Challenges on the Application of Automated Planning for Comprehensive Geriatric Assessment Using an Autonomous Social Robot

Angel García-Olaya(✉) , Raquel Fuentetaja , Javier García-Polo,
José Carlos González , and Fernando Fernández

Universidad Carlos III de Madrid, 28911 Leganés, Spain
{agolaya,rfuentet,fjgpolo,josgonza,ffernand}@inf.uc3m.es,
http://www.plg.inf.uc3m.es

Abstract. Comprehensive Geriatric Assessment is a medical procedure to evaluate the physical, social and psychological status of elder patients. One of its phases consists of performing different tests to the patient or relatives. In this paper we present the challenges to apply Automated Planning to control an autonomous robot helping the clinician to perform such tests. On the one hand the paper focuses on the modelling decisions taken, from an initial approach where each test was encoded using slightly different domains, to the final unified domain allowing any test to be represented. On the other hand, the paper deals with practical issues arisen when executing the plans. Preliminary tests performed with real users show that the proposed approach is able to seamlessly handle the patient-robot interaction in real time, recovering from unexpected events and adapting to the users' preferred input method, while being able to gather all the information needed by the clinician.

Keywords: Automated Planning · Human-Robot Interaction
Planning and execution · Social Robotics
Comprehensive Geriatric Assessment · Health-care robotics

1 Introduction

One of the main challenges of the health-care systems is the aging of the population. According to the World Health Organization [24] between 2000 and 2050, world's population over 60 years will increase from 11% to 22%, reaching a total of 2 billion people, from which almost 400 million will be 80 years or older.

Comprehensive Geriatric Assessment (CGA) [6] is a medical procedure to evaluate the physical, social, cognitive and psychological status of elder patients. CGA is the prerequisite for personalized treatment and follow-up. It is performed usually every 6 months and involves three phases: in phase 1, Clinical Interview, patients and relatives inform to health-care professionals about the patient status

and their perception about her evolution since last evaluation. During phase 2, Multidimensional Assessments, different measurement tests, performed either by patients or relatives, evaluate the functional, cognitive, motor and social status of the patient. Finally, in phase 3, Individualized Care Plan, the results from the two previous steps are taken into account and the health-care professional designs a personalized care plan to be followed until next visit.

This paper shows the challenges faced when applying Automated Planning [11] to control a robot able to assist health-care professionals collecting information during phase 2. Currently it is able to autonomously conduct tests without health-care professional intervention. Meanwhile, the clinician can concentrate on more added-value tasks like discussing with patient or relatives or evaluating the evolution since the last visit. The use of an automatic solution to perform the tests could significantly reduce the total time spent in the CGA process.

The paper is structured as follows: Next section describes the background of the work. Section 3 shows the challenges faced and the decisions taken when developing the deliberative module. Section 4 presents a brief description of the domain and problems. In Sect. 5, preliminary results are shown. Section 6 shows the closest related work. Finally the paper conclusions and the future work are described in Sect. 7.

2 Background

2.1 CGA Tests

Several tests have been proposed in the medical literature for CGA's phase 2. Our system is currently able to perform three of the most popular ones: a functional test, the Barthel's Index Rating Scale [17]; a cognitive test, the Mini-Mental State Examination (MMSE) [8]; and a physical test, the Get Up & Go test [18]. In addition to measure different aspects of the patient status, these three tests are quite different in nature and pose different challenges from the Human-Robot Interaction (HRI) point of view:

– The Barthel's test is performed by the patient or a relative and can refer to present or past patient's conditions. It consists of ten questions, each one with three or four possible closed answers. Questions deal with the patient's ability to perform daily living activities. From the HRI perspective this is the simplest test to automatize as answers are closed and the patient just needs to answer verbally or using a tablet.
– The MMSE test evaluates cognitive impairment and changes in patients suffering from dementia. It must be performed by the patient. In 5 to 15 min it examines orientation, immediate and short-term memory, attention, calculation abilities, recall, language understanding, and ability to follow simple commands. It is more complex to automatize than the previous one as it does not only include closed-answer questions, but also open-answer questions ("What day is today?") and also questions that require monitoring simple patient movements ("Close your eyes"), painting or hand-writing.

– The Get Up & Go test is used to measure balance and fall risk, detecting deviations from a confident, normal performance. The patient must stand up from a chair, walk a short distance, turn around, return to the initial location and sit down again. The robot has not only to speak to guide the patient, but it needs also to place itself in a position where it can monitor patient movements. In addition, the robot needs to perceive the gait and analyze balance and time.

