Path-finding in dynamic environments with PDDL-planners

Path-finding in dynamic environments with PDDL-planners

Vladimir Estivill-Castro *School of ICT, Griffith University Nathan, 4111, QLD, Australia*
Email: v.estivill-castro@griffith.edu.au

Jonathan Ferrer-Mestres *DTIC, Universitat Pompeu Fabra Barcelona, Spain*
Email: jonathan.ferrer21@gmail.com

***Abstract*—The standardization of planning problems by their descriptions in the PDDL has resulted in clear benchmarking of planners, and thus, in significant advances in reliable and efficient planning packages. The output of these classical planners is a plan as sequence of actions for the controllable robots in the environment. We show here that, provided that the adversaries follow a deterministic behavior, PDDL-planners can also be used in dynamic environments where uncontrollable adversaries may obstruct paths at some time in the future. Therefore, these environments can be used by mobile robots without the need to use more sophisticated planners where environments are modeled by Markov Decision Processes (MDPs). We created a planning API for integrating any PDDL-solver and use it to elaborate platform independent planning behavior. We also have the ability of switching between PDDL-solvers or to change the integration cycle of the planner. We show that these two features are essential for the dynamic environments considered here.**

***Keywords*-Navigation, Behaviour-based robotics, Path and task planning, Adversarial Planning, Robotic Architectures.**

I. Introduction

STRIPS-like planning [1, page 49] is perhaps one of the simplest forms of planning, and in particular, a rather common planning approach for navigation tasks. While the robotics community has moved forward into problems that include temporal environments with uncontrollable obstacles, the planning community has standardized STRIPS-like planning with the introduction of the Planning Domain Definition Language (PDDL) and its extensions [2], [3] in order to consider *mixed discrete-continuous domains*.

PDDL support for the AIPS Planning Competition has resulted benchmark planning problems (see www.plg.inf.uc3m.es/ipc2011-deterministic/Domains). The original fully deterministic and observable planning challenges have been complemented by *planning and learning* domains as well as *uncertainty* challenges (where the effects of the actions are uncertain and usually modeled by some probability distribution of the outcomes). Nevertheless, we are motivated by the fact that the domains, under the original (or "classical") planning part, seem mostly derived from scheduling challenges. This is not surprising, as the plan is usually a schedule of the actions of the agent/robot in the domain. If the domain contains more controllable agents, they are also specified as part of the actions in a sequence. A prototype of these challenges is 'Sokoban' [4]. In this transport puzzle, the agent/robot pushes crates (or boxes) in the interior of a warehouse. The goal is to place all objects in special cells of the grid environment that are designated as storage locations. The agent is confined to the warehouse, and actions are horizontal/vertical moves onto empty squares (never through walls or boxes). The player can also move into a box, which pushes it into the square beyond. Boxes may not be pushed into other boxes nor into walls, and they cannot be pulled.

We consider this type of problems in a robotic environment with the additional challenge that planning may happen while carrying out a particular plan (which may imply re-planning), and also, in the presence of other robots/agents for which the robot has no control. These adversaries will perform deterministic trajectories governed by logic-labeled finite-state machines (LLFSMs).

Therefore, this paper proposes (1) a methodology to describe (in PDDL) scenarios where the controllable robot is not the only moving object in the environment. It also indicates (2) a software architecture for carrying out model-driven development of the integration of the planners into the capacities of a robot. We demonstrate these proposals with a scenario of a robot navigating in an environment that includes fixed obstacles and that includes roving adversaries.

It is important not only that the planner be PDDL-based for standard use; the fundamental heuristic and searching algorithm behind the planner may be more suitable for dynamic environments where other robots are present [5]. Moreover, it is recognized within the agents community that there is a trade-off between the effort/time invested in planning and the effort of carrying out with business (acting) [6]. Such trade-off has been manifested in the robotics community with the suggestion of the "execution" driven cycle [7] as an alternative to the plan-execute cycle. Namely, in the plan-execute cycle, the planning module remains idle while the robot is performing a plan. Re-planning occurs once the robot has found an unknown obstacle. This is suitable for static environments where we rarely encounter unknown obstacles. However, in a much more dynamic environment, long plans will rarely be useful, specially the tail of the plan. In such settings, the planner could be concurrently exploring alternative plans (and not

