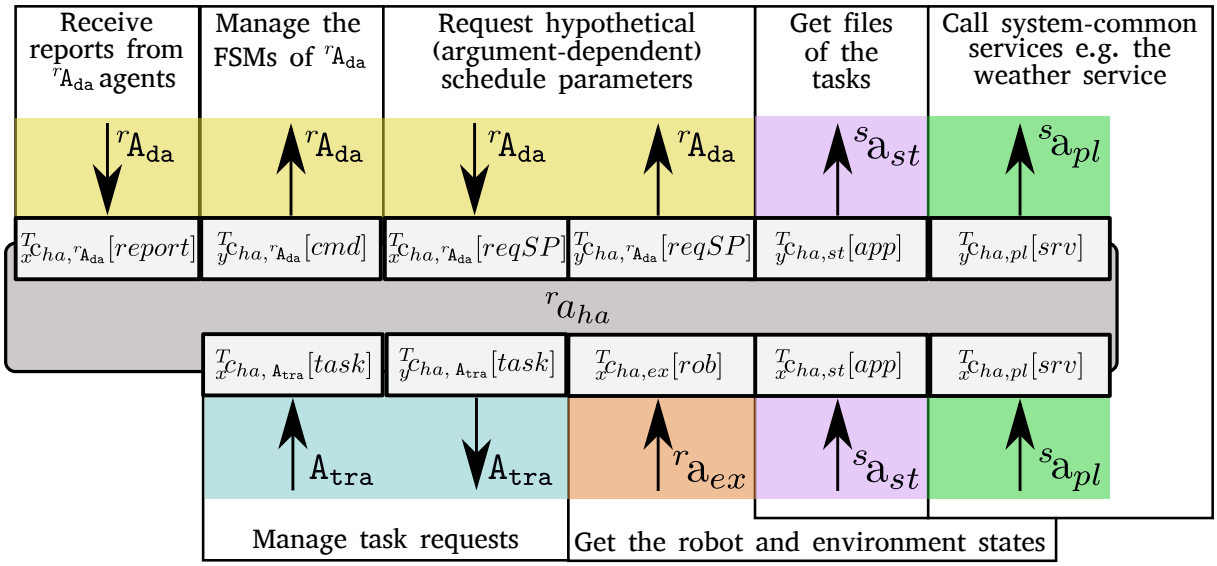


Figure 3.6: The buffers of the agents of tha class.

The buffers ${}^{T}_{x}\mathrm{c}_{ha,tr}[task]$ and ${}^{T}_{y}\mathrm{c}_{ha,tr}[task]$ are an interface (denoted as $taskIF_{ha,tr}$) for the ${}^{s}\mathrm{a}_{\mathrm{tr}}$ agent 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 and output buffers that connect ${}^{r}\mathrm{a}_{\mathrm{ha}}$ with all tra-class agents in the system:

$$taskIF_{ha,tr} := \{{}^{T}_{x}\mathrm{c}_{ha,tr}[task], {}^{T}_{y}\mathrm{c}_{ha,tr}[task]\}, \tag{3.30}$$

$$taskIF_{ha,tr} \in \mathrm{bs}_{ha,\mathtt{A}_{\mathtt{tra}}}[task] \Leftrightarrow \mathrm{a}_{\mathrm{tr}} \in ({}^{r}\mathtt{A}_{\mathtt{tra}} \cup {}^{s}\mathtt{A}_{\mathtt{tra}}), \tag{3.31}$$

where $\mathrm{bs}_{ha,\mathtt{A}_{\mathtt{tra}}}[task]$ is the set of the task buffers of ${}^{r}\mathrm{a}_{\mathrm{ha}}$ for receiving new task requests and responding to them. Moreover, the model supplies ${}^{r}\mathrm{a}_{\mathrm{ha}}$ with two sets of buffers, namely, ${}_{x}\mathrm{bs}_{ha,{}^{r}\mathtt{A}_{\mathtt{da}}}[report]$ and $\mathrm{bs}_{ha,{}^{r}\mathtt{A}_{\mathtt{da}}}[reqSP]$, for collecting status and task-dependent schedule parameters from the ${}^{r}\mathtt{A}_{\mathtt{da}}$ agents:

$${}^{T}_{x}\mathrm{c}_{ha,dy}[report] \in {}_{x}\mathrm{bs}_{ha,{}^{r}\mathtt{A}_{\mathtt{da}}}[report] \Leftrightarrow {}^{r}\mathrm{a}_{\mathrm{dy}} \in {}^{r}\mathtt{A}_{\mathtt{da}}, \tag{3.32}$$

$$\{{}^{T}_{x}\mathrm{c}_{ha,dy}[reqSP], {}^{T}_{y}\mathrm{c}_{ha,dy}[reqSP]\} \in \mathrm{bs}_{ha,{}^{r}\mathtt{A}_{\mathtt{da}}}[reqSP] \Leftrightarrow {}^{r}\mathrm{a}_{\mathrm{dy}} \in {}^{r}\mathtt{A}_{\mathtt{da}}. \tag{3.33}$$

The buffers of these sets are used to:

1. ${}_{x}\mathrm{bs}_{ha,{}^{r}\mathtt{A}_{\mathtt{da}}}[report]$ – collect the current states of the ${}^{r}\mathtt{A}_{\mathtt{da}}$ agents and their schedule parameters (3.15), (3.16);

2. $\mathrm{bs}_{ha,{}^{r}\mathtt{A}_{\mathtt{da}}}[reqSP]$ – request and receive hypothetical schedule parameters from the ${}^{r}\mathtt{A}_{\mathtt{da}}$ agents.

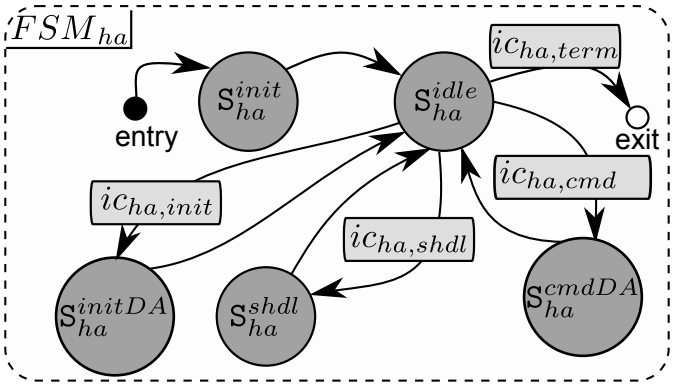


Figure 3.7: Finite state machine that governs the behaviour of ${}^{r}\mathrm{a}_{\mathrm{ha}}$.

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

Finally, there is the input buffer ${}^{T}_{x}\mathrm{c}_{ha,ex}[rob]$ that can be used by the scheduling algorithm implemented in ${}^{r}\mathrm{a}_{\mathrm{ha}}$ to obtain current states of the robot and its environment.

3.3.2 The basic behaviours of `tha`-class agents

Management of ${}^{r}\mathrm{a}_{\mathrm{ha}}$ basic behaviours is specified by the FSM shown in Fig. 3.7. The predicates that define the initial conditions of the FSM are given by (3.34)-(3.38).

$$\begin{aligned}
\mathrm{ic}_{ha,init} :=& \big(\exists x \in \mathrm{bs}_{ha,\mathtt{A_{tra}}}[task] : newData(x) = True\big) \\
& \wedge \neg \mathrm{ic}_{ha,term}, && (3.34)\\
\mathrm{ic}_{ha,shdl} :=& \big(timer({}^{c}\mathrm{c}_{ha}[sFreq]\,) \vee shdl_{event}\big) \\
& \wedge \neg\big(\mathrm{ic}_{ha,term} \vee \mathrm{ic}_{ha,init}\big), && (3.35)\\
\mathrm{ic}_{ha,cmd} :=& \big({}^{c}\mathrm{c}_{ha}[irrDA] \neq \emptyset \vee term_{event}\big) \\
& \wedge \neg\big(\mathrm{ic}_{ha,init} \vee \mathrm{ic}_{ha,shdl} \vee \mathrm{ic}_{ha,term}\big), && (3.36)\\
shdl_{event} :=& ({}^{c}\mathrm{c}_{ha}[exeDA] = \emptyset \wedge {}^{c}\mathrm{c}_{ha}[irrDA] = \emptyset \wedge {}^{c}\mathrm{c}_{ha}[idleDA] \neq \emptyset) \\
& \vee\ newData({}_{x}\mathrm{bs}_{ha,{}^{r}\mathtt{A_{da}}}[report]), && (3.37)\\
term_{event} :=& \exists \alpha \in {}_{x}\mathrm{bs}_{ha,{}^{r}\mathtt{A_{da}}}[report] : \alpha\ = \{end\}, && (3.38)
\end{aligned}$$

where $newData(x)$ represents a trigger set by data changes in the x buffer and $timer(x)$ represents a repetitive trigger set with a frequency specified by x. Interpretations of the initial conditions are as follows:

- $\text{ic}_{ha,init}$ – is $True$, if a new task request comes and ${}^{r}\text{a}_{\text{ha}}$ is not shutting down,
- $shdl_{event}$ – is $True$, if:
  - there is no da-class agent executing its task,
  - there is no da-class agent set as candidate for execution,
  - there are da-class agents awaiting for execution,
  - any of da-class agents updates its report,
- $\text{ic}_{ha,shdl}$ – is $True$, with a given frequency or if $shdl_{event}$ is satisfied, but ${}^{r}\text{a}_{\text{ha}}$ is not terminated and $\text{ic}_{ha,init}$ is not satisfied,
- $term_{event}$ – is $True$, if any of da-class agents ended (a buffer from ${}_{x}\text{bs}_{ha,{}^{r}\text{A}_{\text{da}}}[report]$ set consists the flag end,
- $\text{ic}_{ha,cmd}$ – is $True$, if there is a candidate agent for task execution or $term_{event}$ is satisfied (a da-class agent needs to be terminated), but ${}^{r}\text{a}_{\text{ha}}$ is not terminated and $\text{ic}_{ha,init}$ and $\text{ic}_{ha,shdl}$ are not satisfied,

The FSM of ${}^{r}\text{a}_{\text{ha}}$ is composed of:

1. S_{ha}^{init} – the basic behaviour executed in this state depends on the implementation of ${}^{r}\text{a}_{\text{ha}}$ as it is responsible for the agent's memory (${}^{c}\text{c}_{ha}$) initialisation and the connection of the ${}^{r}\text{a}_{\text{ha}}$ buffers. The terminal condition of this basic behaviour is: $\text{tc}_{ha,init} := True$.
2. S_{ha}^{idle} – the basic behaviour executed in this state is composed of an empty transition function and its terminal condition given by (3.39):

$$\text{tc}_{ha,idle} := \text{ic}_{ha,term} \vee \text{ic}_{ha,init} \vee \text{ic}_{ha,shdl} \vee \text{ic}_{ha,cmd}. \tag{3.39}$$

3. S_{ha}^{initDA} – the agent reads data from ${}_{x}^{T}\text{c}_{ha,tr}[task]$, assigns an identifier to the received task (e.g., *dy*), sets up ${}^{r}\text{a}_{\text{dy}}$ and adds it to D_{r}. Finally, the process of FSM_{dy} of the ${}^{r}\text{a}_{\text{dy}}$ is initiated, and the initial arguments are passed from the requester to the process of ${}^{r}\text{a}_{\text{dy}}$. The basic behaviour executed in this state consists of the transition function given by (3.40):

$${}^{c}\text{c}_{ha}[idleDA] := \text{f}_{ha}^{initDA}\big(\text{bs}_{ha,\text{A}_{\text{tra}}}[task]\big), \tag{3.40}$$

and the terminal condition given by (3.41):

$$\text{tc}_{ha,initDA} := True. \tag{3.41}$$

4. $\mathtt{S}_{ha}^{shdl}$ – the agent executes a specified scheduling algorithm. The algorithm results with a schedule decision. The transition function that executes the algorithm takes values of the schedule parameters (received via $\mathrm{bs}_{ha,{}^{r}\mathtt{A}_{\mathtt{da}}}[report]$ and $\mathrm{bs}_{ha,{}^{r}\mathtt{A}_{\mathtt{da}}}[reqSP]$) and assigns $\mathtt{D}_r$ agents to the field ${}^{c}\mathrm{c}_{ha}[irrDA]$. The algorithm of this transition function is configurable because it depends on a specified system and its objectives; hence, it is left to be defined for a specific robot system. The transition function of the basic behaviour executed in this state is given by (3.42):

$$ {}^{c}\mathrm{c}_{ha}[irrDA] := \mathrm{f}_{ha}^{shdl}\big({}_{x}\mathrm{bs}_{ha,{}^{r}\mathtt{A}_{\mathtt{da}}}[report], \mathrm{bs}_{ha,{}^{r}\mathtt{A}_{\mathtt{da}}}[reqSP]\big), \tag{3.42} $$

and the behaviour's terminal condition is given by (3.43):

$$ \mathrm{tc}_{ha,shdl} := True. \tag{3.43} $$

Based on the reports from da-class agents and schedule parameters, the transition function chooses a candidate agent to be set as executing. If no agent is selected, the current executing agent will continue its task.

5. $\mathtt{S}_{ha}^{cmdDA}$ – the agent reads the value of the ${}^{c}\mathrm{c}_{ha}[irrDA]$ field and sets the values of the ${}_{y}\mathrm{bs}_{ha,{}^{r}\mathtt{A}_{\mathtt{da}}}[cmd]$ buffers to either suspend, start, or terminate agents from the set ${}^{r}\mathtt{A}_{\mathtt{da}}$. The basic behaviour executed in this state consists of a transition function given by (3.44):

$$ \begin{aligned} \big[{}_{y}\mathrm{bs}_{ha,{}^{r}\mathtt{A}_{\mathtt{da}}}[cmd], {}^{c}\mathrm{c}_{ha}[exeDA], {}^{c}\mathrm{c}_{ha}[irrDA], {}^{c}\mathrm{c}_{ha}[idleDA]\big] := \\ := \mathrm{f}_{ha}^{cmdDA}\big({}^{c}\mathrm{c}_{ha}[irrDA], {}^{c}\mathrm{c}_{ha}[exeDA]\big), \end{aligned} \tag{3.44} $$

and the behaviour's terminal condition is: $\mathrm{tc}_{ha,cmdDA} := True.$ The transition function based on the identifiers of the candidate, and the executing agents sends commands to these agents to switch their modes of operation (defined in their FSMs). An example algorithm of (3.44) is presented in Alg. 1.

3.4 The Executor, Cloud and Task Requester Agent classes

Whereas da-class agents are client-like entities that are responsible for the execution of the tasks that are implemented in them, agents of exa and cla classes are server-like entities that da-class agents call to complete their tasks.

An exa-class agent commands the robot's hardware. According to (3.7), the example agent of this type is ${}^{r}\mathrm{a}_{\mathrm{ex}}$. It has:

Algorithm 1: Algorithm of f_{ha}^{cmdDA}.

Result: Suspension of the task of ${}^{r}a_{exeDA}$ and start/resumption of the ${}^{r}a_{irrDA}$ task

```
1  //check if ra_exeDA executes its task
2  if xTc_ha,exeDA[report] == [exeTsk, scheduleParams] then
3  |  yTc_ha,exeDA[cmd] = [susp, data];
4  //check if ra_exeDA has suspended its task
5  else if xTc_ha,exeDA[report] == [wait, scheduleParams] then
6  |  D_r = D_r ∪ {cc_ha[exeDA]};
7  |  cc_ha[exeDA] = {};
8  //check if there is no agent assigned to ra_exeDA and there is ra_irrDA
9  if cc_ha[exeDA] == {} ∧ cc_ha[irrDA] ≠ {} then
10 |  cc_ha[exeDA] = cc_ha[irrDA];
11 |  cc_ha[irrDA] = {};
12 |  yTc_ha,exeDA[cmd] = [start, data];
13 end
14 //check if a buffer of xbs_ha,rA_da[report] consists of 'end' flag and if so, remove
     the appropriate agent from rA_da
15 if ∃ xTc_ha,dy[report] ∈ xbs_ha,rA_da[report] : xTc_ha,dy[report] == {end} then
16 |  rA_da = rA_da \ ra_dy;
17 end
```

- ${}_{y}^{T}\mathrm{c}_{ex,dy}[rob]$, ${}_{y}^{T}\mathrm{c}_{ex,rr}[rob]$, ${}_{y}^{T}\mathrm{c}_{ex,ha}[rob]$ buffers to inform other agents about the robot and its environment states,

- ${}_{x}^{T}\mathrm{c}_{ex,exeDA}[rob]$ buffer to configure the robot hardware and command the robot's effectors.

A `cla` class (for this description ${}^{r}\mathrm{a}_{\mathrm{cl}}$ is an example) awaits requests from the `da` class that is associated with it (according to (3.9), ${}^{r}\mathrm{a}_{\mathrm{cl}}$ supports ${}^{r}\mathrm{a}_{\mathrm{dy}}$), calculates the response and stores it in its buffer (in the example namely ${}_{y}^{T}\mathrm{c}_{cl,dy}[cloud]$). Implementation of `cla`-class agents and the support they supply differ among tasks and systems; hence, the structures of `cla` class buffers may differ as well. Symbolic planning is one of the example services that can be provided by `cla` class to `da` class.

