Supporting the Formalization of Use Cases in Social Robotics

Alba Gragera, Ángel García-Olaya, Fernando Fernández
Universidad Carlos III de Madrid, Leganés, 28911, Madrid, Spain
agragera@pa.uc3m.es, {agolaya, ffernand}@inf.uc3m.es

Abstract

The implementation of use cases in Social Autonomous Robotics is a complex and time consuming task to be developed by domain experts and engineers, involving a large knowledge acquisition process. To achieve correct operation on the robotic platform, the resulting use case description must also be formalized taking into account stochastic events that may occur in the real world. There are several control architectures based on Automated Planning (AP) to deploy robotic use cases, where the standard Planning Domain Description Language (PDDL) is assumed. This manuscript focuses on bridging the gap between the domain expert definition of the use case and its formalization in PDDL. We propose a novel tool which facilitates the description of the use case through state transition diagrams representing nominal behaviours, exogenous events and some extra features. From this diagram, the system automatically generates the PDDL files, which can be injected in a standard control architecture to setup the robotic platform. The tests show that the proposed system is feasible, encouraging the use of AP in Social Robotics and further research about tools for facilitating the development of AP problems to experts and non-experts in the field. In addition, our approach is general enough to also formalize classical planning tasks.

1 Introduction

Social Robotics (Breazeal, Dautenhahn, and Kanda 2016) is the branch of robotics where autonomous systems have to interact with people, expecting them to have a natural behaviour in order to reduce the sense of frustration in the user when dealing with a machine. However, real world's uncertainty and stochasticity make hard to ensure the proper performance of the robot, where actions can be interrupted by external events. For that reason, developing robots that deal autonomously in these scenarios is a challenging task (Tapus, Mataric, and Scassellati 2007) addressed in the study of Human-Robot Interaction. Several control systems have been proposed over time for that purpose, such as Finite State Machines (Lera et al. 2018) or distributed systems (Prenzel, Feuser, and Graser 2005). Although these methods perform well, they are highly domain dependent and their use requires extra effort to create new applications since they have to be built from scratch in a complex development process, specially in those ones with sophisticated operating modes.

Some approaches in the literature (Cashmore et al. 2015; González, Pulido, and Fernández 2017; Rajan and Py 2012; Tran et al. 2017; Bandera et al. 2016) have dealt with this through Automated Planning (AP) (Ghallab, Nau, and Traverso 2004), enabling the robot control and coordination by using a problem solver and a control architecture. Nevertheless, AP has not been widely used in Social Robotics due, among other reasons, to the complexity of identifying and formalizing all possible actions that the robot must implement or how to recover the normal behaviour from an unexpected situation. Extracting these requirements to model the robot behaviour also presents a problem: it is a time consuming task which involves an intensive knowledge engineering process (Kambhampati 2007) between the domain expert (for instance, therapists or clinicians in assistive environments) and engineers. Therefore, modelling AP domains for Social Robotics is usually a bottleneck for robotics developers and not a straightforward task, not only for previous issues, but also for some other engineering problems identified in the literature (García-Olaya et al. 2019).

This work addresses the solution for the issues above by modelling HRI tasks using a formalization based on Classical Planning through the use of PDDL 2.1 (Fox and Long 2003). To relieve the knowledge engineering process, we propose a design tool where the domain experts can model their own use cases and automatically generate the appropriate code which can be injected into the control architecture. In our case, PELEA (Planning, Execution and Learning Architecture) (Alcázar et al. 2010) is assumed, but other control systems as ROSPLAN (Cashmore et al. 2015) could be used. The proposed system tries to solve the open questions about managing Social Robotics with AP and ease the process of developing new use cases, bringing the modelling and formalization closer to domain experts.

The paper is organized as follows: next section shows the background about control architectures based on AP, while Section 3 introduces the usual process to generate Social Robotics use cases and the features of the graphic editor we propose. Section 4 details the main concepts that must be specified to describe an use case in AP, whose formal-

ization using the PDDL language is detailed in Section 5. Further details about non fully guided use cases are shown in Section 6. To conclude, we show the tests performed, the conclusions and future lines.

2 Background

Planning, execution and monitoring architectures are essential for implementation of AP in real social robotics. These systems typically involve a planner and a formal planning model to generate the sequence of actions the robot must perform. Once generated they are executed using the appropriate robot commands, while monitoring the correct execution of the plan. Some examples of control architectures are the aforementioned PELEA (Alcázar et al. 2010) and ROSPLAN (Cashmore et al. 2015), both using Classical Planning (Ghallab, Nau, and Traverso 2004). Architectures like T-REX (Rajan and Py 2012) support other planning paradigms like timeline-based planning. In general, these architectures implement the robot deliberative behaviour according to the information sensed from the environment and provide a sequence of actions to manage the current situation. In the case of PELEA, the architecture applied in our work, the formal model is based on Classical Planning and uses the PDDL language to describe the planning problem. It is divided in two parts: the domain model, which contains a generic definition of the world and the possible behaviours; and the specific problem to solve related to that domain. Figure 1 shows the architecture of PELEA.