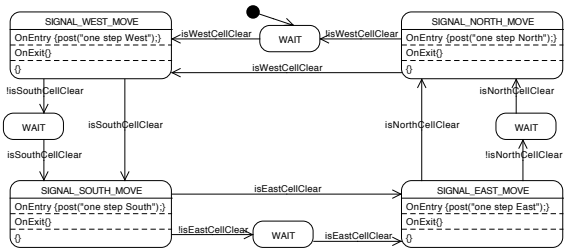

Figure 1: A LLFSM that encodes the behavior of a simple adversary that loops in 4 positions.

just policies) while the execution of the current plan is going on. Under the name of *reactive executive monitoring* [8] a reactive planning engine [9] monitors the systems low-level states against a declarative model of the robot's functionality, while continuously performing sense-plan-act cycles [10]–[13]. In the planning stage, the reactive planner finds a plan (a sequence of robot actions) under some parameters; which may include a planning horizon. Such a parameter balances long planning times with a reactive behavior. During the monitoring, the belief of the robot can be contrasted with operator directives and enable supervised autonomy [14] and integration of intervention by a human operator [15].

We will not focus in this paper on the re-planning aspect and the plan-while execute aspect; suffice to say that our architecture accommodates this and enables our robots to navigate in dynamic environments where the initial positions of other moving robots/adversaries (for which the main robot does not have control) are also unknown, but using a classical planner. The current main stream approach (to model partial observability as non-deterministic search [16]) for both a) contingent planning [17] and b) POMDP planning [18], has shown serious scalability issues [19]. Therefore, several researchers have studied classes of problems and produced methods to translate the partial observability problem description into a fully observable planning problem solved by a classical planner [19]. We emphasize that this paper is also an effort in this direction.

II. Deterministic planning

Our argument here is that, such LLFSMs can describe the behavior of Logic-labeled finite-state machines (LLFSMs) are models of behavior that use the ubiquitous model of state machines widely used for representing software behavior [20] and the behavior of embedded systems (with tools like *MathWorks® StateFlow* with *Symlink*). But, as opposed to the event-driven finite-state machines in UML [21], LLFSMs use the alternative model of time [22] where transitions are labeled by statements of a formal logic. For robotic systems, LLFSMs have been used in the language XABSL (where the transitions are labeled by decision trees [23], [24]) and with Defeasible Logic (DPL) [25], [26]. The version using DPL has also been used to model embedded systems as arrangements of LLFSMs can model concurrent behaviors. Moreover, the arrangement of LLFSMs is scheduled deterministicly enabling formal verification [27], [28].

other components of an environment for a robot. Moreover, while deterministic planners may have been conceived for environments where the solution is a sequence of actions structured as a plan for the components of the environment that there is control, we suggest that we can use such deterministic planners even for environments where there is dynamic behavior of environment elements for which the planner and the robot do not have control — as long as those other components have their behavior described by LLFSMs. That is, we will use deterministic planners in deterministic environments, but the environments we consider are much more dynamic as they will include other objects that act on the environment. The idea is similar to the planning happening in *Cooperative* $A^\star$ [5], (that moves from reasoning on space to reasoning on time and space), but rather than having a dedicated planner with $(position, time)$ states, we create the description in PDDL and any PDDL-planner can attempt[1] to build a plan.

To illustrate this point, we again refer to the Sokoban grid-based environment. This is considered an extremely challenging environment for deterministic planners. At the international planning competition, some instances have been recycled to the next competition as all solvers performed poorly in the puzzle. However, only one robot moves around and solvers consider only the moves of this robot. As a result, only the effects of the plan modify in any way the environment. We illustrate our suggestion of a dynamic environment by considering other adversaries in the environment that also move. An example of such adversary could be another robot in the environment for which our controllable robot only knows its deterministic behavior as it is expressed in a LLFSM. This is an extension on Sokoban that makes the environment dynamic.

Navigation behavior as a LLFSM is natural for patrolling robots and can be synthesized out of specifications from a temporal logic [29]. In fact, our robot could actually know the LTL specification of the other robots and use such approach [29] to infer the LLFSM of the other adversaries in the environment. For illustration purposes, Fig. 1 presents an example of a LLFSM for this extended grid-based environment. This agent loops around 4 states, successively moving one cell West, one cell South, one cell East and one cell North in the grid environment. Clearly, grid-environments are an abstraction (for planning and reasoning) of the real-world environments robots navigate in (we make the same remark as others [29] that lower level behaviors achieve grid-environment abstractions such as moving a robot from one cell to another, or from one room to another).

