Towards an Easy Use Case Implementation in Social Robotics

Alba Gragera[1], **Carmen Díaz de Mera**[1], **Alberto Tudela**[2], **Alejandro Cruces**[2], **Fernando Fernández**[1], **Ángel García-Olaya**[1]

[1]Universidad Carlos III de Madrid, Spain
[2]Universidad de Málaga, Spain
{agragera, cdiazmer}@pa.uc3m.es, {ajtudela, acruces}@uma.es, {ffernand, agolaya}@inf.uc3m.es

Abstract

Social Autonomous Robotics aims to deploy robots in scenarios characterized by an intensive and continuous interaction with humans. The use of Automated Planning (AP) within a control architecture has been proposed to specify the behavior of robotic platforms in such environments. However, the design of AP models is a time consuming task developed by domain experts and engineers. It involves a large knowledge acquisition process, where the use case description has to be properly defined by specifying the different tasks the robot can perform. In this paper we contribute a system to graphically design the use case and to configure the robot platform adapted to the desired execution. Through state transition diagrams, users can define the different tasks the robot can perform, which are automatically translated to the standard Planning Domain Definition Language (PDDL). It also permits an easy generation of the configuration files to setup the robotic platform. The proposed framework has been tested in a real environment in a retirement home.

Introduction

In recent years, there is a growing interest in robots that operate in public environments. Much of research in this field is focused on Social Autonomous Robotics (SAR) (Breazeal, Dautenhahn, and Kanda 2016), which must act according to the data collected from sensors and show flexible capabilities and robust behaviours, useful in dynamic and changing environments (Ingrand and Ghallab 2017). Automated Planning (AP) (Ghallab, Nau, and Traverso 2004) has been previously used to achieve this autonomous behaviour (Bandera et al. 2016; Cashmore et al. 2015; Chen, Yang, and Chen 2016; González, Pulido, and Fernández 2017; Mohseni-Kabir, Veloso, and Likhachev 2020; Rajan and Py 2012; Tran et al. 2017) by using a problem solver and a control architecture: the problem solver creates a plan of actions to be performed and the control architecture deals with execution and monitoring.

Despite these examples, developing such systems to work autonomously in Human-Robot Interaction (HRI) scenarios is still a challenging task (Tapus, Mataric, and Scassellati 2007). The models used have to be represented explicitly and declaratively, so that they can be processed by specific reasoning engines (Korf 1985). It requires both knowledge on the application field and expertise in robot programming, in the frame of a time-consuming knowledge engineering process (Kambhampati 2007) between the domain experts and software developers. This difficulty is applicable to any other control technique, not only AP, and becomes a bottleneck for developers and an entry barrier for novice users to deploy Social Robotics use cases, creating a growing demand for frameworks helping users to program their robots in a seamless manner (Kramer and Scheutz 2007).

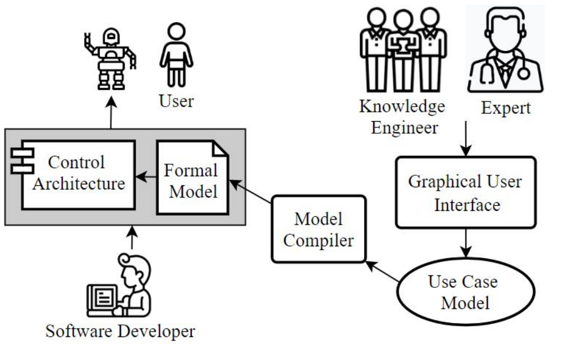

Figure 1: Proposed development process using the graphical interface

In this paper we present our work to alleviate the development of SAR use cases. Given AP as a paradigm to describe the use case and a control architecture implementing its model, we contribute with a novel tool to represent the expected behaviour of the robot and, in addition, to set the control architecture ready to be connected with the robotic platform (Figure 1). Such models are automatically translated into the Planning Domain Definition Language (PDDL) formalization (McDermott et al. 1998), along with the configuration files to be injected into an AP-based control architecture. This process is based on AP concepts but also suitable for users with limited expertise. We tested our implementation through two use cases drawn from the needs encountered in a retirement home. They were implemented from scratch and exposed to execution with a robot interacting with senior adults in a real environment, aimed to be the first steps towards a long term adherence and acceptability in HRI (Iglesias et al. 2020).

In the remainder of the paper we present the motivation of this work, introducing two use cases we have deployed in a retirement home, followed by the background on AP and control architectures. After that, we detail the main concepts to generate models based on AP, whose development is supported by the novel interface we present. The formal models are automatically extracted from the tool, which have to be injected into the control architecture as explained. The results obtained in the execution on the retirement home are shown. Finally, we discuss the conclusions and future lines.

Motivation

In this work we captured two real use cases, both arising from the need to optimize the time of healthcare professionals on their daily tasks on a retirement home. Activities such as informing patients about the meals of the day or helping residents to talk to their relatives require staff members and a precise schedule, which is sometimes difficult to reconcile. Their needs have been captured in the use cases below.

Announcer The residence has a pre-established monthly lunch and dinner menu with different options depending on the required diet. The proposal is to have a robot in charge of announcing the menu when lunchtime or dinnertime approaches. This is the most simple scenario, in which the robot is waiting in the charging base until the selected time when the menu has to be announced. Then, it goes to the main room and plays a sound to notify its arrival. The menu is stated by the robot twice. After that, it goes back to the charging base.