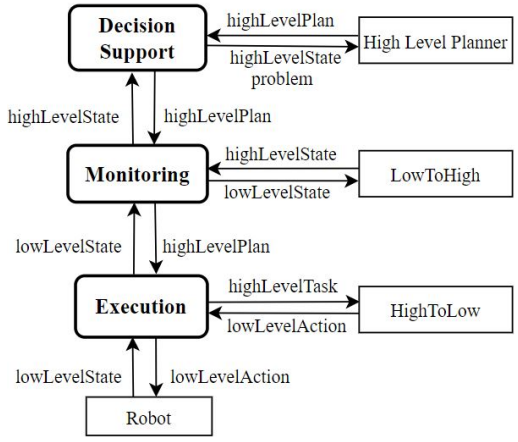

Figure 1: PELEA Architecture

At the beginning, the problem-solver (DECISION SUPPORT) obtains a plan π whose actions will be sent in sequence to the robot platform by EXECUTION, assuming that no events will interrupt the execution. To verify whether the plan goes as expected, external information of the environment is sensed by the robot. If differences are found by MONITORING, it indicates that an unexpected event in the scenario has occurred, needing a new plan π' to solve the current situation. Otherwise, the initial plan can continue with the normal behaviour.

Minor modules HIGHTOLOW and LOWTOHIGH receive as input a catalogue where low and high-level conversions are specified. They are responsible for the mapping between both the architecture and the robot since they do usually work at different abstraction levels. Plan actions such as greeting a person may imply several tasks at a low-level, including arm movement and voice playback. Conversely data coming for example from a 3D person-tracking sensor must be summarised to convert them into predicates like `(person-walking)` or `(person-standing)`. High Level planning is performed via the Metric-FF (Hoffmann 2003) PDDL2.1 compliant planner.

3 Graphic Editor for Domain Modelling

Despite the architecture used, the general case to use AP in Social Robotics usually involves a software developer, a knowledge engineer and an expert in the application field. The knowledge engineer is in charge of generating a planning model of the use case following the instructions of the expert. This model will be generated in a planning language like PDDL. Meanwhile, the software developer adapts the control architecture to the specifications of the use case. Our proposal is to foster the implication of the domain experts in the process, allowing them to participate in modelling their own use cases in collaboration with knowledge engineers. This is done thanks to a graphical interface and a model compiler to automatically generate the formal model, which can then be injected into the control architecture, as shown in Figure 2. This user interface is intended to ease the modelling and maintenance of social robotics domains and problems through an AP approach.

The interface we propose allows the user to completely edit the models, defining all objects, facts, states and actions involved in the use case, corresponding to the elements needed to create an AP model. As will be explained later, predicates and states may have some relevant features for the use case definition, which are represented by checkboxes and can be chosen by the user during their definition. The graphic model of the use case is reflected in real-time on the visual area, depicted as workflows. Exogenous events in the environment may need corrective actions to be handled, which are defined through minor workflows in new graphical areas, adding or switching them in the upper tabs.

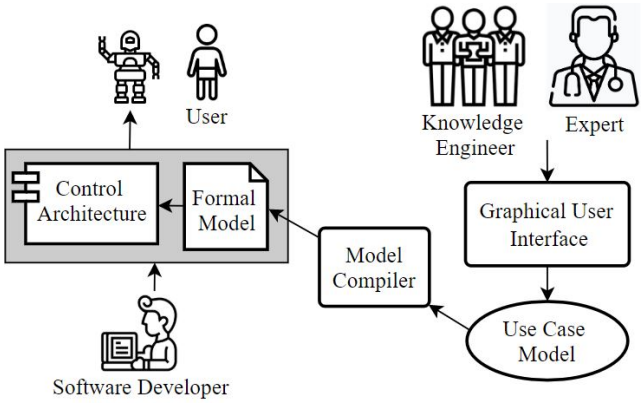

Figure 2: Proposed use case development process

The editor also allows to define problems associated with the domain previously modelled, specifying the initial state and goals to meet. Finally, the management buttons allow to save the model. The Export/Save component stores the graphic design to recover it again if necessary, and generates the XML file which contains the model to be translated into the PDDL formalization. In addition, thanks to the Import/Load feature that can be found in the menu window, the user is able to manipulate existing domains and problems in an intuitive way, without the need for an AP expert.

4 Modelling Social Robotics Use Cases Through Automated Planning

Similar to AP, we define a Robotic Use Case as a tuple $P = \{S, A, I, G\}$, where S is the set of possible states, A is the set of actions that the robot can carry out to transit between states, $I \subset S$ defines the initial state, and $G \subset S$ specifies the goal or conditions that the use case must meet. Thus, the user has to specify all these elements to properly build the use case, where a briefly definition is detailed below.

Objects

Every element in the real world involved in the use case must be defined as an object of the domain. It is also possible to define an object-type hierarchy, which is useful to make generalizations. The types of objects are defined in the domain, but instantiated in the problem definition.

