Automated Planning Theory and Practice
Assignment Report

Eugenio Ferrari 239924
eugenio.ferrari-1@studenti.unitn.it

1. Introduction

The goal of the assignment was to formulate planning problems using PDDL and then to execute them across various planner, in my case those provided by `planutils`. The last phase was about the integration of a temporal planning problem into `PlanSys2`.

1.1. Deliverable content

The zip archive contains five folder, each dedicated to one of the problems of the assignment. The folders for problem 1 to 4, contains a domain.pddl file, a problem.pddl file, the outputs generated by the planners used, and a README.md file. The README.md file specifies the contents of the folder, along with details about the planner employed and the respective command and arguments used to invoke the planner.

The final folder, dedicated to problem 5, includes the source code for the `PlanSys2` part of the assignment. The folder structure follows the *plansys2_simple_example* from the official examples [2], as discussed in the lecture.

1.2. Report content

This report is organized as follows:

- Section 2: Provides an overview of the scenario, including my interpretation and the assumptions underlying my solution.
- Sections 3 to 7: Presents the solutions to the individual problems. Each section details the approach taken, design choices (predicates, actions, etc.), additional assumptions made and any modifications from preceding problems.
- Section 8: Concludes the report with an analysis of the results obtained for each problem.

2. Scenario Understanding and Assumptions

The objective of the planning systems is to orchestrate the activities of a set of different robotic agents to deliver boxes containing needed supplies to each workstation. Here I'll describe the general problem, I'll talk about further details on additions such as carriers in the following sections of the report. The scenario involves four main components: the robotic agents, workstations, boxes, and supplies.

Each workstation is situated at a specific location and requires supplies (zero or more) to operate. Therefore, we need to keep track of which supplies each workstation needs and has. Since multiple workstations may exist within the same location, we need to assign a supply to a single workstation and not just leave the supply to the workstation location.

Supplies, stored at specific locations, are transported by the agent (later with the use of a carrier) via boxes also located at those specific locations. From the assignment description, it seems to me that, supplies can be considered as "general resources". For instance, if `bolts` are stocked in `central_warehouse` and an agent packs them into a box for delivery, other bolts still remain in `central_warehouse`, and so the supply locations, such as `central_warehouse` in this example, function as stockpiles for such supplies. Also, I assumed that once supplies are delivered to a workstation, they cannot be reclaimed. This assumption comes from the understanding that once a workstation begins utilizing the supplies, they are considered expended.

Regarding the boxes, I assumed them to be reusable. Therefore, after emptying the contents, the agent can continue to utilize the box for new deliveries.

The robotic agent is capable of executing the following actions (further details regarding preconditions and effects will be provided in next sections):

- Fill a box with a supply.
- Empty a box to a designated workstation, thereby assigning it to that specific workstation and not simply leaving it at its location.
- Load a box.
- Unload a box, this action was not initially outlined in the assignment but I included it to facilitate tasks such as transferring empty boxes between locations and it will be used later when dealing with carriers.
- Move to another location, given that the starting and destination locations are connected.

I assumed that the agent is capable of filling and emptying a box only when it is currently unloaded (when positioned on the ground / not on any robot). Initially, this assumption may seem not usefull, but I think this becomes important in Problem 2, where multiple boxes can be loaded simultaneously

by carriers. This restriction ensures clear operations and prevents potential conflicts or ambiguities in box handling procedures.

The final consideration is that we may have multiple agents in our scenarios, necessitating the modeling of agents as a distinct type, as specified in the assignment.

3. Problem 1 Approach

Types

- `location`: Represents different locations within the environment.
- `locatable`: Parent type for agents, supplies, boxes, and workstations, it to allows me to define just one predicate `located(locatable, location)` and not one for each child type.
- `agent`: Represents the robotic agents.
- `agent_type_1`: Represents a specific type of agent, for example, Problem 2 talks about different kind of agents (example, drones).
- `supply`: Represents the supplies needed by the workstations.
- `box`: Represents boxes containing needed supplies.
- `workstation`: Represents the workstations.

