Generating scenarios for a mobile robot with an arm

Case study: Assistance for handicapped persons

P. Morignot, M. Soury, C. Leroux
CEA, LIST, Interactive Robotics Laboratory
B.P. 6, Fontenay aux Roses, F-92265, France
{philippe.morignot mariette.soury
christophe.leroux}@cea.fr

H. Vorobieva, P. Hède
CEA, LIST, Vision and Content Engineering Laboratory
B.P. 6, Fontenay aux Roses, F-92265, France
{helene.vorobieva patrick.hede}@cea.fr

Abstract—**In this paper, we present a mobile robot with an arm and a gripper, which can generate its own scenarios before executing them. Hand-written scenarios, as we previously did, are not applicable any longer, since a wider range of scenarios is aimed at, which cannot be predicted in advance. Our approach involves task planning, for scenario generation, and finite state automation, for scenario execution. Our robot is used for servicing handicapped or ageing persons in their apartment.**

***Keywords*—scenarios generation and execution, mobile robotics, control model, task planning, plan execution, service robotics.**

I. INTRODUCTION

Significant research efforts are nowadays devoted to help disabled/ageing persons. Automatic assistance via a mobile robot appears as a rapidly growing subfield, as one possible direction for robotic service to persons (e.g., [1][2][11][12][13]). Current concepts in medical assistance of such persons involve mostly remote control of a robotic agent by a nurse. New concepts involve the disabled/ageing person himself asking requests to the agent (without external help), hence leading to his increased autonomy in his environment. This also entails more autonomy of the robotic agent and adapted communication with the disabled/ageing person.

The ITEA ANSO project [11][12] addressed the feasibility of a robotic assistant for disabled/ageing persons[1]. It proved that (1) typical scenarios in an apartment can be successfully carried out by a mobile robot, and that (2) all quadriplegic people preferred this type of assistant to equipment like a wheel chair with a robot arm.

However, global requests from the disabled/ageing person are limited, due to the hand-coding of scenarios describing the various actions the robot must take, to fulfil the handicapped person's request. A wider range of scenarios is aimed at, which cannot be predicted in advance.

The MIDAS project addresses this constraint (among others). Scenarios are generated using a STRIPS-planning approach [4], enabling the serviced person to specify goals only. Scenarios are then turned into a finite state automaton, for execution.

[1] Videos are available on YouTube under the keywords "CEA" and "SAM".

This paper is organized as follows: First, the context involving scenarios is presented (the robot's sensors and effectors, the intended scenarios and the software architecture for components organization). Then, we focus on task planning and execution, for scenarios generation (section III). The implementation leads to experimental results, described in section IV. Finally, we state differences between our approach and preceding ones, and sum up our contribution.

II. CONTEXT

A. The robot SAM

The robot [12] is a non-holonomic mobile base NEOBOTIX MP-M470 with a 6-DOF MANUS arm ending with a gripper (see Figure 1). Its sensors are forward- and backward-oriented lasers located on the base (for obstacle avoidance), a panoramic camera located on top of the base (for scene detection), 2 webcams located on the arm (for object recognition, distance stereo-measurement, and visual servoing [9]) and an optical barrier located in the gripper (for decision on clamp closure/opening).

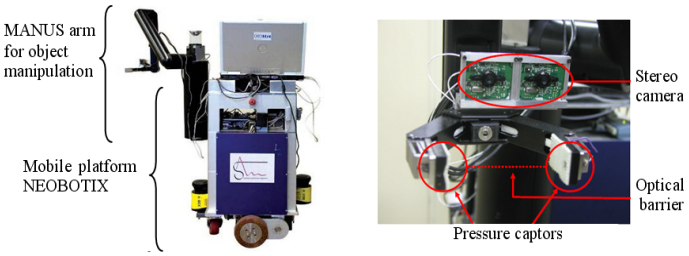

Figure 1: The robot (left) and its gripper (right).

