



UNIVERSITÀ DELLA CALABRIA

DIPARTIMENTO DI
INGEGNERIA INFORMATICA,
MODELLISTICA, ELETTRONICA
E SISTEMISTICA

DIMES

Corso di Laurea Magistrale in Ingegneria Informatica

Progetto di Intelligenza Artificiale

Docenti:

Prof. Francesco Scarcello

Ing. Antonio Bono

Studenti:

Fabio Fusto

Simone Caridà

Francesco Bongiovanni

Indice

Indice	1
1 Introduzione	3
2 Modeling	5
2.1 Dominio applicativo	5
2.1.1 Definizione degli oggetti	5
2.1.2 Definizione dei predicati	7
2.1.3 Definizione delle azioni	9
3 Classical Planning	17
3.1 Istanza 1	17
3.2 Istanza 2	19
3.3 Planner personalizzato	21
3.3.1 Algoritmo di ricerca	22
3.3.2 Euristiche	23
3.4 Risultati ottenuti	25
3.4.1 Istanza 1: A* e Fast-Forward peso 3	25
3.4.2 Istanza 1: A* e AdjustedSum peso 7	26
3.4.3 Istanza 2: A* e Fast-Forward peso 7	27
3.4.4 Istanza 2: A* e AdjustedSum peso 7	28
3.4.5 Confronto tra i risultati ottenuti	29
4 Temporal Planning	30
5 Robotics Planning	35

<i>INDICE</i>	2
6 Organizzazione archivio	40

Capitolo 1

Introduzione

PDDL, acronimo di "Planning Domain Definition Language" (linguaggio di definizione del dominio di pianificazione), è un linguaggio formale utilizzato nell'ambito dell'intelligenza artificiale e della pianificazione automatizzata. È stato creato per descrivere problemi di pianificazione e specificare domini in cui agenti o sistemi intelligenti possono operare. La pianificazione è una branca dell'intelligenza artificiale che si occupa di sviluppare algoritmi e metodi per decidere quali azioni eseguire in sequenza per raggiungere determinati obiettivi in un ambiente dinamico e incerto. Può essere applicata in vari contesti: come robotica, sistemi di controllo automatizzati, giochi, logistica e altro.

Con PDDL, è possibile descrivere il mondo in cui agisce l'agente, specificare le azioni disponibili e definire gli obiettivi che l'agente deve raggiungere. Questo linguaggio offre una rappresentazione standardizzata e precisa dei problemi di pianificazione, facilitando lo sviluppo di algoritmi di pianificazione automatizzata che possono trovare soluzioni efficienti a questi problemi.

Il linguaggio nasce nel '98 in concomitanza con la prima competizione internazionale per il planning. Dei linguaggi antecedenti sono Sprint e ADL, mentre PDDL si sviluppa a fianco di questa competizione.

Le specifiche PDDL sono composte da definizioni di domini (che elencano gli operatori e i predicati disponibili) e problemi (che rappresentano una situazione specifica all'interno di quel dominio). I solver di pianificazione, software che utilizzano PDDL, prendono queste specifiche come input e restituiscono

una sequenza di azioni (piano) che rappresenta una soluzione al problema specificato.

L'obiettivo principale di questo elaborato è quello di illustrare e giustificare le scelte implementative fatte per risolvere i problemi di pianificazione usando il linguaggio PDDL e algoritmi personalizzati.

Capitolo 2

Modeling

Nel contesto di PDDL, il modeling consiste nel definire formalmente il dominio di pianificazione e i problemi associati, specificando gli oggetti, le azioni possibili, le condizioni iniziali e gli obiettivi da raggiungere.

Il dominio descrive gli elementi statici, come le tipologie di oggetti e le azioni che possono essere eseguite, con le loro precondizioni ed effetti. Esso permette quindi di creare una rappresentazione astratta del sistema che un pianificatore automatico può utilizzare per generare un piano ottimale o valido per raggiungere gli obiettivi specificati.