Facts and (Partial) States

After defining the objects we need to define the facts that can appear in the world involving the robot, the user and the environment. In PDDL planning this is done through predicates, which are composed of the objects that they affect. It is important to distinguish among different types of facts that are part of the domain. We define the following categories:

- Dynamic: dynamic facts can be added, removed or modified as necessary. We can distinguish two types:
  1. Sensed: obtained from the robot sensors, as the robot battery level.
  2. Internal: they belong to the control knowledge and their value is internally calculated, such as the number of repetitions performed during the exercise. At the same time, these predicates are divided in persistent and non-persistent. The former are facts that never become false, unless an explicit action is executed to remove them or if information suggesting they are no longer valid is received from the environment. For instance, if the robot puts a bracelet on the patient we assume that the person will keep it during the whole use case.
- Static: Static facts are those that can not change during the plan execution. For instance, it can be assumed that the training area does not change during the session.

Users are required to mark every predicate in the definition stage by choosing the most appropriate characteristic for each case. As a result, we have states composed by a subset of facts that represent the situations which may take place during the robot performance. Although we will call them states, they are partial states since we only specify for them the set of facts that must be true to consider if the robot is in such state. Figure 3 shows an example of state where a patient has already been detected, identified and greeted by the robot and both are in the training area, ready to start the session. Other components of the state, as the location of the training area or the name of the patient, are ignored here as they are not considered relevant for the current step.

```
(detected-patient ?robot ?patient)
(identified-patient ?robot ?patient)
(greeted-patient ?robot ?patient)
(at-training-area ?robot ?tr)
(at-training-area ?patient ?tr)
```

Figure 3: Example of (partial) state in PDDL

Actions

Following a Classical Planning approach, actions are defined as deterministic operators composed by parameters, preconditions and effects. Preconditions are a set of facts that must occur to perform the action, while effects define how the world changes after executing the action by adding, removing or modifying facts. Since actions are carried out in the real world, they have stochastic outcomes. However, to model them according to the previous definition we perform a determination with known effects. The stochasticity of the actions is solved by replanning, which is widely used (Yoon, Fern, and Givan 2007) to address such issues. The control architecture must monitor the execution of actions and compute a new plan if any unexpected situation arises.

Actions allow transitions between successive situations. As usual in AP, we assume that the state from they start represents the minimal facts needed to execute the action, in other words, its preconditions. Since social use cases may present actions which can be repeated several times, we also allow the modelling of sequential or looped actions.

All these elements are the basis for starting to model Social Robotics use cases, that correspond to the AP problems elements. However, this is not enough to achieve a natural interaction. Concepts introduced below are intended to mitigate the issues associated to these models.

The Nominal Flow

Since people in the real world follow a series of ordered steps to build a coherent interaction, we assume a workflow as a suitable way to represent the desired interaction in Social Robotics. The user is requested to model the expected behaviour by describing the states that must be transited during its execution. Along with that, actions are the activities that the robot is able to perform, modelled by connecting states. An action connects two states if the first one goes before in terms of social interaction. As we saw before, that means that this first state is the precondition of that action.

As a running example, Figure 4 shows a robotics Use Case for motor rehabilitation sessions as it is modelled in the graphic editor. It is extracted from a real system driven

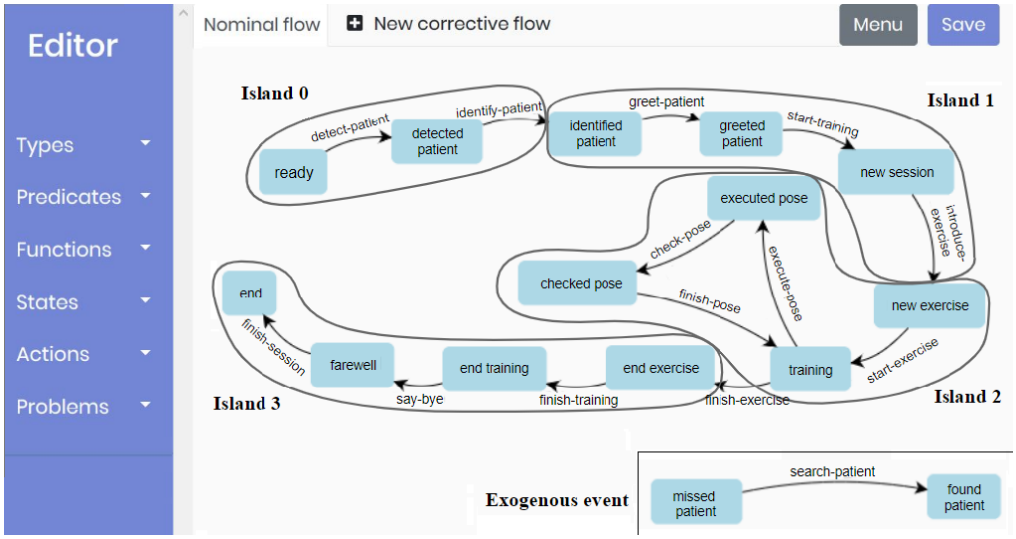

Figure 4: Social Robotics use case modelled in the graphical tool (modified image to provide a clearer vision in the paper)

by AP (Pulido et al. 2017). The project presents a support tool for therapists, based on a humanoid robot which autonomously drives a complete non-contact upper-limb rehabilitation session for children. Each session is composed by repetitions of exercises based on pose execution; first the robot performs each exercise, then it asks the kid to repeat it while controlling she does it correctly. This graph represents the ideal that the robot must follow, where no errors nor stochastic events are taken into account.

