FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
Universitatea POLITEHNICA din București

CELDA 2020

17th International Conference on Cognition and Exploratory Learning in Digital Age, CELDA 2020

AI PLANNING AND REASONING FOR A SOCIAL ASSISTIVE ROBOT

Eng. Bilal HOTEIT, Dr. Ali ABDALLAH, Dr. Ahmad FAOUR,
Dr. Imad Alex AWADA, Dr. Alex SORICI, Prof. Adina Magda FLOREA.

Table of contents

Concept

Artificial Intelligence is a trending in science and engineering:

- AI embodies intelligence in a machine
- Artificial intelligence is divided into several types and branches
- AI often rely an initial data and depends on a set of algorithms
- Robotics and Planning are a main branches of AI

Increase:

- the usability and accessibility
- the autonomy
- the efficiency of the system

Goal

1. Health care

Medical Robot
Health Care

2. Service Robot

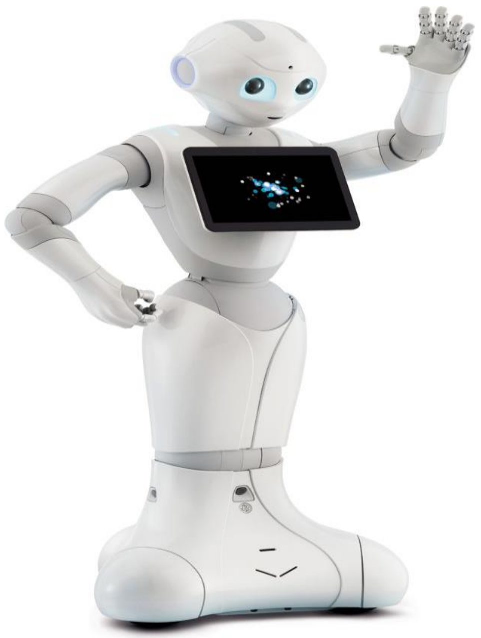
Helps people in home services
Helps in achieving routine tasks

3. Teacher robot

Integrated with our current e-Learning systems.
A teacher Robot.

Current Challenges:

- Robotic Architecture.
- Planning



AI PLANNING AND REASONING FOR A SOCIAL ASSISTIVE ROBOT

Table of contents

Existing Approaches

Planning	Principles
Finite state machine technology	• Defines the required steps to achieve the goal. • Match actions to the situations (the states). Also the goals can been considered. • keep the reactivity fashion of the robot. • prevents the robot to react to novel situations or new goals.
Cognitive Architecture	• Several types of sensory data to produce a range of intelligent behaviors. • A procedural process to produce the behaviour as a production rules. • Decides the required skill that must be executed recursively until the goal is accomplished
AI Symbolic	• Symbolic planners used to generate a plan based on environment and actions' model

Existing Approaches

Planning	Technique	Implemented	References
Finite state machine technology	beforehand Precompiled	Waldo social robot Pepper service robot.	(KC, 2019) (Garvil, 2019)
Cognitive Architecture	Soar BICA	Reem social assistive robot TIAGo robot.	(Martín,2020) (Puigbo,2013)
AI Symbolic	HTN Contingent	PR2 social Nao robot mobile robot	(Lemaignan,2017) (sanelli,2017)