2.1 Dominio applicativo

Di seguito viene descritto tutto il processo di modellazione del dominio in PDDL. In particolare, sono riportati i dettagli riguardo gli oggetti, i predicati e le azioni implementati.

2.1.1 Definizione degli oggetti

Tra gli oggetti definiti troviamo:

- **Location:** rappresenta una posizione o un luogo specifico all'interno dell'ambiente di produzione industriale. Le varie location possono includere il magazzino centrale, le varie workstations e altre aree rilevanti per il

movimento e la consegna di materiali. Tale oggetto viene utilizzato per specificare la posizione di un agente, delle scatole, delle workstation e dei materiali;

- **Locatable**: rappresenta oggetti che possono essere situati in una posizione. In altre parole, sono oggetti che possono occupare una location. Ad esempio, *box* è un oggetto di tipo locatable, poiché può essere situato in una specifica posizione nell'ambiente;
- **Workstation**: rappresenta una stazione di lavoro, che necessita di specifici materiali. Le workstation si trovano all'interno delle varie location, e nulla vieta che una location possa ospitare più di una workstation. Una workstation può richiedere più di un materiale e due workstation diverse possono richiedere due istanze diverse dello stesso materiale.
- **Agent**: rappresenta un agente, ovvero un'entità capace di eseguire azioni nell'ambiente. Un agente è un robot capace di trasportare una scatola per volta o più di una attraverso un trasportatore attaccato ad esso, e si occupa di consegnare le scatole piene di materiali alle workstation che ne fanno richiesta. È in grado di caricare e scaricare il trasportatore stesso, nonché riempire e svuotare le scatole;
- **Carrier**: rappresenta un trasportatore. Esso appartiene ad un singolo agente e può essere caricato di scatole, in modo da trasportarne più di una per volta. La sua capacità è definita attraverso gli slot;
- **Slot**: rappresenta una posizione, ovvero uno spazio all'interno del trasportatore, che può essere occupato da una scatola. Si è deciso di utilizzare un tipo del genere per far fronte all'impossibilità di specificare oggetti di tipo number e alla necessità di definire una specifica capienza per il trasportatore. Ogni slot è associato ad un singolo trasportatore;
- **Box**: rappresenta una scatola che può essere riempita di materiali e svuotata dagli agenti. Essa può essere trasportata da un agente o dal suo trasportatore e viene consegnata alle workstation;

→ **Supply**: rappresenta un materiale specifico, destinato ad una o più workstation. Il loro scopo è quello di essere consegnati, attraverso le scatole trasportate dagli agenti, alle workstation che ne fanno richiesta.