Identifying Exogenous Events

The nominal execution can be interrupted by incidents, called exogenous events, which generate an unexpected state. They are not represented along with the nominal flow as they can occur at any time and none of the actions of the normal behaviour produce them. Identifying exogenous events is essential in order to provide a model with corrective actions which are able to recover the nominal flow, building a robust system and showing a coherent reaction in all situations. Otherwise, the robot will be stuck with no way of restoring the interaction with the user. By this reason, in the use case definition it is necessary to model independently (in upper tabs of the editor window) minor workflows which indicate what is the action or actions to perform to solve an unexpected situation. These actions will have the unexpected fact(s) as a precondition(s), ensuring that they will be executed just in case of nominal behaviour failure.

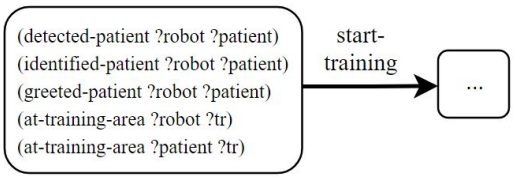

Figure 5: Representation of state - action

Defining *Islands*

As mentioned before, the social behaviour has a number of steps that are usually executed together to achieve a coherent interaction. Actions such as greeting a person before starting to talk and some other activities that follow a particular order are the behaviours we want to ensure in the robot execution. Since the discussed use cases try to imitate social behaviours, we represent these interaction stages as groups of actions and states. In the example depicted in Figure 4 we can see the nominal behaviour split into these different phases, referred to as *islands* in the remainder of this paper. Their definition is aimed to provide a coherent execution of the use case. It allows returning to the nominal behaviour from a logical point when an unexpected event occurs, going back to the beginning of the island to execute it again. Otherwise, the nominal flow would be recover from the point where the error took place. These return points are set by the user to their choice, who just has to mark the beginning of the desired phases, where the execution has to recover the normal flow.

5 Formalizing Social Robots Behaviours

The development of PDDL code is a tedious and complex task, in addition to problems related to how to define deterministic actions to model stochastic domains and still achieve a natural interaction. These challenges slow down the adoption of AP in this field. In this section we present our approach to solve these issues at a high-level, proposing a PDDL formalization which is automatically generated from the input model given by the user through the graphical editor. We present the PDDL formalization of the use case of the social robot for motor rehabilitation previously shown in Figure 4.

Formalizing Sequential Connections

According to the interaction graph previously presented, every state where an action starts defines its preconditions, since it contains the facts that the environment must hold to execute the action. In this way, for the `start-training` action, whose previous state is depicted in Figure 5, we will obtain the formalization shown in Figure 6, which follows the standard PDDL. To simplify the model, effects are not taken from the next state, but specified during the action definition. Action parameters are defined by all the objects that are involved in the facts of the preconditions and effects, getting them automatically.

```
(:action start-training
  :parameters(?r-robot ?p-patient ?tr-location ?s-session)
  :precondition (and (detected-patient ?r ?p)
                     (identified-patient ?r ?p)
                     (greeted-patient ?r ?p)
                     (at-training-area ?r ?tr)
                     (at-training-area ?p ?tr))
   :effect (started_session ?r ?p ?s)
)
```

Figure 6: PDDL code of the `start-training` action

Formalizing Loops

Since loops can be seen as *for* or *while* statements, they need to have an exit condition that makes possible to leave the loop when reached. The difference depends on whether the predicate which controls the loop is internal or sensed, respectively. If it is internal, it acts as a counter, which will be incremented or decreased during the cycles until it reaches the exit condition. Otherwise, the loop will be like a *while* statement if the value of the variable is not modified in a explicit way, but sensed from the environment. In that case, the execution will leave the loop when the correct value is sensed.