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 computational agents (CT) with a control subsystem only and request new tasks, e.g. based on a timer. Agents of `tra` class may also be equipped with a physical device or a sensor that initiates a task request. As there may be many robots in the system, agents of ${}^{s}\mathtt{A}_{\mathtt{tra}}$ (${}^{s}\mathrm{a}_{\mathrm{tr}}$ is an example) must obtain a list of the robots and their addresses from the `pla` class (${}^{s}\mathrm{a}_{\mathrm{pl}}$ is an example) by the *active-robots* interface (${}_{y}^{T}\mathrm{c}_{tr,pl}[active\text{-}robots]$ and ${}_{x}^{T}\mathrm{c}_{tr,pl}[active\text{-}robots]$ are an example).

Each of the `tra`-class and `cla`-class agents has a pair of buffers to communicate with the ${}^{s}\mathrm{a}_{\mathrm{pl}}$ agent to call the system-common services that are supplied by ${}^{s}\mathrm{a}_{\mathrm{pl}}$. For the example agents, the buffers are ${}_{x}^{T}\mathrm{c}_{tr,pl}[srv]$, ${}_{y}^{T}\mathrm{c}_{tr,pl}[srv]$ and ${}_{x}^{T}\mathrm{c}_{cl,pl}[srv]$, ${}_{y}^{T}\mathrm{c}_{cl,pl}[srv]$.

3.5 The harmonisation procedure

The robot systems that utilise TaskER can harmonise their tasks and are composed of agents that belong to the defined classes. Models of the classes are presented in the previous sections; however, for a comprehensive description of the system behaviour, a depiction of the inter-agent interactions is recommended. Thus, in this section workflow of the agents in a typical task harmonisation problem is presented. The system that is considered in the scenarios consists of one robot ($\mathtt{R} = \{r\}$), one `tha`-class (${}^{r}\mathtt{A}_{\mathtt{tha}} = \{{}^{r}\mathrm{a}_{\mathrm{ha}}\}$), one `exa`-class (${}^{r}\mathtt{A}_{\mathtt{exa}} = \{{}^{r}\mathrm{a}_{\mathrm{ex}}\}$) and one `tra`-class (${}^{s}\mathtt{A}_{\mathtt{tra}} = \{{}^{s}\mathrm{a}_{\mathrm{tr}}\}$) agents. As the objective of this study is to describe the robot task harmonisation model, this section focuses on the workflow of ${}^{s}\mathrm{a}_{\mathrm{tr}}$, ${}^{r}\mathrm{a}_{\mathrm{ha}}$ and `da`-class agents, and not the other agents. Behaviours of the other agents are described in detail in [30].

An outline of the task harmonisation procedure in a system that utilises the TaskER model is presented as an activity diagram (Fig. 3.8). The complex activities that are illustrated in the rounded rectangles are described in the following subsections. The interactions between the system agents in each of the complex activities are presented in separate sequence diagrams. The harmonisation procedure is initiated by one of the following events, as marked in the activity diagram:

1. **event 1** – ${}^{r}\mathrm{a}_{\mathrm{ha}}$ receives a new task request from ${}^{s}\mathrm{a}_{\mathrm{tr}}$, therefore, the initial condition to $\mathtt{S}_{ha}^{initDA}$ is satisfied,
2. **event 2** – a trigger repetitively initiates $\mathtt{S}_{ha}^{shdl}$ (3.35),
3. **event 3** – ${}^{c}\mathrm{c}_{ha}[exeDA]$ and ${}^{c}\mathrm{c}_{ha}[irrDA]$ are empty, but there is an idle `da`-class agent or ${}^{r}\mathrm{a}_{\mathrm{ha}}$ receives a new report with schedule parameters (3.37).

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 checks the output of f_{ha}^{shdl} and is implemented as the initial condition to $\mathtt{S}_{ha}^{cmdDA}$. The latter decision node, namely, *D2*, is checked in the conditions of Alg. 1.

3.5.1 Activity 1 – launch a new Dynamic Agent

The interaction between agents in this activity is presented in Fig. 3.9. ${}^{s}\mathrm{a}_{\mathrm{tr}}$ sends a request to ${}^{r}\mathrm{a}_{\mathrm{ha}}$ via ${}^{T}_{y}\mathrm{c}_{tr,ha}[task]$ that is connected to ${}^{T}_{x}\mathrm{c}_{ha,tr}[task]$. Consequently, $\mathtt{S}_{ha}^{initDA}$ is triggered (3.34) and manages initialisation of a new Dynamic Agent, starts the program that implements the `da` class model, passes initial data to it and extends ${}^{c}\mathrm{c}_{ha}[idleDA]$ with the identifier of the new dynamic agent (e.g., *d1*). Thus, ${}^{r}\mathrm{a}_{\mathrm{d1}}$ is created, and FSM_{d1} switches to $\mathtt{S}_{d1}^{init}$, where the system-specific initialisation actions are conducted ($\mathtt{S}_{d1}^{initComm}$). Finally, ${}^{r}\mathrm{a}_{\mathrm{d1}}$ switches to $\mathtt{S}_{d1}^{compSP}$, in which it repetitively calculates its schedule parameters (given by (3.21)) and saves them in ${}^{T}_{y}\mathrm{c}_{d1,ha}[report]$.

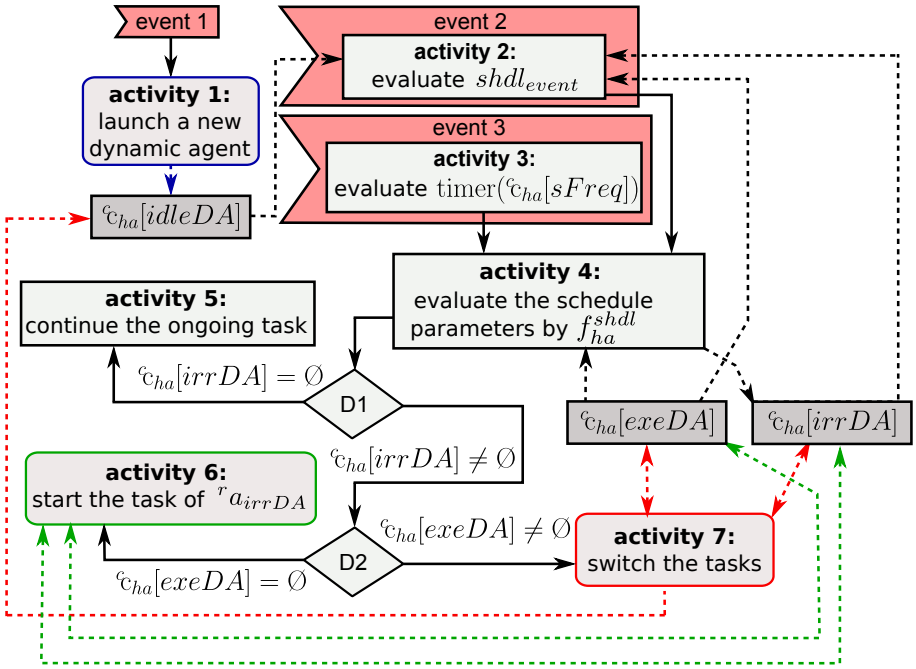


Figure 3.8: The task harmonisation procedure in the TaskER model.

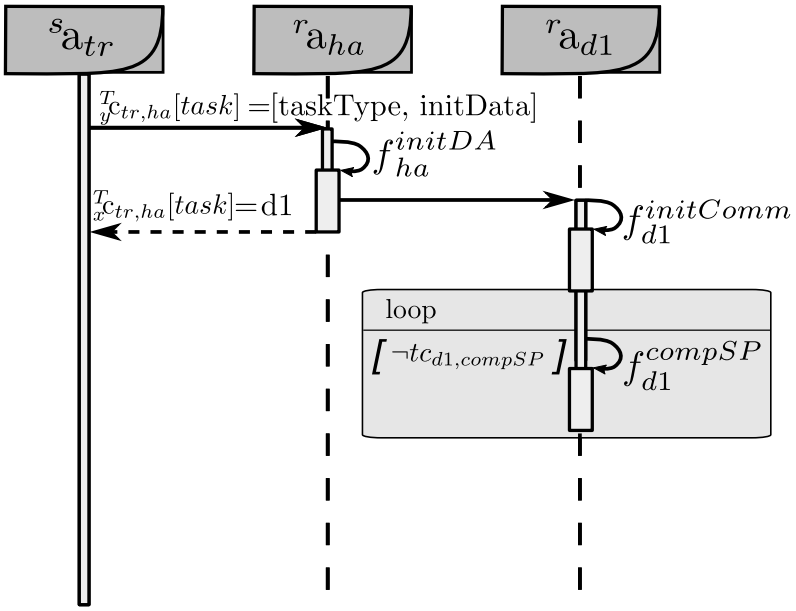


Figure 3.9: The sequence diagram of a new task request management and a da-class agent initialisation.

3.5.2 Activity 6 – start the task of $^{r}a_{irrDA}$

In this case, none of the agents in $^{r}\mathtt{A}_{\mathtt{da}}$ plays the role of $^{r}a_{ha,exeDA}$, and f_{ha}^{shdl} assigns the role of $^{r}a_{irrDA}$ to an agent from $\mathtt{D}_r$. The interaction between the agents in this activity, assuming $^{r}a_{irrDA} = {}^{r}a_{d1}$, is illustrated in Fig. 3.10.

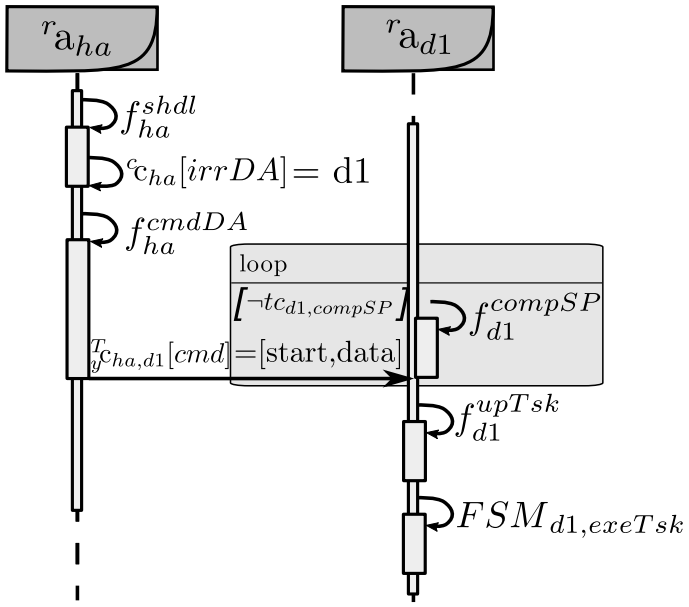


Figure 3.10: The sequence diagram of starting a da class task.

3.5.3 Activity 7 – switch the tasks

This activity is triggered only if f_{ha}^{shdl} assigns an agent from $\mathtt{D}_r$ to $^{r}\mathrm{a}_{\mathrm{irrDA}}$ and if a da-class agent executes its task. The sequence of the agent interactions in this activity is presented in Fig. 3.11. According to (3.36), the initial condition to $\mathtt{S}_{ha}^{cmdDA}$ is satisfied until the identifier of $^{r}\mathrm{a}_{\mathrm{irrDA}}$ is not removed from $^{c}\mathrm{c}_{ha}[irrDA]$ in line 11 of Alg. 1. Depending on a state of $^{r}\mathrm{a}_{\mathrm{exeDA}}$, either the condition in line 2 or in line 5 of Alg. 1 is satisfied. In the former case, $^{r}\mathrm{a}_{\mathrm{ha}}$ sends a suspension signal to $^{r}\mathrm{a}_{\mathrm{exeDA}}$, and in the latter, it removes $^{r}\mathrm{a}_{\mathrm{exeDA}}$ from $^{c}\mathrm{c}_{ha}[exeDA]$. As this activity is started only if $^{c}\mathrm{c}_{ha}[exeDA]\neq\emptyset$, first, f_{ha}^{cmdDA} will access the condition in line 2 and send the suspension signal to $^{r}\mathrm{a}_{\mathrm{exeDA}}$ ($^{T}_{y}\mathrm{c}_{ha,exeDA}[cmd]$=[susp,data]). Henceforth, two cases are considered:

1. If the ongoing stage is suspendable ($\mathtt{S}_{exeDA}^{stage,k}\in\mathtt{U}_{exeDA}$), then $\mathrm{FSM}_{exeDA,exeTsk}$ is terminated (as presented in Fig. 3.4c), and $^{r}\mathrm{a}_{\mathrm{exeDA}}$ is switched to $\mathrm{FSM}_{exeDA,susp}$. Subsequently, the agent prepares a plan for the task suspension ($\mathtt{S}_{exeDA}^{genSusp}$) and executes the plan in $\mathrm{FSM}_{exeDA,exeSusp}$. Finally, when the task is suspended, the agent switches to $\mathtt{S}_{exeDA}^{wait}$ and saves the report message to $^{T}_{y}\mathrm{c}_{exeDA,ha}[report]$ that contains the name of the current state – *wait*. While $^{r}\mathrm{a}_{\mathrm{exeDA}}$ follows the above sequence, the algorithm of f_{ha}^{cmdDA} always enters the condition in line 2 of Alg. 1. When $^{r}\mathrm{a}_{\mathrm{exeDA}}$ enters the $\mathtt{S}_{exeDA}^{wait}$ state, the condition in line 5 is satisfied, and f_{ha}^{cmdDA} moves the identifier of $^{r}\mathrm{a}_{\mathrm{exeDA}}$ from the $^{c}\mathrm{c}_{ha}[exeDA]$ field to the $\mathtt{D}_r$ set. Next, the algorithm enters the condition in line 9 as the $^{c}\mathrm{c}_{ha}[exeDA]$ field is empty. Consequently, f_{ha}^{cmdDA} moves the identifier of $^{r}\mathrm{a}_{\mathrm{irrDA}}$ from $^{c}\mathrm{c}_{ha}[irrDA]$ to the $^{c}\mathrm{c}_{ha}[exeDA]$ field and sends the start signal to $^{r}\mathrm{a}_{\mathrm{irrDA}}$.

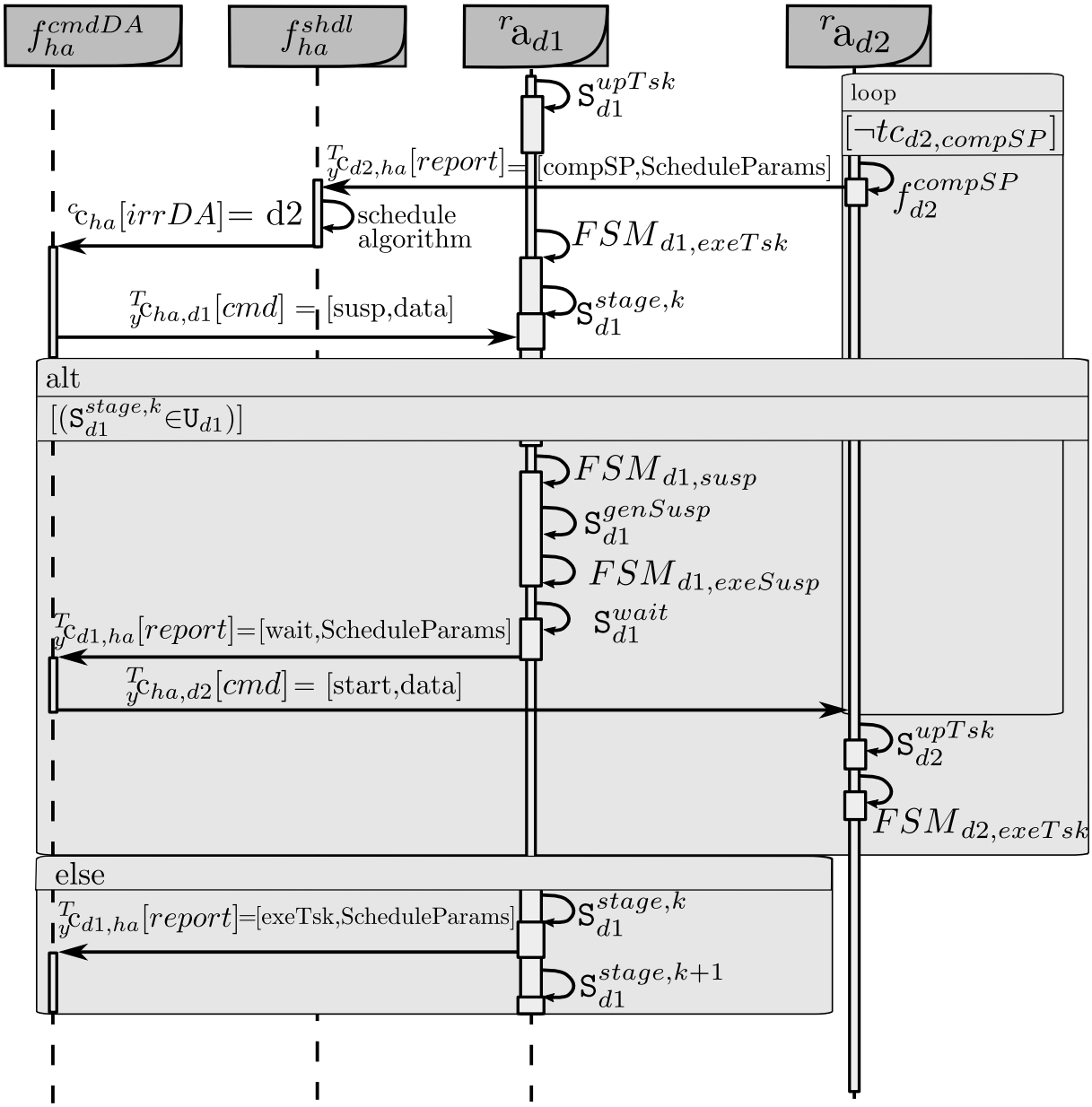


Figure 3.11: The sequence diagram that presents cooperation of the system agents while switching ${}^{r}\mathrm{a}_{\mathrm{irrDA}}$ and ${}^{r}\mathrm{a}_{\mathrm{exeDA}}$ agents (*activity 7*), where $\mathrm{s}_{d1}^{stage,k}\in \mathrm{FSM}_{d1,exeTsk}$, ${}^{r}\mathrm{a}_{\mathrm{d1}}$ is initially ${}^{r}\mathrm{a}_{\mathrm{exeDA}}$ and ${}^{r}\mathrm{a}_{\mathrm{d2}}$ becomes ${}^{r}\mathrm{a}_{\mathrm{irrDA}}$ as a result of the f_{ha}^{shdl} execution.