Videocall In many cases, relatives of patients in a retirement home do not visit them as much as they would like due to lack of time or geographical distance, which was exacerbated by the COVID-19 restrictions. To alleviate that, it is desired to establish a video call system in which relatives of residents can choose a time slot to make a video call. This process can be automatized by using a robot with a tablet instead of relying on a staff member. When the call time approaches, the robot leaves its charging base and goes to the different halls where the person who has the videocall could be, calling them and telling them to follow the robot. Once announced, it will go to the point defined to have the call. When the call is ended, the robot leaves and goes back to the base. A similar use case was deployed in a hospital, where, due to COVID restrictions, the robot also needs to be disinfected after each call. It was implemented though the state machine shown in Figure 2. The interface of the videocall system was previously implemented for this purpose, but the use case in the retirement home had not yet been implemented because of the complexity of developing a new state machine to control the robot.

These use cases were initially proposed to be teleoperated or implement though state machines. The first option was discarded as it still needs staff to control the robot manually, defeating the purpose of alleviating their work. State machines seemed to be a better option, but they are tough to implement and update, being also difficult to be understood by general users, requiring expertise in programming. By contrast, a proposal based on AP with a control architecture provides an easier and more flexible implementation. Main concepts are explained in next section.

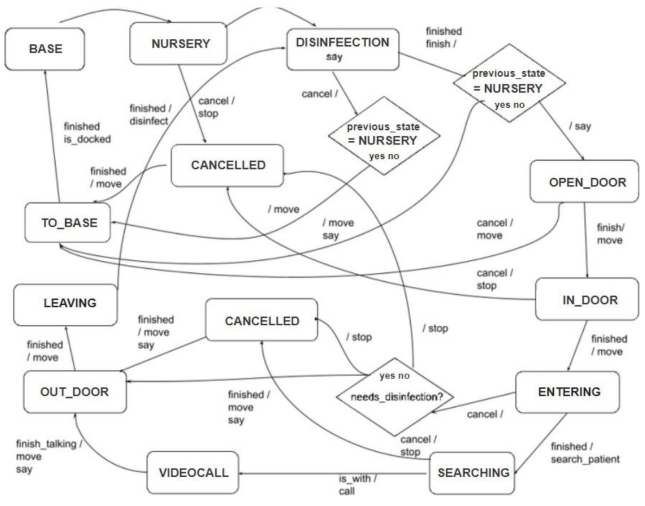

Figure 2: State machine implementing a videocall use case in a hospital, needing disinfection.

Background

The implementation of Social Robotic systems based on AP relies on a formal model and a control architecture, breaking it into two different levels: the high level allows a declarative definition of the use case, whereas the low level deals with monitoring and execution.

A Classical Planning (Ghallab, Nau, and Traverso 2004) task consists of driving a system from a given initial state into a state where the goals are achieved, by applying actions whose effects are deterministic and known (Geffner and Bonet 2013). Formally, a classical planning task can be defined as a tuple $\Pi = \langle S, A, I, G \rangle$, where S is the set of states, A is the set of actions, $I \subseteq S$ defines the initial state and $G \subseteq S$ specifies the goal to achieve. Actions $a \in A$ are tuples $\{pre_a, add_a, del_a\}$. For a_i to be applicable in a state $s_i \in S$, $pre(a_i) \subseteq s_i$ and $s_{i+1} = s_i \cup add(a) \setminus del(a)$. A plan $\pi = \{a_1, a_2, \ldots, a_n\}$ is a solution of Π if it is applicable in I and results in a state s_n such as $G \subseteq s_n$. Classical planning assumes that an action's effects are deterministic and known, without external events interrupting the plan. However, in real environments actions may fail and other agents can change the state of the world. Although there are AP paradigms that consider such non-determinism, a common approach is to tackle the inherent world uncertainty by replanning; if the state changes and the remaining plan cannot be applied, a new plan is created from the current state (Yoon, Fern, and Givan 2007; Geffner and Bonet 2013).

A planning approach with replanning upon failure requires monitoring and execution control architectures. Examples of them are PELEA (Alcázar et al. 2010) and ROSPLAN (Cashmore et al. 2015), both using Classical Planning (Ghallab, Nau, and Traverso 2004). They typically involve a planner and a formal planning model to generate the

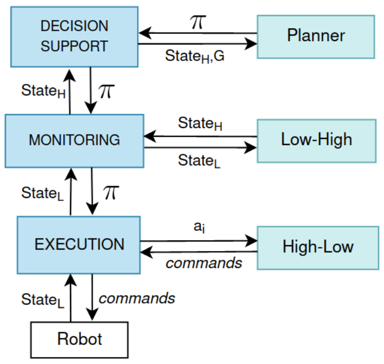

Figure 3: Architecture of PELEA

sequence of actions to be performed, while checking the correct execution of the initial plan. Each action is sent to the robotic platform, assuming that no events will interrupt the execution. To verify whether the plan goes as expected, external information of the environment is obtained from sensors. If divergences are found between the expected value of the state and its observed value, the current plan may not be longer valid, triggering a replanning process to replace it by a new plan to manage the current situation. Even if this approach seems rather simplistic it has been used in real HRI environments with good results (Bandera et al. 2016; Cashmore et al. 2015; Chen, Yang, and Chen 2016; González, Pulido, and Fernández 2017; Mohseni-Kabir, Veloso, and Likhachev 2020; Rajan and Py 2012; Tran et al. 2017).

MLARAS (Multi-layered Architecture for Autonomous Systems), a similar architecture to the ones mentioned, based on Automated Planning, was developed in the context of the NAOTHERAPIST project (González, Pulido, and Fernández 2017). This architecture integrates planning, execution, monitoring, replanning and learning in different layers of abstraction, with a translation between layers. Normally a high-level layer for deliberation and a low-level layer for the information that the robot can directly work with are used. To perform the high-level deliberation and the translation between layers, MLARAS uses PELEA as a sub-architecture. The general process is shown in Figure 3, where the high level planner returns the desired plan $\pi = \{a_1, a_2, \ldots, a_n\}$. Each $a_i \in \pi$ is translated into low-level commands that the robot can execute and sent to the robotic platform. The information from the sensors of the robot is translated into high-level predicates for monitoring purposes. Even if it was implemented for a specific project, MLARAS was developed to be a generic architecture, so that it can be integrated in any robot and with any use case easily by means of a few configuration files. This architecture was used to integrate the use case definitions of the announcer and videocall with any robotic platform.