2.2 The Robot

The used robot is shown in Fig. 1. It is a mobile robot, based on the MetraLabs SCITOS G3 platform[1], with capabilities for localization, navigation and obstacle avoidance. Interaction with patients is done via speech and speech recognition, but also the embedded tablet is used to show the tests' questions and can be used by the patient to answer. For patient's convenience she is provided with a second tablet mirroring the embedded one. The robot is also endowed with a Kinect 3D sensor that detects the patient and tracks her movements, which is needed to check that she has not left the room, but also for some parts of the tests. The software and hardware description of the robot has been performed elsewhere [3] and is not part of this paper.

Kinect sensor v2

Shotgun microphone

Touchscreen monitor

2 omnidirectional speakers

Barebone i7

Embedded PC with intel-i7 CPU and Wi-Fi

SCITOS G5 mobile base

Question 1

Fig. 1. The robot used for CGA, showing sensors (left) and shell (right).

2.3 Automated Planning

Automated Planning (AP) consists of finding a plan (sequence of transitions) that, when executed, takes the system from its current state (initial state) to a final one where the goals are achieved. Most of the research in AP is devoted to the automatic generation of plans using generic problem-solving techniques.

We use AP techniques to provide the robot with the ability of generating its own plans and behave autonomously when conducting the CGA tests. There are

---

[1] http://www.metralabs.com/en/ (Last visit on July 21st 2018).

two main activities that need to be done so that the application of AP is possible: First, the Human Robot Interaction (HRI) needs to be modeled; second, once the model has been created and a plan achieving the goals is found, the plan must be executed and monitored, given that the real interaction with the patient is very likely to be quite different from the "ideal" planned one.

Roughly speaking, there are two main approaches to Automated Planning: *action-based* and *timeline-based*. The main difference between them is the way transitions are modeled: *Action-based* planning [7] represents transitions by means of actions. Following this paradigm, the task is modeled in terms of states and actions; applying an action in a given state produces a new state. The plan will be the sequence of actions achieving the goal from the initial state. Formally:

Definition 1 (Action-based planning task). *An action-based planning task is a tuple $\Pi_{ab} = \{F, A, I, G\}$.*

F is a finite set of facts and numerical state variables. A state $s \subseteq F$ is defined by the set of facts that are true together with the set of values of the numerical state variables. A are the actions that the robot can perform, being each $a \in A$ described by means of four sets: $\{pre_a, add_a, del_a, num_a\}$. pre_a are the preconditions, that must hold in a state for the action to be applicable, add_a and del_a are respectively the true facts appearing and the facts that become false after a is applied, and num_a accounts for the variations that a produces in the numeric state variables. $I \subseteq F$ is the initial state and $G \subseteq F$ describes the goals. A plan, $\pi = \{a_1, a_2, ...a_n\}$ is sequence of actions that applied to I reaches a state s_n such as $G \subseteq s_n$. Classical *action-based* planning assumes instantaneous actions and does not consider time explicitly, even if it has extensions handling time.

On the other hand, *timeline-based* planning focuses on the internal state of the system, how it varies due to changes in the world and how it drives the interaction with the environment. Formally:

Definition 2 (Timeline based-planning task). *A timeline-based planning task is a tuple $\Pi_{tb} = \{S, R, I, G\}$.*

$S = \{s_a, s_b, ...s_k\}$ is set of state variables, where each $s_i \in S$ is defined by (V, T, D), being V the possible values it can take, $T : V \rightarrow 2^v$ is a transition function that for each value $v \in V$ specifies the values that s_i can take after it, and $D : V \rightarrow \mathbb{N} \times \mathbb{N}$ specifies the minimum and maximum times where s_i can take the value v. R is a synchronization rule, a pair of two state variables s_i and s_j temporally related by one of the Allen's temporal relations [2] (*equal*, *before*, *meets*, *overlaps*, etc.). I is the initial value of all the state variables and G are the goals; future desired values for some of the state variables, usually temporarily tagged and with temporal relations among them. A plan is a sequence of state variables transitions that respects the synchronization rules.