Predicates

- `connected ?l1 ?l2 - location`: Indicates that location ?l1 is connected to location ?l2, representing the connectivity of the environment (the "roadmap").
- `located ?x - locatable ?l - location`: Specifies that ?x is located at location ?l.
- `contains ?b - box ?s - supply`: Indicates that box ?b contains supply ?s.
- `isempty ?b - box`: Specifies that box ?b is empty.
- `loaded ?a - agent ?b - box`: Indicates that box ?b is loaded on agent ?a.
- `free ?a - agent`: Specifies that agent ?a is free, not loaded with any box.
- `unloaded ?b - box`: Specifies that box ?b is unloaded, not on any agent.
- `has ?w - workstation ?s - supply`: Indicates that workstation ?w has supply ?s.
- `needs ?w - workstation ?s - supply`: Indicates that workstation ?w needs supply ?s.

Actions

`fill`

- **Parameters**: (?a - agent ?b - box ?s - supply ?l - location)
- **Preconditions**:
  - Agent ?a, box ?b, and supply ?s are located at the same location ?l.
  - Box ?b is empty and unloaded.
- **Effects**: Box ?b is no longer empty and contains supply ?s.

`empty`

- **Parameters**: (?a - agent ?b - box ?s - supply ?w - workstation ?l - location)
- **Preconditions**:
  - Agent ?a, box ?b, and workstation ?w are located at the same location ?l.
  - Box ?b contains supply ?s needed by workstation ?w and is unloaded.
- **Effects**: Supply ?s is gicen to workstation ?w. Box ?b becomes empty. Workstation ?w does not need ?s anymore and has it.

`move`

- **Parameters**: (?a - agent ?from ?to - location)
- **Preconditions**: Agent ?a is located at location ?from. Locations ?from and ?to are connected.
- **Effects**: Agent ?a moves from location ?from to location ?to.

`load`

- **Parameters**: (?a - agent ?b - box ?l - location)
- **Preconditions**:
  - Agent ?a and box ?b are located at the same location ?l.
  - Agent ?a is free, and box ?b is unloaded.
- **Effects**: Box ?b is loaded onto agent ?a. Agent ?a is no longer free.

`unload`

- **Parameters**: (?a - agent ?b - box ?l - location)
- **Preconditions**: Agent ?a is located at location ?l and is loaded with box ?b.
- **Effects**: Box ?b is unloaded from agent ?a. Agent ?a becomes free again.

4. Problem 2 Approach

In Problem 2, the scenario includes carriers. Each agent has a carrier, and each carrier has varying maximum capacities. Each robotic agent loads and unloads boxes onto its assigned carrier and then moves the carrier to deliver supplies.

A crucial constraint is that agents can move their carrier to workstations for supply delivery but must not return to the `central_warehouse` until all the supplies have been delivered. Additionally, carriers obviously cannot transport more than their maximum capacity.

To facilitate the constraint of agents and carriers not returning to the `central_warehouse` until all deliveries are complete, I introduced two subtypes of `location`: `ws_location` for workstations and `warehouse` for the `central_warehouse`. I made this distinction because in the problem description is specified that no workstations can be positioned in `central_warehouse`.

Given my previous assumption of reusable boxes, carriers can load both filled and empty boxes. Therefore, agents are permitted to return to the central warehouse only if the carrier is not moving filled boxes, indicating that all supplies have been delivered. To achive this, I track each carrier's current capacity, encompassing both filled and empty boxes, as well as the number of filled boxes among the loaded ones.

Additionally, I assume that once an agent and its carrier are in the same location, they move together. I included an action allowing the agent to "search" for the carrier if they are not initially in the same place. This action's precondition requires the agent and the carrier to be in different locations, ensuring the action is not applicable and they always move together once reunited.

To keep track of the current capacity and filled boxes, I didn't use `:numeric-fluents` due to limited support across planners. Instead, I encoded numbers as objects and used predicates for increment and decrement operations. This approach resembles the one demonstrated in the laboratory for the `transport.hddl` problem, and my code is also inspired from the example provided in the book *"An Introduction to the Planning Domain Definition Language"* [1], specifically in Section 2.1.3 titled *"EXAMPLE: LOGISTICS"*.

Types

All the types described for the previous problem, Section 3 remains, these are the ones added for addressing the carrier addition:

- `warehouse`: Represents the warehouse location where workstations are not allowed, child of `location`.
- `ws_location`: Represents locations where workstations are allowed, child of `location`.
- `carrier`: Represents the carrier.
- `quantity`: Represents a quantity type, used for increase/decrease and keeping track of capacity and filled boxes.

Constants

- **q0**: Represents an empty quantity constant.