B. Scenarios

The scenarios considered in the MIDAS project consist in going to another room of the indoor environment, grasping an object, coming back to the person's location, and handing the object close to the person's mouth. We also envision scenarios such as searching for objects in the apartment, and helping a person to cook.

Figure 2 shows a metric map of the apartment used for executing the scenarios: yellow lines represent walls (small rectangles are obstacles); Red areas denote forbidden zones.

978-1-4244-7815-6/10/$26.00 ©2010 IEEE

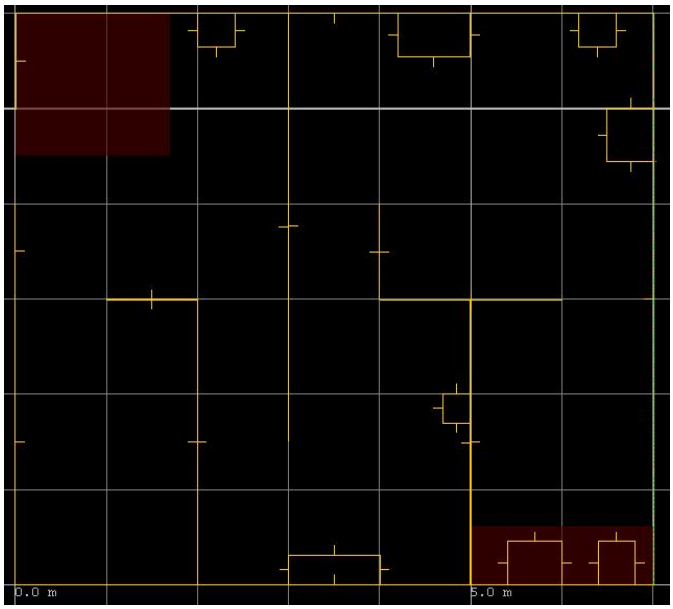

Figure 2: Metric map of the apartment.

C. Architecture

All software components are integrated as web services in a Device Profile Web Services architecture [7]. They include (i) moving the mobile base to a 2D location in the apartment, (ii) moving the mobile base closer to an object to grasp, in case it lays out of range of the arm [2], (iii) visually recognizing a given object in an image given a list of known object images [9] and (iv) determining the successive poses of the arm depending on the shape and orientation of the object to grasp [15].

III. SCENARIOS GENERATION

Among the components of the robot's software architecture, we now focus of the scenario generation one. Our approach uses task planning [4] and finite-state automata execution [10].

A. Task planning

A task planner uses a domain (the descriptions of the operators) and a problem (an initial state and goals), described in the syntax of the Planning Domain Definition Language (PDDL), to generate a sequence of totally instantiated operators which together lead the initial state to a (final) state which satisfies the goals [4].

For this, the CPT planner (Constraint Programming Temporal planner [14]) creates variables with finite domains (possible values for each variable) and constraints (relations between variables, which much always hold). A variable is the start instant of an operator. The constraints represent minimal and maximal time bounds on each task, time bounds on each precondition of each task (support), causal links for each pre-condition of each task, and mutual exclusion constraints between preconditions. Constraint programming is then used to search for an instant for each variable which satisfies the constraints, for a specific plan length. If no solution is found, the plan length is increased, and a solution is searched for in this new space [14].

As a result, CPT is optimal (it finds the plan with the shortest plan length), and is capable of making inferences very quickly, but is canonical (an instantiated action appears only once in the solution plan).

Possible actions taken by the robot are described as PDDL operators. The initial planning state is a description of the environment of the robot. Goals are specified by the user.

B. Finite state automaton

A finite state automaton reasons on entities called states. One state is declared as initial --- the control flow starts on this state. When the control flow reaches one state, handlers (functions in the implementation language) are activated, to represent the semantics of this state. In the ISEN automata executor [10], handlers are divided into ON_ENTRY, ON_DO and ON_EXIT categories, to sort their execution. In each category, handlers are executed in the order in which they are declared. Once handlers have been activated, a condition (a function in the implementation language) is checked: if it succeeds, control flow jumps to another state (also specified in the current state). Several conditions and jumps can be specified in a state. A special state, called "AllStates", is considered as always active: the control flow activates its handlers and checks its conditions at every time step.