In both planning paradigms modeling a task consists in creating a representation of the states and the transitions using a declarative language. *Action-based* planning uses mainly the Planning Domain Definition Language (PDDL) [19] and its variants. Meanwhile there is no standard language in *timeline-based*

planning and various alternatives, like the Domain Description Language (DDL) [5] or the New Domain Definition Language (NDDL) [4], coexist. The models are usually split into two files: the domain file, which contains the definition of the predicates and numerical functions for defining states and the transitions (actions or synchronization rules); and the problem file, which contains the descriptions of the initial state and the goals. This way, a domain file can be reused to represent a family of planning tasks just by changing the problem file.

3 Automated Planning for CGA

In this section we analyze the AP challenges CGA poses from both the modelling and execution perspectives.

3.1 Modeling Challenges and Decisions Taken

There are several ways to model the CGA process using AP. Once a planning approach is selected, specific features of CGA allow to decide among the different alternatives that appear when creating the model.

Selection of the Planning Paradigm. In order to choose the paradigm the following general aspects are usually considered:

– Reasoning about time requirements: Although both paradigms are able to explicitly take time into account when creating the plans, it has been demonstrated that *timeline-based* planning is expressive enough to capture *action-based* temporal planning, while the contrary is yet unclear [13]. If complex temporal reasoning is needed, as in the case of quantitative temporal relations among goals (for example representing that a goal must be achieved between 10 and 15 s after another goal has been reached), *timeline-based* planning is usually a better approach.
– Model complexity: *Timeline-based* models tend to be more complicated than their equivalent *action-based* ones. In addition, changes in *action-based* models tend to be quite straightforward and can be performed even by non-experts, while in the another approach changes in the model usually needed to be carefully studied.
– Availability of planners: *Action-based* planning community is quite bigger than the *timeline-based* one. That means the number of freely available planners is higher and contributions in the field appear at a higher pace. Nevertheless, some applications like autonomy in space are mainly dominated by *timeline-based* approaches.

Although the concept of time is relevant in CGA interaction (for example patients have a limited time to answer before the robot repeats the question or marks it as unanswered), reasoning about time is not required, nor there is any need for concurrent actions or complex temporal relations between goals. Implicitly considering time, as non-temporal *action-based* planning does, seems a good

choice, reducing the complexity of the search and producing better quality plans faster. Non-temporal models are also easier to create and there is a large amount of planners supporting it, consequently, non-temporal *action-based* planning has been selected to model the CGA interaction. As a natural result of this choice, the PDDL language will be used.

Planning Horizon. Next decision is to determine the planning horizon. We can plan in a question by question basis: the robot creates a plan to pose the current question, executes it, and once finished, looks for a new plan for the next one. Most CGA tests are episodic at the level of questions. This means each question can be planned in isolation and the result of one question has no effect in the next ones. In the rare case where some consecutive questions are interrelated they could be considered as a bigger single question. The main advantage of this approach is that it is very likely that plans will be found quickly, as they will comprise a relatively small number of actions. The drawback is that a external control system is needed to control the global interaction flow.

A second alternative is to plan the whole test, modeling each of the episodes by introducing intermediate subgoals or landmarks [16] and a mechanism to impose a total order among them. That way, an external control system is not needed, but it will take longer to find a plan. We could also think about planning the whole CGA, creating a plan for all tests to be carried during the session.

After evaluating the three alternatives, we have decided to plan at the level of a single test. As we will see in Sect. 5 planing times are short enough to guarantee a seamless interaction even when multiple replanning processes are needed. In the usual case that several tests are prescribed sequentially, a control system is needed to concatenate the planning and execution of all of them, but this is way simpler than the one needed if planning at the question level.

Model Partition. Once decided we will plan at the test level. We need to state whether the CGA interaction will be modeled using a single domain or if a domain will be created for each test.

Having a domain for each test allows to tailor them to the specific types of interactions they need; answering questions in some of them, physical activities in other ones, etc. Following this approach domains will be simpler as only the required types of interactions must be encoded. The drawback is having several similar domains, which makes changes and improvements harder. On the other hand, most tests have a common structure composed by salutation, introduction, questions and farewell. Also the order of the questions is usually fixed and most of them follow a common structure: The robot begins the interaction, usually with some kind of introduction that precedes the question. Then the question itself is posed and options, if any, are enumerated. Once this is done, it is the turn of the patient to answer or to perform some action. Questions finish with the robot giving some kind of feedback about the user action ("Thanks, we will proceed to the next question") or asking the user if the answer is right ("Is this

your final answer?"). Attending to this common structure it would make sense to create a single domain to model all tests.