2. If the ongoing stage is blocking ($\mathrm{S}_{exeDA}^{stage,k} \in \mathrm{L}_{exeDA}$), then the suspension signal (${}_{x}^{T}\mathrm{c}_{exeDA,ha}[cmd]$=[susp,data]) is ignored; hence, this stage and all subsequent blocking stages will be completed without interruption. The signal is ignored because the terminal condition of a blocking stage behaviour is given by (3.25). When $\mathrm{FSM}_{exeDA,exeTsk}$ finally enters a suspendable stage, the activity will follow the sequence that is described in the first case.

3.5.4 Configuration method of the TaskER model

In the Model-Driven Engineering approach, models are parametrised, and as the result of their parameters configuration, the model becomes a system adjusted to a specific problem. The requirements for the resulting system declare the goal of the configuration. In the previous sections, the TaskER model is introduced with a set of configuration parameters. In this section, a procedure to configure the model is presented, and the effect of the parameters adjustment is shown.

Expression of a specific system's requirements in the model's formalism

There are several system-wide parameters to be set to configure the TaskER model. However, first, the system requirements need to be defined. Based on them, the following steps are needed:

1. specification of the task request interfaces and the robots composing the system need to be reflected in the sets defined in TaskER (3.6)-(3.8) and names of the agents need to be set,

2. determination of the task harmonisation type (interruptible-task or constant-task),

3. selection of schedule parameters and a scheduling algorithm,

4. classification of the schedule parameters into task-context dependent, which needs to be calculated by `da`-class agents and system-context dependent, which `tha` class agent can compute,

5. based on the above analysis, the structure of ${}_{y}^{T}\mathrm{c}_{dy,ha}[report]$ is to be defined (3.15)-(3.16) and if any task-context schedule parameters are used, complete definition of the *scheduleParams* structure is required,

6. each task-context dependent schedule parameter requires individual primitive transition function composed in $\mathrm{pf}_{dy}^{compSP}$ (3.21), so the functions are to be defined,

7. the scheduling algorithm (for ${}^{r}\mathrm{a}_{\mathrm{ha}}$ designated as f_{ha}^{shdl}) of each `tha` class agent is to be specified.

Configuration of the Dynamic Agent model

Robots in the system can execute only the tasks that are stored in ${}^{s}\mathrm{a}_{\mathrm{st}}$. The tasks need to be defined by the configuration of the model describing `da`-class agents. Therefore, for each type of the tasks, the following configuration procedure of the model is required (assuming ${}^{r}\mathrm{a}_{\mathrm{dy}}$ as an example agent):

1. **The task automata definition** – a static $\mathrm{FSM}_{dy,exeTsk}$ or a function to generate one in pf_{dy}^{upTsk} is to be specified,
2. **The task stages classification** – each stage of the automata needs to be classified either as blocking or suspendable,
3. **The task stages specification** – transition functions, error and terminal conditions for the stages need to be defined,
4. **The task update specification** – a function to update the task automata can be defined, if not, the task while resumed executes a static automaton,
5. **The task suspension specification** – a transition function of $\mathtt{S}_{dy}^{genSusp}$ is to be defined. It can involve a planning method or be a simple decision algorithm selecting basic behaviours to be assigned to $\mathtt{S}_{dy}^{exeSusp}$,
6. **The schedule parameters calculation** – $\mathrm{pf}_{dy}^{compSP}$ needs to be defined, including functions calculating task-dependent schedule parameters.

Chapter 4

Verification – implementation, specification and execution of the example systems

The model specified in the previous chapter is implemented as a framework to facilitate utilisation of this study results and to test it in the verification systems. The first system controls the TIAGo mobile robot [59], shown in Fig. 4.1a. The second system controls the Velma mobile manipulator [60], shown in Fig. 4.1b. Each of the robots operates in the Gazebo simulator and executes the tasks specified in the verification systems' requirements. Additionally, the TIAGo system is integrated with the real robot and launched. In this study case, verification in a simulated world is adequate because the work abstracts from the parts of the robot controllers that differ between the simulated and the real version of the controllers.

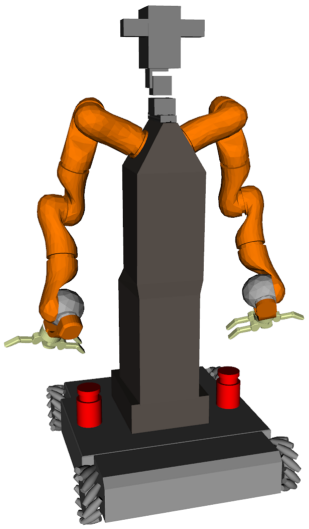
4.1 Implementation of the TaskER model

The TaskER framework delivers two Python classes, namely, `TaskHarmoniser` and `DynamicAgent`, which implement the `tha` and `da` classes, respectively. Implementation of the framework and the agents that are specific to example systems (${}^{r}\mathrm{a}_{\mathrm{ex}}$, ${}^{s}\mathrm{a}_{\mathrm{pla}}$, ${}^{s}\mathrm{a}_{\mathrm{tr}}$, and ${}^{s}\mathrm{a}_{\mathrm{st}}$), is based on the ROS framework.

Agents ${}^{r}\mathrm{a}_{\mathrm{ha}}$ and ${}^{s}\mathrm{a}_{\mathrm{tr}}$ are implemented as separate ROS nodes and are launched upon the system startup. ${}^{r}\mathrm{a}_{\mathrm{ex}}$ is robot-specific and is implemented with ROS nodes that serve common robot actions (e.g., navigation and manipulation).

Upon `TaskHarmoniser` class initialisation, a ROS node is launched. The node exposes a ROS service interface (implementing f_{ha}^{initDA}) to request new tasks and implements a sub-

(a) The TIAGo robot used in the example system launched in simulation and reality.



(b) The Velma robot used in the example mobile manipulation system.

scriber for ROS topic messages. The subscriber receives reports from ${}^{r}\mathtt{A}_{\mathtt{da}}$ agents. The interface of a `tha`-class agent for requesting tasks (${}^{T}_{x}\mathrm{c}_{ha,tr}[task]$ and ${}^{T}_{y}\mathrm{c}_{ha,tr}[task]$ buffers) is implemented as a ROS service. Upon each task request, ${}^{r}\mathrm{a}_{\mathrm{ha}}$ launches a new ROS node that implements a suitable `da`-class agent. Henceforth, the node of the `da` class will publish its report messages to the ROS topic that the `tha`-class agent subscribes. In addition to the above interfaces, the `TaskHarmoniser` class implements example transition functions namely, f_{ha}^{cmdDA} and f_{ha}^{shdl}, respectively, as presented in this dissertation (Alg. 1 and Alg. 2).

A dedicated script is implemented with the `DynamicAgent` Python class for each type of tasks in the system. The task scripts are started by a `tha`-class agent upon 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 ${}^{r}\mathrm{a}_{\mathrm{cl}} \in {}^{r}\mathtt{A}_{\mathtt{cla}}$ (if ${}^{r}\mathrm{a}_{\mathrm{dy}}$ requires support from a `cla`-class agent);

3. interfaces of ${}^{r}\mathrm{a}_{\mathrm{dy}}$ are created:

   (a) ${}^{T}_{x}\mathrm{c}_{dy,ha}[cmd]$ as a ROS topic subscriber, which sets the initial conditions $\mathrm{ic}_{dy,start}$, $\mathrm{ic}_{dy,susp}$, and $\mathrm{ic}_{dy,term}$;

   (b) ${}^{T}_{y}\mathrm{c}_{dy,ha}[report]$ as a ROS topic publisher (which is connected to ${}^{T}_{x}\mathrm{c}_{ha,dy}[report]$),

(c) ${}^{T}_{x}\mathrm{c}_{dy,ha}[reqSP]$ and ${}^{T}_{y}\mathrm{c}_{dy,ha}[reqSP]$ as a ROS service (which are connected to ${}^{T}_{y}\mathrm{c}_{ha,dy}[reqSP]$ and ${}^{T}_{x}\mathrm{c}_{ha,dy}[reqSP]$, respectively);

(d) ${}^{T}_{x}\mathrm{c}_{dy,ex}[rob]$ and ${}^{T}_{y}\mathrm{c}_{dy,ex}[rob]$ by the robot API class initialisation; and

(e) ${}^{T}_{x}\mathrm{c}_{dy,cl}[cloud]$ and ${}^{T}_{y}\mathrm{c}_{dy,cl}[cloud]$ as a ROS service client that is connected to the ROS service that is shared by ${}^{r}\mathrm{a}_{\mathrm{cl}}$ supporting ${}^{r}\mathrm{a}_{\mathrm{dy}}$.

The above initialisations are managed in the $\mathrm{S}^{initComm}_{dy}$ state. Next, ${}^{r}\mathrm{a}_{\mathrm{dy}}$ follows FSM_{dy} and the basic behaviours that are described in Section 3.2.3.

4.2 The system with simple tasks and complex schedule parameters – the TIAGo robot example

4.2.1 Expression of the system's constraints in the model's formalism

This verification system is configured to harmonise simple tasks; however, the parameters used to decide which task should be executed are complex. This system was launched in two TIAGo robot embodiments— simulated and real. The configuration of the 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 (${}^{\mathtt{R}}\mathtt{A}_{\mathtt{tha}} = \{{}^{r}\mathrm{a}_{\mathrm{ha}}\}$) and one `exa` class (${}^{\mathtt{R}}\mathtt{A}_{\mathtt{ca}} = \{{}^{r}\mathrm{a}_{\mathrm{ex}}\}$),
3. there is one `tra`-class agent—${}^{s}\mathrm{a}_{\mathrm{tr}}$,
4. agents of ${}^{r}\mathtt{A}_{\mathtt{da}}$ provide task-dependent schedule parameters that are defined by the following structure:

$$scheduleParams = [tType,\ cost,\ cps,\ cTime,\ endPose]. \tag{4.1}$$

The field $tType$ consists of an identifier of the task type that the `da` class manages. In this verification, ${}^{s}\mathrm{a}_{\mathrm{st}}$ provides two types of tasks (4.2).

$$tType \in \{guideHuman,\ humanFell\}. \tag{4.2}$$

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

consciousness. The symbol $tType_{dy}$ denotes a type of the task that is conducted by ${}^{r}\mathtt{a}_{\mathtt{dy}}$. Agents of ${}^{r}\mathtt{A}_{\mathtt{da}}$ belong to set $\mathtt{G}_r$ if they initialise their report buffers (${}^{T}_{y}\mathtt{c}_{dy,ha}[report] \neq \emptyset$ for ${}^{r}\mathtt{a}_{\mathtt{dy}}$) with the first report (by $\mathrm{pf}^{compSP}_{dy}$ for ${}^{r}\mathtt{a}_{\mathtt{dy}}$) and manage a task of $guideHuman$ type. Agents that initialise their report buffers and their tasks are of $humanFell$ type belong to the set $\mathtt{F}_r$.

$$e = {}^{r}\mathtt{a}_{\mathtt{dy}} \in {}^{r}\mathtt{A}_{\mathtt{da}} \wedge {}^{T}_{y}\mathtt{c}_{dy,ha}[report] \neq \emptyset,$$

$${}^{r}\mathtt{a}_{\mathtt{dy}} \in \mathtt{F}_r : e \wedge tType_{dy} = humanFell, \tag{4.3}$$

$${}^{r}\mathtt{a}_{\mathtt{dy}} \in \mathtt{G}_r : e \wedge tType_{dy} = guideHuman. \tag{4.4}$$

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

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

$$cost = \frac{w_{stand}}{t_{stand}} + \frac{w_{sit}}{t_{sit}}, \tag{4.5}$$

where:

i. t_{stand} is an estimated time for which the guided person stands. It is measured starting from the receipt of the task request until the task objective is completed;

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

iii. t_{sit} is an estimated time for which the guided person sits. It is measured starting from the receipt of the task request until the task objective is completed; and

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

(b) For $\mathtt{F}_r$ agents, the $cost$ parameter represents the distance to the human that fell in consideration of the robot's current pose.

The greater the value of the $cost_{dy}$ parameter, the less urgent the task of ${}^{r}\mathtt{a}_{\mathtt{dy}}$ is. cps is an estimate of the first derivative of the task cost, considering the current state of the robot and its environment. $cTime$ is an estimate of the task duration if it was started at the estimation time (considering the robot's current state and its environment). $endPose$ is the pose of the robot when the task is completed.

5. ${}^{r}\mathrm{a}_{\mathrm{ha}}$ schedules ${}^{r}\mathtt{A}_{\mathtt{da}}$ agents (using f_{ha}^{shdl}) based on algorithm 2. Hence, tasks of the same type compete with one another based on the $cost$ parameter and the hypothetical parameters that are requested by f_{ha}^{shdl} (via $\mathrm{bs}_{ha,{}^{r}\mathtt{A}_{\mathtt{da}}}[reqSP]$ buffers). The algorithm always causes an interruption of a guideHuman task by a humanFell task. In line 22, the algorithm requests two hypothetical parameters: *cc* and *ccps*. The former is an estimated cost of the recipient's task completion if the task would be started from a specified pose. The *ccps* parameter is the cost per second while the recipient's task awaits start.

4.2.2 Configuration of the Dynamic Agent model for the task types

Configuration of the *guideHuman* type tasks The model of the da class (section 3.2) must be configured for a specified task type to satisfy constraints of the task. The configuration procedure is presented for the example da class—${}^{r}\mathrm{a}_{\mathrm{gh}} \in {}^{r}\mathtt{A}_{\mathtt{da}}$ managing a task of *guideHuman* type:

- **The task automata specification** – $\mathrm{FSM}_{gh,exeTsk}$ is illustrated in Fig. 4.2,
- **The task stages classification** – most of the stages are suspendable, but $\mathtt{S}_{gh}^{stage,4}$ is blocking as it is the final stage finishing the task,
- **The task stages specification** – the basic behaviours executed in the task stages are described in the caption of Fig. 4.2,
- **The task update specification** – f_{gh}^{upTsk} is defined in Alg. 3,
- **The task suspension specification** – is given by the following formula:

$$\mathrm{f}_{gh}^{genSusp} = \begin{cases} \mathcal{B}_{gh}^{exeSusp} = \mathcal{B}_{gh}^{save}, & \text{if } \neg greeted \\ \mathcal{B}_{gh}^{exeSusp} = \mathcal{B}_{gh}^{apologize}, & \text{if } greeted \end{cases}, \tag{4.6}$$

where $greeted$ is $True$ if the robot already greeted the person being guided and $False$ if did not. In the first case of (4.6), the basic behaviour of $\mathtt{S}_{gh}^{exeSusp}$ consists of the function f_{gh}^{save}. It saves already computed results, which are useful after the task resumption (e.g., the person's pose can be used as an initial value for the person search algorithm after the task resumption). The terminal condition of $\mathcal{B}_{gh}^{save}$ is always satisfied ($\mathrm{tc}_{gh,save}$=$True$); hence, after one iteration of $\mathcal{B}_{gh}^{exeSusp}$, the transition to $\mathtt{S}_{gh}^{wait}$ is triggered. In the second case, $\mathcal{B}_{gh}^{apologise}$ is assigned to $\mathtt{S}_{gh}^{exeSusp}$. This basic behaviour stops the robot, saves important data (as in the $\mathcal{B}_{gh}^{save}$ case), and asks the person to stay and not follow the robot because it received another important task.

Algorithm 2: The algorithm of an example f_{ha}^{shdl} that is implemented in TIAGo verification system.