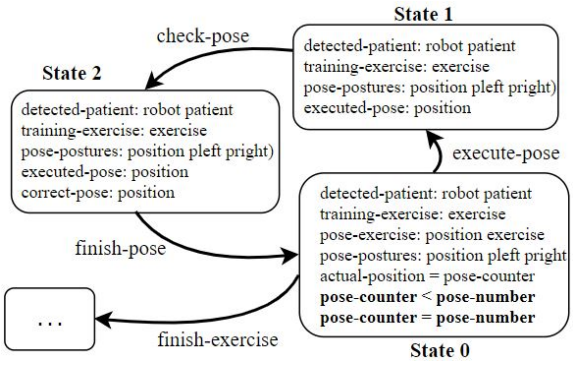

Figure 7: Representation of states involved in a loop

As an example, the loop found in Figure 4 (Island 2), is composed by three actions whose states are represented in Figure 7. In this case, the objective is to achieve the desired number of poses to complete the exercise. All of the states represent the current situation according to the exercise stage, where poses have been executed and corrected. However, in state 1 there is one more detail, where two situations can occur: the number of executed poses (`pose-counter`) is below the desired number of repetitions (`pose-number ?e`) or it is equal to that value.

The user has to model both options in that stage. With the first one the loop is still running, whereas reaching the number of poses allows leaving the loop. Mapping it into PDDL code means to associate the statement `(< (pose-counter) (pose-number ?e))` with the preconditions of the action `execute-pose`. On the other hand, the action `finish-exercise` only will be executed when the poses counter reaches the exit condition, so the function `(= (pose-counter) (pose-number ?e))` will be part of the precondition of this action. This distinction is automatically done when mapping the conceptual model to its formalization. The user just has to indicate which is the action that allows to leave the loop. The counter must be increased after the execution of the last action of the loop, `finish-pose`.

Formalizing Exogenous Events

Actions of the nominal behaviour will be executed while no external events interrupt the robot work. For that purpose, we automatically introduce a flag called `(can-continue)` on the precondition of each nominal behaviour action, which forces the flow to continue only if everything goes according to the established plan. Otherwise, this flag will be removed and the robot feedback will include the error found, which will be translated to high-level predicates by the LOWTOHIGH module.

Exogenous events are closely related to island definition. As we explained before, the user has to decide which states are the beginning of the islands, acting as a restoration point. Considering the Island 1 in Figure 4, going back to the first state means that the next action to execute again after recovering the normal flow is `greet-patient`. This implies that this action must be able to be executed even in error case. This is easily done by adding the disjunction `(or (can-continue)(restore))` in its precondition, declaring that it can be performed in case of no error or when returning at the beginning of that island.

Regarding to the effects, passing through that action in-

```
(:action greet-patient
  :parameters (?r-robot ?p-patient)
  :precondition (and (detected-patient ?r ?p)
                     (identified-patient ?r ?p)
                     (or(can-continue)(restore)))
  :effect    (and (greeted_patient ?r ?p)
                  (can-continue)
                  (not (restore))
                  (assign (island-number) 1))
)
```

Figure 8: PDDL code for greet-patient action, marked as a restoration point

```
(:action search-patient
  :parameters (?r-robot ?p-patient)
  :precondition (and (not (can-continue))
                     (not (detected-patient ?r ?p)))
  :effect (and (detected-patient ?r ?p)
               (restore-action))
)
```

Figure 9: PDDL code for corrective action

dicates that the execution is already in the first island, reason why it assigns this value to `island-number`. In case of running this action after an unexpected state, we assume that the nominal flow is restored, indicating that the workflow can continue again and it is not necessary to go back to previous restoration points. The formal definition of this action can be seen in the Figure 8. In this way, if the patient is lost during the training, the right way to respond is greeting again the patient if they were in island 1 or restarting the exercise if they were in island 2. Users have the possibility to define corrective actions to handle those situations by specifying the events that they want to solve and how these actions manage the situation. These actions will be only included in the plan if the nominal flow can not continue.

Taking the previous example, we have defined in the editor the corrective action `search-patient`, formalized in Figure 9. It will be executed in case the patient leaves the training area and its effect is finding her again. For this example we assume that only one task applies as corrective action, but a number of them can be included as needed.

In order to recover the nominal flow properly, we add in the effects of these actions another flag called `(restore-action)`, which activate special actions that enable to go back to a correct state to continue with the nominal behaviour. The objective of these restorer actions is to delete all intermediate effects added by actions of the island, forcing to restart the nominal flow from the desired point. Otherwise, the current state would still include all these facts, going back to the exit point when the exogenous event happened. Our compiler will generate one of these actions for each defined island, removing all facts added except those ones marked as persistent. As an effect, it allows to find the appropriate restoration point to recover the nominal behaviour. An example can be seen in Figure 10.

It is important to remind that the PDDL formalizations

```
(:action restore-from-one
  :parameters (?e-exercise ?r-robot ?p-patient ?s-session)
  :precondition  (and (restore-action)
                      (= (island-number) 1))
  :effect  (and (not (training-exercise ?e))
                (not (started-session ?r ?p ?s))
                (not (greeted-patient ?r ?p))
                (not (restore-action))
                (restore))
)
```

Figure 10: PDDL code for restore action

showed above are automatically generated by our compiler, working as a black box. As a result we obtain the PDDL domain file corresponding to the use case defined by the user in the graphical interface. Our system also provides PDDL problem generation by instantiating the predicates which define the initial state and the goals to meet.