Considering pros and cons of each alternative, we have decided to create a single domain. This domain is able to model the different types of questions that appear in CGA. Using this unified domain, a planning problem will encode each test. Although finding a plan in this common domain is harder than if several simpler domains were considered, exploiting the sequential nature of CGA makes planning times suitable for real-time interaction.

Action Modeling: Handling Uncertainty in Actions. An *action-based* planning model comprises a description of the states and of the actions the system can perform. Actions modify the state either by changing the truth value of facts or by altering the value of numerical ones. When modeling a task using planning it is necessary to make some assumptions or predictions about the effects of the actions [12]. As we are planning the full test we need to guess which will be the patient's behavior for each question. Probabilistic planning, which extends the model of *action-based* planning assigning probabilities to each of the outcomes of an action, seems a natural approach to handle uncertainty in action execution, as for example the patient asking for the robot to repeat a question. But in practice probabilistic planning has some drawbacks: probabilities for each outcome are usually unknown and finding a plan taking them into account becomes much harder. In fact, in domains where there are no dead-ends (states from which we cannot escape, or that once reached prevent us from achieving the goals), a common procedure to handle uncertainty is to assume that the action execution will result in its most likely effect, and replan in any other case [25].

That is the approach followed in our work: A *nominal behavior*, referring to the desired flow of interaction, has been defined. It corresponds to a seamless interaction between the robot and the patient, where the first one poses the questions and the second one answers them with no errors or external events modifying the expected flow. If the interaction diverges from the predicted flow a replanning episode is launched and a new plan is created.

Action Modeling: Handling Unpredictable Events. Many things can go wrong while performing the CGA, but the ultimate goal of the robot is to find a new plan to come back to the nominal behavior in order to finish the test gathering the required information. The new plan must contain some actions to correct the unexpected issue and to return to the normal flow of the nominal behavior. These are *corrective actions*, which are never included in the initial plan. For instance, if the patient leaves the room, the robot must interrupt the test execution. The new plan should start calling the patient and searching for her before continuing with the test. After the execution of these corrective actions, the nominal behavior can continue. Modeling these corrective actions is important as it endows the system with much more responsiveness to the environment and a more coherent interaction. Dividing actions into

nominal-flow ones and corrective ones simplifies both types of actions, which results in reduced planning times. Nominal behavior plus corrective actions are a good alternative to probabilistic planning approaches. They allow to easily recover from undesirable situations, scale better and do not need a probabilistic model of each action outcome to be created.

One peculiarity of CGA interaction is that often after an interruption the interaction should not come back to the point where it was when stopped, but to a previous point. For example if the robot is interrupted while reading the options of a question, it is required to start the question again. If the unexpected event occurs when waiting for the answer, it makes sense that a summary of the question is performed again when the nominal flow is resumed.

Modeling these *restart from here* states and the way to come back to them in case of interruption is somewhat special because the corrective plan must reset the values of some fluents to repeat one or more previously executed actions to achieve a coherent interaction. To solve it, from a conceptual point of view the execution has been divided in interaction episodes of different number of actions. An interaction episode must be completely executed or it has to be repeated again from its beginning. As a consequence, the number of actions to execute again after a replanning depends on the interruption but also on the moment in which it occurs.

State Modeling: Handling Numerical Information. While performing a CGA test the state must contain some numerical information, like the number of the question being currently asked, the number of times it has been repeated or how many questions the patient answered incorrectly, among others. This is represented by means of numerical functions or numerical fluents. Despite being part of the language specification since a long time ago and their obvious utility in many real applications, modern planners still have problems to integrate them in their heuristics, and indeed many state-of-the-art planners do not fully support them. A common *trick*, widely used when the range of numeric values is limited, is to codify them by using logical predicates. That way, the heuristic can consider them while looking for the solution. But this increases the number of possible states, which in turn increases the search space and the pre-processing time. In our case, where plans must be found in real time modeling numbers or order relations with predicates could increase pre-processing time to unbearable values, not suitable for a fluid interaction.

State Modeling: Types of State Variables. Two states will be different if they contain different facts or different values for the numerical state variables. Both will change as an effect of an applied action or due to external events. Taking that into account, we can conceptually divide the state variables (both logical and numerical) into three different categories:

– *Internal variables*: They are used to control the execution of the test, organize the flow of actions and specify properties of the questions of a test. Their value

at any moment of the plan execution can be fully determined as they will only change after an action has been applied.