[1]PDDL-description facilitates using any PDDL-planner, but different search strategies and driving heuristic, result in different planning behavior.

```
(define (domain patrolling)
        (:requirements :strips :action-costs)
        (:predicates
                (connect ?p0 ?p1)
                (next ?p0 ?s1 ?p1 ?s2)
                  (at-robot ?p0)
                (at-adversary ?p0 ?s)
)
(:functions
        (total-cost) -number
)

(:action move-is-possible
        :parameters (?pos0 ?pos1  ?pos3 ?pos4 ?s1 ?s2)
        :precondition (and(at-robot ?pos0) (connect ?pos0 ?pos1) (not(=?pos1 ?pos4)) (not(=?pos1 ?pos3)) (at-adversary ?pos3 ?s1) (next ?pos3 ?s1 ?pos4 ?s2))
        :effect (and(at-robot ?pos1) (not(at-robot ?pos0)) (at-adversary ?pos4 ?s2) (not(at-adversary ?pos3 ?s1)) (increase(total-cost)1.5))
        )

(:action move-is-not-possible-adversary
        :parameters (?pos0 ?pos1 ?pos3 ?pos4 ?s1 ?s2)
        :precondition (and(at-robot ?pos0) (connect ?pos0 ?pos1) (=?pos1 ?pos4) (at-adversary ?pos3 ?s1) (next ?pos3 ?s1 ?pos4 ?s2))
        :effect (and(at-robot ?pos1) (not(at-robot ?pos0)) (at-adversary ?pos3 ?s2) (not(at-adversary ?pos3 ?s1)) (increase(total-cost)1.5))
        )

(:action wait-and--adversary-moves
        :parameters (?pos0   ?pos3 ?pos4 ?s1 ?s2)
        :precondition (and(at-robot ?pos0)  (not (=?pos0 ?pos4)) (at-adversary ?pos3 ?s1) (next ?pos3 ?s1 ?pos4 ?s2))
        :effect (and(at-robot ?pos0)  (at-adversary ?pos4 ?s2) (not(at-adversary ?pos3 ?s1)) (increase(total-cost)1))
        )

(:action wait-and--adversary-cannot-move
        :parameters (?pos0   ?pos3 ?pos4 ?s1 ?s2)
        :precondition (and(at-robot ?pos0)  (=?pos0 ?pos4) (at-adversary ?pos3 ?s1) (next ?pos3 ?s1 ?pos4 ?s2))
        :effect (and(at-robot ?pos0)  (at-adversary ?pos3 ?s2) (not(at-adversary ?pos3 ?s1))(increase(total-cost)1))
        )
)
```

Figure 2: File that provides the PDDL domain description for the adversary of Fig. 1.

Transitions are labeled by logic expressions. This LLFSM illustrates the process of describing the dynamic environment entirely in PDDL, in order to apply deterministic planning software packages. For this example, we let the granularity of the adversary's time-step to be the same as for the mobile robot under the controller. However, if the time steps of the adversaries and robot are different, we just choose a common divisor of the time steps for the PDDL description to reason about the states of the world.

For each motion by an adversary, we re-classify the actions that describe the motion of the robot into 4 new action descriptions.

1) `move-is-possible`**.** This is the first case, when the robot and the adversary do not collide. This collision is not existent because the context, verified in the preconditions, is as follows.
   a) The robot does not perform a move to a grid-cell currently occupied by the adversary.
   b) The adversary does not perform a move to a grid-cell currently occupied by the robot.
   c) Both, robot and adversary, do not perform a move to the grid-cell occupied by the other.

   The PDDL code for this is illustrated in Fig. 2 under the first action labeled `move-is-possible`. In general, we use a predicate `connect`(p_0, p_1) between two positions to indicate that the robot (or the adversaries) can move from position p_0 to position p_1. The collection of connected positions is the static description of the environment, in other words, the map of the world describing open space and fixed obstacles. The predicate `next`(p_0,s_1,p_1,s_2) encodes all the position and state changes of the LLFSM by the adversary. For example that the adversary changes state as it moves East is expressed by declaration of the PDDL problem of the following form.