High Level

Users have to identify and specify the concepts that shape the SAR model following the formalism of a planning task. In this section we show some examples of the information needed for the proposed use cases as formalized in PDDL.

Types are abstract representations of the kind of objects which are involved in the execution: places, people, items, etc. Object-type hierarchies can be created to define generalizations and specializations. For the use cases considered, only the types `patient`, to identify different residents, and `point`, useful to define the main locations, were required.

Predicates are mostly used to create relations between objects to build information useful during the execution. Using predicates logic, a declaration like `(room-number patient01 room01)` represents the relation between patients and their room number. It is the job of the knowledge engineer to recognize this information and specify it as part of the knowledge of the use case.

States are sets of facts that represent the current information the agent must reason with. Interpreted as a conjunctive formula, a state is any subset of relevant predicates that must be true to consider if the platform is in such state. In most cases it is not necessary for the states to be fully defined: reasoning usually involves just some of the facts present in the state, so a partial state definition is enough to reason about it. Figure 4 shows an example state in which the robot is looking for a patient to make the videocall.

```
(robot-location hall01)
(current-patient patient01)
(in-hall patient01 hall01)
```

Figure 4: Example of a partial state definition when searching for a patient.

Actions are seen as the high level operators to be carried out by the robot. They are applied to make changes in the environment according to a desired purpose. Through their use facts are added, removed or modified, giving rise to new situations that have to be managed again by the robot. A simple way to elicit actions is to ask the expert what characteristics the scenario must hold to carry out the action and how it changes after its execution.

Goals must be defined to specify the real purpose of the robot. They may be a simple predicate or to be composed by a set of facts that the agent must meet. All the decision-making process is guided to accomplish them.

The specification of these components shapes an agent-based model which, injected into a cognitive architecture, has the ability to reason about the behaviour it should perform to act as an autonomous system.

Supporting Users in High Level Development

Creating complete models based on the previous abstractions is a not straightforward task. It is usually carried out by engineers with AP background and guided by experts in the domain where the robot will operate. To relieve the development process we propose workflows as a visual knowledge representation of the use case. Figure 5 shows the videocall use case as modelled in the proposed interface, where blue boxes are partial states composed by the facts that the scenario must hold to execute the action, and actions are the edges connecting them, representing the expected transitions between states.