– *Predicted variables*: Although in the general case the patient can behave non-deterministically and unpredictably, we can have some expectations about her behavior while performing the test, what we called *nominal behavior*. Predicted variables represent the expected behavior of the patient during the interaction, their true value does not depend on the system, but on the patient, but planning is done assuming certain outcomes as a result of the patient's actions.

– *Unpredictable variables*: They are used to represent the effects of actions of the user that do not directly respond to an ideal interaction flow. They are not considered in the planning process as their occurrence cannot be predicted in advance. They appear as consequence of unexpected events that break the nominal flow and trigger corrective actions, whose aim is indeed to make them disappear.

This differentiation allows us to conceptually divide a state of the world into two parts: one made of variables whose values will be (almost) always as expected (internal ones), and another one comprising the variables that can change unexpectedly, invalidating the current plan in the middle of its execution (predicted and unpredictable ones).

3.2 Execution

Once the plan has been created it must be executed in the real and dynamic environment. In fact, the mechanism to join planning and execution is usually up to each developer and depends usually on the chosen architecture. In our case we are using the PELEA architecture [1]. This section describes the generic items that must be considered in executing the generated plans of actions for CGA.

Continuous Monitoring. While planning we assume actions are instantaneous since we do not need to reason about time and there are not concurrent or parallel actions. But of course actions do have duration while executing. This implies that for some preconditions it is not enough to monitor whether they hold when the action starts. It must be ensured they are kept during the whole execution. Then, we have adopted a solution similar to that of temporal *action-based* planning [9], specifying for each precondition whether it must be true at the beginning, end or during all the action execution. To simplify the search process this information is not taken into account when planning, it is only used while monitoring: the system is continuously monitoring the real state and comparing it to the expected one. This ensures the robot can react properly when something unexpected occurs.

Triggering Replanning. Different options exist in order to decide when to replan. Always replanning after a single change in the state can lead to unnecessary replanning as this change may not affect the plan, but allows also to

take advantage of opportunities. Replanning only if the current plan is not valid requires a more complex monitoring mechanism, but reduces the need for replan. Finally is it also possible to replan only if the change invalidates the next action; this reduces even more the number of replanning episodes, but can lead to dead ends. In our domain there are no opportunities as the nominal behavior plan needs always to be followed, being the shorter path to reach the goals. Nor there are dead ends as it is always possible to find a plan that finishes the test, either successfully or not. Taking that into account we opted for the last approach, replanning only if the next action cannot be applied or if the preconditions of the current action are invalidated during its execution.

Interrupting Actions. A true interactive system must be responsive in any moment of its execution. This is especially important while it is executing an action but it has not finished yet. For example, if the robot is executing a `Say` action, which involves a 20 s speech and the patient leaves the room in the fifth second, the robot must be able to interrupt the speech in the middle of its execution. If not, it would continue speaking to nobody during the next 15 s before realizing that it has to call the patient to return to the test area.

Planning Time Restrictions. After an unexpected event, the robot stops in the middle of the current interaction while replanning, as it does not know the next action until the new plan is created. This imposes strict planning time restrictions: planning must be performed so there is no detectable delay in the interaction. It is very important to keep replanning times as low as possible to achieve a fluid interaction. Replanning times higher than 3 s are considered non-assumable for a proper and seamless interaction.

4 Example of Domain and Problems

In a first version of the work, a different domain was created for each of the tests. Actually, a first domain was designed to conduct the Barthel's test, which was later updated to be able to cope with MMSE and Get Up & Go tests. As a result three similar but slightly different domains were created, which resulted in a kind of *spaghetti code*. Although preliminary tests with real patients have been conducted using those domains, updates and maintenance became a problem. For that reason, taking into account the learned lessons, we designed a second unique domain, able to represent the three tests. Each test will be instantiated using a different problem.

The domain contains two types of actions, those that conform the nominal flow and the corrective ones that return to it in case of unexpected events.