A handler can use variables as arguments, and return a value in another variable. Initial values of variables (of type INTEGER, BOOLEAN, STRING, or VECTOR) can be defined. Since several handlers can be stated in sequence in a category (see above), a basic programming language is hence available [10].

C. Connecting task planning to finite state automation

Task planning produces a sequence of instantiated operators, given an initial state and goals. Finite state automation produces the behaviour of the robot, given a scenario. A scenario is composed of states, to which handlers are attached; the control can jump from the current state to a next state through triggering conditions.

The initial planning state is a description of the environment of the robot; goals are specified by the user (i.e., the handicapped person). The possible operators (the planning domain) are description of the actions that the robot can take. Therefore the instantiated operators of the generated plan set the nature and order in which these physical actions must be executed, in order to reach the goals.

Once an action plan is built, an instantiated operator is identified to a state of a finite state automaton. The semantics is: *in a state (of a finite state automaton), the robot is executing an action (of a task planner)*.

Then, the handlers, controlling the actual motion of the robot' sensors and effectors, are attached to each instantiated operators / state. States are linearly chained, since the structure of a plan is also linear in our case. The next state's triggering conditions are given in the handlers' specification.

Finally, the scenario is thus complete and can be executed.

IV. Experimental Results

In this section, we describe our implementation and the results which we obtain with it.

A. Task planning

Task planning is performed using the CPT planner [14] encoded in the programming language C using the GDSL library. A domain and a problem (initial state and goals) are encoded in the Planning Domain Definition Language (PDDL).

The PDDL operators, which represent the actions that the robot can take, are listed in