   ```
   (next _pos_0_3_ c0 _pos_1_3_ c1)
   ```

   The current position of the robot is the position p_0 for which the predicate `at-robot`(p_0) is true. The current state s and position p_0 of the adversary is the pair (p_0, s) for which `at-adversary`(p_0,s) is true. Then, the parameters of the action in the case `move-is-possible` are the current position and the next position of a robot's move while for the adversary are the current (position,state)-pair and the next (position,state)-pair. As a precondition for the action we do need to check these positions are current and that the map allows to move from the current to the next position for both the robot and the adversary (the predicate `connect` does this for the robot while `next`(p_0,s_1,p_1,s_2) implies `connect`(p_0,p_1), and handles the free space of the robot). We also need as part of the precondition the 3 cases of this situation. For example, the following test in the precondition

   ```
   {(not(=?pos1 ?pos4))}
   ```

   checks that the robot and the adversary do not share the same grid-cell as the target of their respective moves. If the precondition holds, the effects are that the robot is in its new position and the adversary is as well in its new position and state (and they are not in their earlier positions). While there is an effect on the cost, we delay the justification until we complete the 4 cases.
2) `move-is-not-possible-adversary`**.** This is

the same action for our robot, but in this case the adversary's next position in the grid-cell map is blocked by the robot. Thus, the adversary must wait in its current position. That the robot blocks the adversary move is now the predicate `(=?pos1 ?pos4)` of the action's precondition in the PDDL of Fig. 2.

3) `wait-and-adversary-moves`**.** The same action of the robot is qualified by the condition that the robot cannot move to a cell occupied currently by the adversary. But, for planning, we do have the effect that the adversary will carry on its behavior as per its LLFSM, and therefore, it may vacate the area where the robot desires to move. Nevertheless, the robot would have to carry a wait action, at least in this time step.
4) `wait-and--adversary-cannot-move`**.** In this case, not only the action of the robot is blocked by the adversary, but the adversary's behavior is also blocked by the robot. In this case, the effect is that both remain in the same position.

We hope the reader appreciates the generality of the derivation of the PDDL domain description from the behavior of one or more adversaries as long as they are specified by LLFSMs. Also, the PDDL is generic in the structure of the map by the `connect` predicate. If moves/actions of the robot are more complex on what, when and where is an action applicable, the corresponding predicates for this specification would be replacing (expanding) this predicate.

The last point to discuss is the fact that we have associated weights with the 4 qualifications placed at each action of the robot. The actions when the robot actually moves are assigned a slightly larger cost than those where the robot waits. We anticipate that normally not performing an action and remaining idle is less costly (energy consumption, for example), than actually performing an action (a move that may require energy for the motors). However, this is also a condition to assist the solver in focusing on the types of plans we would expect to be realistic. If no cost is provided, PDDL solvers produce plans that may be regarded as anomalous, because they may contain spurious sequences of actions; for example, they move out and in from one position rather than wait for two time steps in the position.

A. *Illustration*

We clarify the earlier presentation with the running example of Fig. 1 and its PDDL (see Fig. 2) with two scenarios where the initial position of one patrolling adversary is known. We thus suggest an environment as per Figure 3[2]. This map has a fixed obstacle on (0,2) and the goal is always at (0,4). In the first setting, the initial position of the automata (the adversary) is (0,3) and its patrolling behavior is as per Fig. 1. We place our robot in (0,0), and configure

[2]The original image of the e-Puck was released by its author Stéphane Magnenat on http://commons.wikimedia.org.

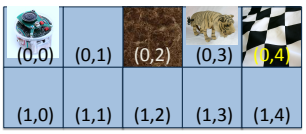

Figure 3: Instance of a dynamic environment.

different problem instances by selecting different cells as the initial position of the adversary. For illustration, we also specify in the domain instance that the only actions for the robot are rectilinear motions of one grid cell in the map.