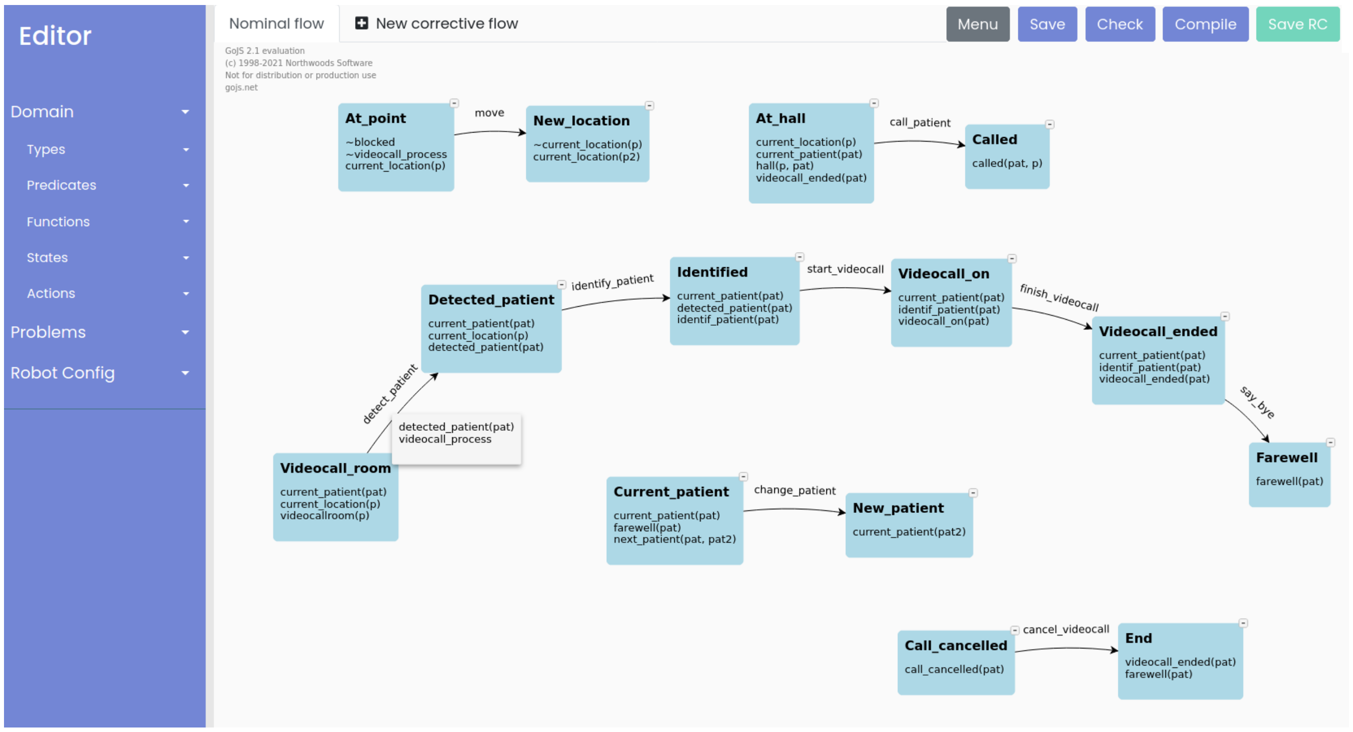

Figure 5: Social Robotics use case modelled in the graphical tool

In most cases the sequence of actions to solve the problem is unknown or tough to represent using approaches as state machines, which require a full specification of the states and all possible transitions, as shown in Figure 2. By contrast, in our proposed workflow representation states and transitions do not need to be fully specified. Instead, partial definitions with some of the facts can be enough to reason about the state and are also easier to define by the user. Let $\mathcal{S}_\pi$ be the set of all totally specified states that can be reached while the robot is operating and $\mathcal{P}_s$ the set of partial states defined by the user, the nominal behaviour of the use case is depicted as a directed graph composed by partial states an actions $\langle \mathcal{P}_s, A \rangle$ in such a way that applying $a \in A$ in $ps_t \in \mathcal{P}_s$ results in ps_{t+1}. The directed graph $\langle \mathcal{P}_s, A \rangle$ can be design using the concept of *options* (Sutton, Precup, and Singh 1999):

Definition 1 *An **option** is a tuple $o = \langle \mathcal{I}, \pi_o, \beta \rangle$ where $\mathcal{I} \subseteq \mathcal{P}_s$ is an initiation set, π_o a partial policy and β the termination condition .*

An option is applicable only if the state $ps_t \in \mathcal{I}$. If the option is taken the next action a_t is selected, making a transition to state ps_{t+1}. If the action a_{t+1} is not applicable in such state, it reaches the termination condition and has the opportunity to select any other option. In contrast to macro-actions (Korf 1985), which specify a sequence of actions that has to be executed as a whole, options can be partially executed and resumed from any point. If it is desired to go back to the option, it does not have to be taken from the beginning, but can be taken at any point. *Options* assume that all states where an option might continue are also states where the option may be taken (Sutton, Precup, and Singh 1999).

Similar to partial order plans (Weld 1994), options are graphically specified by depicting the causal links between actions $a_i \xrightarrow{p} a_n$, in which p is both an effect of a_i and a precondition of a_n, establishing an order constraint $a_i < a_n$.

However, neither transitions between options nor termination conditions are explicitly stated, and order restrictions are only used to easily depict and understand the current use case, they are not taken into account during the planning process. Users only have to define them as a reasonable outline of the different use case tasks, and it is the task of the reasoning engine to choose the most promising options choice to solve the use case.

Considering the exemplified use case in Figure 5, after calling a patient there exist options to move to another hall to call them again or to go to the videocall room to start the videocall. In addition, options can be interrupted by other options thanks to the replanning process. Let us consider a robot identifying the person who is going to have a call, but in this moment the request is cancelled. Such option is no longer valid, and the replanning process will include the *call cancelled* option instead. The tool also presents features to handle exogenous events breaking the nominal behaviour of the use case. Interruptions such as the person leaving during the call, if the robot is stuck, etc., could be considered, along with *checkpoint* states from which the plan is desired to be recovered in such cases, instead of the point where the exogenous event took place. Although this feature is not used in our current use cases, it is interesting to take it into account for future larger scenarios.

Automated Planning Compilation

Once the model has been graphically created by defining the possible options of the use case, it can be translated to the AP standard language PDDL 2.1 to obtain its formal description. Although many versions of PDDL (Haslum et al. 2019) with different levels of expressiveness exist, all of them separate the model definition into a domain, containing the object types, the predicates and functions, as well as the set of available actions, and the problem, specifying the initial state and the goals to achieve. Following the PDDL approach, an action is defined as a deterministic operator composed by parameters, preconditions and effects. Action's preconditions are set as the predicates of the state where the action starts, since they represent the relevant information needed to execute the action. Effects are taken from the user specification of which predicates change after the action execution. Parameters are taken from the object's types involved in the action. Figure 5 shows the PDDL formalization for the *detect-patient* action (in Figure 5). Since the model does not contain exogenous events or checkpoint states defined, it shows the most simple translation. If exogenous events had been defined, they will be automatically added to each domain action as negative precondition, since actions can only be executed in case of nominal behaviour. The checkpoint recovery is managed through special actions which remove all intermediate effects added between two different checkpoints, which forces to restart the nominal flow from the last checkpoint visited.

```
(:action detect-patient
  :parameters(?pat - patient ?p - point)
  :precondition (and (current-patient ?pat)
                     (current-location ?p)
                     (videocallroom ?p))
   :effect (and (detected-patient ?pat)
                (videocall-process)))the
```

Figure 6: PDDL code of the `detect-patient` action

From High to Low Level and Vice-Versa

When using deliberation architectures such as the aforementioned MLARAS, knowledge is generally divided in two levels of abstraction. The external information that comes from the sensors of the robot is low-level, and should be translated into a higher level of abstraction so that the planner system integrated in the architecture can reason with it. In the case of PELEA, which uses planners based on PDDL, the low-level information would be translated into high-level PDDL predicates, which represent the state of the system.

In the other direction, high-level actions defined with PDDL should be translated into the low-level instructions that a robot can immediately execute, which are considered low-level actions. For example, in a robot with color-coded eyes, when saying a speech the high-level action *say* could be decomposed into changing the eyes to an appropriate color and then saying the actual speech.

These translations are generally hard-coded directly into the architecture by technical experts, which makes it difficult to change. To alleviate this problem, a declarative language was proposed in (González et al. 2018). This declarative language eases the translation, but it is still necessary to understand the Extended Backus–Naur Form (EBNF) description of the grammar for both translations (Figure 7, Figure 8).

```
High: SAY(speech)
Lows: play_sound()
      show_subtitles(?speech)
      say(?speech)
```

Figure 7: High to Low declaration.