Table 1.

```
(:action move-mobile-base
   :parameters (?pt1 - point2D ?pt2 - point2D ?pt - point3D
                ?r - rotation3D ?a - angle)
   :precondition (and (mobile-base ?pt1 ?a) (path ?pt1 ?pt2)
                      (arm-transportation ?pt ?r) (arm-pose ?pt ?r)
                      (not-open-clamp))
   :effect (and (mobile-base ?pt2 ?a) (not (mobile-base ?pt1 ?a))))

(:action arm-position-for-drop
   :parameters (?pt - point3D ?rot - rotation3D ?pt2 - point3D
                ?rot2 - rotation3D)
   :precondition (and (arm-pose ?pt ?rot) (arm-drop ?pt2 ?rot2))
   :effect (and (not (arm-pose ?pt ?rot)) (arm-pose ?pt2 ?rot2)))

(:action arm-position-in-front-of-scene
   :parameters (?s - scene ?pt2 - point3D ?rot2 - rotation3D
                ?pt - point3D ?rot - rotation3D)
   :precondition (and (approach-x-y-z ?s ?pt)
                      (approach-rx-ry-rz ?s ?rot)
                      (arm-pose ?pt2 ?rot2) (not-clamp-open))
   :effect (and (arm-pose ?pt ?rot) (not (arm-pose ?pt2 ?rot2))))

(:action arm-position-in-front-of-object
   :parameters (?o - object ?pt2 - point3D ?rot2 - rotation3D
                ?pt - point3D ?rot - rotation3D)
   :precondition (and (approach-x-y-z ?o ?pt)
                      (approach-rx-ry-rz ?o ?rot)
                      (arm-pose ?pt2 ?rot2) (not-clamp-open))
   :effect (and (arm-pose ?pt ?rot) (not (arm-pose ?pt2 ?rot2))))

(:action arm-position-for-transportation
   :parameters (?pt1 - point3D ?rot1 - rotation3D ?pt2 - point3D
                ?rot2 - rotation3D)
   :precondition (and (arm-pose ?pt1 ?rot1)
                      (arm-transport ?pt2 ?rot2))
   :effect (and  (not (arm-pose ?pt1 ?rot1)) (arm-pose ?pt2 ?rot2)))

(:action open-clamp
   :parameters ()
   :precondition     (not-open-clamp)
   :effect (and  (not (not-open-clamp)) (open-clamp)))

(:action drop-object
   :parameters (?o - object ?pt - point2D ?a - angle ?pt2 - point3D
                ?rot2 - rotation3D)
   :precondition (and (clamp-holds ?o) (mobile- base ?pt ?a)
                      (arm-drop ?pt2 ?rot2)
                      (bras-pose ?pt2 ?rot2))
   :effect (and (not (clamp-holds ?o)) (not (not-clamp-open))
                (open-clamp) (position ?o ?pt)))

(:action close-clamp
   :parameters ()
   :precondition     (open-clamp)
   :effect  (and (not (open-clamp)) (not-open-clamp)))

 (:action blindly-move-gripper-towards-object
   :parameters (?o - object ?pt - point3D ?r - rotation3D
                ?pto - point2D ?a - angle)
   :precondition (and (arm-pose ?pt ?r)
                      (evt- selected-object ?o)
                      (approach-x-y-z ?o ?pt)
                      (approach-rx-ry-rz ?o ?r)
                      (position ?o ?pto) (mobile- base ?pto ?a)
                      (open-clamp))
   :effect (evt-optical-barrier-crossed))

 (:action close-clamp-on-object
   :parameters (?o - object ?pt - point3D ?r - rotation3D
                ?pto - point2D  ?a - angle)
   :precondition (and (arm-pose ?pt ?r)
                      (evt-selected-object ?o)
                      (approach-x-y-z ?o ?pt)
                      (approach-rx-ry-rz ?o ?r)
                      (position ?o ?pto) (mobile- base ?pto ?a)
                      (open-clamp) (evt-optical-barrier-crossed))
   :effect (and (not-open-clamp)
                (not (open-clamp)) (clamp-holds ?o)))

(:action choose-object-in-scene
   :parameters (?s - scene ?o - object ?pt - point3D ?r - rotation3D)
   :precondition (and (approach-x-y-z ?s ?pt)
                      (approach-rx-ry-rz ?s ?r)
                      (arm-pose ?pt ?r)
                      (usr-selected-zone ?s ?o))
   :effect (evt- selected- object ?o)))
```

Table 1: PDDL operators for the domain SAM.

PDDL typing and reduction of the number of variables in operators are necessary: CPT performs a cross product on the variable's domains of an operator, which might lead to computer memory exhaustion (before plan search).

The robot's arm is controlled by setting a 3D location to the gripper's basis in Cartesian or angular mode (no need to re-write direct and reverse geometric models for the arm). This entails that general 3D points can be encoded in a symbolic way into the initial state of the planning problem. For example, precondition *(approach-x-y-z ?o ?pt)* in the *blindly-move-gripper-towards-object* operator of Table 1 represents the way object *?o* must be approached by the robot's arm in Cartesian mode. This 3D position can be encoded as variable *?pt* --- the arm can infer at execution time the desired motion for each arm's engine until the desired 3D location is reached (see [6] for a discussion on geometric constraints to extend PDDL).

Encoding the mobile base's 2D motion on a topological map is performed by using a fluent *(path ?pt1 ?pt2)*, and fluents stating that the mobile base now is at location *?pt2* and not at location *?pt1* any longer.

Similarly, opening and closing the gripper is performed using fluents *(clamp-open)* and *(not-clamp-open)* --- the latter, instead of *(not (clamp-open))*, is due to the use of strict STRIPS formalism [4], excluding the use of negated preconditions.