6 Non Fully Guided Social Robotics Scenarios

The above scenario represents a full sequential use case, where the expected behaviour has a totally preset order. In these cases, the specialist has to define all the connections between states and actions to generate a domain model that allows the proper performance of the robot. However, there might be some situations where the sequence to solve the problem is unknown. Actually this kind of scenarios are the ones where AP shows all its potential against other control techniques like Finite State Machines.

Such scenarios could be depicted by disconnected graphs, as it is shown in Figure 11. This example represents a similar use case, involving a social interactive blocks game. This is an adaptation of the classical planning *blocksworld* domain, with misplaced blocks that must be relocated in a certain order by the children and the robot, seeking collaboration among participants in a turn-based activity. The domain is modelled taking into account that both the child and the robot can carry out actions such as pick up a block or unstack it. However, we can not guess which block will be moved first, since it depends on the initial location of the blocks and on the child decision, i.e., on the current use case to solve.

Since disconnected graphs do not specify the full order of the actions, it is necessary to correctly define when the game has to take place. This is easily done by adding a predicate that enables/disables the execution of the game. In this case we use the expression `(play-time)`, that has to be true throughout all the game. The action `finish-exercise` will indicate the end of the game, removing this predicate in its effects, as is shown in Figure 12.

As said, identifying appropriate islands is important to achieve a natural behaviour when recovering the interaction from an unexpected event. Restorer actions (see Figure 10) suppose the main difference in these scenarios. For instance, if a child is holding a block to stack it on top of another, but during the process the block falls on the table, recognizing that situation and recovering from it depends on monitoring and replanning. Since at this point there is no nominal flow (because there is not a unique general sequence) and all predicates involved in the action are sensed ones, no restorer actions will be applied and solving the current situation relies on finding a new plan from the sensed scenario. Figure 13 shows all sensed predicates involved in the game, which specify the location of the blocks and whether a player is able to pick a block. The only way to know if they are true or false is sensing them from the environment. Then, it is not necessary for the user to model actions such us `recover-fallen-block`. In the case of applying such a normal restorer action would mean that all blocks would

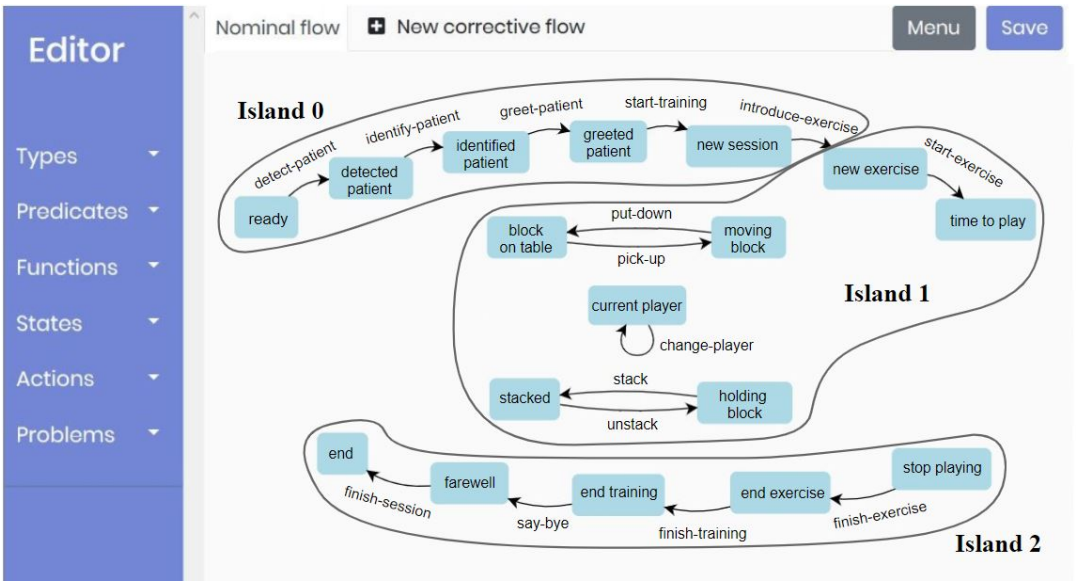

Figure 11: Social Robotics use case modelled using a disconnected graph

```
(:action finish-exercise
  :parameters (?r - robot ?p - patient ?e - exercise)
  :precondition (and (detected-patient ?r ?p)
                     (training-exercise ?e))
  :effect  (and (not (play-time))
                (finished-exercise ?e))
)
```

Figure 12: PDDL code for finish-exercise action

be located as at the beginning of the game, which is an assumption that can not be made. The control architecture will obtain a new plan with the expected behaviour from that new situation instead.