Predicates

Again, **some of the predicates for the previous problem, Section 3 remains**, these are the new ones and the changes:

- `has_carrier ?a - agent ?c - carrier`: Indicates that carrier ?c is assigned to agent ?a.
- `inc ?q ?q_new - quantity`: Specifies an increment of quantity.
- `dec ?q ?q_new - quantity`: Specifies a decrement of quantity.
- `available_slots ?c - carrier ?q - quantity`: Indicates that carrier ?c has ?q available slots left. If $?q = q0$, it means the carrier is full.
- `current_filled_boxes ?c - carrier ?q - quantity`: Indicates the number of filled boxes in carrier ?c.
- `loaded ?c - carrier ?b - box`: This predicate was present before, but now it has carrier instead of agent, indicating that box ?b is loaded on carrier ?c.
- `free ?a - agent`: This predicate has been removed because now the boxes are loaded on the carrier.

Actions

As evident from the following, certain actions could have been simplified and merged together using `:conditional-effects`. For instance, both the `load` and `unload` actions have versions for both empty and filled boxes. However, to keep greater compatibility with the planners, I opted not to use them.

`fill` and **`empty`** does not change.

`move-to-carrier`

- **Parameters**: (?a - agent ?c - carrier ?from ?to - location)
- **Preconditions**:
  - Agent ?a is located at location ?from.
  - Carrier ?c is not located at location ?from.
  - Agent ?a has carrier ?c.
  - Locations ?from and ?to are connected.
- **Effects**: Agent ?a moves from location ?from to location ?to.
- This is the action that allows the agent to "find" his carrier if initally not together.

`move-carrier-to-workstations`

- **Parameters**: (?a - agent ?c - carrier ?from - location ?to - ws_location)
- **Preconditions**:
  - Agent ?a is located at location ?from.
  - Carrier ?c is located at location ?from.
  - Agent ?a has carrier ?c.
  - Locations ?from and ?to are connected.
  - I didn't add constraints for the agent to move to workstations only when it has supplies to deliver. This decision was made

due to the reusability of boxes, which might necessitate the agent to go gather empty boxes and transport them back to the `central_warehouse` to refill.

- **Effects**: Agent ?a and carrier ?c move from location ?from to location ?to.

`move-carrier-to-warehouse`

- **Parameters**: (?a - agent ?c - carrier ?from - location ?to - warehouse)
- **Preconditions**:
  - Agent ?a is located at location ?from.
  - Carrier ?c is located at location ?from.
  - Agent ?a has carrier ?c.
  - Locations ?from and ?to are connected.
  - Carrier ?c has no filled boxes left.
- **Effects**: Agent ?a and carrier ?c move from location ?from to warehouse ?to.

`load-filled`

- **Parameters**: (?a - agent ?c - carrier ?b - box ?l - location ?q ?q_new ?n ?n_new - quantity)
- **Preconditions**:
  - Agent ?a, carrier ?c, and box ?b are located at the same location ?l.
  - Agent ?a has carrier ?c.
  - Box ?b is unloaded and filled.
  - Carrier ?c has available slots.
- **Effects**: Box ?b is loaded onto carrier ?c. Carrier's available slots **and** number filled boxes are updated.

`load-empty`

- **Parameters**: (?a - agent ?c - carrier ?b - box ?l - location ?q ?q_new - quantity)
- **Preconditions**:
  - Agent ?a, carrier ?c, and box ?b are located at the same location ?l.
  - Agent ?a has carrier ?c.
  - Box ?b is unloaded and empty.
  - Carrier ?c has available slots.
- **Effects**: Box ?b is loaded onto carrier ?c. Carrier's available slots are updated.

`unload-filled`

- **Parameters**: (?a - agent ?c - carrier ?b - box ?l - location ?q ?q_new ?n ?n_new - quantity)
- **Preconditions**:
  - Agent ?a and carrier ?c are located at the same location ?l.
  - Agent ?a has carrier ?c.
  - Box ?b is loaded on carrier ?c and is filled.
- **Effects**: Box ?b is unloaded from carrier ?c. Carrier's available slots **and** number filled boxes are updated.

`unload-empty`

- **Parameters**: (?a - agent ?c - carrier ?b - box ?l - location ?q ?q_new - quantity)
- **Preconditions**:
  - Agent ?a and carrier ?c are located at the same location ?l.
  - Agent ?a has carrier ?c.
  - Box ?b is loaded on carrier ?c and is empty.
- **Effects**: Box ?b is unloaded from carrier ?c. Carrier's available slots are updated.