To make the translation from high to low level and vice-versa even easier, it has been integrated in the graphical user interface used to model use cases, so that when the high-level behaviour of the use case is specified with sequences of actions, the translation of these actions can be defined too. In the same way, the values of the low-level variables that come from the sensors of the robot will be translated into high-level predicates. These translations are done by means of if/then statements, depending on the value of a low-level variable, predicates can be added or deleted from the state of the problem. The interface takes such conditions input by the user and converts them into the high-to-low and low-to-high translation files as described in (González et al. 2018). This way the output of the interface can be directly integrated into the architecture of the robot, as detailed in the next section. Different catalogues describing what the robot can do are also provided by its designers as input. These catalogues are in comma separated values (csv) format and divided in:

- LowActions: Includes all the low level instructions that the robot can execute. Each line includes the name of the action and the required parameters separated by commas. The user will then be able to choose between these actions to detail the decomposition of the high-level operators.
- Variables: Includes all the low level variables that the robot perceives from its sensors. In the graphical interface the user will have to specify all the effects that would happen depending on the values of these variables as a series of conditions. The effects could be to add a PDDL predicate to the current state, to delete it, or to change the value of the numerical predicates (functions) of the state.
- Speech: Optional catalogue to specify the possible utterances that the robot can say when speaking. For each line, the id of the speech, its type and the actual text is specified. There can be different speeches that share the same type. This catalogue is only necessary if the robot has the ability to speak and a low-level action to do so, where one of the parameters is the type of speech to say.
- Animations: Optional catalogue to define the names of the animations implemented in the robot. These anima-

```
If: $call_cancelled is true
   add(call_cancelled $patient, state_pddl)
```

Figure 8: Low to High declaration.

tions will be used as parameter of a low level action to execute the animations.

These catalogues will be associated to a type of robot, so that when a user designs a use case with the interface they will be able to choose among several available robots.

Integration in an AP Control Architecture

The main objective of the graphical user interface is that the designers of a use case can do so directly in the tool, where the output of the interface can be directly integrated in an autonomous robot. For the demonstration, the cognitive architecture MLARAS and the two use cases previously stated have been implemented in a social robot developed for geriatric assessment in the context of the CLARC project (Martínez et al. 2018). This robot works by means of the ROBOCOMP framework (Manso et al. 2010), a component-oriented robotic architecture whose main aim is to ease the development of robotic frameworks. This middleware automates the communication among different components through TCP/IP using Ice (Internet Communication Engine) interfaces. The components also share a common world view through the CORTEX architecture, which uses a Deep State Representation (DSR) graph with symbolic and geometric information (Bustos et al. 2015)

The planning architecture MLARAS is integrated as a component of this framework, together with an *agent*, another component that can read and modify the DSR. The agent will receive the low level actions of the planner and write them into the DSR, so that the other components of the robot will read the information and react accordingly, executing the instructions. In the same way, the agent will read the low-level information written in the DSR and communicate it to the planner, by means of the low-level variables. As ROBOCOMP takes care of the implementation of the communication, to integrate the planner in the robot's architecture the main task is to define the interfaces between the planner and its agent, one from the planner to the agent and vice-versa.

The full architecture of MLARAS in the CLARC robot is shown in Figure 9. A high level layer is used to define the use case by means of the graphical user interface as proposed in this paper. From the interface, the PDDL domain and problem definition is obtained, as well as the translation from high to low level and vice-versa as explained in the previous section. This output is received by the medium layer, which is the deliberation layer of the architecture. The Execution component centralises all operations, performing the translation between high and low levels. This deliberation layer uses PELEA as a sub-architecture. The Execution component communicates directly with the ROBOCOMP agent, getting access to the world view of the DSR, as discussed.

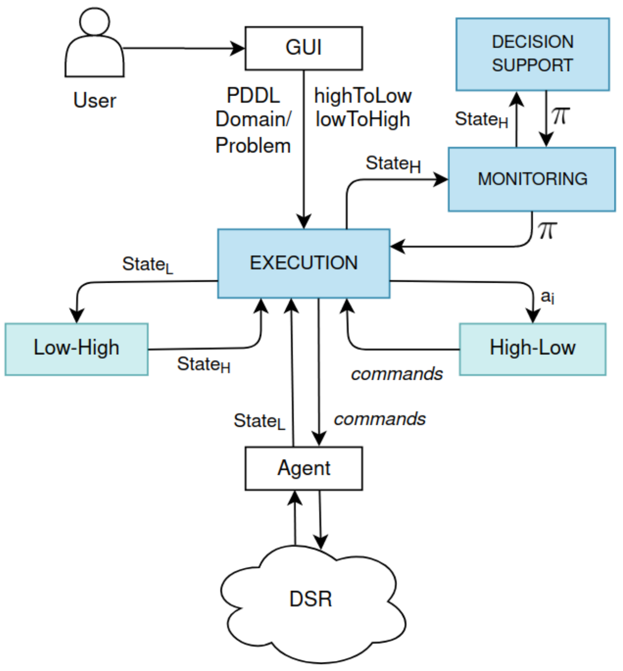

Figure 9: Architecture of the deliberative module of the CLARC robot

Field Trials

The present system has been fully tested on a CLARC robot (Voilmy et al. 2017), previously used in Comprehensive Geriatric Assesment procedures, helping healthcare professional in their daily tasks. The proposed use cases have been designed in the graphical interface and tested on a retirement home with real users, exposing the generated plans to execution in a real environment.

The high level design and implementation of the use cases were carried out during one week at the Vitalia Teatinos Residence (Málaga, Spain) in collaboration with Universidad de Málaga. The generated files from the graphical interface were injected into the control architecture, where High Level planning is performed via the Metric-FF (Hoffmann 2003) PDDL2.1 compliant planner. During the use cases the robot interacted with 5 people with no experience in activities with robots, in addition to the residents located in the corridors and halls. In all executions the robot was totally autonomous, including navigation. Although initially some executions failed and required a manual restart of the platform, by the third day all interactions were correctly executed as described below.