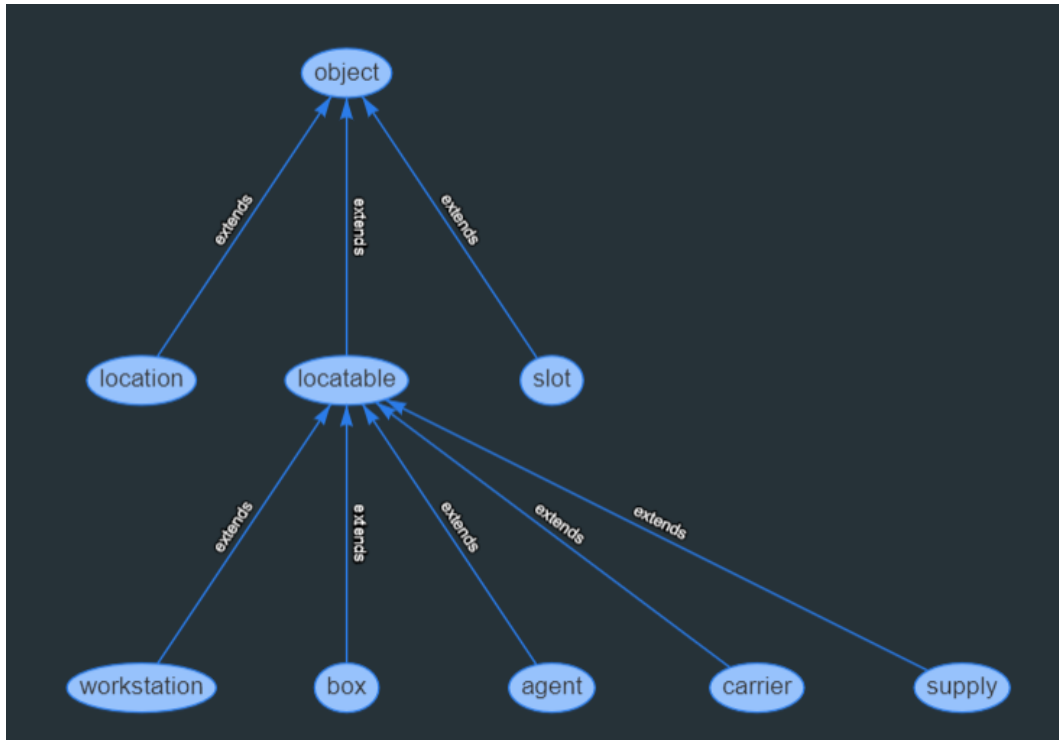


Figura 2.1: Gerarchia dei tipi

Nella figura 2.1 è possibile visualizzare uno schema della gerarchia dei tipi definiti nel file di dominio.

2.1.2 Definizione dei predicati

I predicati descrivono le proprietà e le relazioni tra gli oggetti appartenenti al dominio. Questi predicati sono essenziali per modellare correttamente lo stato del sistema e definire successivamente le azioni.

Predicati comuni

→ **(at_location ?ltb - locatable ?l - location)**: indica che un oggetto specifico di tipo *locatable* ?ltb si trova in una specifica location ?l;

- (**content_in_box ?b - box ?c - supply**): indica che il materiale ?c è contenuto all'interno della scatola ?b;
- (**empty_box ?b - box**): indica che la scatola ?b è vuota, cioè non contiene nessun materiale;
- (**workstation_has_content ?w - workstation ?c - supply**): indica che la workstation ?w ha ottenuto il materiale ?c, quindi ha ricevuto il materiale di cui aveva bisogno;
- (**connected ?loc1 ?loc2 - location**): indica che la location ?loc1 è collegata alla ?loc2. Questo predicato serve a definire le strade che un agente può percorrere.

Predicati specifici della prima istanza

- (**free ?r - agent**): indica che l'agente ?r è libero, ovvero non sta trasportando nessuna scatola;
- (**is_carrying ?r - agent ?b - box**): indica che l'agente ?r sta trasportando la scatola ?b.