5. Problem 3 Approach

In Problem 3, I solved the problem by using hierarchical task networks. Following the suggestion, I keept the actions outlined in Section 4, with a few additional noop actions, then I defined all the tasks and methods. Also the goal specification here changes, as it is specified in for of a `htn`.

These are the noops actions added:

- `noop_agent_to_carrier`
- `noop_agent_to_deliver`
- `noop_agent_back_warehouse`

These are the tasks:

- `deliver`
- `find_carrier`
- `fill_load_supply`
- `go_to_deliver`
- `unload_empty_supply`
- `back_to_warehouse`

These are the methods:

- `m_deliver_1_supply`
- `m_find_carrier`
- `m_find_carrier_via`
- `m_find_carrier_base`
- `m_fill_load_supply`
- `m_go_to_deliver`
- `m_go_to_deliver_via`
- `m_go_to_deliver_base`
- `m_unload_empty_supply`
- `m_back_to_warehouse`
- `m_back_to_warehouse_via`
- `m_back_to_warehouse_base`

The move of the robot and carrier, tasks `find_carrier`, `go_to_deliver` and `back_to_warehouse`, has been implemented as the `get_to` task in the `Transporter` example seen in the laboratory.

Among all, the most important method is `m_deliver_1_supply`, it handles a series of subtasks to complete the delivery process. The method involves these tasks, finding the carrier, transporting the carrier back to the starting location, filling and loading a box with the required supply, moving to the delivery location, unloading

and emptying the supply to the workstation, and it brings back the empty box so that is available for the next use.

This is a suboptimal solution, as it not consider the carrier's capability to transport multiple boxes up to its maximum capacity. A naive solution to this could be to simply copy this method and just list multiple times the subtasks doing the load/fill and unload/empty actions up to the desired amount. However, this approach lacks scalability, as it leads to redundant code. Therefore, I decided to not add it, as it is not a real solution to the problem and I have not been able to find an "elegant" solution for the the carrier's capacity limitation without introducing redundant code.

6. Problem 4 Approach

In Problem 4, `:durative-actions` has been added. This step involves defining a duration for each action and `at start`, `at end`, and `over all` quantifiers have been introduced in preconditions and effects to delineate the temporal constraints.

These are the action durations I choose:

- fill: 1
- empty: 1
- move-to-carrier: 3
- move-carrier-to-workstations: 5
- move-carrier-to-warehouse: 5
- load-filled: 3
- load-empty: 2
- unload-filled: 3
- unload-empty: 2

Since the temporal planners used are not supporting negative preconditions, I had to remove them by adding these predicates (and change preconditions and effects accordingly):

- `(is_not_zero ?q)`
- `(notlocated ?c - carrier ?l - location)`, this one is defined only for carriers as they were the only one that was using a negative precondition.

I have introduced a new predicate, `(not_busy ?a)`, aimed at ensuring that actions do not run concurrently. This predicate serves to prevent scenarios such as moving and loading a box simultaneously with the same agent.

7. Problem 5 Approach

Problem 5 involved integrating the planning problem from Problem 4 into the `PlanSys2` framework. All relevant source code for this task can be located within the `Problem5` directory of the delivery. The initial setup is derived from the `plansys2_simple_example` tutorial in the official `PlanSys2` documentation [2]. The main components of a `PlanSys2` package include:

- `CMakeLists.txt` to build the package.
- `pddl` folder containing the PDDL domain file.
- `launch` folder containing the launch script responsible for selecting the domain and executing action implementations and a `commands` file containing the commands needed to set the instances, predicates and goals in the `plansys2_terminal`.
- `src` folder containing the implementation of domain actions, for our assignment, fake actions.

In order to adapt the PDDL solution described in Section 6 to be compatible with `PlanSys2`, a couple of adjustments were necessary. First, the `q0` constant had to be removed, as `:constants` are not supported by `POPF` utilized in `PlanSys2`. Instead, the `(is_zero ?q - quantity)` predicate was introduced to serve as a replacement. Additionally, the types declaration such as `agent_type_1 drone - agent` had to be removed, maybe this last problem is related to this issue in the `PlanSys2` Github repository, which can be found at this link.