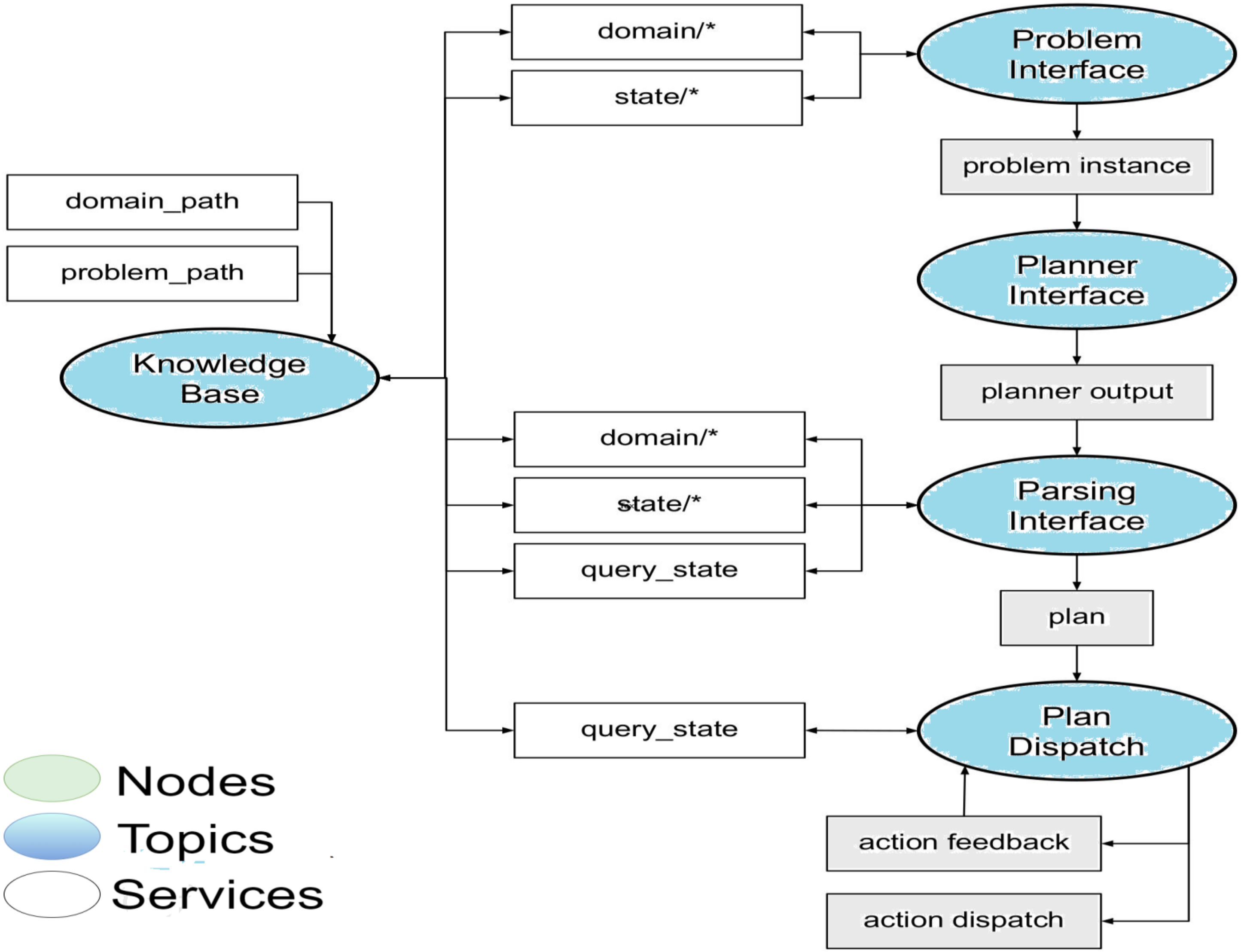
State of the Art

ROSPlan	Robot type	Planner	Language	References
Temporal planning	AUVs AUVs	POPF POPF2	PDDL 2.1 PDDL 2.2	(Cashmore, 2015). (Cashmore, 2016).
Opportunistic planning	AUVs	POPF2	PDDL 2.2	(Cashmore, 2016).
Contingent planning	Mobile Robot	Contingent-FF	PDDL +	(Sanelli, 2017).
Probabilistic planning	Mobile Robot	PROST	RDDL	(Canal, 2019).