Input: $\{{}_{x}\mathrm{bs}_{ha,{}^{r}\mathtt{A}_{\mathtt{da}}}[report], \mathrm{bs}_{ha,{}^{r}\mathtt{A}_{\mathtt{da}}}[reqSP], \mathtt{G}_r, \mathtt{F}_r, {}^{c}\mathtt{c}_{ha}\}$
Result: $\{{}^{c}\mathtt{c}_{ha}[irrDA]\}$

```
dac = Ø;
//get the agents' identifiers from F_r and G_r sets that have the lowest costs
cHF = arg min_{dy:ra_dy ∈ F_r} cost_dy ;
cGH = arg min_{dy:ra_dy ∈ G_r} cost_dy ;
if cHF == Ø ∧ cGH == Ø then
    return;
end
else if c_ha[irrDA] == Ø then
    if cHF ≠ Ø then
        if ra_exeDA ∉ F_r then
            c_ha[irrDA] = {cHF} ;
            return;
        end
        //set the identifier of the candidate for the task switch
        dac = cHF;
    end
    else if cGH≠Ø ∧ ra_exeDA ∉ F_r then
        dac = cGH;
    end
    if dac ≠Ø then
        //request the candidate for hypothetical schedule parameters, where 1 is the identifier
          of the primitive transition function of f_dac^compSP
        T_y c_ha,dac[reqSP] = {1, endPose_exeDA};
        {cc_dac, ccps_dac} = T_x c_ha,dac[reqSP];
        //request of the estimated cost of c_ha[exeDA] task suspendion and completion after
          resumption of the ra_exeDA task (cc_exeDA) and cost per second while waiting for resumption
          (ccps_exeDA)
        T_y c_ha,exeDA[reqSP] = {1, endPose_dac};
        {cc_exeDA, ccps_exeDA} = T_x c_ha,exeDA[reqSP];
        c_switch=cost_dac + cc_exeDA + cps_exeDA * cTime_dac;
        c_wait=cost_exeDA + cc_dac + cps_dac * cTime_exeDA;
        if c_switch < 0.9 · c_wait then
            c_ha[irrDA] = {dac}
        end
    end
end
```

- **The schedule parameters calculation** – $\mathrm{pf}_{gh}^{compSP}$ is given by the following formula:

$$\mathrm{f}_{gh}^{compSP} = \begin{cases} \mathrm{pf}_{gh}^{report}, & \text{if } timer({}^{c}\mathtt{c}_{gh}[stsRate]) \\ \mathrm{pf}_{gh}^{sp,1}, & \text{if } I^1 \end{cases}, \tag{4.7}$$

$$I^1 = newData({}^{T}_{x}\mathtt{c}_{gh,ha}[reqSP]), \tag{4.8}$$

$$\mathrm{pf}_{gh}^{sp,1} : \{{}^{T}_{x}\mathtt{c}_{gh,ha}[reqSP], {}^{T}_{x}\mathtt{c}_{gh,ex}[rob], {}^{c}\mathtt{c}_{gh}\} \rightarrow \{cc_{gh}, ccps_{gh}\}. \tag{4.9}$$

Algorithm 3: The algorithm of f_{gh}^{upTsk}, where $greeted$ is *True* if the task was suspended after greeting the human and *False* if it was not suspended.

Input: $\mathrm{FSM}_{gh,exeTsk}$
Result: Modification of $\mathrm{FSM}_{gh,exeTsk}$

```
if greeted == True then
    B_gh^{s,2} = B_gh^{greet_and_apologise};
else
    B_gh^{s,2} = B_gh^{greet};
end
```

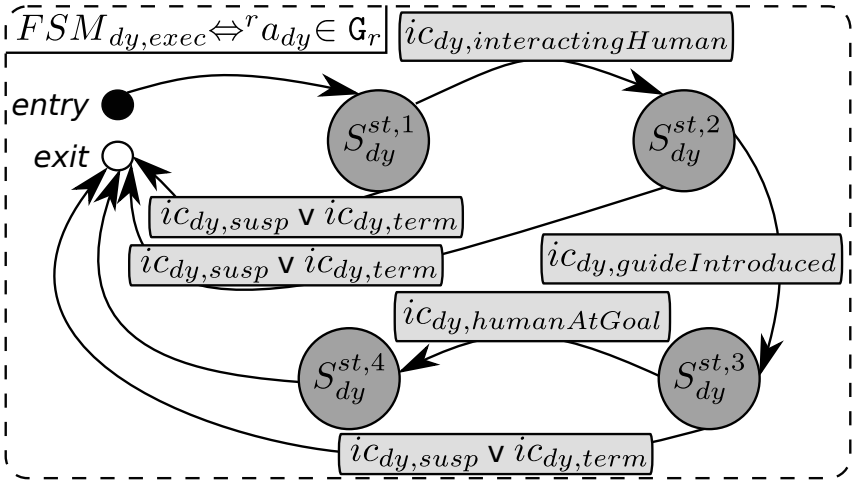


Figure 4.2: The FSM of human guidance tasks, where the basic behaviour of $\mathtt{S}_{dy}^{stage,1}$ moves the robot to the requested human, the basic behaviour of $\mathtt{S}_{dy}^{stage,2}$ interacts with the human to introduce the task and ask him to follow the robot, the basic behaviour of $\mathtt{S}_{dy}^{stage,3}$ moves the robot to the destination and checks if the human follows the robot. The basic behaviour of $\mathtt{S}_{dy}^{stage,4}$ finishes the interaction with the human.

Configuration of *humanFell* type tasks The constraints for a humanFell type task is as follows (considering ${}^{r}\mathrm{a}_{\mathrm{hf}}$ as an example):

- **The task automata definition** – $\mathrm{FSM}_{hf,exeTsk}$ is given by the graph shown in Fig. 4.3,
- **The task stages classification** – $\mathtt{S}_{hf}^{stage,move}$ is suspendable and $\mathtt{S}_{hf}^{stage,report}$ is blocking,
- **The task stages specification** – the basic behaviours executed in the task stages are described in Fig. 4.3,
- **The task update definition** – the task automata (Fig. 4.3) is not required to be updated in this simple example,

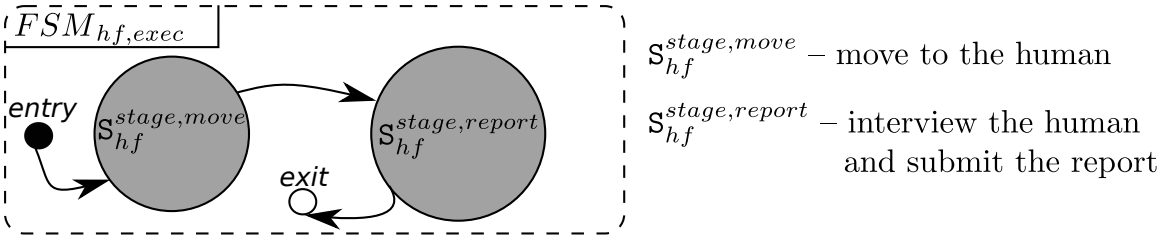


Figure 4.3: The graph describing $\mathrm{FSM}_{hf,exeTsk}$ of a *humanFell* type task.

- **The task suspension specification** – as there is just one suspendable stage ($\mathrm{S}_{hf}^{stage,move}$), so there is just one suspending behaviour, namely $\mathcal{B}_{hf}^{stop}$ that is assigned by $\mathrm{f}_{hf}^{genSusp}$ to $\mathcal{B}_{hf}^{exeSusp}$. The robot is stopped if this basic behaviour is executed.

- **The schedule parameters calculation** – $\mathrm{pf}_{hf}^{compSP}$ is given by the following formula:

$$\mathrm{f}_{hf}^{compSP} = \begin{cases} \mathrm{pf}_{hf}^{report}, & \text{if } timer(\mathrm{c}_{hf}[stsRate]) \\ \mathrm{pf}_{hf}^{sp,1}, & \text{if } I^1 \end{cases}, \tag{4.10}$$

$$I^1 = newData(\mathrm{c}_{hf,ha}^{T}[reqSP]), \tag{4.11}$$

$$\mathrm{pf}_{hf}^{sp,1} : \left\{\mathrm{c}_{hf,ha}^{T}[reqSP], \mathrm{c}_{hf,ex}^{T}[rob], \mathrm{c}_{hf}\right\} \rightarrow \left\{cc_{hf}, ccps_{hf}\right\}. \tag{4.12}$$

4.2.3 Execution of the example system

The system was executed in the example scenario in simulation and the task harmonisation procedure (Sec. 3.5) was executed over 80 times in a changing environment. The scenario involved:

1. requests of `da`-class agents that manage tasks of various types,

2. calculation of dynamic schedule parameters while the robot moves and conducts various tasks,

3. selection and execution of task suspension strategies. Additionally, task plans are updated before execution.

4. a task switch due to either a request for a higher priority task or a change in the environmental setup (human movement), and

5. termination of an agent from set $\mathtt{D}_r$ if its task is no longer beneficial.

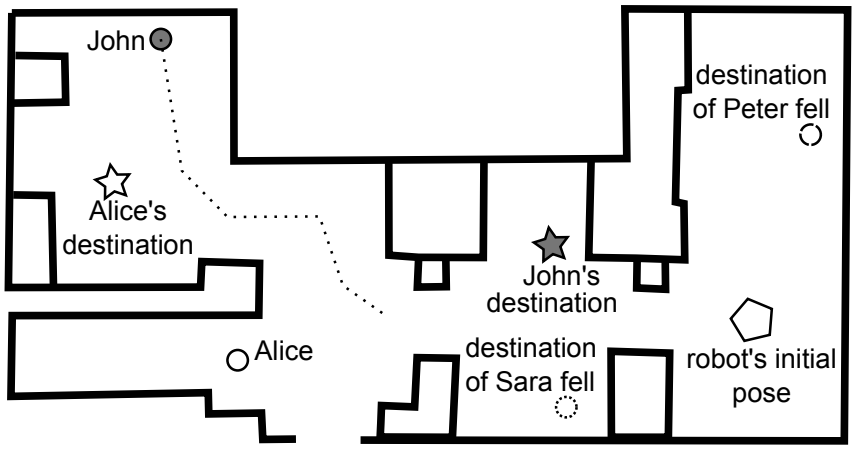


Figure 4.4: Verification environment setup.

The initial setup of the environment is shown in Fig. 4.4. Four people, namely, John, Alice, Peter and Sara, were in the environment. In the following paragraph, a period of the tests is described in detail. In this period, four tasks were requested:

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

The robot and human actions during the scenario's period are shown in the sequence diagram (Fig. 4.5). There are actions of TIAGo, actions of the moving human (John) and primary states executed by Dynamic Agents visualised over time. In Fig. 4.4 there are initial poses of the robot (as a pentagon), John (as a filled circle) and Alice (as an empty circle) visualised. The path that John traversed while waiting for the robot is represented as a dotted line. Filled and empty stars represent John's and Alice's destinations. 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 [71].

In Fig. 4.6, the values of the following schedule parameters calculated by f_{ha}^{shdl} are plotted: $cost_{dac}$, $cost_{exeDA}$, cc_{dac}, cc_{exeDA}, $ccps_{dac}$, $ccps_{exeDA}$, c_{switch}, and c_{wait}. The rectangularly highlighted sections of the figure show values of the fields ${}^{\mathrm{c}}\mathrm{c}_{ha}[idleDA]$, ${}^{\mathrm{c}}\mathrm{c}_{ha}[exeDA]$ and ${}^{\mathrm{c}}\mathrm{c}_{ha}[irrDA]$ in crucial moments of the test period. First, at $time = 2450$, ${}^{r}\mathrm{a}_{\mathrm{d1}}$ is initialised and executes its task. Then, between $time = 2450$ and $time = 2509$, ${}^{r}\mathrm{a}_{\mathrm{d2}}$ is initialised, which computes the schedule parameters, and eventually, at $time = 2509$, f_{ha}^{shdl} sets $dac = d2$ (line 18 of the Alg. 2). After comparing c_{switch} and c_{wait} (line 28 of the Alg. 2), it assigns the identifier

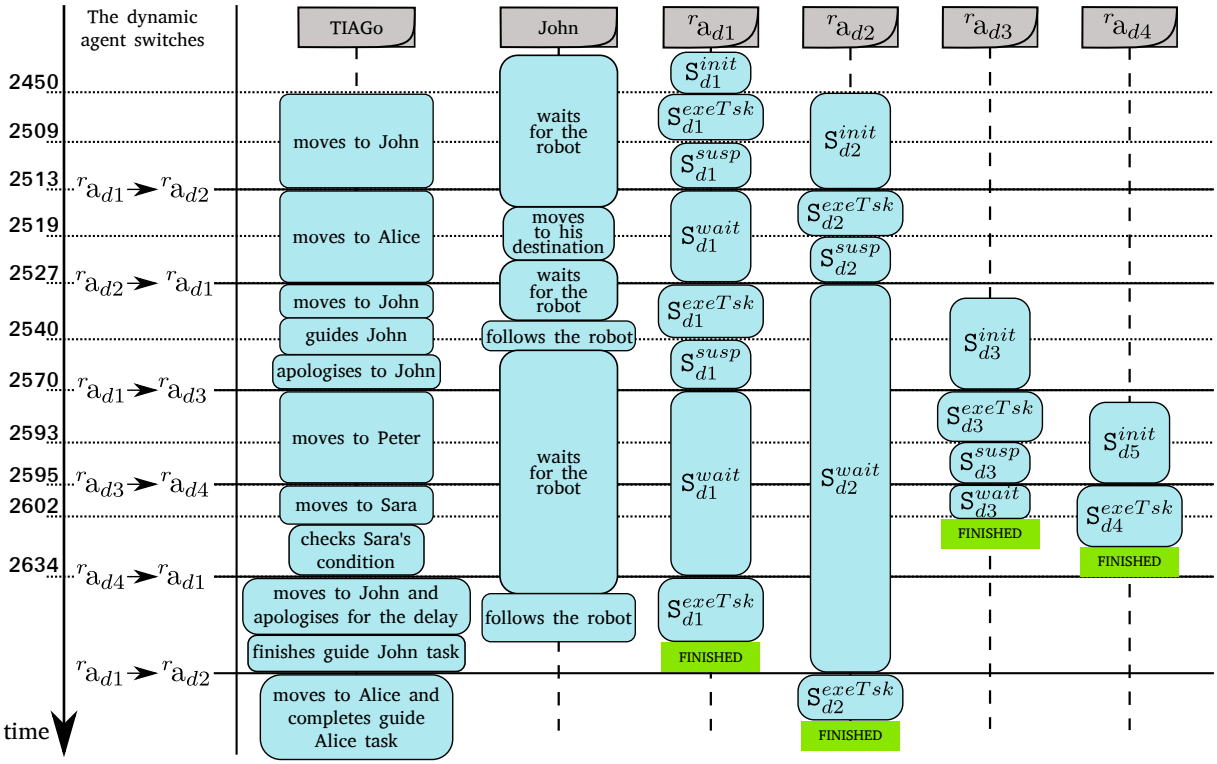


Figure 4.5: The sequence of TIAGo and John actions in the scenario with timestamps. There are Dynamic Agent switches and the states executed by the agents during the verification.

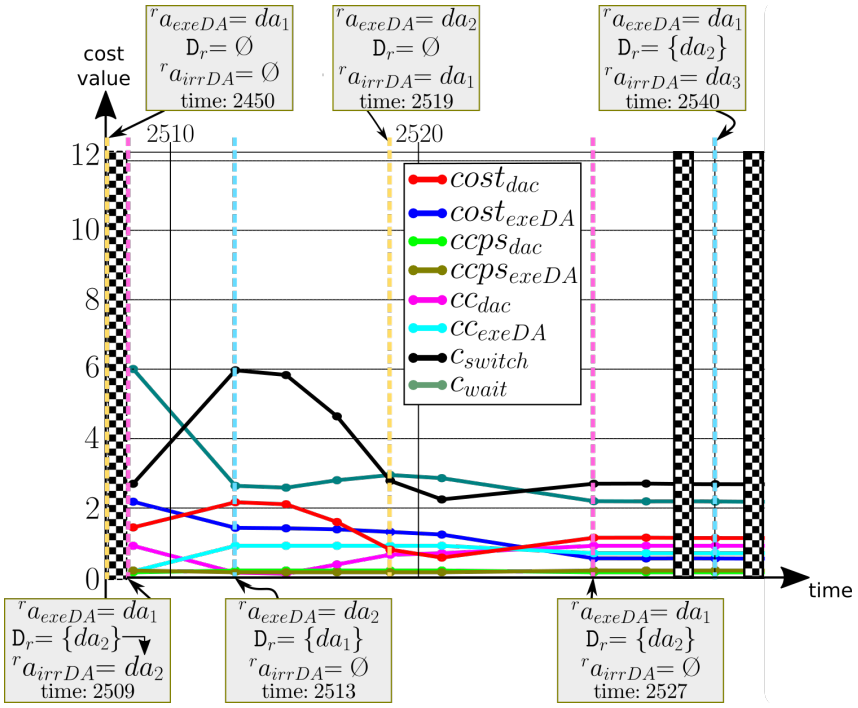
of $^{r}a_{d2}$ to $\mathfrak{c}_{ha}[irrDA]$. In response to this, S_{ha}^{cmdDA} is triggered and sends a suspension signal to $^{r}a_{d1}$, which is in $S_{d1}^{stage,1}$(the robot approaches John). The agent switches to the $S_{d1}^{genSusp}$ state (which, following (4.6), assigns $\mathcal{B}_{d1}^{save}$ to $\mathcal{B}_{d1}^{exeSusp}$) and subsequently to $S_{d1}^{exeSusp}$ and S_{d1}^{wait}. As $^{r}a_{d1}$ notifies $^{r}a_{ha}$ that the suspension strategy ($FSM_{d1,exeSusp}$) has been completed, f_{ha}^{cmdDA} sends a start signal to $^{r}a_{d2}$. Finally, at $time = 2513$, $\mathfrak{c}_{ha}[exeDA] = d2$ and $\mathfrak{c}_{ha}[idleDA] = D_r = \{d1\}$.