The implementation of the action in our scenario just provides feedback on its progress. It is a `plansys2::ActionExecutorClient`, which requires as constructor parameters the action's name and the rate at which it invokes the `do_work` method. In our context, this method simply sends the feedback. I choose the rate (and the associated `progress_` increment) to align with the action durations specified earlier in the PDDL outlined in Section 6.

8. Results

The planners utilized with various configurations are documented in the README.md files corresponding to each problem. Additionally, all outputs for each problem and configuration can be located within the zip archive.

8.1. Results Problem 1

I used Fast Downward in planutils, trying two different configurations, the pre-defined `lama-first` and `A* with landmark heuristic`. The results are shown in Table 1.

Configuration	Solution Cost	Planning Time (s)
lama-first pre-defined	35	0.45
A* with landmark heuristic	27	0.57

TABLE 1: Results Problem 1

Has expected, using A* we are able to find the optimal solution, while the lama-first configuration finds a worse solution. It can be seen for example that with the lama-first plan, the agent performs useless action, here are the first lines of the lama-first plan:

```
(load agent1 box1 central_warehouse)
(fill agent1 box2 bolt
central_warehouse)
(move agent1 central_warehouse loc1)
(unload agent1 box1 loc1)
(move agent1 loc1 central_warehouse)
```

```
(load agent1 box2 central_warehouse)
(move agent1 central_warehouse loc2)
...
```

As it can be seen, it loads, moves and unloads an empty box, `box1`, and then goes back, load the filled box `box2` and continue the delivery. This does not happen with A*, as it finds the optimal plan. Here are the first lines of the A* solution:

```
(fill agent1 box1 valve
central_warehouse)
(load agent1 box1 central_warehouse)
(move agent1 central_warehouse loc2)
(unload agent1 box1 loc2)
(empty agent1 box1 valve ws3 loc2)
...
```

The complete plans can be found in the zip archive.

8.2. Results Problem 2

I used again Fast Downward in planutils, with the same two configurations. The results are shown in Table 2. As

Configuration	Solution Cost	Planning Time (s)
lama-first pre-defined	53	0.78
A* with landmark heuristic	30	106.51

TABLE 2: Results Problem 2

the difficuty of the problem increases with the addition of the carrier, we can see that the difference in cost between the optimal solution and the other start to become more important, as well as the time for computing it.

Once again, the comprehensive plans are available in the zip archive. Similarly to previous problem, upon reviewing the plans, the suboptimal one includes actions that aren't particularly useful towards achieving the goal.

8.3. Results Problem 3

For HDDL, I used the PANDA planner. As I already explained in Section 5, my HTN solution only delivers one supply at the time,The sequence of actions outlined here pertains to the first supply delivery generated by PANDA, the following deliveries in the plan, follows the same sequence of actions, with different parameters depending on which supply and to which workstation we're delivering.

```
0: noop_agent_to_carrier(agent1,carrier1,
    central_warehouse)
1: noop_agent_back_warehouse(agent1,carrier1
    ,central_warehouse)
2: fill(agent1,box3,tool,central_warehouse)
3: load-filled(agent1,carrier1,box3,
    central_warehouse,q2,q1,q0,q1,q0)
4: move-carrier-to-workstations(agent1,
    carrier1,central_warehouse,loc1)
5: unload-filled(agent1,carrier1,box3,loc1,
    q1,q2,q1,q0)
6: empty(agent1,box3,tool,ws1,loc1)
7: load-empty(agent1,carrier1,box3,loc1,q2,
    q1,q0)
8: move-carrier-to-warehouse(agent1,carrier1
    ,loc1,central_warehouse,q0)
9: unload-empty(agent1,carrier1,box3,
    central_warehouse,q1,q2)
... starts a new delivery ...
```

As always, the complete plan is in the zip archive.

8.4. Results Problem 4

For the durative actions part I tried two planners, OPTIC and Temporal Fast Downward, both from *Planutils*. The metric I used in the problem file is `(:metric minimize (total-time))`. The results are shown in Table 3.

Planner	Number of actions	Total Time (s)
TFD	48	144.471
OPTIC	42	120.041

TABLE 3: Results Problem 4

As always, the complete plans for each planner is in the zip archive.

8.5. Results Problem 5

For this last part, all the commands used to build and launch the package are described in the file `Problem5/plansys2_assignment/README.md`, after building the package, we can launch it, and then using the `plansys2_terminal`, we can set the problem instances, predicates and goals, compute a plan using the command `get plan`, and run the plan with the `run` command.