Announcer

It was the simplest use case and the first one implemented. Its design contains two main points where the robot can be, the charging base and the hall where the menu has to be announced. Given the generated domain and problem, the planner returns as solution the grounded sequence of actions shown in Figure 10. They represent the high level actions taken by the robot, which are sent in commands to the robotic platform. Such decomposition is detailed in Figure 11. It represents a good example about how the same high level action may implement different behaviours in its corresponding low level, depending on the current situation. In this case, the movement has a particularity if the point to which it is directed is the charging base, where the robot will also introduce a speech called "rest" which warns about the end of the use case, and that the robot is going to charge.

```
0: (MOVE CHARGING_BASE HALL_ANNOUNCE)
1: (PLAY_SOUND HALL_ANNOUNCE)
2: (SAY_MENU HALL_ANNOUNCE)
3: (MOVE HALL_ANNOUNCE CHARGING_BASE)
```

Figure 10: Resulting plan for the announcer use case

```
High: move(point1, point2)
Lows: print("MOVE TO " + $point2)
      move($point2)
High: move(point1, point2), $point2 is charging_base
Lows: print("MOVE TO " + $point2)
      say("rest")
      move($point2)
High: say_menu(point)
Lows: print("SAY_MENU " + $point)
      say("menu")
High: play_sound(point1)
Lows: print("PLAY_SOUND")
      playSound()
```

Figure 11: High to low level decomposition for the announcer use case

Videocall

For this demonstration, we establish various halls where the videocall has to be announced. Residents of the retirement home spend the day at different places according to their level of dependency. This information is an input to build the initial state, so when receiving a call with a patient it is already known where they could be. The other location previously established is the hall where the call will be held, which is given by the retirement home. The initial plan generated is shown in Figure 12, where the call is assumed to be executed normally. But in this use case it was also tested an example of a change in the expected state of the world: if the call is cancelled, the robot receives the current new state and replan according to that, stopping the use case and going back to the charging base.

Although from an AP point of view the generated plans are simple (low number of actions), for more complex scenarios our approach also would provide shorter deployment

```
0: (MOVE CHARGING_BASE HALL_ANNOUNCE)
1: (CALL_PATIENT HALL_ANOUNCE PATIENT01)
2: (MOVE HALL_ANOUNCE HALL_CALL)
3: (DETECT_PATIENT PATIENT01 HALL_CALL)
4: (IDENTIFY_PATIENT PATIENT01)
5: (START_VIDEOCALL PATIENT01)
6: (FINISH_VIDEOCALL PATIENT01)
7: (SAY_BYE PATIENT01)
8: (MOVE HALL_CALL CHARGING_BASE)
```

Figure 12: Resulting plan for the videocall use case

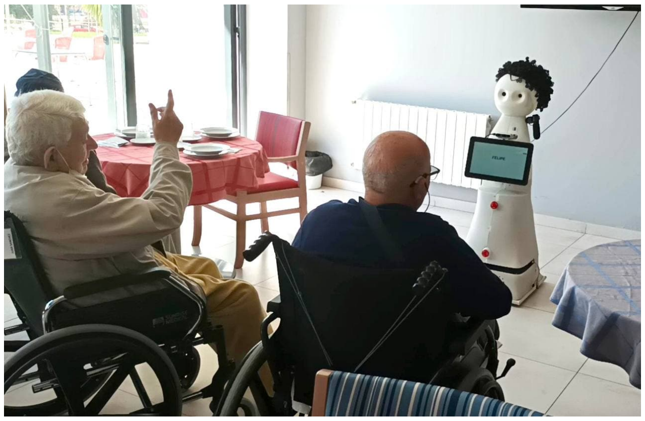

Figure 13: CLARC robot during the use case testing at the retirement home.

times compared to other techniques. In just one week we were able to design two prototypes, integrate the control architecture on the robotic platform and test both uses cases in a real environment. Once all the components are working and MLARAS is integrated in the robotic platform, this time may be reduced for future developments, as it is only necessary to change the PDDL domain and problem definition, and the high to low decomposition of the new actions defined. Those actions are sent in commands by consulting the given high to low and low to high translation files. In addition, these models are easier to generate and understand compared, for instance, to the Finite State Machine shown in Figure 2.

Related Work

In this work we propose a link between the graphical workflows and the PDDL formalism through the definition of partial options. The concept of *option* is borrowed from (Sutton, Precup, and Singh 1999) and is based on *macro-operators* (Fikes, Hart, and Nilsson 1972): sets of actions that are usually applied sequentially in a plan. But macro-operators focus on the actions applied, abstracting out the partial states traversed, while in our approach both actions and states are equally important and must be specified by the user. Partial options have some similarities with timelines (Jonsson et al. 2000) and partial plans (Minton, Bresina, and Drummond 1994). A timeline is a temporal description of the different values a state variable takes. In timeline-based planning the use case is modeled in terms of a series of state variables and temporal constrains among their values. For example, variable A can only take value a_i, after variable B has taken value b_j. Initial state and goals are expressed as current and future values of some of the variables, respectively. The planner has then to *fill the gaps* in the timelines to reach the desired values. In a similar way, our planner has to *fill the gaps* in the partial options interleaving other options. The main difference is that there is no concept of *action* in timeline-based planning, while in our approach actions are a crucial element. Partial options can also be seen as totally ordered partial plans, as actions in each partial option must appear in the plan one after the other, and there can be some other actions in between. But partial plans do not make any assumption about the states the plan traverses, while our partial options force the plan to reach the states included in them.

Related to the knowledge engineering process behind the development of AP models for Social Robotics systems, various tools have been introduced to support the implementation of such planning domains. In addition to some number of PDDL editors[1] which require deep knowledge about the specification language, we can find in literature systems characterized by automatically translating the resulting visual model into its PDDL formalization (Vaquero et al. 2013; Simpson, Kitchin, and McCluskey 2007; Hatzi et al. 2010). Although all of these systems use different graphical representations to specify planning domains, they focus on users with a deep knowledge on software engineering and become unmanageable for large domains.