Figure 2 shows the PDDL description of the *Say* action, the parameters are the current question and the specific label inside the question that identifies what will be said. In the preconditions, (1) accounts for all the external events, if any appears, this fact will be false preventing the action from being executed.

```
(:action say
   :parameters    (?q - question ?l - label)
   :precondition  (and (can_continue)                      (1)
    (test_configured)                                       (2)
    (= (question_position ?q) (current_question))          (3)
    (belongs ?l ?q)                                         (4)
    (= (label_order ?l) (current_label))                   (5)
    (< (repetitions ?l) (max_repetitions ?l))              (6)
    (not (answer_received ?q))                             (7)
    (not (waiting_for_behavior ?l))                        (8)
    (not (answer_needed))                                  (9)
    (not (pending_confirmation)))                         (10)
   :effect  (and  (increase (repetitions ?l) 1)
    (when (needs_feedback ?l) (answer_needed))
    (when (not (needs_feedback ?l)) (increase (current_label) 1))))
```

Fig. 2. Example of an action: the *Say* action.

Next preconditions are used to check the current question and the current label to be said (3–5), while controlling the maximum number of repetitions for this question has not been achieved yet, nor the answer has been already received (6–7). Finally we check too we are not waiting for the robot to perform any action (8), or for the patient to answer (9) or to confirm a previously entered answer (10). In the effects, the number of repetitions is increased. If after this action

```
(= (question_order q1) 3) ← the position of the question in the test
(behavior_of_event q1 question_failed ignore) ← what to do if an answer is not
provided (ignore means the question is marked for later answer)
(behavior_of_event q1 doctor_needed call_doctor) ← what to do if doctor needed
(behavior_of_event q1 patient_absent call_patient) ← what to do if the patient
is not visible
(behavior_of_event q1 max_q_failed call_doctor) ← what to do if after failing this
question the max number of failed questions is reached
(behavior_of_event q1 max_a_failed ignore) ← what to do if the patient fails
max_failed_answers in this question (notice that the system can try max_repetitions
to get an answer before marking it as missing)
(= (answers q1) 0) ← the number of answers received for this question
(= (answers_required q1) 1) ← the number of answers this question requires
(= (number_failed_answers q1) 0) ← the number of non-provided answers
(= (max_failed_answers q1) 1) ← the maximum number of non-provided answers
for this question before the max_a_failed event is triggered
(belongs q1_s1 q1) ← the label
(= (label_order q1_s1) 0) ← this is the first label
(= (max_repetitions q1_s1) 1) ← this label will be repeated at most once
(= (repetitions q1_s1) 0) ← the number of times this label has been repeated
```

Fig. 3. A partial example of the definition of a question of the Barthel's test.

the patient has to answer, a fact is inserted to account for it, if not, we continue with the question. Figure 3 shows how a question is encoded in a problem.

5 Preliminary Results

A preliminary evaluation was conducted in a retirement home in Seville, Spain, on November 2017 involving 8 patients. It was shown the robot is able to seamlessly conduct the CGA for the three selected tests, although many suggestions were made by the clinicians. Among many other improvements and changes, this resulted in a modification of the planning domains and problems. Practical reasons forced us to do those modifications while creating in parallel the unified domain. Prior to it inclusion in the real robot, severe testing in simulation has been performed to verify both domains are equivalent and the unified domain can be used in real practice with no changes in the rest of the system.

In this section we compare in simulation the planning and replanning times of both the original domain and the unified domain for the Barthel's and Get Up & Go tests. In the simulations, there is a 10% chance that the robot loses track of the patient. Additionally, in Barthel test, we have included an interaction error for each question (i.e., the robot does not receive the answer from the patient, or the patient does not respond) with a probability of 30%. Finally, in Get up & Go test, we have included different detection errors: the robot does not detect the patient near the chair (10%), it does not detect the patient seated (20%), and it detects the patient has walked more than the required distance (10%). Experiments were conducted on a 64 bits Intel Xeon 2.93 GHZ Quad Core processor with 2 GB RAM, using Linux and the Metric-FF planner [15].

Table 1 provides the results for each domain and test, analyzed across five dimensions: the accumulated time (seconds) used to perform the test , the number of actions in the executed plan, the number of times it is necessary to replan, the average time (seconds) needed to build a plan, and the average time needed to start executing each action (milliseconds). Means and standard deviations computed after ten different executions for each domain and test are shown.

Table 1. Results in Barthel and Get up & Go tests for the unified and specific domains.