Predicati specifici della seconda istanza

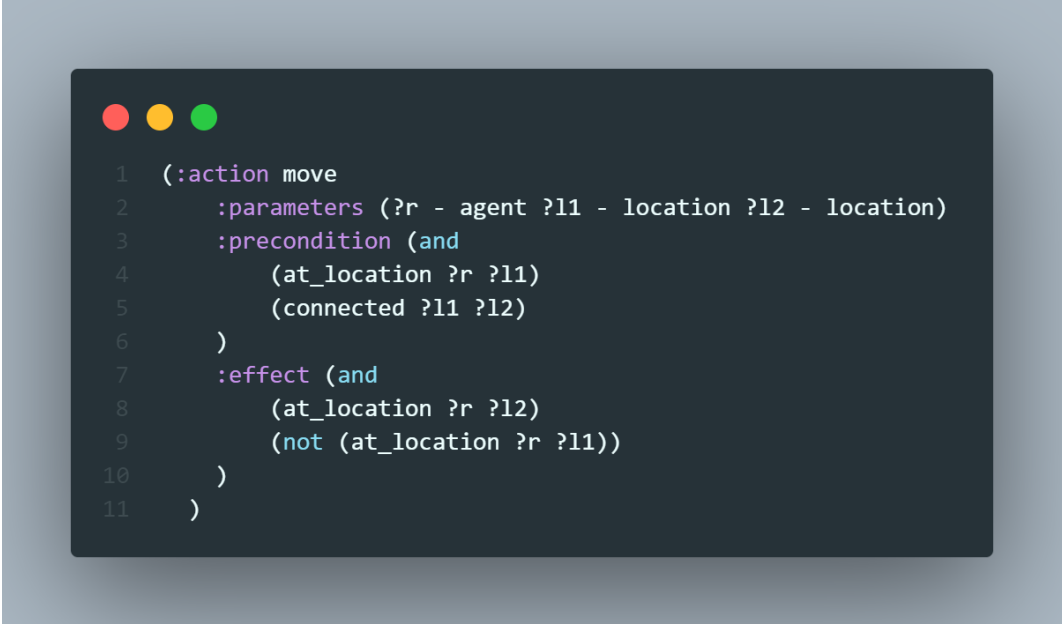
- (**carrier_of_robot ?c - carrier ?r - agent**): indica che il trasportatore ?c appartiene all'agente ?r. Questo perché ogni agente ha un trasportatore personale attaccato a sé;
- (**slot_of_carrier_free ?s - slot ?c - carrier**): indica che lo slot ?s del trasportatore ?c è libero, quindi pronto ad accogliere nuove scatole;
- (**box_on_carrier ?b - box ?c - carrier**): indica che la scatola ?b è stata caricata e si trova momentaneamente sul trasportatore ?c.

2.1.3 Definizione delle azioni

Le azioni consentono agli agenti robotici di interagire con l'ambiente e di effettuare gli step necessari per giungere al soddisfacimento degli obiettivi. Un'azione è costituita dai parametri (oggetti necessari per lo svolgimento dell'azione), dalle precondizioni (condizioni da verificare prima dello svolgimento dell'azione) e dagli effetti (ciò che accade in seguito allo svolgimento dell'azione).

Azioni specifiche della prima istanza

→ **move:**

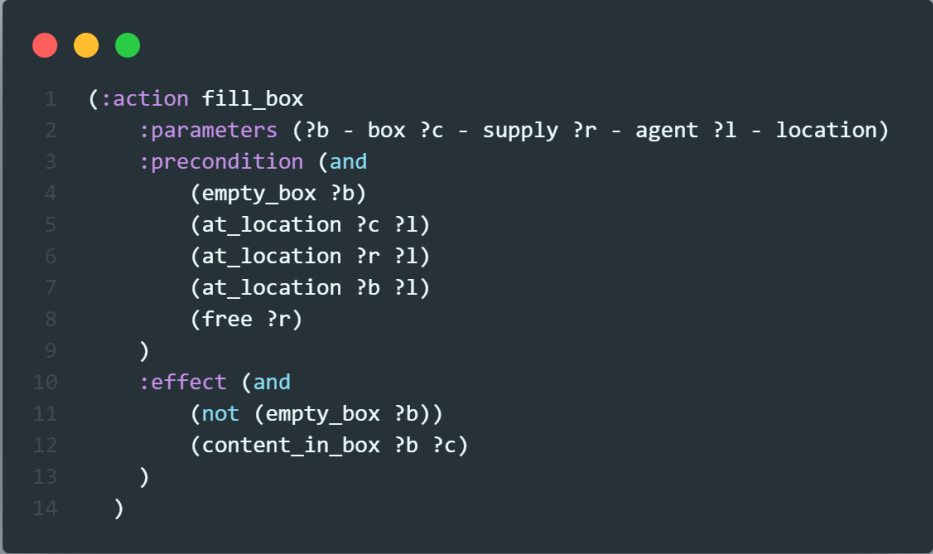


```
1 (:action move
2   :parameters (?r - agent ?l1 - location ?l2 - location)
3   :precondition (and
4     (at_location ?r ?l1)
5     (connected ?l1 ?l2)
6   )
7   :effect (and
8     (at_location ?r ?l2)
9     (not (at_location ?r ?l1))
10  )
11 )
```