AI PLANNING AND REASONING FOR A SOCIAL ASSISTIVE ROBOT

Table of contents

ROSPlan components and communications overview



ROSPlan components and communications overview

Component	Principibles
KB	• Stores symbolic information to represent states, and facts. • KB node launched with domain and problem file. • Several services fetch and update state's variables. • Updated based on the perception to represent states.
Problem Generator	• Produces a problem instance. • Uses a KB node's services to get the states' details. • Publishes instances to a ROS topic, or to a file.
Planner node	• Is a wrapper for AI planner. • Solves problems by Generating plans. • publishes plan to a ROS topic, or to a file.
Parsing node	• Translates the plan into a ROS representation.. • Each abstract action is changed to a ROS action message. • Provides a valid structure specifying the execution details.
Dispatching node	• Executes the parsing plan considering the action times. • Restricts to same representation of the parsing node • Encapsulates an action interface node that links the action and the lower level actuators.

Prerequisites to the execution phase

1. The Domain File

```
(define (domain social-robot)
(:requirements :strips :typing :fluents :disjunctive-preconditions :durative-
actions)
(:types
waypoint
robot              )
 (:predicates
(robot_at ?v - robot ?wp - waypoint)
(connected ?from ?to - waypoint)
(visited ?wp - waypoint)
                  (notified ?wp - waypoint)          )
 (:functions
(distance ?wp1 ?wp2 - waypoint)          )
 (:durative-action goto_waypoint
:parameters (?v - robot ?from ?to - waypoint)
:duration ( = ?duration 10)
:condition (and
                             (at start (robot_at ?v ?from)))
:effect (and
                             (at end (visited ?to))
                             (at start (not (robot_at ?v ?from)))
                             (at end (robot_at ?v ?to)))       )
(:durative-action notify_waypoint
:parameters (?m - robot ?loc - waypoint)
:duration ( = ?duration 20)
:condition (and
                             (at start (visited ?loc))
                (at start (robot_at ?m ?loc)))
:effect (and
        (at end (notified ?loc))
                 (at end (robot_at ?m ?loc)))
) )
```

2. The Problem File

```
(define (problem task)
(:domain social-robot)
(:objects
    wp0 wp1 wp2 wp3 – waypoint
      kenny – robot  )
(:init
    (robot_at kenny wp0)
-------------------------------------     )
      (connected wp0 wp0)
    (connected wp1 wp0)
    (connected wp1 wp2)
--------------------    )
    (= (distance wp0 wp0) 0)
    (= (distance wp1 wp0) 0.5)
    (= (distance wp2 wp1) 0.707107)
    (= (distance wp1 wp2) 0.707107)
    (= (distance wp2 wp2) 0)
-------------------------------------     )
(:goal (and
    (notified wp0)
    (visited wp1)
    (notified wp1)
    (visited wp2)
----------------------------------------    )))
```

3. Knowledge base

KB should be initialised.

The execution phase

1. The Generated Plan

```
Number of literals: 12
Constructing lookup tables: [10%] [20%] [30%] [40%] [50%]
--------------------------
Post filtering unreachable actions:  [10%] [20%] [30%] [40%]
----------------------------
All the ground actions in this problem are compression-safe
Initial heuristic = 8.000
b (7.000 | 10.000)b (6.000 | 30.001)b (5.000 | 40.002)b
 --------------------------
; States evaluated: 15
; Cost: 120.007
-----------------------------------------
0.000: (goto_waypoint kenny wp0 wp0)  [10.000]
10.001: (notify_waypoint kenny wp0)  [20.000]
30.002: (goto_waypoint kenny wp0 wp1)  [10.000]
40.003: (notify_waypoint kenny wp1)  [20.000]
60.004: (goto_waypoint kenny wp1 wp2)  [10.000]
70.005: (notify_waypoint kenny wp2)  [20.000]
90.006: (goto_waypoint kenny wp2 wp3)  [10.000]
100.007: (notify_waypoint kenny wp3)  [20.000]
```

2. The ROS Message


```
node_type: 1
node_id: 14
name: "goto_waypoint_end"
action:
  action_id: 6
  name: "goto_waypoint"
  parameters:
    -
      key: "v"
      value: "kenny"
    -
      key: "from"
      value: "wp2"
    -
      key: "to"
      value: "wp3"
  duration: 10.0
  dispatch_time: 90.0059967041
edges_out: [27]
edges_in: [25, 26]
```


3. Action Controller

- Action interface implements abstract execution method to link ROS action with the lower controller through ROS communication.
- Action interface exchanges information with the dispatching node.