Test	Domain	Time (s)	# Actions	# Replans	Planning (s.)	Response (ms)
Barthel	Unified	723.2 ± 42.9	123.5 ± 8.4	6.3 ± 2.8	0.08 ± 0.01	281.5 ± 5.3
	Specific	716.5 ± 64.5	102.0 ± 5.3	5.8 ± 3.3	0.33 ± 0.01	277.7 ± 6.8
GetUpGo	Unified	167.9 ± 8.9	33.0 ± 2.7	3.0 ± 0.9	0.01 ± 0.00	232.5 ± 4.9
	Specific	166.7 ± 9.2	20.0 ± 1.8	2.2 ± 0.7	0.05 ± 0.03	236.1 ± 5.6

The unified domain requires a higher number of actions to solve the tests, however, the accumulated time needed to perform each test is just slightly affected by this increment in the number of actions. Most of these additional

actions control the internal flow of the planning process and they do not have associated low-level actions to be performed on the robot. The planner finds plans much faster using the unified domain than the specific one. The unified domain simplifies the previous ones, containing just the information needed to reason and to drive the interaction between the patient and the robot (e.g., pauses between speech segments are not planned). The number of replanning episodes is not affected by the use of one or other domain and less than 250 ms. are needed to start executing each action.

6 Related Work

The use of Automated Planning for language and dialog generation, which is an important part of social interaction, goes back to the early 80s [20], but these efforts suffered from the poor performance of planners available that time. More recently, AP has been used for situated natural language generation, by encoding a PDDL problem that can be solved by any planner [10]. Instead of generating the language, our phrases are prerecorded and is the interaction flow what is generated using planning. In [22] domain-independent planning is used to generate conversations. They use a planner able to handle incomplete information and sensing actions by means of a non-standard representation language, which can be compiled to PDDL. The planner divides the information into five different databases and reasons to generate new knowledge that is stored into them. A similar approach, using the same planner, is used to perform action selection in a robot bartender scenario [21]. In particular, the task of interacting with human customers is mixed with the physical task of ensuring that the correct drinks are delivered to the correct customers. Conditional plans are generated if the needed information is not available yet. In the case of unexpected situations a new plan is generated. This application is quite similar to ours as both interaction and robot behavior must be planned, but we use standard PDDL, with no conditional plans, being able to use any planner released by the community.

The STRANDS project [14] aims to provide long-term autonomy to indoor robots, using also the MetraLabs SCITOS platform. One of the scenarios where their solutions has been deployed has been a large elder-care facility, where it acted as an information point, guided visitors and helped in walking-based therapies. Unlike in our case tasks are considered to be atomic so they are scheduled, not planned. An interesting feature, which we would like to explore in the future, is the robot ability to learn from experience. It is able to predict future states (for example if a door will we open at certain times), update traverse times and success ratio (using a Markov Decision Process), or learn user interaction patterns (probability of users wanting to interact at given times and locations).

Action-base Planning, Timeline-based Planning and Constrain-based Scheduling have been compared to control multiple robots deployed to assist elderly residents in a retirement home [23]. For that specific scenario, Constraint-based scheduling seems to be the most appropriate technique: PDDL-based planning finds always low quality plans, while Timeline-based planning is unable to

model the problem unless the solver is modified accordingly. Their approach involves multiple robots in a temporal problem that lies in the intersection of planning and scheduling. Our problem only involves a single robot, thus temporal aspects and concurrency are not imperative, and given the sequential nature of tests, plans are always optimal or near optimal. Scheduling of tests is currently done by the healthcare professionals, and although we plan to provide automatic mixed-initiative scheduling capacities to our system, scheduling a test and running it are to some extent independent and different techniques can be used for each of them.

7 Conclusions and Future Work

In this paper we have presented the decisions taken while modeling CGA interaction and showed that using the created domain and problems we can find plans able to control the robot while performing the Barthel's and Get Up & Go tests. Plans are found in less than a tenth of second, which fully meets the real time requirements for CGA interaction. The domain has been created to follow a nominal behavior of the patient. In the likely case that the patient does not stick to it, we use replanning to come back to the expected flow, and again this takes less than 0.1 s.

In a future we want to use the unified domain to encode also the MMSE test and any other test required by the clinical experimentation. A large scale pilot with real patients will be carried out in fall 2018 in retirement homes at Seville, Spain and Troyes, France.