From $time = 2513$ to $time = 2519$, John moves closer to his destination; 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 a sum of the following costs:

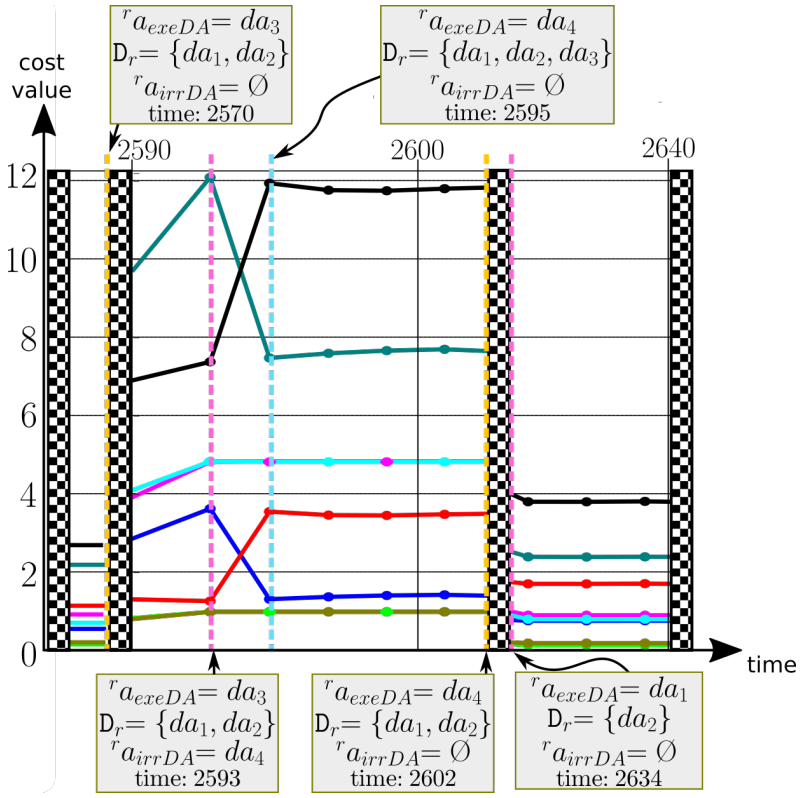
- suspension of the task of $^{r}a_{d2}$,
- completion of the task of $^{r}a_{d1}$ (approaching John and guiding John to his destination),
- completion of the task of $^{r}a_{d2}$ starting from the John's destination pose,

and c_{wait} is the sum of the following costs:

- completion the ongoing task of $^{r}a_{d2}$,
- completion of the task of $^{r}a_{d1}$ starting from the Alice's destination pose.



(a) The first part of the verification period.



(b) The second part of the verification period.

Figure 4.6: Data that were calculated by f_{ha}^{shdl} during the verification.

Therefore, in response to the change in the environment (John's movement), the former is less than the latter. Until $time = 2527$, ${}^{r}\mathrm{a}_{\mathrm{d2}}$ suspends its task; finally, at this time, ${}^{r}\mathrm{a}_{\mathrm{d1}}$ can resume its task. ${}^{r}\mathrm{a}_{\mathrm{d1}}$ completes stages $\mathtt{S}_{d1}^{stage,1}$ and $\mathtt{S}_{d1}^{stage,2}$.

At $time = 2540$, while ${}^{r}\mathrm{a}_{\mathrm{d1}}$ was in $\mathtt{S}_{d1}^{stage,3}$ (guiding John to his destination), the robot received a request for the Peter fell task. As a result of this, ${}^{r}\mathrm{a}_{\mathrm{d3}}$ is initialised, and as the Peter fell task is of humanFell type (${}^{r}\mathrm{a}_{\mathrm{d3}} \in \mathtt{F}_r$) and the guide John task is of guideHuman type (${}^{r}\mathrm{a}_{\mathrm{d1}} \in \mathtt{G}_r$), the condition in line 10 of the Alg. 2 is satisfied, and ${}^{r}\mathrm{a}_{\mathrm{d3}}$ is immediately assigned to ${}^{c}\mathrm{c}_{ha}[irrDA]$. At this point, ${}^{r}\mathrm{a}_{\mathrm{d1}}$ switches to $\mathtt{S}_{d1}^{genSusp}$, which, according to (4.6), assigns $\mathcal{B}_{d1}^{apologise}$ to $\mathcal{B}_{d1}^{exeSusp}$. Then, ${}^{r}\mathrm{a}_{\mathrm{d1}}$ switches to $\mathtt{S}_{d1}^{exeSusp}$, and the robot apologises to John, asks him to stop following the robot and stores his current pose. Finally, ${}^{r}\mathrm{a}_{\mathrm{d1}}$ switches to $\mathtt{S}_{d1}^{wait}$, and f_{ha}^{cmdDA} sends the start signal to ${}^{r}\mathrm{a}_{\mathrm{d3}}$ at $time = 2570$.

Next, while the robot is moving to the Peter fell destination (the dashed circle in Fig. 4.4), 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, ${}^{r}\mathrm{a}_{\mathrm{d3}}$, following FSM_{d3}, switches to the $\mathtt{S}_{d3}^{wait}$ state, and the robot starts the Sara fell task.

Next, at $time = 2602$, ${}^{r}\mathrm{a}_{\mathrm{d3}}$ is in $\mathtt{S}_{d3}^{wait}$ and calculates f_{d3}^{wait} (in this case $\mathrm{pf}_{d3}^{report}$). 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 ${}^{r}\mathrm{a}_{\mathrm{ha}}$ based on (3.16):

$$ {}_{y}^{T}\mathrm{c}_{d3,ha}[report] = [end, scheduleParams]. \tag{4.13} $$

As a result, $term_{event}$ is triggered (3.36) and f_{ha}^{cmdDA} removes ${}^{r}\mathrm{a}_{\mathrm{d3}}$ from the ${}^{r}\mathtt{A}_{\mathtt{da}}$ set. During the verification, there was no candidate for ${}^{c}\mathrm{c}_{ha}[irrDA]$ in the following periods:

- between the initialisation of ${}^{r}\mathrm{a}_{\mathrm{d1}}$ ($time = 2450$) and ${}^{r}\mathrm{a}_{\mathrm{d2}}$ ($time = 2509$),
- 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, in these periods dac variable of Alg. 2 is: $dac = \emptyset$, and no values of the considered parameters were calculated by f_{ha}^{shdl}.

Additionally, the example system was integrated with the real TIAGo robot, and a scenario similar to the presented one was played with cooperation with humans. The video showing the experiment is available in [72].

4.2.4 Satisfaction of the requirements in the example system

In the example system, ${}^{s}\mathrm{a}_{\mathrm{tr}}$ can request multiple tasks at any time. The requests are processed, suitable `da`-class agents are created, and in each iteration of f_{ha}^{shdl}, a `da` class can be assigned to ${}^{r}\mathrm{a}_{\mathrm{irrDA}}$; thus, the requirement `R1`[1] is satisfied.

Schedule parameters can be computed repetitively (3.21) by $\mathrm{pf}_{dy}^{report}$, which fills the ${}^{T}_{y}\mathrm{c}_{dy,ha}[report]$ buffer (3.16). Additionally, the scheduling algorithm can request hypothetical schedule parameters, which are computed by dedicated primitive transition functions of f_{dy}^{compSP}; thus, the requirement `R2`[2] is satisfied.

The procedure of scheduling `da`-class agents is divided into multiple primitive transition functions, which are distributed among `da`-class agents and ${}^{r}\mathrm{a}_{\mathrm{ha}}$. The latter with the f_{ha}^{shdl} function computes the decision on switching an ongoing task with a new task or continuing the current task. Therefore, the example system can be easily configured to use different schedule parameters and scheduling algorithms. Thus, the requirement `R3`[3] is satisfied.

Tasks are created independently and stored in the cloud. A `tha` class agent of a robot downloads them upon its user's request. The initiated tasks become `da`-class agents and compute the schedule parameters considering the expert knowledge and the task's context. For example, expert knowledge can be used to calculate an estimated time for completing a task (e.g., a speed limitation can be set due to transportation of a delicate object or liquid). Thus, the requirement `R4`[4] is satisfied.

The TaskER model divides tasks into stages that can be suspended and stages that cannot. It also defines the FSM of `da` class with dedicated states triggered if the agent receives a suspension signal and the current stage of the agent's task is suspendable. Hence, the agent, executing its task, manages the actions in case of an interruption. Therefore, it prevents possible inconvenient behaviour or damage to the system and its environment. The executing agent is chosen to execute the suspending actions because it 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 the 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

[1] The robot controller maintains additional activities to enable an advised task switch. It constantly listens to task requests and executes a schedule decision and switches tasks, even if the robot executes another task.

[2] Values of the schedule parameters (e.g., priorities) used to compute schedule decisions are dynamically changing even if a task awaits execution. If one of the parameters changes, then the scheduling algorithm is initiated. All classes of the schedule parameters shown in Fig. 2.2 can be used in the system.

[3] Both the algorithm and the parameters that are used in the scheduling procedure depend on a system application and must be configured based on individual system's requirements.

[4] The tasks that are available in the system are created independently and differ in terms of knowledge base and contexts.

task's execution. Another example is to exit a secret room that access code is known only to the current task. Hence, the requirement R5[5] As a result of this, a possible robot/environment damage or other loss due to the task interruption is limited. is satisfied.

Due to the frequently triggered $\mathrm{pf}_{dy}^{report}$ function, ${}^{r}\mathrm{a}_{\mathrm{dy}}$ can react to changes in a dynamic environment and update values of the schedule parameters. The changes in the environment also affect the plans of da-class agents. Therefore, the TaskER model foresees the need to update the plans (in the state S_{dy}^{upTsk}) before an agent executes its task. The most straightforward example result of $\mathcal{B}_{dy}^{upTsk}$ can be an extension or reduction of a plan due to actions carried out by a user in the environment. Thus, the requirement R6[6] is satisfied.

4.3 The system with a complex task and simple schedule parameters – the mobile manipulator example

4.3.1 Expression of the system's constraints in the model's formalism

Verification in a mobile manipulation system is conducted by applying the TaskER model and its harmonisation procedure (Fig. 3.8) in a simulated environment inhabited by the Velma robot [61]. The configuration of the system is as follows.

1. task harmonisation is of interruptible-task type,
2. there is one robot in the system ($\mathtt{R} = \{v\}$), one tha-class agent (${}^{\mathrm{R}}\mathtt{A}_{\mathtt{th}} = \{{}^{v}\mathrm{a}_{\mathrm{ha}}\}$) and one exa class (${}^{\mathrm{R}}\mathtt{A}_{\mathtt{ca}} = \{{}^{v}\mathrm{a}_{\mathrm{ex}}\}$),
3. there is one tra-class agent—${}^{s}\mathrm{a}_{\mathrm{tr}}$,
4. agents composing ${}^{v}\mathtt{A}_{\mathtt{da}}$ do not provide any schedule parameters to ${}^{v}\mathrm{a}_{\mathrm{ha}}$,
5. ${}^{s}\mathrm{a}_{\mathrm{st}}$ provides two task types: *bringObject* and *humanFell*. Agents managing tasks of *bringObject* type compose the set ${}^{s}\mathtt{A}_{bo}$ and these managing tasks of *humanFell* type compose the set ${}^{s}\mathtt{A}_{hf}$,
6. the scheduling algorithm managing tasks in this system is given by the following rules:

[5] The developed model must raise awareness of the task developer to foresee possible dangerous situations caused by a task switch. Additionally, the proposed task execution method enables independent tasks to oversee changes in the environment and set various schedule parameters. Furthermore, the ongoing task is carefully suspended before the controller switches to another task.

[6] A plan of a task that awaits execution is updated before the task execution.

(a) agents managing tasks of the same type are stored in separate queues following First In First Out paradigm,

(b) the queue aggregating the agents from the ${}^{s}\mathtt{A}_{hf}$ set is preferred over the other queue, so the algorithm favours agents from the ${}^{s}\mathtt{A}_{hf}$ set,

(c) if ${}^{v}\mathrm{a}_{\mathrm{exeDA}} \in {}^{s}\mathtt{A}_{bo}$ and ${}^{s}\mathtt{A}_{hf} \neq \emptyset$, then the first agent from the queue of ${}^{s}\mathtt{A}_{hf}$ agents is set as ${}^{v}\mathrm{a}_{\mathrm{irrDA}}$,

(d) tasks of the same type do not interrupt each other.

4.3.2 Configuration of the Dynamic Agent model for the task types

The model of `da` class (Sec. 3.2) must be configured for a specified task type to satisfy its constraints. Configuration of the tasks of *bringObject* type is as follows (considering ${}^{v}\mathrm{a}_{\mathrm{bo}}$ as an example):

- **The task automata definition** – $\mathrm{FSM}_{bo,exeTsk}$ is given by the graph shown in Fig. 4.7,
- **The task stages classification** – most of the stages of `bringObject` type tasks are suspendable, but $\mathtt{S}_{bo}^{stage,5}$ and $\mathtt{S}_{bo}^{stage,6}$ are blocking. It means that the robot should finish these stages before a suspension strategy can be executed, and the task can be switched with the interrupting task,
- **The task stages specification** – the basic behaviours executed in the task stages are described in Fig. 4.7,
- **The task update definition** – the task automata (Fig. 4.7) is not required to be updated in this simple example. Only the parameters used in the task behaviours change (e.g. pose of the object to be transported),
- **The task suspension specification** – there are two alternative behaviours, which can be assigned by $\mathrm{f}_{bo}^{genSusp}$ to $\mathcal{B}_{bo}^{exeSusp}$ and they are as follows:
  - $\mathcal{B}_{bo}^{stop}$ – stop the mobile base and take a safe posture,
  - $\mathcal{B}_{bo}^{setAside}$ – set aside the object being transported,

  They are chosen following (4.14):

$$\mathrm{f}_{bo}^{genSusp} = \begin{cases} \mathcal{B}_{bo}^{exeSusp} = \mathcal{B}_{bo}^{stop}, & \text{if } \neg grabbed \\ \mathcal{B}_{bo}^{exeSusp} = \mathcal{B}_{bo}^{setAside}, & \text{if } grabbed \end{cases}, \tag{4.14}$$

  where $grabbed$ is $True$ if the robot already caught the object.

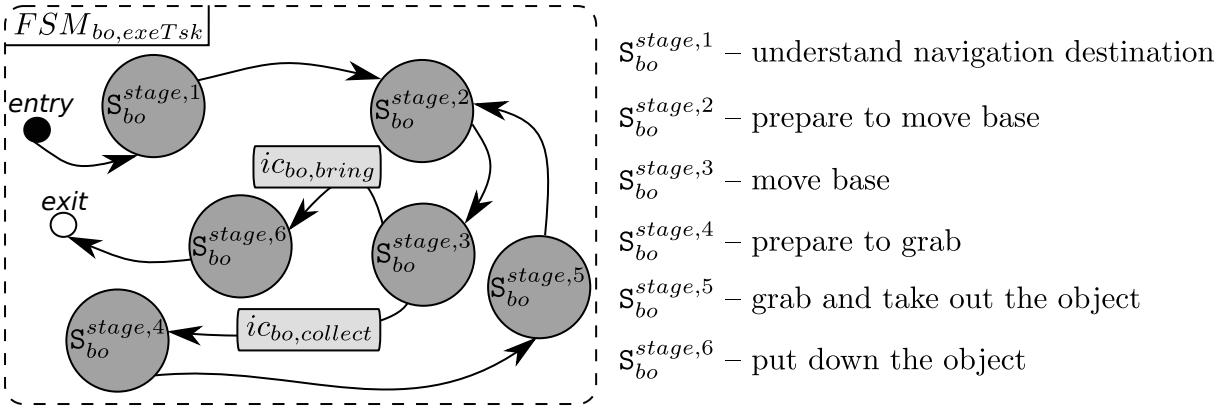


Figure 4.7: The graph describing $\mathrm{FSM}_{bo,exeTsk}$ of ${}^{v}\mathrm{a}_{\mathrm{bo}}$ managing a `bringObject` type task.

- **The schedule parameters calculation** – in this system Dynamic Agents do not compute schedule parameters, so the status buffer (${}_{y}^{T}\mathrm{c}_{bo,ha}[report]$) has the basic structure given by (3.15) and $\mathrm{pf}_{bo}^{compSP}$ (3.21) consists only $\mathrm{pf}_{bo}^{report}$, which calculates the content of the report buffer (parameter p of (3.21) equals 0).

Configuration of the tasks of *humanFell* type is as follows (considering ${}^{v}\mathrm{a}_{\mathrm{hf}}$ as an example):

- **The task automata definition** – $\mathrm{FSM}_{hf,exeTsk}$ is given by the graph shown in Fig. 4.3,
- **The task stages classification** – $\mathtt{S}_{hf}^{stage,move}$ is suspendable and $\mathtt{S}_{hf}^{stage,report}$ is blocking,
- **The task stages specification** – the behaviours executed in the task stages are described in Fig. 4.3,
- **The task update definition** – the task automata (Fig. 4.3) is not required to be updated in this simple example,
- **The task suspension specification** –there is one suspendable stage ($\mathtt{S}_{hf}^{stage,move}$) and one suspending basic behaviour—$\mathcal{B}_{hf}^{stop}$. The basic behaviour simply stops the robot.
- **The schedule parameters calculation** – in this system Dynamic Agents do not compute schedule parameters, so the status buffer (${}_{y}^{T}\mathrm{c}_{hf,ha}[report]$) has the basic structure given by (3.15) and $\mathrm{pf}_{hf}^{compSP}$ (3.21) consists only $\mathrm{pf}_{hf}^{report}$, which calculates the content of the report buffer (parameter p of $\mathrm{pf}_{hf}^{compSP}$ equals 0).