Now, using our coded domain of Fig. 2, and our planning API for a robot, then solvers like `regression` or `lama` [30] find a plan that goes along the following sequence of actions while the automata carries out the corresponding moves.

robot		adversary	
(0,0)	(0,1)	(0,3)	(1,3)
(0,1)	(1,1)	(1,3)	(1,4)
(1,1)	(1,2)	(1,4)	(0,4)
(1,2)	(1,3)	(0,4)	(0,3)
(1,3)	(1,4)	(0,3)	(1,3)
(1,4)	(0,4)	(1,3)	(1,4)

However, if the adversary's initial position is (1,3), then the plan found by the solvers `regression` and `lama` has the following form (where the adversary is forced to wait).

robot		adversary	
(0,0)	(0,1)	(1,3)	(1,4)
(0,1)	(1,1)	(1,4)	(0,4)
(1,1)	(1,2)	(0,4)	(0,3)
(1,2)	(1,3)	(0,3)	(0,3) (wait)
(1,3)	(1,4)	(0,3)	(1,3)
(1,4)	(0,4)	(1,3)	(1,4)

But if the adversary's initial position is (1,4), then our solvers' plan must include an action to wait.

robot		adversary	
(0,0)	(0,0) (wait)	(1,4)	(0,4)
(0,0)	(0,1)	(0,4)	(0,3)
(0,1)	(1,1)	(0,3)	(1,3)
(1,1)	(1,2)	(1,3)	(1,4)
(1,2)	(1,3)	(1,4)	(0,4)
(1,3)	(1,4)	(0,4)	(0,3)
(1,4)	(0,4)	(0,3)	(1,3)

III. PLANNING API

The use of a planner in a robot can be organized in a cycle where a robot control provides a planner with a planning request (usually consisting of the robot's current state and a goal state). Our architecture here takes advantage of the fact that inputs to a domain-independent planner can be provided in PDDL, and therefore we integrate several planners with the only requirement that they conform to such standardized planning language. We also integrate the standardized output so that the resulting fully-ordered or partially-ordered sequence of actions is transmitted to the other elements of the architecture that execute them.

Our first step is to emulate the behavior of the adversaries (represented as a LLFSM) into the spatial map abstraction of the world. The constructed sequence of maps contains the elements and distribution on the environment and this also constitutes an abstract representation of the behavior of

the adversaries. In a sense we get each spatial map also for each time-step in a temporal dimension. Once we obtain the map abstraction and the adversarial behavior, the PDDL is automatically constructed and we can invoke the solvers to generate plans for the environment. However, the plan may be updated as a result of sensory information that discovers obstacles or adversaries.

We provide a module that enables a planner (or several planners) to be integrated with a whiteboard architecture through the following API.

1) LOAD_ENVIRONMENT(*a map abstraction*) : A map abstraction is provided to the robot. It contains a representation of a real map and the behavior of the dynamic adversaries. We can have as many adversaries as we want, and the behaviors of each one can be completely different.
2) CONSTRUCT_PDDL(*environment*): Once the distribution of the map has been loaded and the environment is constructed taking into account all entities, our program constructs a PDDL that corresponds to that environment and that can be used by the robot.
3) LOAD_PDDL(*name of problem description*) : This enables the planner to retrieve and load a planning problem in PDDL. If there is no file with that name or it does not conform to PDDL, an error is posted to the whiteboard; otherwise, success is reported.
4) LOAD_PLANNER(*planner*) : This enables to select a planner. In our prototype implementation we can choose between several planners that can work with our PDDL domain. The planning module posts to the whiteboard either success (a known planner was provided) or failure. These planners can work and interact with our architecture.
5) START_PLANNER(*depth*) : This starts the planner with a certain maximum depth of actions (a horizon). The depth parameter is optional, and if not supplied, plans of any finite depth are sought. The planner can construct one or several plans for a problem, depending on the search process and provided heuristics and weights. This allows creating a library of plans that can be used by the robot depending on what plan is considered better. If a plan exists, confirmation is posted to the whiteboard. Failure is reported when there is no sequence of actions from the source to the goal. For example, a high number of adversaries with long and complex patrolling routes can prevent the robot from reaching the goal.
6) NEXT_ACTION(*rank*) : This requests the rank-th action in the current plan. The planner responds also with the action's number (and the action parameters). This provides robustness to lost exchanges when we distribute the whiteboard over a network and use UDP. We point out that 'action' here means the next activity in the plan. In typical robots, such action may consist of doing concurrent tasks (in RoboCup, an action like searching for the ball may consist of moving the head around, switching camera, and even some walking, depending of the level the planning is being performed).
7) IS_OBSTACLE_KNOWN(*position*) : This reports to the planner an obstacle at the supplied position. This may be an obstacle found along the way or an explanation for why the last action failed. The planner module responds whether this obstacle was detailed in the problem description or if it is an obstacle the planner was not aware of. In the later case, the planner updates the problem description to now include the obstacle, but, first, the planner module can check if we have another constructed plan that avoids this obstacle and allows the robot to continue its way without re-planning.
8) REPLAN(*source,depth*) : If there is something in the world that was not represented and that the robot is unaware, like an obstacle, and early planning did not take into account, then the module finds a new plan as with START_PLANNER, but from a new source.