The default planner used by `PlanSys2` is `POPF`. The output plan and the Behavior Tree generated by the `PlanSys2 executor`, which are stores by `PlanSys2` in the `/tmp` directory by default, are available in the `Problem5` folder of the zip archive.

Finally, Figure 1 and 2 demonstrate the execution of the package.

References

[1] Haslum, Patrik and Lipovetzky, Nir and Magazzeni, Daniele. *An Introduction to the Planning Domain Definition Language*. Morgan & Claypool Publishers, 2019.

[2] PlanSys2 example https://github.com/PlanSys2/ros2_planning_system_examples/tree/rolling/plansys2_simple_example

```
> get plan
plan:
0:      (load_empty agent1 carrier1 box2 central_warehouse q4 q3)       [2]
2.001:  (fill agent1 box3 tool central_warehouse)       [1]
3.002:  (load_filled agent1 carrier1 box3 central_warehouse q3 q2 q0 q1 tool)   [3]
6.003:  (load_empty agent1 carrier1 box1 central_warehouse q2 q1)       [2]
8.004:  (move_carrier_to_workstations agent1 carrier1 central_warehouse loc1)   [5]
13.005: (unload_filled agent1 carrier1 box3 loc1 q1 q2 q1 q0 tool)      [3]
16.006: (empty agent1 box3 tool ws1 loc1)       [1]
17.007: (load_empty agent1 carrier1 box3 loc1 q2 q1)    [2]
19.008: (move_carrier_to_warehouse agent1 carrier1 loc1 central_warehouse q0)   [5]
24.009: (unload_empty agent1 carrier1 box2 central_warehouse q1 q2)     [2]
26.01:  (fill agent1 box2 bolt central_warehouse)       [1]
27.011: (load_filled agent1 carrier1 box2 central_warehouse q2 q1 q0 q1 bolt)   [3]
30.012: (move_carrier_to_workstations agent1 carrier1 central_warehouse loc2)   [5]
35.013: (unload_filled agent1 carrier1 box2 loc2 q1 q2 q1 q0 bolt)      [3]
38.014: (empty agent1 box2 bolt ws3 loc2)       [1]
39.015: (load_empty agent1 carrier1 box2 loc2 q2 q1)    [2]
41.016: (move_carrier_to_warehouse agent1 carrier1 loc2 central_warehouse q0)   [5]
46.017: (unload_empty agent1 carrier1 box2 central_warehouse q1 q2)     [2]
48.018: (fill agent1 box2 tool central_warehouse)       [1]
49.019: (load_filled agent1 carrier1 box2 central_warehouse q2 q1 q0 q1 tool)   [3]
52.02:  (move_carrier_to_workstations agent1 carrier1 central_warehouse loc2)   [5]
57.021: (unload_filled agent1 carrier1 box2 loc2 q1 q2 q1 q0 tool)      [3]
60.022: (empty agent1 box2 tool ws3 loc2)       [1]
61.023: (load_empty agent1 carrier1 box2 loc2 q2 q1)    [2]
63.024: (move_carrier_to_warehouse agent1 carrier1 loc2 central_warehouse q0)   [5]
68.025: (unload_empty agent1 carrier1 box2 central_warehouse q1 q2)     [2]
70.026: (fill agent1 box2 valve central_warehouse)      [1]
71.027: (load_filled agent1 carrier1 box2 central_warehouse q2 q1 q0 q1 valve)  [3]
74.028: (move_carrier_to_workstations agent1 carrier1 central_warehouse loc2)   [5]
79.029: (move_carrier_to_workstations agent1 carrier1 loc2 loc3)        [5]
84.03:  (unload_filled agent1 carrier1 box2 loc3 q1 q2 q1 q0 valve)     [3]
87.031: (unload_empty agent1 carrier1 box1 loc3 q2 q3)  [2]
89.032: (move_carrier_to_workstations agent1 carrier1 loc3 loc2)        [5]
94.033: (move_carrier_to_warehouse agent1 carrier1 loc2 central_warehouse q0)   [5]
99.034: (unload_empty agent1 carrier1 box3 central_warehouse q3 q4)     [2]
101.035:        (fill agent1 box3 valve central_warehouse)      [1]
102.036:        (load_filled agent1 carrier1 box3 central_warehouse q4 q3 q0 q1 valve)  [3]
105.037:        (move_carrier_to_workstations agent1 carrier1 central_warehouse loc2)   [5]
110.038:        (unload_filled agent1 carrier1 box3 loc2 q3 q4 q1 q0 valve)     [3]
113.039:        (empty agent1 box3 valve ws3 loc2)      [1]
114.04: (move_carrier_to_workstations agent1 carrier1 loc2 loc3)        [5]
119.041:        (empty agent1 box2 valve ws4 loc3)      [1]
> run
[INFO] [1710175771.813801755] [executor_client]: Plan Succeeded

Successful finished
```