Table 2 shows a plan generated by CPT for object grasping, given the goal *(position migralgine pt5)* and an initial state assigning locations to objects and describing a topological map: the robot has to move to the location of the constant *migralgine*, grab it, move to constant location *pt5* and drop *migralgine*. (In Table 2, the first figure denotes the execution instant, the last one denotes the operator's duration.) This plan is successfully built in 0.08s on a computer with 4-CPUs at 2.66GHz with 3.37Go RAM.

We notice that the only recursive fluent (a post-condition of an operator which is also stated in pre-condition of the same operator) only occurs in the operator *move-mobile-base* of the

SAM domain: the *path* fluent (cf. Table 1). This could lead to combinatorial explosion if the planner had to search in a large tree or graph along the *path* fluent --- as this is the case for the domain *rover* of the International Planning Competition [8], for example. Fortunately, the apartment in which the robot moves has a small scale topology (cf. Figure 2). Hence this combinatorial explosion, which is possible in theory, is *not* encountered in our practical environment.

We also notice that the planning speed is less than one second for a plan length of approx. 20 operators (in our configuration). Given the motion speed of the robot (20 cm/s) and of the arm, this is acceptable for future on line re-planning and, for now, for user acceptability.

Finally, we notice that the plans, generated by the task planner CPT in our domain, are linear. A non linear plan can be represented by the finite state automaton, but would lead to increased complexity in the translation from plans to scenarios.

0: (arm-position-for-transportation ptr3 rotr3 ptrt rotrt) [1]
1: (move-mobile-base pt1 pt2 ptrt rotrt abm) [1]
2: (arm-position-in-front-of-scene table ptrt rotrt ptr1 rotr1) [1]
3: (choose-object-in-scene table migralgine ptr1 rotr1) [1]
4: (arm-position-in-front-of-object migralgine ptr1 rotr1 ptr2 rotr2) [1]
5: (open-clamp) [1]
6: (blindly-move -gripper-towards-object migralgine ptr2 rotr2 pt2 abm) [1]
7: (close-clamp-on-object migralgine ptr2 rotr2 pt2 abm) [1]
8: (arm-position-for-transportation ptr2 rotr2 ptrt rotrt) [1]
9: (move-mobile-base pt2 pt3 ptrt rotrt abm) [1]
10: (move-mobile-base pt3 pt4 ptrt rotrt abm) [1]
11: (move-mobile-base pt4 pt5 ptrt rotrt abm) [1]
12: (arm-position-for-drop ptrt rotrt ptrd rotrd) [1]
13: (drop-object migralgine pt5 abm ptrd rotrd) [1]

Table 2: Plan of instantiated operators, for object grasping.

B. *Finite state automata*

Scenarios are encoded and executed by the finite state automaton executor ISEN [10], written in the programming language C++ with the ALTOVA library for handling XML structures. An automaton / scenario for object dropping is presented in Figure 3, in the IsenEdit graphical interface.

Typical scenarios, translated from the previously generated plans, involve less than 20 operators / states --- consider for example the 13 actions plan of Table 2. This corresponds to the length of existing scenarios, which were previously written manually.

The state "AllStates" cannot be encoded simply from a plan: we chose to use it as an emergency condition catcher. This state is added by the translator, and contains the emergency conditions which were common in hand written scenarios (e.g., stopping the arm motion, stopping the base motion).

C. *Translation of plans into scenarios*

The translator, encoded in the programming language C++, converts a plan generated by the CPT task planner into an XML scenario for the finite state automaton executer ISEN.

It calls the CPT planner, parses the generated plan, parses the handlers' descriptions, attach each handler description to the corresponding instantiated operator / state, and generate a scenario in XML format.