This API allows integration of the planning on board of a robot for environments that can differ from the initial knowledge (the original map or problem description). Unknown fixed obstacles are discovered by the robot and re-planning adapts plans to new situations. Our API provides the tools to perform continuous sense-plan-act cycles [10]–[13] as well as a *plan-while execute* approach [7]. In that sense, we do not depart from the common "free-space assumption" [31] where we can plan until we must revise because we discover an obstacle or an adversary.

We underline here with another example, that even with the suitable sensors model and localization algorithms, a robot that can plan and act reliably those actions, faces challenges with deterministic (re-planning) planners when the environment presents dynamic elements if it does not apply the translation we propose. A dynamic element is one that does not sustain its position all the time. It can move through the environment following a predefined behavior. An example where the plan, execute, re-plan cycle fails is provided in the example problem represented in Fig. 4. Here, the adversary moves from cells (2,2) to (2,1), from (2,1) to (2,2) and so on, one cell at each time. The robot is on (0,1), it does a forward movement for the goal to (1,1), here at (1,1) the sensors detect an obstacle blocking (2,1). A message asking if that obstacle is known is posted into the whiteboard and two things can happen:

1) The planner module has a library of plans where the current robot position corresponds to a position in the path of one of that plans. So now, the planner module gives another possible plan to the robot that can reach the goal following another path.
2) The planner module does not have a library of plans,

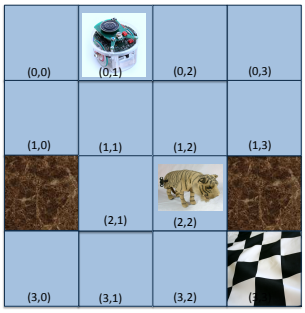

Figure 4: An environment where deterministic planning with obstacle sensing and re-planning fails.

or another available plan does not solve the problem. So the planner module re-plans again in order to find a new path to the goal, taking into account the new discovered obstacle.

In both situations, the robot will move to the next position (1,2) and it will detect the cell (2,2) as blocked, — the same as the other case, but now the adversary has moved to (2,2). The same problem will occur, the planner will not find a plan unless it models the behavior of the moving adversary and performs a wait.

Thus, enabling dynamic elements with deterministic planners requires the construction we have described earlier.

IV. Discussion

This paper explores to what the extent a deterministic planner can be used to guide the motions of a robot in a dynamic environment. We have shown that objects that move in the environment with a deterministic behavior can be integrated into the PDDL descriptions for these types of planners. The general method consists of specifying the states and the transitions of each of such object as connections in the problem specification part of the PDDL. Then, the actions of our robot (as specified in the domain part of the PDDL) are decorated with the preconditions and effects of each of the moving objects.

This general methodology is not without its challenges. However, it already proves to be superior to the approaches that use planner and re-planning by deterministic planners in a dynamic environment. Recall the early discussion regarding the environment of Fig. 4

Our planning API (and its associated architecture) has been successful in the integration of PDDL-solvers for different domains where robots must navigate an environment with known and unknown obstacles. We have also integrated simulators into our architecture for PDDL-planners.

The execution of plans where the `lama` planner is underneath is demonstrated in the videos for the *Webots*[3] simulator. The video is accelerated and a speech module is used to record the actions received for each action request. The length of the side of all squares is equivalent to 40cm. For evidence of the proposal here we actually provide the video http://www.youtube.com/watch?v=6MJIlOZTeVc where we display 3 scenarios. All were actually computed by the `lama` planner as the `regression` planner actually fails to find a plan. This means that the state explosion of the environment here surpasses the heuristic of this solver. This aspect illustrates the relevance of our API enabling the selection of a different PDDL-solver. In the first scenario we keep one adversary fixed, while another does move under a deterministic behavior (specified by a LLFSM). The LLFSM for this adversary is significantly more complicated that the one in Fig. 1 (the adversaries repeat position but moving in different directions). The second part of the video shows the scaling to even more adversaries, as both become moving adversaries, and we have chosen one of the `lama` plans that illustrate the choosing of all actions possible by the robot. Finally, we show that under static planning conditions, all classical planners would guide the robot to a crash. However, the planning control has been factored out to a common FSM with the option of choosing the `lama` planner if we suspect more alternative paths are required besides just one.