4.3.3 The verification scenario of the mobile manipulation system

The mobile manipulation system's verification environment consists of a human, a table with shelves, an object to transport, and another table. The object is on a shelf, and the Velma robot is

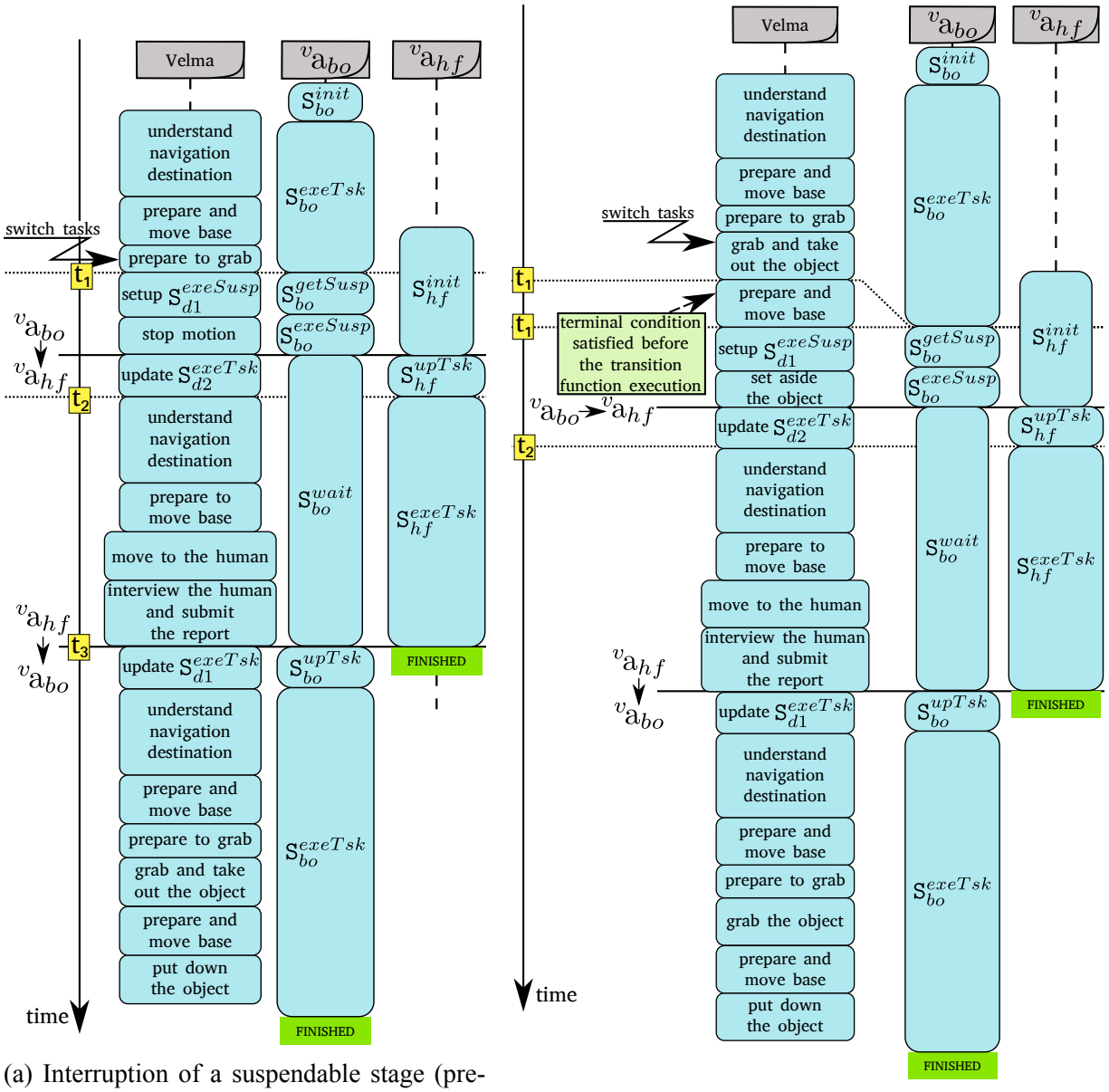
requested to transport it to the other table. The setup of the environment is visualised in Fig. 4.8. The conducted verification covers two moments while the robot executes a `bringObject` task

Figure 4.8: The setup of the environment utilised in the verification system

and is requested to manage a `humanFell` task. The first request is received when ${}^{v}\mathrm{a}_{\mathrm{bo}}$ executes $\mathtt{S}_{bo}^{stage,4}$ (prepares to grab) and the second during the object manipulation in $\mathtt{S}_{bo}^{stage,5}$. Following the proposed model of `da`-class agents, ${}^{v}\mathrm{a}_{\mathrm{bo}}$ manages suspension requests differently in these two situations. The robot's overall behaviour and the states of the Dynamic Agents in the verification are shown in Fig. 4.9.

In the first scenario, the suspension signal received by ${}^{v}\mathrm{a}_{\mathrm{bo}}$ in $\mathtt{S}_{bo}^{stage,4}$ triggers transition from $\mathtt{S}_{bo}^{exeTsk}$ to $\mathtt{S}_{bo}^{susp}$. As a result of this the robot executes suspending actions. In this case the robot takes a safe posture, as it has not caught the object yet. Next, it executes the interrupting task (managed by ${}^{v}\mathrm{a}_{\mathrm{hf}}$), and when it is finished ${}^{v}\mathrm{a}_{\mathrm{bo}}$ receives a start signal and continues its task.

In the second scenario, the robot receives the suspension request in $\mathtt{S}_{bo}^{stage,5}$. It is different and more complex than the previous scenario. First, the stage of the currently managed task is blocking, so the interruption signal is managed by ${}^{v}\mathrm{a}_{\mathrm{bo}}$ in the next stage ($\mathtt{S}_{bo}^{stage,2}$). Second, the robot has the object in its hand; therefore, the proper suspension behaviour is chosen in $\mathtt{S}_{bo}^{genSusp}$ and executed in $\mathtt{S}_{bo}^{exeSusp}$. In this case, the behaviour sets aside the object to free the robot's effector. Finally, the interrupting task (managed by ${}^{v}\mathrm{a}_{\mathrm{hf}}$) can be executed. As soon as it is completed, ${}^{v}\mathrm{a}_{\mathrm{bo}}$ receives a start signal and resumes its task. Thanks to $\mathtt{S}_{bo}^{upTsk}$, the agent adapts the task to the world's current state. Therefore, the task stays feasible, even if the object's position changes. The above-described behaviour of the system was recorded, and the video is available [73].



(a) Interruption of a suspendable stage (prepare to grab)

(b) Interruption of a blocking stage (grab and take out the object)

Figure 4.9: The sequences of Velma actions and Dynamic Agents states during verification.

Chapter 5

Conclusions

5.1 Discussion and related works

The conducted study regards principally three problems: robotic systems modelling, their integration with cloud support and management of their tasks. Therefore, the analysis of related works is presented in three germane sections.

Structures and behaviour models for robot systems

One of the most popular approaches for modelling robot control systems is a layered architecture, e.g., the 3T architecture [55], [74]. It is thoroughly described, compared and used in SmartMDSD Toolchain [75], [76]. The authors of [70] design the middle layer of the 3T architecture that is structured with a finite-state machine (FSM). It integrates a symbolic plan with a geometric planner that instructs the robot. Each description layer of a task (symbolic planning, behaviour sequencing, or command execution) uses a different context. Each layer fulfils objectives and handles exceptions specified in a context, e.g., a symbolic planner uses logic variables and functions, and a behaviour sequencer uses basic behaviours and transition functions.

The study described in this dissertation regards all three layers that are defined in the above architectures. An agent of `exa`-class that is used in this work, is the command execution layer, `da`- and `tha`-class agents constitute the sequencer layer. The symbolic planners (implemented in either `cla` class or `pla` class) that the sequencer layer calls, 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 [77]; however, the study focuses on the specification and execution of a specified task. The work described in this dissertation concentrates on robot tasks management by switching independent agents of sequencer layer that can utilise high-level task description and planning and affect the environment by instructing the agents of command execution layer.

Distributed robot control systems Authors of the survey [78] present the current state of robot-cloud cooperation. They distinguish four potential profits of cloud implementation in robotic systems.

- Big data – access to large packages of images, trajectories or descriptive data,
- Cloud computing – parallel grid computing, learning, planning,
- Collective Robot Learning – sharing trajectories and outcomes in multiple robot systems,
- Human Computation – image/video analysis, classification, learning and error recovery.

Among other robot behaviours, the cloud is used in robot navigation. The work [79] presents the C^2TAM system that is used to process visual SLAM – vSLAM. In this solution, the cloud builds the environment map using heavy computational power algorithms and processes RGB-D data from multiple robots. In work [80], authors focus on different approaches to processing distribution among build-in robot computer and the cloud. They consider stereo pair cameras' image processing in the mobile robot teleoperation task. They present processing and transmission time of data in different image resolution and different wireless communication cases. The work shows that the proper distribution of processing has a crucial influence on the robot control system's stability and robustness. The DAvinCi [81] project developers based the cloud system on ROS (Robot Operating System [82], [83]) communication mechanisms and the Hadoop cluster [84]. It was used to process the FastSLAM (Fast Simultaneous Localization And Mapping) – algorithm in parallel. Rapyuta platform [85] allows dynamic allocation of robot safe processing environments (that are based on ROS). These environments are strictly conjugated one to another to allow information and service exchange between robots.

Above systems, apart from using the robot computer, benefit from the second platform – the cloud. This solution has many advantages. It supplies robot controllers with additional computation power and storage capacity and reduces the cost of robots. Among the advantages of cloud computing, researchers also note the reliability, large memory capacity, energy-saving, stable power supply, better use of resources and more straightforward modernization of the cloud than the onboard robot platform.

Most of the known robot systems supported by the cloud platform are dedicated to a robot type, an application, an algorithm, or a single service. However, the RAPP system is a distributed, modular robot system that can extend its capabilities. It allows robots to:

- store large packages of data,
- process data and conclude,

- request computationally heavy services,
- learn objects,
- share robotic tasks even if the robots types differ,
- share data with other robots.

The cloud part of the RAPP system is based on multi-thread services; thus, it can handle multiple robots at once. Furthermore, a well-defined API of the RAPP system allows a different type of robots to share the same tasks, and the tasks can be implemented by a non-experienced programmer using the RAPP API. When a robot requires a specific ability, the robot downloads a required task and interprets the RAPP API methods depending on the specific robot platform controller implementation.

Multi-task robot systems

The authors of [86] describe a study on planning schemes for human-robot interaction. The schemes are scheduled by a symbolic planner depending on the situation. The authors focus on social constraints considered in the 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 low robot-controller level (e.g., [87], [88]). Domain switching problems arise in robotics in switching different-domain tasks and in the execution of some tasks. For example, there are robot-human communication tasks. The authors of [89] describe a model of a system that is capable of interruption and switching the conversation domain.

The authors of the works [29], [90] present a method to choose the robot's next action from the tasks awaiting completion. The tasks based on the observations propose subsequent actions, while the task selection algorithm chooses one of them based on the previously learned policy. The concept of this solution is shown in Fig. 5.1. The system model proposed in this dissertation defines a dedicated state of the task in which the task manages its suspension. The harmonisation procedure enables the system to calculate various schedule parameters (e.g. time to complete tasks). Based on this knowledge, the system decides if the task switch is safe or if any actions are required to suspend the current task. The referenced work does not consider task suspensions. Possibly, the task suspension could be considered in the task selection learning procedure by a negative reward generation every time the task interruption resulted in an unsafe situation. However, the design of an appropriate set of experiments for learning prudent task management is challenging or, in some cases, impossible. In contrast, the TaskER model uses a formal method (HFSM) to ensure the system will consider safety while switching the robot tasks. Furthermore,

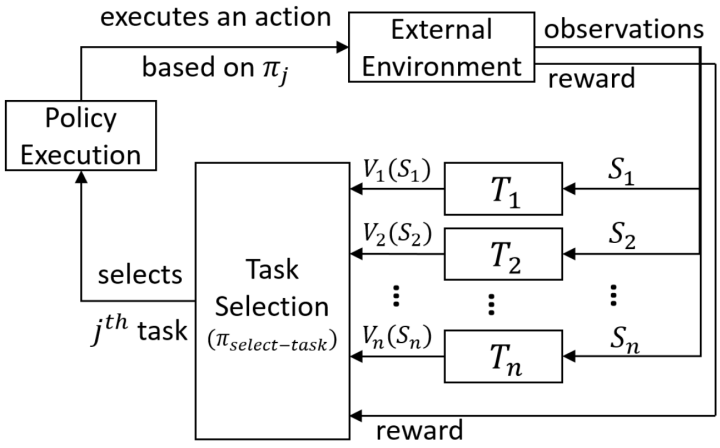


Figure 5.1: Machine learning to choose an action from the awaiting tasks [90].

the model classifies stages of tasks as suspendable and blocking. So even the stages can be interrupted when they are executed. In the referenced work, actions are the equivalent of the task stages, and their execution cannot be interrupted. In an example situation where the action is 'move to a far destination', the action should be interruptible if the environment changes.

The SMACH library [91] is a known implementation of the robot tasks modelled with FSMs. The TaskER, besides the task FSMs defines configurable scheduling algorithms and suspension/replanning actions. SMACH does not support replacement/modification of HFSMs natively, and it is required in systems featured with planning algorithms. In this case, in one state, a planning algorithm develops the HFSM describing the subsequent robot's behaviour; thus, the initial HFSM of an agent must be modified. The model of TaskER can be used for systems with planning; therefore, in this case, the native form of SMACH can not be used.

Multiple tasks execution in state-of-the-art robot systems is realised with high-level deliberation. Reasoning algorithms compute a multi-objective plan that uses temporal constraints [92] and uncertainties in hypothetical planning [93]. This approach utilises a semantic planner, which requires a knowledge base 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 a task's priority changes, if a task is cancelled, or if a task's parameters change, this approach requires re-planning. The available frameworks that utilise semantic planning ROSPlan [94], SmartTCL [76] do not provide a mechanism for re-initialising planning in the face of modification of a task schedule parameter (like priority).

Robot systems are cyber-physical systems (CPSs). Various articles consider the task harmonisation problem in this group [95]–[97]. However, they are focused mostly on algorithms for optimising the task switching moment or coordination of the tasks delegated to multiple

devices to complete a specified objective [98]. The authors of [95] present a methodology for describing, managing and realising objectives of a device community. This study shows how to describe an objective of device community. The authors propose to delegate simple roles to each device. The second paper [96] presents an algorithm for minimising the deadline miss ratio. The algorithm considers the time required by the servicing node for moving from one place to another. However, the authors do not demonstrate how to manage a task switch. The study [97] considers a dynamic allocation of computational tasks among distributed CPS devices. The authors of [99] 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 [100] or task activation is statically defined by rules as in home automation applications), robots are being unpredictably requested for tasks. The robot user would like to change his/her requests and preferences while the robot carries out the requested task.

Task queueing and management is an old topic in cybernetic system studies [101], [102] and a hot topic in cyber-physical systems [103], [104]. The areas differ because the physical environment, which CPS percept and affect, is more unpredictable. It is difficult to observe accurately [105], [106]; therefore, it should be affected with uncertainty [107], [108] and according to a plan [109]. The robot task harmonisation problem has similarities with process scheduling in operating systems (e.g., sporadic scheduling in real-time systems [110]). Both consider unit work management: in robotics, a robot executes multiple tasks, and in operating systems, a CPU manage multiple processes. The correlations between scheduling operating system processes and harmonisation considered in the TaskER are presented in Tab. 5.1.

Table 5.1: The comparison of scheduling in typical operating systems and robot systems that use TaskER.

Feature	**OS process scheduling**	**TaskER harmonisation**
Reject mechanism	Available (sporadic scheduling)	All accepted, 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. 3.4)
Execution unit	CPU	Robot (`exa`-class agent)
Utilisation of a sophisticated calculation of priorities and deadlines, which are dependent on a job/task context	Unavailable	Available
Effect of an atomic instruction	The memory state change, or an elementary action on i/o devices	The 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 `tha`- and `da`-class agents. The latter are aware of the task context and constraints of the real world environment
Medium-term scheduling entity	Swapping mechanism	————”————
Instant execution of the scheduler decision	Available	When it is safe, otherwise suspends the ongoing task first

Feature	OS process scheduling	TaskER harmonisation
Procedure of a job/task suspension and resumption	Save/load the process memory (relatively is a quick action)	Suspension and resumption procedures enable foreseeing consequences of a task interruption and mitigate them (relatively is a time-consuming action)
Re-scheduling jobs/tasks on their requests	Unavailable	Available, because `da`-class agents calculate schedule parameters and any change in the parameters triggers the scheduling algorithm
Abstract parameters are used in the scheduling process	Unavailable	Available (e.g., maximisation of human convenience, or for a restaurant runner task, the time at which a new client approaches a table is a deadline for cleaning the table)
Hard restriction of deadlines	Available	Depends on the robot application: for social and service robots, deadlines can be fuzzy
The types of jobs/tasks can have dedicated schedule parameters	Processes have priorities which can be directly compared	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)

5.2 Summary

The conducted study regards the problem of task management of an autonomous service robot. The problem arises in multiple applications and limits the level of robots autonomy. It significantly affects the systems ordered by multiple users who do not agree their requests for the system.