Instead, the design and implementation of robotic platforms are usually covered by specific toolkits (Pot et al. 2009; Touretzky and Tira-Thompson 2011; Kim and Jeon 2007; Jackson 2007). Some of them provide visual programming utilities for novice users, but they are restricted to hardware configurations, not being able to build general models. So a major missing feature in current robotic development tools is the possibility to have both visual programming and general model generators, in addition to mechanisms to properly manage the human-robot interaction. These are some of the highlighted features of the interface we propose.

Conclusions and Future Work

Automated Planning has been already reported in the literature as a general approach to SAR development. However, it is not extensively used due to the bottleneck of model development, where an extensive knowledge engineering process is required beforehand. In this paper we propose a tool where non-experts can participate in the design of complex real world use cases by the definition of the possible *options* to execute by the robotic platform. It depicts the expected behavior of the robotic platform through simple conducts which can be interleaved to create more sophisticated and robust behaviors. The representation of these *options* provides a simpler and a more general approach against other techniques such as FSM's, particularly in complex use cases or where there is no fixed sequence of actions to solve the use case. In addition, we include the generation of files to set up the control architecture embedded in the robotic platform, without the need of hard-coding components which have to be changed for each different use case.

As they have been implemented, the use cases described are launched manually, having to decide which one of them the robot must perform at any time. As future lines, we propose implementing another layer of deliberation that uses AP to determine the use case, checking for opportunities to perform small tasks in between execution.

[1]http://editor.planning.domains/

Acknowledgments

This work has been partially funded by the EU ECHORD++ (FP7-ICT-601116) project, FEDER/Ministerio de Ciencia e Innovación - Agencia Estatal de Investigación RTI2018-099522-B-C43 project. SUSTAIN ("EU project DIH-Hero SUSTAIN) is funded by Digital Innovation Hub Healthcare Robotics platform, EU Horizon 2020 research and innovation programme under grant agreement No 825003. It has been also supported by the Madrid Government (Comunidad de Madrid, Spain) under the Multiannual Agreement with UC3M in the line of Excellence of University Professors (EPUC3M17) in the context of the V PRICIT (Regional Programme of Research and Technological Innovation), and by the National Research Project RTI2018-099522-B-C41. The authors would like to thank the people living and working at Vitalia Teatinos retirement house for their collaboration.

References

Alcázar, V.; Guzmán, C.; Prior, D.; Borrajo, D.; Castillo, L.; and Onaindía, E. 2010. PELEA: Planning, learning and execution architecture. In *Workshop of Planning and Scheduling*.

Bandera, A.; Bandera, J. P.; Bustos, P.; Calderita, L. V.; Duenas, A.; Fernández, F.; Fuentetaja, R.; Garcıa-Olaya, A.; Garcıa-Polo, F. J.; González, J. C.; et al. 2016. CLARC: a Robotic Architecture for Comprehensive Geriatric Assessment. In *WAF*, 1–8.

Breazeal, C.; Dautenhahn, K.; and Kanda, T. 2016. Social Robotics. In *Springer Handbook of Robotics*, Springer Handbooks, 1935–1972. Springer.

Bustos, P.; Manso, L. J.; Rubio, J. P. B.; Romero-Garcés, A.; Calderita, L. V.; Marfil, R.; and Bandera, A. 2015. A Unified Internal Representation of the Outer World for Social Robotics. In *Robot 2015: Second Iberian Robotics Conference - Advances in Robotics, Lisbon, Portugal, 19-21 November 2015, Volume 2*, volume 418 of *Advances in Intelligent Systems and Computing*, 733–744. Springer.

Cashmore, M.; Fox, M.; Long, D.; Magazzeni, D.; Ridder, B.; Carrera, A.; Palomeras, N.; Hurtós, N.; and Carreras, M. 2015. ROSPlan: Planning in the Robot Operating System. In *ICAPS*, 333–341.

Chen, K.; Yang, F.; and Chen, X. 2016. Planning with Task-Oriented Knowledge Acquisition for a Service Robot. In *IJCAI*, 812–818.

Fikes, R.; Hart, P. E.; and Nilsson, N. J. 1972. Learning and Executing Generalized Robot Plans. *Artif. Intell.*, 3(1-3): 251–288.

Geffner, H.; and Bonet, B. 2013. *A Concise Introduction to Models and Methods for Automated Planning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. ISBN 9781608459698.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory & Practice*. Elsevier.

González, J. C.; Pulido, J. C.; and Fernández, F. 2017. A three-layer planning architecture for the autonomous control of rehabilitation therapies based on social robots. *Cognitive Systems Research (CSR)*, 43: 232–249.

González, J. C.; Pulido, J. C.; and Fernández, F. 2017. A three-layer planning architecture for the autonomous control of rehabilitation therapies based on social robots. *Cognitive Systems Research*, 43: 232–249.

González, J. C.; Garcia, J.; Fuentetaja, R.; Olaya, A.; and Fernández, F. 2018. From High to Low Level and Vice-Versa: A New Language for the Translation between Abstraction Levels in Robot Control Architectures. In *3rd Workshop on Semantic Policy and Action Representations for Autonomous Robots (SPAR)*.

Haslum, P.; Lipovetzky, N.; Magazzeni, D.; and Muise, C. 2019. *An Introduction to the Planning Domain Definition Language*. Synthesis Lectures on Artificial Intelligence and Machine Learning.

Hatzi, O.; Vrakas, D.; Bassiliades, N.; Anagnostopoulos, D.; and Vlahavas, I. P. 2010. A visual programming system for automated problem solving. *Expert Syst. Appl.*, (6): 4611–4625.