The overall behavior is encapsulated by the deployment of a FSM that invoke the necessary planner. That is, the responses from the planning module are significantly high level. As a result, we can perform planning with the same architecture and the same behavior on different robotic platforms. For illustration, we have videos of an *NXT*[4] robot with tethered control (in the form of a differential robot), a *Nao*[5] humanoid robot and also the e-puck robot (differential robot) inside the Webots simulator. From 1min-30s until 3min, our video used for classification for RoboCup-2013 illustrates the execution of plans with the regression planner on two architectures; namely on the *NXT* and on the *Nao* robot (see http://vimeo.com/mipalgu/qualification2013 or http://youtu.be/cQgCrqRznCo from 1min-30s to 3min). Another video (http://youtu.be/-mvppFPWfMU) shows alternative paths run with `lama`.

Acknowledgment

We thank all members of the MiPal team for their help.

References

[1] S. M. LaValle, *Planning Algorithms*. UK: Cambridge Univ. Press, 2006.

[2] M. Fox and D. Long, "PDDL2.1 : An extension to PDDL for expressing temporal planning domains," *J. of Artificial Intelligence Research*, vol. 20, pp. 61–12, 2003.

[3] *Webots* is a development environment used to model, program and simulate mobile robots distributed by Cyberbotics (http://www.cyberbotics.com).

[4] LEGO MINDSTORMS NXT, is a configurable robotic system distributed by LEGO (http://mindstorms.lego.com).

[5] NAO is a programmable, 58cm tall humanoid robot distributed by Aldebaran Robotics.

[3] ——, "Modelling mixed discrete-continuous domains for planning," *J. of Artificial Intelligence Research*, vol. 27, pp. 235–297, 2006.

[4] A. Junghanns and J. Schaeffer, "Sokoban: Enhancing general single-agent search methods using domain knowledge," *Artificial Intelligence*, vol. 129, no. 1-2, pp. 219–251, 2001.

[5] D. Silver, "Cooperative pathfinding," in *Proc. First Artificial Intelligence and Interactive Digital Entertainment Conf.*, Marina del Rey, CA, USA: AAAI Press, 2005, pp. 117–122.

[6] B. Brenner and B. Nebel, "Continual planning and acting in dynamic multiagent environments," *Autonomous Agents and Multi-Agent Systems*, vol. 19, no. 3, pp. 297–331, 2009.

[7] F. Teichteil-Königsbuch, C. Lesire, and G. Infantes, "A generic framework for anytime execution-driven planning in robotics," in *2011 IEEE Int. Conf. on Robotics and Automation (ICRA)*, Shanghai, China, 2011, pp. 299–304.

[8] A. Carbone, A. Finzi, A. Orlandi, and F. Pirri, "Model-based control architecture for attentive robots in rescue scenarios," *Autonomous Robots*, vol. 24, pp. 87–120, 2008.

[9] M. Beetz and D. V. McDermott, "Improving robot plans during their execution," in *Proc. of artificial intelligence planning systems*. Menlo Park: AAAI Press, 1994, pp. 7–12.

[10] D. J. Musliner, E. H. Durfee, and K. G. Shin, "CIRCA: A cooperative intelligent real time control architecture," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 23, no. 6, pp. 1561–1574, 1993.

[11] B. C. Williams and P. P. Nayak, "A reactive planner for a model-based executive," in *Proc. of the 15th Int. Joint Conf. on Artificial Intelligence (IJCAI-97)*, San Mateo, CA: Morgan Kaufmann Publishers, 1997, pp. 1178–1185.

[12] N. Muscettola, G. A. Dorais, C. Fry, R. Levinson, and C. Plaunt, "IDEA: planning at the core of autonomous reactive agents." in *Proc. of NASA workshop on planning and scheduling for space*, 2002.