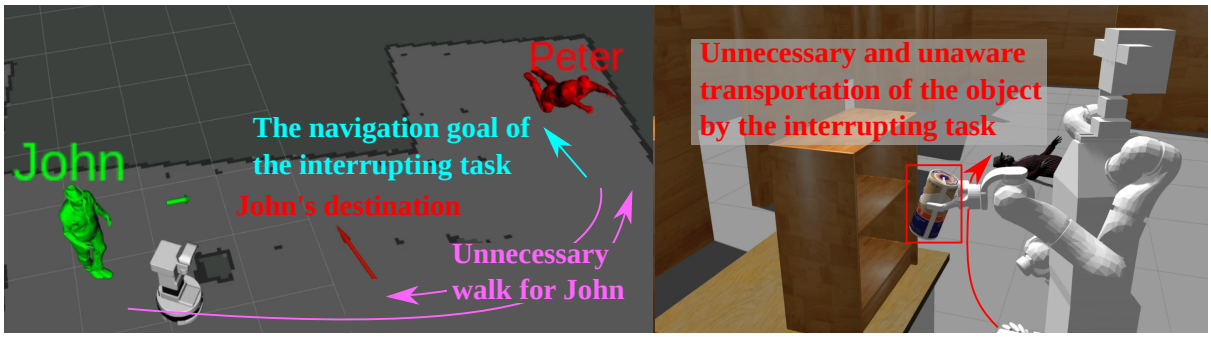
There are various applications for robots, and their control system's versatility can be either integrated or modular (Fig. 1.2). The problem of user request management arises in both types; however, various parts of the system resolve it in each of them. In integrated systems, the system's overall behaviour is a result of a composition of elementary actions; thus, the user request prioritisation and management is a duty of the composer (either the system developer, if the composition is static or a planning algorithm). In a modular versatility case, elementary actions compose independently developed task modules deployed on a robot according to the user requests. Unfortunately, with the benefit of enhanced expandability of modular versatility systems, the following disadvantage arises. Actions of the robot are coordinated in each of the tasks; however, when tasks switch, the subsequent task's actions are not coordinated with the actions completed by the previous task. As a solution to this problem, the role of harmoniser arise. Accordingly, the model of the task-related part of the system requires extra behaviours to enable safe task switches and coordination.

Uncoordinated task switch can injure the environment inhabitants, lead to surroundings damage or even disasters. Exemplary consequences that were avoided by application of the TaskER model in the situations described in this dissertation verification were:

- an unnecessary walk of a human in case of the task switch while guiding the human.
- an unintended and unmaintained transport of the can grabbed while executing one task, leading to a spill off the liquid from the can. It is possible because the tasks are independent, and the subsequent task has no information about the object being carried and cannot configure the safe motion constraints.

Visualisation of the consequences prevented by the TaskER model implementation in the verification systems is shown in Fig. 5.2 Besides the above possible consequences, there are more situations where a switch of independent tasks without care can be dangerous:

- leaving a cooker on for a long time without supervision,
- an unsafe configuration of robot arms or a dangerous mobile base speed for a situation (e.g. carrying a sharp object or liquid container collected during the previous, interrupted task),



The consequences prevented by a prudent task switch during verification:

1) John is irritated due to unnecessary walk
2) John can be injured if he is disabled or has trouble walking

1) Spill of liquid contained in the can (because the interrupting task does not adapt robot motion speed to the situation)

Figure 5.2: Visualisation of the consequences prevented by the tasks management defined in the TaskER model.

- completion of an interrupting task can be impossible because the robot resources were not freed before the task switch, and the interrupting task has no knowledge of how to free it (e.g. the interrupted task directed the robot to grab an object, but the interrupting task does not implement any manipulation actions),

- a deadlock of a robot (e.g., a task directed the robot to a classified room using a secret code, but the interrupting task does not have access to the code).

The above summary regards the first thesis:
"A service robot can compromise the convenience and safety of humans and the safety of robots/objects in their environment while it switches independent tasks".

The related work section shows different algorithms for planning robot actions and respecting some criteria (e.g. deadlines, priorities) of requests; however, the whole problem of harmonisation is not resolved. The algorithms of this kind mostly resolve the request management problem in integrated versatility systems. The modular versatility systems feature a barrier between domains of the tasks. The barrier complicates the solution to the request management problem in modular versatility systems. Additionally, analysis of the related works discloses a lack of a model for system managing user requests with dynamic criteria. Such criteria are expected in service robotics, where the users' preferences change over time.

This dissertation constitutes the problem of request management in modular versatility systems and proposes a solution to it. The analysis of the problem resulted in the concept of the harmonisation procedure (Fig. 2.4). The procedure involves a configurable scheduling algorithm that uses schedule parameters to compute schedule decisions. Furthermore, the analysis

revealed three crucial orthogonal classification criteria: range, constancy and factuality of the schedule parameters (Fig. 2.2). The analysis described in Sec. 2.1.2 shows the need for ease configuration of the scheduling algorithm and the parameters for various robot applications. For example, different algorithms (and their parameters) should be used for robots working in a factory and versatile service robots working in an elderly house. In the first case, the algorithm can be configured to maximise production quantity and quality, and in the other, to maximise convenience of encountered humans. The above summary regards the second thesis:
"In various robot applications, different algorithms and parameters for task scheduling should be used".

The solution to the stated problem is the model for modular versatility systems featured with user request management. The robot controller structure is variable, and it is composed of multiple agents derived from various agent classes. The scheduling algorithm implemented in the Task Harmoniser Agent is configurable with ease, and the schedule parameters can be of various classes.

In this study, sample use cases describe the desired behaviours of a robot control system faced with the request management problem. Then, based on the use cases, six requirements for the control system are formulated. Next, the model is described. The model adapts and extends the known variable structure for robot control systems, namely, the RAPP architecture, to satisfy the previously specified requirements. Finally, the TaskER framework was implemented, enabling the efficient creation of the robot tasks and injecting them as modules into the systems that utilise the proposed model. The algorithm that defines the task scheduling strategy differs among robots, applications, and environments; hence, the TaskER framework applies the algorithm as an interchangeable function.

The model that is described in this dissertation 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 system's current state and the progress of each task.

Model of the Dynamic Agent, being an encapsulation of a task in the proposed approach, defines constraints and specifies the tasks to enable task harmonisation feature to the system. A significant part of the task is roughly constrained by the model proposed in this work. For example, the methods to update the task in $\mathtt{S}_{dy}^{upTsk}$, generation of suspension strategy in $\mathtt{S}_{dy}^{genSusp}$ and task context-dependent parameters computation for the scheduling algorithm can be adjusted. The methods to plan actions executed by the task is another important problem in robotics, and any which results with a plan representable with FSM can be used. In the model, it is defined when and by which agent the planning method is executed. Various planning methods require different world models. As the proposed task management method does not depend on any

planning methods or world model, any can be used. Furthermore, the described work can be applied even in the 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, `da`-class agents:

1. compute task-dependent parameters that are used for the task scheduling,
2. suspend their operation safely in the case of a task switch,
3. update their plans before 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 example system satisfies the specified requirements. Moreover, they demonstrate the benefits for a system that follows the model. The above summary regards the third thesis:
"The proposed model of a service robot control system (named TaskER) fosters safety and user's convenience while the robot manages independent, suspendable tasks in a dynamic environment, and the model is configurable in the aspects of:

- *the task scheduling algorithm,*
- *the parameters used to compute schedule,*
- *the interfaces for task requests,*
- *the set of tasks available for robots in the system."*

5.3 Future work

The study revealed that the task switch problem in modular versatility systems is not resolved, and several research objectives require further investigation. One of them is verification of the proposed model with a robot control architecture featured with semantic planning (such as ROSPlan [94]) and to call the planner in both $\mathtt{S}_{dy}^{upTsk}$ and $\mathtt{S}_{dy}^{genSusp}$ to generate $\mathrm{FSM}_{dy,exec}$ and $\mathrm{FSM}_{dy,es}$, respectively. With this feature, the system's tasks will use planned strategies for task suspension and update actions. Furthermore, the following opened problems were identified:

1. a method for choosing f_{ha}^{shdl} and f_{dy}^{compSP} functions and schedule parameters for various tasks, applications and environment classes (e.g., using an objective optimisation algorithm such as [111]),
2. a method for classification of the task stages into suspendable and blocking,

3. an extension of the PDDL standard to support the proposed approach and enable semantic planners to consider the task scheduling problem while composing $\text{FSM}_{dy,exeTsk}$ and $\text{FSM}_{dy,exeSusp}$ FSMs,

4. a metric to evaluate task harmonisation quality.

5. a self-tuning procedure for the scheduling algorithm to learn desired task switch strategy [112], [113], e.g. based on machine learning, and

6. integration of the model with one of the formally defined task models featured with a consistent language for scheduling algorithm specification and planning method [114].

Bibliography

[1] R. F. Adler and R. Benbunan-Fich, "Juggling on a high wire: Multitasking effects on performance," *International Journal of Human-Computer Studies*, vol. 70, no. 2, pp. 156–168, 2012.

[2] L. L. Bowman, L. E. Levine, B. M. Waite, and M. Gendron, "Can students really multitask? an experimental study of instant messaging while reading," *Computers & Education*, vol. 54, no. 4, pp. 927–931, 2010.

[3] D. L. Strayer, J. M. Watson, and F. A. Drews, "Cognitive distraction while multitasking in the automobile," in *Psychology of Learning and Motivation*, vol. 54, Elsevier, 2011, pp. 29–58.

[4] A. Mazur, "Trajectory tracking control in workspace-defined tasks for nonholonomic mobile manipulators," English, *Robotica*, vol. 28, no. 1, pp. 57–68, Jan. 2010, Prawa autorskie - Copyright © Cambridge University Press 2009; Ostatnia aktualizacja - 2015-08-15. [Online]. Available: `https://search.proquest.com/scholarly-journals/trajectory-tracking-control-workspace-defined/docview/211837370/se-2?accountid=27375`.

[5] B. Cybulski, A. Wegierska, and G. Granosik, "Accuracy comparison of navigation local planners on ros-based mobile robot," in *2019 12th International Workshop on Robot Motion and Control (RoMoCo)*, IEEE, 2019, pp. 104–111.

[6] W. Kowalczyk, M. Michałek, and K. Kozłowski, "Trajectory tracking control with obstacle avoidance capability for unicycle-like mobile robot," *Bulletin of the Polish Academy of Sciences. Technical Sciences*, vol. 60, no. 3, pp. 537–546, 2012.

[7] T. Gawron and M. M. Michałek, "Planning the waypoint-following task for a unicycle-like robot in cluttered environments," *Journal of Automation Mobile Robotics and Intelligent Systems*, vol. 9, 2015.

[8] M. M. Michałek, "Cascade-like modular tracking controller for non-standard n-trailers," *IEEE Transactions on Control Systems Technology*, vol. 25, no. 2, pp. 619–627, 2016.

[9] M. Visinsky, J. Cavallaro, and I. Walker, "Robotic fault detection and fault tolerance: A survey," *Reliability Engineering & System Safety*, vol. 46, no. 2, pp. 139–158, 1994, ISSN: 0951-8320. DOI: https://doi.org/10.1016/0951-8320(94)90132-5.

[10] S. Haddadin, "Physical safety in robotics," in *Formal Modeling and Verification of Cyber-Physical Systems: 1st International Summer School on Methods and Tools for the Design of Digital Systems, Bremen, Germany, September 2015*, R. Drechsler and U. Kühne, Eds. Wiesbaden: Springer Fachmedien Wiesbaden, 2015, pp. 249–271, ISBN: 978-3-658-09994-7. DOI: 10.1007/978-3-658-09994-7_9.

[11] M. Mohanan and A. Salgoankar, "A survey of robotic motion planning in dynamic environments," *Robotics and Autonomous Systems*, vol. 100, pp. 171–185, 2018.

[12] W. Domski, A. Mazur, and M. Kaczmarek, "Extended factitious force approach for control of a mobile manipulator moving on unknown terrain," *Journal of Intelligent & Robotic Systems*, vol. 93, no. 3, pp. 699–712, 2019.

[13] T. S. Tadele, T. de Vries, and S. Stramigioli, "The safety of domestic robotics: A survey of various safety-related publications," *IEEE Robotics Automation Magazine*, vol. 21, no. 3, pp. 134–142, 2014. DOI: 10.1109/MRA.2014.2310151.

[14] W. Domski and A. Mazur, "Emergency control of a space 3R manipulator in case of one joint failure," in *2017 22nd International Conference on Methods and Models in Automation and Robotics (MMAR)*, 2017, pp. 384–389. DOI: 10.1109/MMAR.2017.8046858.

[15] M. Ben-Ari and F. Mondada, "Robots and their applications," in *Elements of Robotics*, Springer, 2018, pp. 1–20.

[16] T. S. Dahl and M. N. K. Boulos, "Robots in health and social care: A complementary technology to home care and telehealthcare?" *Robotics*, vol. 3, no. 1, pp. 1–21, 2014.

[17] A. Mazur, L. Podsędkowski, A. Szymański, and M. Zawierucha, "Compliance force control for polish cardiosurgical manipulator RobIn Heart," in *9th International Workshop on Robot Motion and Control*, 2013, pp. 92–97. DOI: 10.1109/RoMoCo.2013.6614590.

[18] E. Ruiz, R. Acuña, 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 *Robotics Symposium and Competition (LARS/LARC), 2013 Latin American*, IEEE, 2013, pp. 55–60.

[19] 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. 55 885–55 897, 2019.

[20] J. Malec, "Some thoughts on robotics for education," in *AAAI spring symposium on robotics and education*, 2001.

[21] I. Zubrycki and G. Granosik, "Teaching robotics with cloud tools," in *International Conference on Robotics and Education RiE 2017*, Springer, 2017, pp. 301–310.

[22] L. Guevara, M. M. Michałek, and F. A. Cheein, "Headland turning algorithmization for autonomous N-trailer vehicles in agricultural scenarios," *Computers and Electronics in Agriculture*, vol. 175, p. 105 541, 2020.

[23] A. Lopez, R. Paredes, D. Quiroz, G. Trovato, and F. Cuellar, "Robotman: A security robot for human-robot interaction," in *Advanced Robotics (ICAR), 2017 18th International Conference on*, IEEE, 2017, pp. 7–12.

[24] T. Chen and C. Kemp, "A direct physical interface for navigation and positioning of a robotic nursing assistant," *Advanced Robotics*, vol. 25, pp. 605–627, Mar. 2011. DOI: 10.1163/016918611X558243.

[25] 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 *2009 IEEE international conference on automation and logistics*, IEEE, 2009, pp. 289–294.

[26] I. Zubrycki, M. Kolesiński, and G. Granosik, "A participatory design for enhancing the work environment of therapists of disabled children," in *2016 25th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, IEEE, 2016, pp. 781–786.

[27] I. Zubrycki and G. Granosik, "Understanding therapists' needs and attitudes towards robotic support. the roboterapia project," *International Journal of Social Robotics*, vol. 8, no. 4, pp. 553–563, 2016.

[28] S. A. Frennert, A. Forsberg, and B. Östlund, "Elderly people's perceptions of a telehealthcare system: Relative advantage, compatibility, complexity and observability," *Journal of technology in human services*, vol. 31, no. 3, pp. 218–237, 2013.

[29] J. Aldrich, D. Garlan, C. Kästner, C. Le Goues, A. Mohseni-Kabir, I. Ruchkin, S. Samuel, B. Schmerl, C. S. Timperley, M. Veloso, *et al.*, "Model-based adaptation for robotics software," *IEEE Software*, vol. 36, no. 2, pp. 83–90, 2019.

[30] C. Zieliński, M. Stefańczyk, T. Kornuta, M. Figat, W. Dudek, W. Szynkiewicz, W. Kasprzak, J. Figat, M. Szlenk, T. Winiarski, K. Banachowicz, T. Zielińska, E. G. Tsardoulias, A. L. Symeonidis, F. E. Psomopoulos, A. M. Kintsakis, P. A. Mitkas, A. Thallas, S. E. Reppou, G. T. Karagiannis, K. Panayiotou, V. Prunet, M. Serrano, J.-P. Merlet, S. Arampatzis, A. Giokas, L. Penteridis, I. Trochidis, D. Daney, and M. Iturburu, "Variable structure robot control systems: The rapp approach," *Robotics and Autonomous Systems*, vol. 94, pp. 226–244, 2017, ISSN: 0921-8890. DOI: 10.1016/j.robot.2017.05.002.

[31] 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," *Science Robotics*, vol. 5, no. 40, 2020. DOI: 10.1126/scirobotics.abb5589.

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

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

[34] 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 *2017 E-Health and Bioengineering Conference (EHB)*, 2017, pp. 591–594.