Hoffmann, J. 2003. The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables. *J. Artif. Intell. Res.*, 20: 291–341.

Iglesias, A.; Viciana-Abad, R.; Pérez-Lorenzo, J. M.; Ting, K. L. H.; Tudela, A. J.; Marfil, R.; Dueñas-Ruiz, A.; and Rubio, J. P. B. 2020. Towards long term acceptance of Socially Assistive Robots in retirement houses: use case definition. In *2020 IEEE International Conference on Autonomous Robot Systems and Competitions, ICARSC 2020, Ponta Delgada, Portugal, April 15-17, 2020*, 134–139. IEEE.

Ingrand, F.; and Ghallab, M. 2017. Deliberation for autonomous robots: A survey. *Artif. Intell.*, 247: 10–44.

Jackson, J. 2007. Microsoft robotics studio: A technical introduction. *IEEE Robotics Autom. Mag.*, 14(4): 82–87.

Jonsson, A. K.; Morris, P. H.; Muscettola, N.; Rajan, K.; and Smith, B. 2000. Planning in Interplanetary Space: Theory and practice. In *Proceedings of AIPS*, 177–186. Breckenridge, CO, USA.

Kambhampati, S. 2007. The Challenges of Planning with Incomplete and Evolving Domain Models. In *Proceedings of AAAI*, 1601–1605.

Kim, S. H.; and Jeon, J. W. 2007. Programming lego mindstorms nxt with visual programming. In *ICCAS*, 2468–2472.

Korf, R. E. 1985. Macro-Operators: A Weak Method for Learning. *Artif. Intell.*, 26(1): 35–77.

Kramer, J. F.; and Scheutz, M. 2007. Development environments for autonomous mobile robots: A survey. *Auton. Robots*, (2): 101–132.

Manso, L.; Bachiller, P.; Bustos, P.; Núñez, P.; Cintas, R.; and Calderita, L. 2010. RoboComp: A Tool-Based Robotics Framework. In *Simulation, Modeling, and Programming for Autonomous Robots - Second International Conference, SIMPAR 2010, Darmstadt, Germany, November 15-18, 2010. Proceedings*, volume 6472 of *Lecture Notes in Computer Science*, 251–262. Springer.

Martínez, J.; Romero-Garcés, A.; Suarez, C.; Marfil, R.; Ting, K. L. H.; Iglesias, A.; García, J.; Fernández, F.; Dueñas-Ruiz, A.; Calderita, L. V.; Bandera, A.; and Rubio, J. P. B. 2018. Towards a robust robotic assistant for Comprehensive Geriatric Assessment procedures: updating the CLARC system. In *27th IEEE International Symposium on Robot and Human Interactive Communication, RO-MAN 2018, Nanjing, China, August 27-31, 2018*, 820–825. IEEE.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL-the planning domain definition language.

Minton, S.; Bresina, J.; and Drummond, M. 1994. Total-Order and Partial-Order Planning : A Comparative Analysis. *Journal of Artificial Intelligence Research*, 2: 227–262.

Mohseni-Kabir, A.; Veloso, M.; and Likhachev, M. 2020. Efficient Robot Planning for Achieving Multiple Independent Partially Observable Tasks That Evolve over Time. In *Proceedings of the ICAPS*, 202–211.

Pot, E.; Monceaux, J.; Gelin, R.; and Maisonnier, B. 2009. Choregraphe: a graphical tool for humanoid robot programming. In *RO-MAN*, 46–51. IEEE.

Rajan, K.; and Py, F. 2012. T-REX: partitioned inference for AUV mission control. *Further advances in unmanned marine vehicles*.

Simpson, R. M.; Kitchin, D. E.; and McCluskey, T. L. 2007. Planning domain definition using GIPO. *Knowledge Eng. Review*, (2): 117–134.

Sutton, R. S.; Precup, D.; and Singh, S. P. 1999. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artif. Intell.*, 112(1-2): 181–211.

Tapus, A.; Mataric, M. J.; and Scassellati, B. 2007. Socially assistive robotics [Grand Challenges of Robotics]. *IEEE Robot. Automat. Mag.*, (1): 35–42.

Touretzky, D. S.; and Tira-Thompson, E. J. 2011. The Tekkotsu robotics development environment. In *ICRA*, 6084–6089.

Tran, T. T.; Vaquero, T. S.; Nejat, G.; and Beck, J. C. 2017. Robots in retirement homes: Applying off-the-shelf planning and scheduling to a team of assistive robots. *J. Artif. Intell. Res.*, 58: 523–590.

Vaquero, T. S.; Silva, J. R.; Tonidandel, F.; and Beck, J. C. 2013. itSIMPLE: towards an integrated design system for real planning applications. *Knowledge Eng. Review*, 28(2): 215–230.

Voilmy, D.; Suarez, C.; Romero-Garcés, A.; Reuther, C.; Pulido, J. C.; Marfil, R.; Manso, L. J.; Ting, K. L. H.; Iglesias, A.; González, J. C.; García, J.; Olaya, A. G.; Fuentetaja, R.; Fernández, F.; Dueñas-Ruiz, A.; Calderita, L. V.; Bustos, P.; Barile, T.; Rubio, J. P. B.; and Bandera, A. 2017. CLARC: A Cognitive Robot for Helping Geriatric Doctors in Real Scenarios. In *ROBOT 2017: Third Iberian Robotics Conference - Volume 1, Seville, Spain, November 22-24, 2017*, volume 693 of *Advances in Intelligent Systems and Computing*, 403–414. Springer.

Weld, D. S. 1994. An Introduction to Least Commitment Planning. *AI Mag.*, 15(4): 27–61.

Yoon, S. W.; Fern, A.; and Givan, R. 2007. FF-Replan: A Baseline for Probabilistic Planning. In *ICAPS*, 352. AAAI.