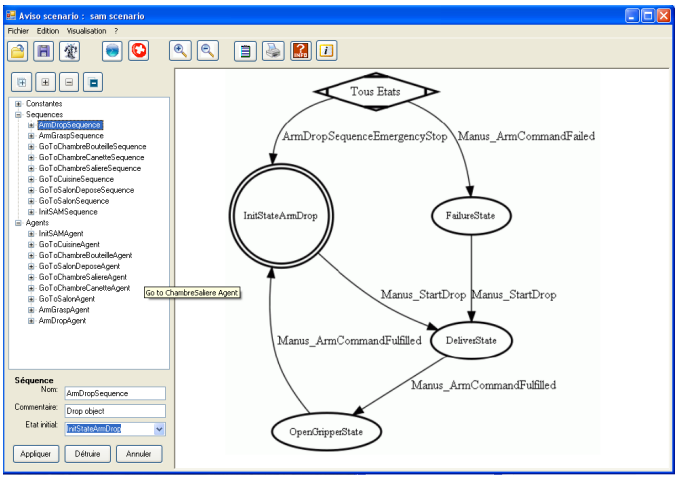

Figure 3: Finite state automaton for object dropping.

The translator is used before any action is executed on the robot (off-line planning).

```
<?xml version="1.0" encoding="UTF-8"?>
<Model_Xml xmlns="http://www.Root.com/schema">
 <IsenPropertiesSet xmlns="">
  <constant name="c1" type="string" contents="Display
                        message"/>
  <constant name="c2" type="vector"
                    contents="(11,22,33,44,55,66,77,88,99)"/>
  <constant name="c3" type="boolean" contents="true"/>
  <constant name="c4" type="boolean" contents="false"/>
  <constant name="pi" type="integer" contents="314159"/>
 <sequence name="Collection1">
  <state name="arm-position-for-transportation ptr3 rotr3 ptrt
                                    rotrt" type="INITIAL_STATE">
   <on_entry> <action>on_entry1()</action></on_entry>
   <on_event name="event2">
    <transition note="event2" next_state="move- mobile- base
                                        pt1 pt2 ptrt rotrt abm"/>
   </on_event>
  </state>
  <state name="move-mobile-base pt1 pt2 ptrt rotrt abm"
        type="NORMAL_STATE">
   <on_entry>
      <action>on_entry21()</action>
      <action>on_entry22()</action>
      <action>on_entry23()</action>
   </on_entry>
   <on_do> <action>on_do2()</action> </on_do>
   <on_exit> <action>on_exit2()</action> </on_exit>
    <on_event name="event3">
     <transition note="event3" next_state="arm-position-in-front-
                                of-scene table ptrt rotrt ptr1 rotr1"/>
```

```
      </on_event>
      </state>
      <state name="blindly-move-towards-object soda-can ptr2 rotr2
                                  pt2 abm" type="NORMAL_STATE">
        <on_entry> <action>on_entry7()</action> </on_entry>
        <on_do> <action>on_do7(c3=$c3)</action> </on_do>
        <on_exit>
          <action>on_exit7(c3=$c3, c4=$c4)</action>
        </on_exit>
    <on_event name="event8">
     <transition note="event8" next_state="close-clamp-on-object
                                    soda-can ptr2 rotr2 pt2 abm"/>
    </on_event>
   </state>
   <state name="open-gripper-holding-object soda-can pt15 abm ptrd
                rotrd" type="FINAL_STATE"
                note="Open the gripper">
     <on_entry>
        <action>DisplayAgentStatus(Brief=true)</action>
        <action>CommandGripperOpen(GripperAction=$OPEN_
                GRIPPER_FULL)</action>
     </on_entry>
     <on_event name="Manus_ArmCommandFulfilled">
        <action>displayMessage(Message="Object drop. Please
                Click on Arm Drop ", TYPE=$HIGH)</action>
    </on_event>
   </state>
   </sequence>
   <agent name="Automate1" start_sequence="Collection1"
                                start_priority="1"/>
</IsenPropertiesSet>
</Model_Xml>
```

Table 3: Structure of a generated scenario (object fetching and dropping), in XML format, corresponding to the instantiated operators of Table 2. (Only 3 states are displayed.)