[35] W. Szynkiewicz, W. Kasprzak, C. Zieliński, W. Dudek, M. Stefańczyk, A. Wilkowski, and M. Figat, "Utilisation of Embodied Agents in the Design of Smart Human–Computer Interfaces – A Case Study in Cyberspace Event Visualisation Control," *Electronics*, vol. 9, no. 6, p. 976, 2020. DOI: 10.3390/electronics9060976.

[36] A. Zalewski, K. Borowa, and A. Ratkowski, "On cognitive biases in architecture decision making," in *European Conference on Software Architecture*, Springer, 2017, pp. 123–137.

[37] A. Zalewski and S. Kijas, "Beyond atam: Early architecture evaluation method for large-scale distributed systems," *Journal of Systems and Software*, vol. 86, no. 3, pp. 683–697, 2013.

[38] T. Kornuta, C. Zieliński, and T. Winiarski, "A universal architectural pattern and specification method for robot control system design," *Bulletin of the Polish Academy of Sciences: Technical Sciences*, vol. 68, no. No. 1 February, pp. 3–29, 2020. DOI: 10.24425/bpasts.2020.131827.

[39] S. Friedenthal, A. Moore, and R. Steiner, *A practical guide to SysML: the systems modeling language*. Morgan Kaufmann, 2014.

[40] 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 - 379, 2020, ISSN: 2079-9292. DOI: 10.3390/electronics9020379.

[41] G. Schuh, V. Zeller, M.-F. Stroh, and P. Harder, "Finding the right way towards a cps–a methodology for individually selecting development processes for cyber-physical systems," in *Working Conference on Virtual Enterprises*, Springer, 2019, pp. 81–90.

[42] A. R. Da Silva, "Model-driven engineering: A survey supported by the unified conceptual model," *Computer Languages, Systems & Structures*, vol. 43, pp. 139–155, 2015.

[43] E. Seidewitz, "What models mean," *IEEE Software*, vol. 20, no. 5, pp. 26–32, 2003. DOI: 10.1109/MS.2003.1231147.

[44] T. Kühne, "Matters of (meta-) modeling," *Software & Systems Modeling*, vol. 5, no. 4, pp. 369–385, 2006.

[45] B. Selic, "The pragmatics of model-driven development," *IEEE software*, vol. 20, no. 5, pp. 19–25, 2003.

[46] J. Bézivin and O. Gerbé, "Towards a precise definition of the omg/mda framework," in *Proceedings 16th Annual International Conference on Automated Software Engineering (ASE 2001)*, IEEE, 2001, pp. 273–280.

[47] C. Zanabria, F. P. Andrén, T. I. Strasser, and W. Kastner, "A model-driven and ontology-based engineering approach for smart grid automation applications," in *IECON 2019-45th Annual Conference of the IEEE Industrial Electronics Society*, IEEE, vol. 1, 2019, pp. 6635–6641.

[48] P. Iyenghar and E. Pulvermueller, "A model-driven workflow for energy-aware scheduling analysis of iot-enabled use cases," *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 4914–4925, 2018.

[49] A. Dorri, S. S. Kanhere, and R. Jurdak, "Multi-agent systems: A survey," *Ieee Access*, vol. 6, pp. 28 573–28 593, 2018.

[50] W. Dudek, W. Szynkiewicz, and T. Winiarski, "Nao Robot Navigation System Structure Development in an Agent-Based Architecture of the RAPP Platform," in *Recent Advances in Automation, Robotics and Measuring Techniques*, R. Szewczyk, C. Zieliński, and M. Kaliczyńska, Eds., ser. Advances in Intelligent Systems and Computing (AISC), vol. 440, Springer, 2016, pp. 623–633. DOI: 10.1007/978-3-319-29357-8_54.

[51] ——, "Cloud computing support for the multi-agent robot navigation system," English, *Journal of Automation Mobile Robotics and Intelligent Systems*, vol. 11, no. 2, pp. 67–74, 2017. DOI: 10.14313/JAMRIS_2-2017/18.

[52] M. Michalek and K. Kozlowski, "Trajectory tracking for a threecycle mobile robot: The vector field orientation approach," in *Proceedings of the 44th IEEE Conference on Decision and Control*, IEEE, 2005, pp. 1119–1124.

[53] 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*, Springer, 2014, pp. 497–506.

[54] K. Łakomy and M. M. Michałek, "Robust output-feedback VFO-ADR control of underactuated spatial vehicles in the task of following non-parametrized paths," *European Journal of Control*, vol. 58, pp. 258–277, 2021.

[55] E. Gat, R. P. Bonnasso, R. Murphy, *et al.*, "On three-layer architectures," *Artificial intelligence and mobile robots*, vol. 195, p. 210, 1998.

[56] W. Dudek, K. Banachowicz, W. Szynkiewicz, and T. Winiarski, "Distributed NAO robot navigation system in the hazard detection application," in *21th IEEE International Conference on Methods and Models in Automation and Robotics, MMAR'2016*, IEEE, IEEE, 2016, pp. 942–947. DOI: 10.1109/MMAR.2016.7575264.

[57] C. Zieliński, W. Szynkiewicz, M. Figat, M. Szlenk, T. Kornuta, W. Kasprzak, M. Stefańczyk, T. Zielińska, and J. Figat, "Reconfigurable control architecture for exploratory robots," in *10th International Workshop on Robot Motion and Control (RoMoCo)*, K. Kozłowski, Ed., IEEE, 2015, pp. 130–135. DOI: 10.1109/RoMoCo.2015.7219724.

[58] W. Dudek and T. Winiarski, "Scheduling of a robot's tasks with the TaskER framework," *IEEE Access*, vol. 8, pp. 161 449–161 471, 2020. DOI: 10.1109/ACCESS.2020.3020265.

[59] J. Pagès, L. Marchionni, and F. Ferro, "Tiago: The modular robot that adapts to different research needs," 2016.

[60] Institute of Control and Computation Engineering, WUT, *Velma robot description*, WUT, Institute of Control and Computation Engineering, 2020. [Online]. Available: https://www.robotyka.ia.pw.edu.pl/robots/velma/.

[61] T. Winiarski, J. Sikora, D. Seredyński, and W. Dudek, "Daimm simulation platform for dual-arm impedance controlled mobile manipulation," in *2021 7th International Conference on Automation, Robotics and Applications (ICARA)*, IEEE, 2021, pp. 180–184. DOI: 10.1109/ICARA51699.2021.9376462.

[62] INCARE, *INCARE project web page*, WUT, Institute of Control and Computation Engineering, 2020. [Online]. Available: https://www.robotyka.ia.pw.edu.pl/projects/incare/.

[63] S. Bedaf, G. J. Gelderblom, D. S. Syrdal, H. Lehmann, H. Michel, D. Hewson, F. Amirabdollahian, K. Dautenhahn, and L. Witte, "Which activities threaten independent living of elderly when becoming problematic: Inspiration for meaningful service robot functionality," *Disability and rehabilitation. Assistive technology*, vol. 9, Oct. 2013. DOI: 10.3109/17483107.2013.840861.

[64] J. F. Engelberger, *Robotics in practice: management and applications of industrial robots*. Springer Science & Business Media, 2012.

[65] K. M. Tsui and H. A. Yanco, "Assistive, rehabilitation, and surgical robots from the perspective of medical and healthcare professionals," in *AAAI Workshop on Human Implications of Human-Robot Interaction, Technical Report WS-07-07 Papers from the AAAI 2007 Workshop on Human Implications of HRI*, 2007.

[66] F. Hegel, M. Lohse, A. Swadzba, S. Wachsmuth, K. Rohlfing, and B. Wrede, "Classes of applications for social robots: A user study," in *RO-MAN 2007-The 16th IEEE International Symposium on Robot and Human Interactive Communication*, IEEE, 2007, pp. 938–943.

[67] M. Vukobratovic, *Dynamics and robust control of robot-environment interaction*. World Scientific, 2009, vol. 2.

[68] V. Kumar, E. Todorov, and S. Levine, "Optimal control with learned local models: Application to dexterous manipulation," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2016, pp. 378–383.

[69] B. P. Gerkey and M. J. Matarić, "A formal analysis and taxonomy of task allocation in multi-robot systems," *The International Journal of Robotics Research*, vol. 23, no. 9, pp. 939–954, 2004.

[70] 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)," *Robotics and Computer-Integrated Manufacturing*, vol. 33, pp. 42–55, 2015.

[71] W. Dudek, *Harmonizing a complex robot tasks with TaskER*, WUT, Institute of Control and Computation Engineering, 2020. [Online]. Available: `https://vimeo.com/403391725`.

[72] ——, *Example system execution using a real TIAGo robot*, WUT, Institute of Control and Computation Engineering, 2021. [Online]. Available: `https://vimeo.com/521756050`.

[73] ——, *TaskER framework verification in mobile manipulation tasks*, WUT, Institute of Control and Computation Engineering, 2020. [Online]. Available: `https://vimeo.com/483480184`.

[74] R. Volpe, I. Nesnas, T. Estlin, D. Mutz, R. Petras, and H. Das, "The claraty architecture for robotic autonomy," in *Aerospace Conference, 2001, IEEE Proceedings.*, IEEE, vol. 1, 2001, pp. 1–121.

[75] S. Dennis, L. Alex, L. Matthias, and S. Christian, "The smartmdsd toolchain: An integrated mdsd workflow and integrated development environment (ide) for robotics software," 2016.

[76] A. Steck and C. Schlegel, "SmartTCL: An execution language for conditional reactive task execution in a three layer architecture for service robots," in *Int. Workshop on DYnamic languages for RObotic and Sensors systems (DYROS/SIMPAR)*, 2010, pp. 274–277.

[77] M. Stenmark, J. Malec, and A. Stolt, "From high-level task descriptions to executable robot code," in *Intelligent Systems'*, Springer, 2015, pp. 189–202.

[78] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg, "A survey of research on cloud robotics and automation," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 398–409, Apr. 2015, ISSN: 1545-5955. DOI: `10.1109/TASE.2014.2376492`.

[79] J. Riazuelo, J. Civera, and J. M. M. Montiel, "C2TAM: A cloud framework for cooperative tracking and mapping," *Robotics and Autonomous Systems*, vol. 62, no. 4, pp. 401–413, Apr. 2014.

[80] J. Salmerón-Garcıa, P. Íñigo-Blasco, F. Dıaz-del-Rıo, and D. Cagigas-Muñiz, "A trade-off analysis of a cloud-based robot navigation assistant using stereo image processing," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 444–454, Apr. 2015. DOI: `10.1109/TASE.2015.2403593`.

[81] R. Arumugam, V. R. Enti, L. Bingbing, W. Xiaojun, K. Baskaran, F. F. Kong, A. S. Kumar, K. D. Meng, and G. W. Kit, "DAvinCi: A cloud computing framework for service robots," in *Robotics and Automation (ICRA)*, IEEE, 2010, pp. 3084–3089.

[82] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA workshop on open source software*, vol. 3, 2009.

[83] O. S. R. Foundatioin, *Robot Operating System*, `http://ros.org/`, [Online; accessed 21-May-2015].

[84] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*, IEEE, 2010, pp. 1–10.

[85] G. Mohanarajah, D. Hunziker, R. D'Andrea, and M. Waibel, "Rapyuta: A cloud robotics platform," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 481–493, Apr. 2015. DOI: `10.1109/TASE.2014.2329556`.

[86] R. Alami, A. Clodic, V. Montreuil, E. A. Sisbot, and R. Chatila, "Toward human-aware robot task planning.," in *AAAI spring symposium: to boldly go where no human-robot team has gone before*, 2006, pp. 39–46.

[87] T. Winiarski, K. Banachowicz, M. Walęcki, and J. Bohren, "Multibehavioral position–force manipulator controller," in *21th IEEE International Conference on Methods and Models in Automation and Robotics, MMAR*, IEEE, 2016, pp. 651–656. DOI: `10.1109/MMAR.2016.7575213`.

[88] M. Stilman, "Task constrained motion planning in robot joint space," in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2007, pp. 3074–3081.

[89] 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 *IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2005, pp. 3329–3335.

[90] A. Mohseni Kabir and M. Veloso, "Robot task interruption by learning to switch among multiple models," Jul. 2018, pp. 4943–4949. DOI: `10.24963/ijcai.2018/686`.

[91] J. Bohren and S. Cousins, "The smach high-level executive [ros news]," *IEEE Robotics & Automation Magazine*, vol. 17, no. 4, pp. 18–20, 2010.

[92] S. Amador, S. Okamoto, and R. Zivan, "Dynamic multi-agent task allocation with spatial and temporal constraints," in *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

[93] A. Nouman, I. F. Yalciner, E. Erdem, and V. Patoglu, "Experimental evaluation of hybrid conditional planning for service robotics," in *International Symposium on Experimental Robotics*, Springer, 2016, pp. 692–702.

[94] 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 *Twenty-Fifth International Conference on Automated Planning and Scheduling*, 2015.

[95] 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.

[96] S. Park, J.-H. Kim, and G. Fox, "Effective real-time scheduling algorithm for cyber physical systems society," *Future Generation Computer Systems*, vol. 32, pp. 253–259, 2014.

[97] H. Mora, J. F. Colom, D. Gil, and A. Jimeno-Morenilla, "Distributed computational model for shared processing on cyber-physical system environments," *Computer Communications*, vol. 111, pp. 68–83, 2017.

[98] A. Węgierska, K. Andrzejczak, M. Kujawiński, and G. Granosik, "Using labview and ros for planning and coordination of robot missions, the example of erl emergency robots and university rover challenge competitions," *Journal of Automation Mobile Robotics and Intelligent Systems*, vol. 13, 2019.

[99] J. Yao, X. Xu, and X. Liu, "Mixcps: Mixed time/event-triggered architecture of cyber–physical systems," *Proceedings of the IEEE*, vol. 104, no. 5, pp. 923–937, 2016.

[100] J. Kim, K. Lakshmanan, and R. R. Rajkumar, "Rhythmic tasks: A new task model with continually varying periods for cyber-physical systems," in *Proceedings of the 2012 IEEE/ACM Third International Conference on Cyber-Physical Systems*, IEEE Computer Society, 2012, pp. 55–64.

[101] J. Xu and D. L. Parnas, "Scheduling processes with release times, deadlines, precedence and exclusion relations," *IEEE Transactions on software engineering*, vol. 16, no. 3, pp. 360–369, 1990.

[102] E. D. Jensen, C. D. Locke, and H. Tokuda, "A time-driven scheduling model for real-time operating systems.," in *Rtss*, vol. 85, 1985, pp. 112–122.

[103] F. Zhang, K. Szwaykowska, W. Wolf, and V. Mooney, "Task scheduling for control oriented requirements for cyber-physical systems," in *Real-Time Systems Symposium*, IEEE, 2008, pp. 47–56.

[104] 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," *Transactions on Emerging Telecommunications Technologies*, e3770, 2019.

[105] M. Stefańczyk and W. Kasprzak, "Model-based 3D object recognition in RGB-D images," in *Bridging the Semantic Gap in Image and Video Analysis*, H. Kwaśnicka and L. C. Jain, Eds. Springer, 2018, pp. 73–96, ISBN: 978-3-319-73891-8. DOI: 10.1007/978-3-319-73891-8_5.

[106] T. Winiarski, W. Kasprzak, M. Stefańczyk, and M. Walęcki, "Automated inspection of door parts based on fuzzy recognition system," in *2016 21st International Conference on Methods and Models in Automation and Robotics (MMAR)*, 2016, pp. 478–483. DOI: 10.1109/MMAR.2016.7575182.

[107] W. Szynkiewicz, "Robot grasp synthesis under object pose uncertainty," *Journal of Automation Mobile Robotics and Intelligent Systems*, vol. 9, no. 1, pp. 53–61, 2015. DOI: 10.14313/JAMRIS_1-2015/7.

[108] ——, "Skill-based bimanual manipulation planning," *Journal of Telecommunications and Information Technology*, no. 4, pp. 54–62, 2012. [Online]. Available: http://yadda.icm.edu.pl/baztech/element/bwmeta1.element.baztech-article-BATA-0018-0007/c/httpwww_itl_waw_plczasopismajtit2012454.pdf.

[109] W. Kasprzak, W. Szynkiewicz, D. Zlatanov, and T. Zielińska, "A hierarchical CSP search for path planning of cooperating self-reconfigurable mobile fixtures," *Engineering Applications of Artificial Intelligence*, vol. 34, pp. 85–98, 2014, ISSN: 0952-1976. DOI: https://doi.org/10.1016/j.engappai.2014.05.013.

[110] K. Jeffay, "Scheduling sporadic tasks with shared resources in hard-real-time systems," North Carolina Univ At Chapel Hill Dept Of Computer Science, Tech. Rep., 1990.

[111] J. Ota, "Goal state optimization algorithm considering computational resource constraints and uncertainty in task execution time," *Robotics and Autonomous Systems*, vol. 57, no. 4, pp. 403–410, 2009.

[112] 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. 77 308–77 327, 2019.

[113] T. Winiarski, W. Dudek, M. Stefańczyk, Ł. Zieliński, D. Giełdowski, and D. Seredyński, "An intent-based approach for creating assistive robots' control systems," *arXiv preprint arXiv:2005.12106*, 2020.

[114] G. Gierse, T. Niemueller, J. Claßen, and G. Lakemeyer, "Interruptible task execution with resumption in golog," in *Proceedings of the Twenty-second European Conference on Artificial Intelligence*, 2016, pp. 1265–1273.