Then, by using a workflow representation we can also model generic domains without knowing beforehand the predicted sequence to achieve the goal. This gives autonomy to the robot, being able to solve the current use case and recover itself from exogenous events that may appear during its execution by monitoring and replanning. Here we can see the strength of the use of AP in these cases, avoiding the use of FSMs that have to be built for each specific case, and making easier the deployment of robotic platforms.

```
(on ?obj - block ?underObj - block)
(on-table ?obj - block)
(clear ?obj - block)
(arm-empty ?pl - player)
(holding-by ?obj - block ?pl - player)
```

Figure 13: Sensed predicates

7 Framework Testing

We have tested our tool in four different domains. Two of them are classical domains from the International Planning Competition (IPC)[1] and the remaining two ones are aforementioned real social robotics use cases. In the four cases the PDDL domains created have been tested to be executable and to provide appropriate plans to solve the task in hand[2].

From the IPC, we have modelled the well-known logistics and blocksworld planning domains. Figure 14 shows how the *logistics* is described using our interface by defining the three actions which allow to transit among the different partial states that are identified for this task, where packages and vehicles can be located in different places. The blocksworld model is shown on Island 1 in Figure 11, regardless of the player change action.

The rehabilitation use case has been integrated in a control architecture using the PDDL domain and problem files generated by our system as input of PELEA, which was configured with two identity mappings in the *highToLow* and *lowToHigh* modules, and tested in simulation. We also deployed a ROS (Robot Operating System) (Quigley et al. 2009) layer on PELEA to simulate the communication between the control architecture and the robotic platform, responsible for messages passing between both systems. Actions are sent in their respective low level, waiting for the robot feedback. As it receives a right response, next action is transmitted. To check restore actions, we force an unexpected state after islands in different executions, reporting that the patient has left the training area. Once noticed that the current situation does not match the expected plan, the problem-solver obtains a new plan which recovers the nominal behaviour by applying a corrective action. The replanning time provides a

[1] http://icaps-conference.org/index.php/Main/Competitions

[2] The resulting domains can be found in https://bit.ly/2ptcwyM

rapid response in less than 0.01 seconds, essential in human-robot interaction.

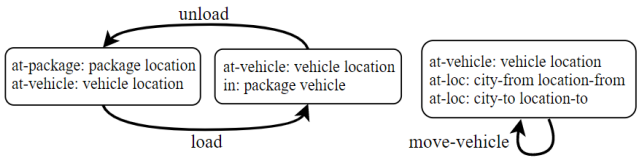

Figure 14: State diagram for logistics domain

8 Related Work

Developing PDDL code is not a straightforward task, requiring skills and understanding of the formalization language, in addition to the effort to model the domain to achieve good performance in real environments. Most of the times the designer is not a planning expert, needing help to build the system which will control the robotic platform. For that reason, an important issue is the lack of tools that support the development of planning domains. Besides some number of PDDL editors[3] which require extensive knowledge about the specification language to be used, there are only a few systems providing a simple and intuitive interface to support the design and development process. Similar to our approach, we focus this section on visual interfaces, which can save a lot of time and facilitate the design and modelling task to people who is not a planning expert.

itSIMPLE (Vaquero et al. 2013) is a Knowledge Engineering environment to support the develop of planning applications, centred on the initial design phases. This is done using UML (Unified Modelling Language), by which the requirements are gathered. They also use PDDL as representation language, translating the input model to this language. Objects and actions are defined with OCL (Object Constraint Language), a pure specification language used to describe types and preconditions and effects of operations and methods. GIPO (Simpson, Kitchin, and McCluskey 2007) presents a planning domain definition in an object-centred perspective. For each object the properties and the possible transitions that can occur within each one are defined, modelling in this way the domain. This definition is performed with abstract state machines, from which propositional descriptions of domain can be derived. To specify associations between objects, different state machines can be connected, allowing the proper translation to PDDL code. On the other hand, VLEPPO (Hatzi et al. 2010) offers an intuitive graphical interface to simplify the modelling task, even for non-expert users. Close to our approach, relations between objects are represented by predicates and actions allow transitions between successive situations. Every element in the domain is thus defined graphically, enabling the generation of the corresponding PDDL code.

All of these systems show different representations to specify planning domains. However, none of them is suitable for modelling social interaction use cases, either to represent a drawback for users who do not have deep knowledge about software engineering or become unmanageable for extensive domains. For that reason, to design Social Robotics behaviours we choose the workflow definition, a simple way to define the behaviour the expert expects the robot to perform, even in those cases where there is no fixed sequence to solve the use case. Furthermore, none of the tools mentioned provide characteristics to ease the definition of usual properties of Social Robotics, issues that we exploit in our work allowing the user to model features such as exogenous events and *islands*, thus creating a more robust model to operate in real world environments.

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

9 Conclusions and Future Work

Applying Automated Planning to Social Robotics has been already tested in the literature. However, it is not extensively used due, among other reasons, to the time-consuming process of the development of PDDL models, particularly in this field, where an extensive knowledge engineering process is required beforehand. Once the requirements have been specified by the domain expert, the task of formalizing them in the representation language is aimed to professionals in AP, where the complexity of modelling uncertain and stochastic environments with deterministic operators arises. We have developed a tool where domain experts can specify the use case through a state transition diagram, defining the actions that the robot must perform. Using our approach it is also possible to model exogenous events and sequences of actions that should be executed again after an error case, intending to achieve a natural interaction. This is useful to ease the knowledge acquisition process, in addition to provide details about characteristics that help to achieve natural interaction between the robot and the human. Going a step further, the mentioned model is automatically translated to PDDL code, generating the domain and problem files which can be inserted in a control architecture based on Classical Planning to set the robotic platform. All of these features ease the development of Social Robotics use cases, providing extended autonomy to those who are domain experts but are not developers.