The scenario corresponding to the plan of instantiated operators of Table 2 is shown in Table 3 (only 3 states are shown, due to paper space limitation).

First, a block of constants ("*c1*" to "*c4*", using types INTEGER, BOOLEAN, STRING and VECTOR) is declared; Then the sequence "*Collection1*" is declared, composed of the following states (only an excerpt is shown in this table); State "*arm-position-for-transportation*" matches the first instantiated operator of Table 1 (the plan indicates that this is the first operator, the translator follows the same order). Its unique handler is the C++ function "on_entry1()", called in ON_ENTRY category. This state jumps to the state "*move-mobile-base*", which appears in Table 3 as the second (it also appears as the second instantiated operator of the plan of Table 2). That state owns 3 successive handlers in the ON_ENTRY category. Later in Table 3, some variables ("*c3*" and "*c4*") are used as arguments of ON_DO and ON_EXIT handlers. The last state (in this excerpt) is "*open-gripper-holding-object*", which contains two ON_ENTRY handlers' calls, and calls a display-message handler on the "Manus_ArmCommandFulfilled" event. Finally, the sequence "*Collection1*" is part of the ISEN agent "*Automate1*".

The scenario of Table 3 is typical of XML format files which can be executed by the finite state automata executer ISEN [10].

V. Discussion

There is a long trend of planning systems which are used for robotic applications, starting with the STRIPS task planner / PLANEX executer on the robot SHAKEY / FLAKEY of Richard Fikes in 1971 [3] --- the STRIPS task planner initiated the scientific field of domain-independent task planning. The principle of our approach follows the planning / execution cycle of SHAKEY, but in a more efficient way: Plans actually used in our scenarios, containing approximately 15 instantiated operators, are generated in less than 0.1 seconds (in our configuration), hence leading to scenario complexity which was unreachable in 1971 (the task planner represents a mobile base, with an arm and a gripper, which the robots SHAKEY / FLAKEY had not [3]).

Second, the Care-O-Bot robot [5] uses a symbolic planner to generate possible actions which fulfil a goal, and organizes the tasks into an execution module, representing time primitives (sequential / parallel, cyclic / discrete, wait-for). In contrast, our approach generates a whole sequence of actions (see Table 2), not just plans of length 1, as fulfilment of one or several goal(s); And Care-O-bot's dedicated execution module is a specific case of ISEN finite state automata executor --- it can encode Care-O-Bot's time primitives using events.

Cablé *et al.* use case-based reasoning to learn scenarios (a list of pairs *(location, action)*) to drive the electric wheelchair of a handicapped person [1]. Within this view, our approach appears as always generating the first-case scenario (no previous scenario to be adapted), then leaving room for later scenarios adaptation in a case-based approach.

When compared to these approaches, our approach, using domain-independent task planning for scenario generation and goal satisfaction, does not suffer these limitations: the robot reasons on its own actions through operators and task planning, hence re-programs its own future behaviour, as goals change --- this is made possible by the DPWS software architecture, for separating behaviors in single web services. The unique limit for scenario complexity, and adaptation capability, is the computation time devoted to task planning and the expressive power of PDDL for goal explicitation.

VI. Conclusion and Future Work

In this paper, we have presented a robot capable of generating its own scenarios, before executing them. Scenarios were previously hand coded, and the handicapped person chose one of them. However, a wider range of scenarios is aimed at, which cannot be predicted in advance.

Now the handicapped person specifies goals only: the robot uses its knowledge of its own actions (planning operators), and of its current environment (initial planning state) to generate a linear plan of instantiated operators, using CPT [14].

Each instantiated operator is then considered as a state of a final state automaton. Trigger conditions and handlers,

representing functions implementing the robot's behaviors, are attached to each operator/state.

The resulting finite state automaton is then executed by an automaton executor [10]. Hence the generated scenario is executed, and the goal (specified by the handicapped person) fulfilled.