[13] A. Finzi, F. Ingrand, and N. Muscettola, "Model-based executive control through reactive planning for autonomous rovers," in *IROS*, 2004, pp. 879–884.

[14] K. Z. Haigh and M. M. Veloso, "Interleaving planning and robot execution for asynchronous user requests," *Autonomous agents*, pp. 79–95, 1998.

[15] A. Finzi and A. Orlandini, "Human-robot interaction through mixed-initiative planning for rescue and search rovers," in *AIIA*, 2005, pp. 483–494.

[16] B. Bonet and H. Geffner, "Planning with incomplete information as heuristic search in belief space," in *Proc. of the Fifth Int. Conf. on Artificial Intelligence Planning Systems*, Breckenridge, CO, USA: AAAI, 2000, pp. 52–61.

[17] J. Hoffmann and R. I. Brafman, "Conformant planning via heuristic forward search: a new approach," *Artif. Intell.*, vol. 170, no. 6, pp. 507–541, 2006.

[18] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artif. Intell.*, vol. 101, no. 1-2, pp. 99–134, 1998.

[19] B. Bonet and H. Geffner, "Planning under partial observability by classical replanning: Theory and experiments," in *Proc. of the 22nd Int. Joint Conf. on Artificial Intelligence*, Barcelona, Spain: IJCAI/AAAI, 2011, pp. 1936–1941.

[20] J. Rumbaugh, M. R. Blaha, W. Lorensen, F. Eddy, and W. Premerlani, *Object-Oriented Modelling and Design*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1991.

[21] S. J. Mellor and M. Balcer, *Executable UML: A foundation for model-driven architecture*. Addison-Wesley, 2002.

[22] D. Harel and A. Naamad, "The STATEMATE semantics of statecharts," *ACM Transactions on Software Engineering Methodology*, vol. 5, no. 4, pp. 293–333, 1996.

[23] M. Lötzsch, J. Bach, H.-D. Burkhard, and M. Jüngel, "Designing agent behavior with the extensible agent behavior specification language XABSL," in *7th Int. Workshop on RoboCup 2003 (Robot World Cup Soccer Games and Conf.)*, ser. LNAI, vol. 3020. Springer, 2004, pp. 114–124.

[24] M. Risler and O. von Stryk, "Formal behavior specification of multi-robot systems using hierarchical state machines in XABSL," in *AAMAS08-Workshop on Formal Models and Methods for Multi-Robot Systems*, Estoril, Portugal, 2008.

[25] D. Billington, V. Estivill-Castro, R. Hexel, and A. Rock, "Chapter 3: Non-monotonic reasoning on board a sony AIBO," in *Robotic Soccer*, Vienna, Austria: I-Tech Education and Publishing, 2007, pp. 45–70.

[26] V. Estivill-Castro and R. Hexel, "Module interactions for model-driven engineering of complex behavior of autonomous robots," in *The Sixth Int. Conf. on Software Engineering Advances. ICSEA 2011*, Barcelona, Spain: IARIA, 2011, pp. 84–91.

[27] V. Estivill-Castro, R. Hexel, and D. A. Rosenblueth, "Efficient modelling of embedded software systems and their formal verification," in *The 19th Asia-Pacific Software Engineering Conf. (APSEC 2012)*, Hong Kong: IEEE Computer Soceity, Conf. Publishing Services, 2012, pp. 428–433.

[28] ——, "Efficient model checkign and FMEA analysis with deterministic scheduling of transition-labeled finite-state machines," in *2012 3rd World Congress on Software Engineering (WCSE 2012)*, Wuhan, China, 2012, pp. 65–72.

[29] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Where's waldo? sensor-based temporal logic motion planning," in *IEEE Int. Conf. on Robotics and Automation, ICRA*, Roma, Italy, 2007, pp. 3116–3121.

[30] S. Richter and M. Westphal, "The LAMA planner: Guiding cost-based anytime planning with landmarks," *J. of Artificial Intelligence Research (JAIR)*, vol. 39, pp. 127–177, 2010.

[31] S. Koenig, C. A. Tovey, and Y. V. Smirnov, "Performance bounds for planning in unknown terrain," *Artif. Intell.*, vol. 147, no. 1-2, pp. 253–279, 2003.