Figure 1: PlanSys2 terminal: optain and execute plan

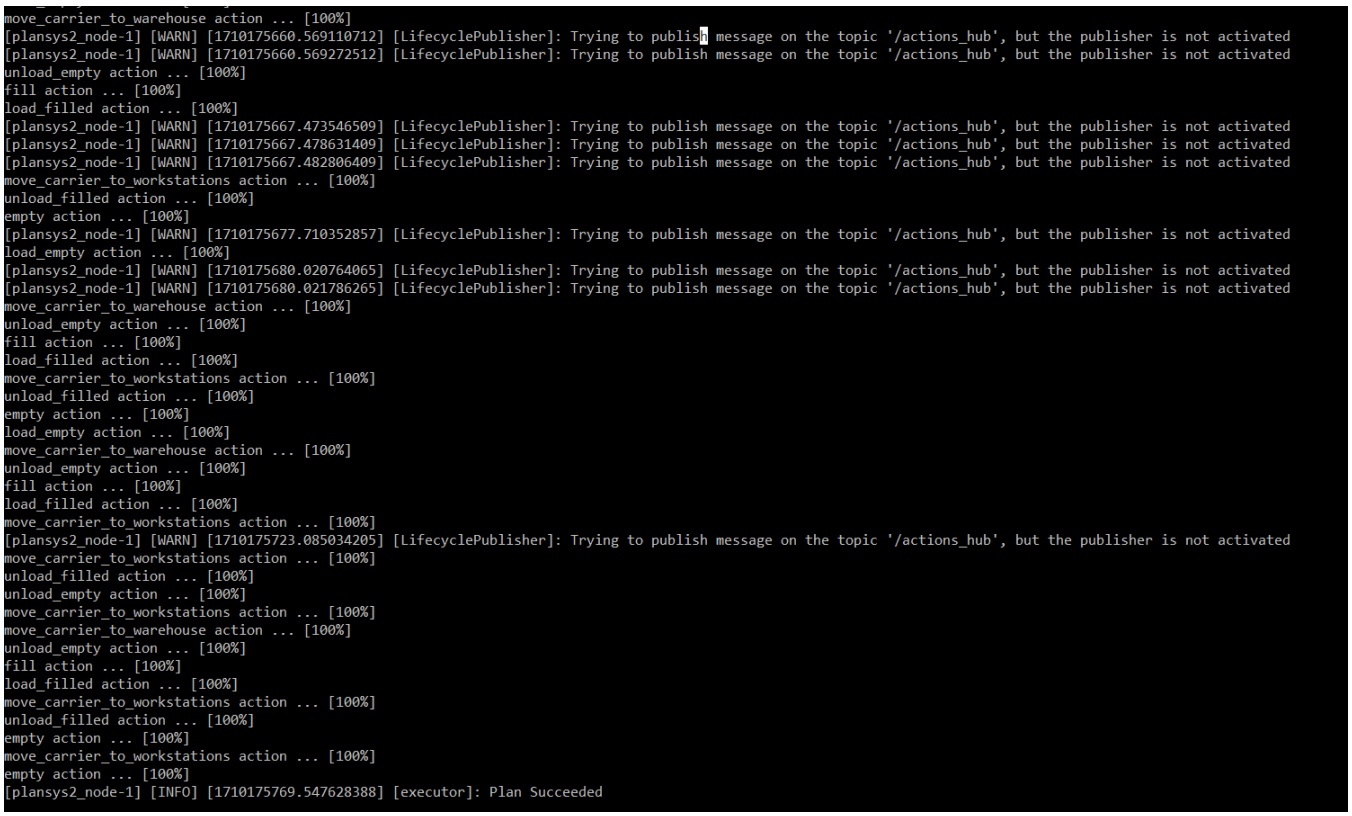

Figure 2: PlanSys2 package node showing the execution of the actions