References

1. Alcázar, V., Guzmán, C., Prior, D., Borrajo, D., Castillo, L., Onaindia, E.: PELEA: planning, learning and execution architecture. In: Proceedings of the 28th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG), Brescia, Italy (2010)
2. Allen, J.: Maintaining knowledge about temporal intervals. Commun. ACM **26**(11), 832–843 (1983)
3. Bandera, A., et al.: CLARC: a robotic architecture for comprehensive geriatric assessment. In: Workshop on Physical Agents, pp. 1–8 (2016). ISBN 978-84-608-8176-6
4. Barreiro, J., Boyce, M., Do, M., Frank, J., Iatauro, M., Kichkaylo, T., Morris, P., Ong, J., Remolina, E., Smith, T., et al.: EUROPA: a platform for AI planning, scheduling, constraint programming, and optimization. Technical report, 4th International Competition on Knowledge Engineering for Planning and Scheduling (ICKEPS) (2012)

5. Cesta, A., Oddi, A.: DDL.1: a formal description of a constraint representation language for physical domains. In: Ghallab, M., Milani, A. (eds.) New Directions in AI Planning. IOS Press, Amsterdam (1996)
6. Ellis, G., Langhorne, P.: Comprehensive geriatric assessment for older hospital patients. Br. Med. Bull. **71**(1), 45–59 (2005). https://doi.org/10.1093/bmb/ldh033
7. Fikes, R., Nilsson, N.J.: STRIPS: a new approach to the application of theorem proving to problem solving. Artif. Intell. **2**(3/4), 189–208 (1971)
8. Folstein, M., Folstein, S., McHugh, P.: Mini-mental state: a practical method for grading the cognitive state of patients for the clinician. J. Psychiatr. Res. **12**, 189–198 (1975)
9. Fox, M., Long, D.: PDDL2.1: an extension to PDDL for expressing temporal planning domains. J. Artif. Intell. Res. (JAIR) **20**(1), 61–124 (2003)
10. Garoufi, K., Koller, A.: Automated planning for situated natural language generation. In: Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, pp. 1573–1582 (2010)
11. Ghallab, M., Nau, D., Traverso, P.: Automated Planning: Theory and Practice. Elsevier, Amsterdam (2004)
12. Ghallab, M., Nau, D., Traverso, P.: The actor's view of automated planning and acting: a position paper. Artif. Intell. **208**, 1–17 (2014)
13. Gigante, N., Montanari, A., Mayer, M.C., Orlandini, A.: Timelines are expressive enough to capture action-based temporal planning. In: 23rd International Symposium on Temporal Representation and Reasoning (TIME), pp. 100–109. IEEE (2016). https://doi.org/10.1109/TIME.2016.18
14. Hawes, N., Burbridge, C., et al.: The STRANDS project: long-term autonomy in everyday environments. IEEE Robot. Autom. Mag. **24**(3), 146–156 (2017). https://doi.org/10.1109/MRA.2016.2636359
15. Hoffmann, J.: The metric-FF planning system: translating "ignoring delete lists" to numeric state variables. J. Artif. Intell. Res. (JAIR) **20**(1), 291–341 (2003)
16. Hoffmann, J., Porteous, J., Sebastia, L.: Ordered landmarks in planning. J. Artif. Intell. Res. (JAIR) **22**, 215–278 (2004)
17. Mahoney, F., Barthel, D.: Functional evaluation: the barthel index. Maryland State Med. J. **14**, 56–61 (1965)
18. Mathias, S., Nayak, U., Isaacs, B.: Balance in elderly patients: the get-up and go test. Arch. Phys. Med. Rehabil. **67**, 387–389 (1986)
19. McDermott, D.: PDDL - the planning domain definition language. Technical report. CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control (1998)
20. Perrault, C.R., Allen, J.F.: A plan-based analysis of indirect speech acts. Comput. Linguist. **6**(3–4), 167–182 (1980)
21. Petrick, R.P.A., Foster, M.E.: Planning for social interaction in a robot bartender domain. In: Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling, pp. 389–397 (2013)
22. Steedman, M., Petrick, R.P.A.: Planning dialog actions. In: Proceedings of the 8th Workshop on Discourse and Dialogue (SIGDIAL), pp. 265–272 (2007)
23. Tran, T.T., Vaquero, T., Nejat, G., Beck, J.C.: Robots in retirement homes: applying off-the-shelf planning and scheduling to a team of assistive robots. J. Artif. Intell. Res. **58**, 523–590 (2017)

24. World Health Organization: Ageing and life-course. Facts about ageing (2014). http://www.who.int/ageing/about/facts/en/
25. Yoon, S.W., Fern, A., Givan, R.: FF-Replan: a baseline for probabilistic planning. In: International Conference on Automated Planning and Schedulling (ICAPS), pp. 352–359 (2007)