In future work, we are planning to extend the editor by including the *highToLow* and *lowToHigh* files generation, by building a catalogue of actions which simplifies the development and integration of new use cases, avoiding to write *ad hoc* files. Considering that PELEA also has a cloud version, we expect that bringing the two systems together will allow any person who has a robot and some technical background to smoothly create and implement her own use case using AP.

Acknowledgements

This work has been partially funded by the European Union ECHORD++ project (FP7-ICT-601116), and grant RTI2018-099522-B-C43 of FEDER/Ministerio de Ciencia e Innovación - Ministerio de Universidades - Agencia Estatal de Investigación.

References

Alcázar, V.; Madrid, I.; Guzmán, C.; Prior, D.; Borrajo, D.; Castillo, L.; and Onaindía, E. 2010. PELEA: Planning, learning and execution architecture. In *Proceedings of the 28th Workshop of the UK 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 *Proceedings of the 17th Workshop of Physical Agents (WAF)*, 1–8.

Breazeal, C.; Dautenhahn, K.; and Kanda, T. 2016. Social robotics. In Siciliano, B., and Khatib, O., eds., *Springer Handbook of Robotics*, Springer Handbooks. Springer. 1935–1972.

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 *Proceedings of the 25th International Conference on Automated Planning and Scheduling, ICAPS 2015, Jerusalem, Israel, June 7-11, 2015*, 333–341.

Fox, M., and Long, D. 2003. PDDL2.1: an extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res.* 20:61–124.

García-Olaya, A.; Fuentetaja, R.; García-Polo, J.; González, J. C.; and Fernández, F. 2019. Challenges on the Application of Automated Planning for Comprehensive Geriatric Assessment Using an Autonomous Social Robot. In *Advances in Intelligent Systems and Computing*, volume 855. Springer International Publishing. 179–194.

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.

Hatzi, O.; Vrakas, D.; Bassiliades, N.; Anagnostopoulos, D.; and Vlahavas, I. P. 2010. A visual programming system for automated problem solving. *Expert Syst. Appl.* 37(6):4611–4625.

Hoffmann, J. 2003. The Metric-FF Planning System: Translating "ignoring delete lists" to numeric state variables. *Journal of Artificial Intelligence Research* 20:291–341.

Kambhampati, S. 2007. Model-lite planning for the web age masses: The challenges of planning with incomplete and evolving domain models. In *Proceedings of the 22th Conference on Artificial Intelligence AAAI , July 22-26, Vancouver, British Columbia, Canada*, 1601–1605.

Lera, F. J. R.; Olivera, V. M.; González, M. Á. C.; and Rico, F. M. 2018. Himop: A three-component architecture to create more human-acceptable social assistive robots. *Cognitive Processing* 19(2):233–244.

Prenzel, O.; Feuser, J.; and Graser, A. 2005. Rehabilitation robot in intelligent home environment-software architecture and implementation of a distributed system. In *9th International Conference on Rehabilitation Robotics, ICORR*, 530–535. IEEE.

Pulido, J. C.; González, J. C.; Suárez-Mejías, C.; Bandera, A.; Bustos, P.; and Fernández, F. 2017. Evaluating the child–robot interaction of the naotherapist platform in pediatric rehabilitation. *International Journal of Social Robotics* 1–16.

Quigley, M.; Conley, K.; Gerkey, B.; Faust, J.; Foote, T.; Leibs, J.; Wheeler, R.; and Ng, A. Y. 2009. ROS: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3. Kobe, Japan.

Rajan, K., and Py, F. 2012. T-rex: Partitioned inference for auv mission control. *Further advances in unmanned marine vehicles* 171–199.

Simpson, R. M.; Kitchin, D. E.; and McCluskey, T. L. 2007. Planning domain definition using GIPO. *Knowledge Eng. Review* 22(2):117–134.

Tapus, A.; Mataric, M. J.; and Scassellati, B. 2007. Socially assistive robotics [grand challenges of robotics]. *IEEE Robot. Automat. Mag.* 14(1):35–42.

Tran, T. T.; Vaquero, T. S.; Nejat, G.; and Beck, J. C. 2017. Robots in Retirement Homes: Applying Fff-the-shelf Planning and Scheduling to a Team of Assistive Robots. *Journal of Artificial Intelligence Research* 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.

Yoon, S. W.; Fern, A.; and Givan, R. 2007. FF-Replan: A Baseline for Probabilistic Planning. In *International Conference on Automated Planning and Schedulling (ICAPS)*, 352–359.