The execution phase

Consequently,The dispatching node executes the ROS action by:

- Invoking the action interface that is related to the abstract action.
- Exchange information with the action interface through *action dispatch* and *action feedback* topic.
- Manage the whole execution and can cancel the execution.
- Can make a decision about the next step of the plan execution, revise or replan when required.

While, Action interface node do the job by:

- Achieve the action by implementing a ROS SimpleActionClient mechanism or a ROS service mechanism.
- Fetches and updates the KB by the real states through a monitoring phase.

With such an implementation, a closed loop composing of planning, observing, monitoring, and executing processes is constructed.

The monitoring phase

Through the executor:

- Check if the plan is still valid.

Through the controller:

- Supported by KB To check the applicability of the action.
- Supported by ROS libraries response to check the execution success.

The Re-Planning phase

1. After a Plan validation process.

```
RUN – Plan (D, KB, P)
Generateproblem (D, P);
GeneratePlan (D, P);
FilterPlan (π, KB);
While Loop (If A exists in π)
If Advice-Plan (D, KB, P, π) =  Failure Then Run – Plan
(D, KB, P);
 While Loop Execute (A)
  If CheckApplicability (A, KB) = Failure Then Run –
  Plan (D, KB, P);
  If Execute (A) = Failure Then Run – Plan (D, KB, P);
  If Monitor (A) = Failure Then Run – Plan (D, KB, P);
  Return;
 End Loop
End If
Return;
End Loop
End
```

2. only when Failure.

```
RUN – Plan (D, KB, P)
Generateproblem (D, P);
GeneratePlan (D, P);
FilterPlan (π, KB);
While Loop (If A exists in π)
 While Loop Execute (A)
   If CheckApplicability (A, KB) = Failure Then Run –
   Plan (D, KB, P);
   If Execute (A) = Failure Then Run – Plan (D, KB,
   P);
   If Monitor (A) = Failure Then Run – Plan (D, KB,
   P);
   Return;
  End Loop
Return;
End Loop
End
```

The dispatching time is over, the action is not applicable, the action duration is exceeded, or even when the action is failed.

.

AI PLANNING AND REASONING FOR A SOCIAL ASSISTIVE ROBOT

Table of contents

Simulation

Gazebo simulation using TurtleBot3 robot.

A case scenario presents a complex mission for a service robot interacting with humans.

The robot navigates to five exam rooms and alert students about the remaining time for the exam.

Two out of box ROS packages, *sound_play* and *baidu_asr* to provide the NLP capabilities.

Two custom action interfaces have been implemented to support the robot to move and say.

The goto_waypoint action controller relies on the move-base node to navigate the points, while the notify_waypoint action controller relies on the sound_play node to play sounds on the sound card.

The controllers check the preconditions of their associated actions such as, for the notify_action, the *visited (wp1)* predicate in the KB should be true.

The *notify_waypoint* controller invokes the *say* service to accomplish the *notify_waypoint* action.

The controller scans the output topic propagating a message from the baidu_asr node.

Simulation

Gazebo simulation using TurtleBot3 robot.

The *goto_waypoint* action controller handles the execution of the *move_base* action service by sending a goal to the *move_base* action server and waiting for the result message.

When the execution is succeeded, the controllers update the knowledge base with the predicates such as, *notify(wp1)* and *visited(wp1)* are updated to true.

Then, the controllers inform the executor that the execution is accomplished. The executor will handle another action.

We assumed a 'most likely' duration for each action based on our scenario, such as 10 seconds for the navigation action and 20 seconds for the notification action.

The same duration is used for the controller. The planner, POPF, is used to generate a temporal plan.

Table of contents

Conclusions

The system:

- Achieve the mission efficiently
- During Failure the system triggers re-planning
- The root behave in a goal directed approach
- Fully responsive

In the future:

- A custom sensing interface will be implemented
- The second approach of the re-planning mechanism

Thank you for your attention!

Questions?