Future work involves (i) reacting to on-the-fly events by re-planning (e.g., a door is closed, leading to generate a plan for opening the door first, or finding another path in the apartment), (ii) using an ontology for extracting the planning initial state, and (iii) building a graphical interface for easiness of goal specification by the handicapped person.

ACKNOWLEDGMENT

This work is funded by the DGE of the French Ministry of Economy, Finance and Industry through contract ITEA 2 MIDAS, as part of a EUREKA European project. We thank Vincent Vidal.

REFERENCES

[1] B. Cablé, J.– M. Nigro, S. Loriette, Y. Barloy, "Reasoning based on scenarios for motion assistance », 18th Workshop on Case-Based Reasoning (Raisonnement à partir de scénarios pour l'assistance au déplacement, 18e Atelier sur le Raisonnement à Partir de Cas), Strasbourg, France, June 2010, pages 13-24.

[2] J. Dumora. "Design of behaviors for a robot assisting handicapped persons" ("Conception de comportements pour un robot d'assistance aux personnes handicapées"). Technical Report, CEA, LIST, DTSI/SRI/08-XXX, August 2008, unpublished.

[3] R. Fikes, N. Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving", Artificial Intelligence, Vol. 2 (1971), pp 189-208.

[4] M. Ghallab, D. Nau, P. Traverso. "Automated planning : theory and practice". Morgan Kaufmann, Elsevier, San Francisco, 2004, 635 pages.

[5] B. Graf, M. Hans,, R. Schraft, "Care-o-bot II development of a next generation robotic home assistant". Auton. Robots, 16(2), pp, 193—205, 2004.

[6] J. Guitton, J.- L. Farges, "Taking into account geometric constraints for task-oriented motion planning", In Proc. Bridging the gap Between Task And Motion Planning, (BTAMP'09), ICAPS Workshop, Thessaloniki, Greece Sept. 2009.

[7] F. Jammes, A. Mensch, H. Smit. "Service-oriented device communications using the devices profile for web services", In AINA Work., pages 947–955, Washington, USA, May 2007.

[8] International Planning Competition, http://idm-lab.org/wiki/icaps/index.php/Main/Competitions

[9] M. Joint, P.-A. Moëllic, P. Hède, P. Adam, "HEKA : A general tool for multimedia indexing and research by content". 16th Annual Symposium Electronic Imaging (SPIE), San Jose 2004, Image Processing, Algorithms and Systems III.

[10] C. Leroy. ISENEdit: User Manual. CEA, LIST, LRI, Technical Report, 2005, unpublished.

[11] A. Remazeilles, C. Leroux, G. Chalubert, "SAM : A majorduomo for helping persons with reduced mobility – description of the remote grasping function" ("SAM : un majordome au service des personnes à mobilité réduite - description de la fonction de saisie à distance"), 5th Conference HANDICAP '08, June 10-12, 2008, Paris, France.

[12] A. Remazeilles, C. Leroux , G. Chalubert, "SAM: a robotic butler for handicapped people", 17th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN - 2008), 01/08/2008-03/08/2008, Munich, Germany.

[13] H. Van der Loos, Va/stanford rehabilitation robotics research and development program: Lessons learned in the application of robotics technology to the field of rehabilitation. In: IEEE Trans. on Neural Systems and Rehabilitation Engineering,1995, pp, 46—55.

[14] V. Vidal, H. Geffner, "Branching and Pruning: An Optimal Temporal POCL Planner based on Constraint Programming", Artificial Intelligence 170 (3), pp. 298-335, 2006.

[15] H. Vorobieva, C. Leroux, P. Hède, M. Soury, P. Morignot, "Object Recognition and Ontology for Manipulation with an Assistant Robot ", In Proceedings of the Eighth International Conference on Smart Homes and Health Telematics (ICOST'10), L.N.C.S., Aging Friendly Technology for Health and Independence, Springer, Berlin - Heidelberg, Germany, vol. 6159, pages 178-185.