Figura 2.2: Azione *move*

Questa azione consente all'agente ?r di spostarsi dalla location ?l1 alla location ?l2. È necessario che l'agente si trovi nella location ?l1 e che le due location siano connesse. Dopo aver eseguito l'azione, l'agente si troverà nella location ?l2 e non più nella location ?l1;

→ `fill_box`:

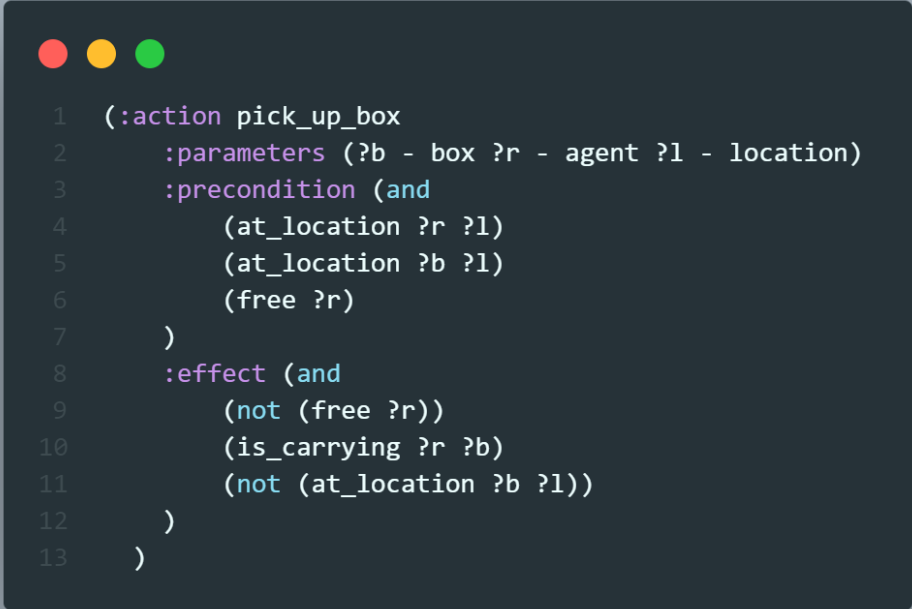


```
1  (:action fill_box
2    :parameters (?b - box ?c - supply ?r - agent ?l - location)
3    :precondition (and
4      (empty_box ?b)
5      (at_location ?c ?l)
6      (at_location ?r ?l)
7      (at_location ?b ?l)
8      (free ?r)
9    )
10   :effect (and
11     (not (empty_box ?b))
12     (content_in_box ?b ?c)
13   )
14 )
```

Figura 2.3: Azione *fill_box*

Questa azione consente all'agente ?r di riempire la scatola ?b con il materiale ?c. È necessario che agente, scatola e materiale si trovino tutti nella stessa location ?l, che la scatola sia vuota e che l'agente non stia trasportando nulla. Dopo aver eseguito l'azione la scatola non sarà più vuota, ma riempita del materiale ?c;

→ `pick_up_box`:




```
1  (:action pick_up_box
2    :parameters (?b - box ?r - agent ?l - location)
3    :precondition (and
4      (at_location ?r ?l)
5      (at_location ?b ?l)
6      (free ?r)
7    )
8    :effect (and
9      (not (free ?r))
10     (is_carrying ?r ?b)
11     (not (at_location ?b ?l))
12   )
13 )
```

Figura 2.4: Azione *pick_up_box*

Questa azione consente all'agente ?r di prendere la scatola ?b per trasportarla. È necessario che agente e scatola si trovino nella stessa location ?l e che l'agente non stia trasportando nulla. Dopo aver eseguito l'azione, l'agente non sarà più libero perché starà trasportando la scatola ?b, e la scatola non si troverà più nella location ?l visto che è stata riempita per essere trasportata;

→ `empty_box`:



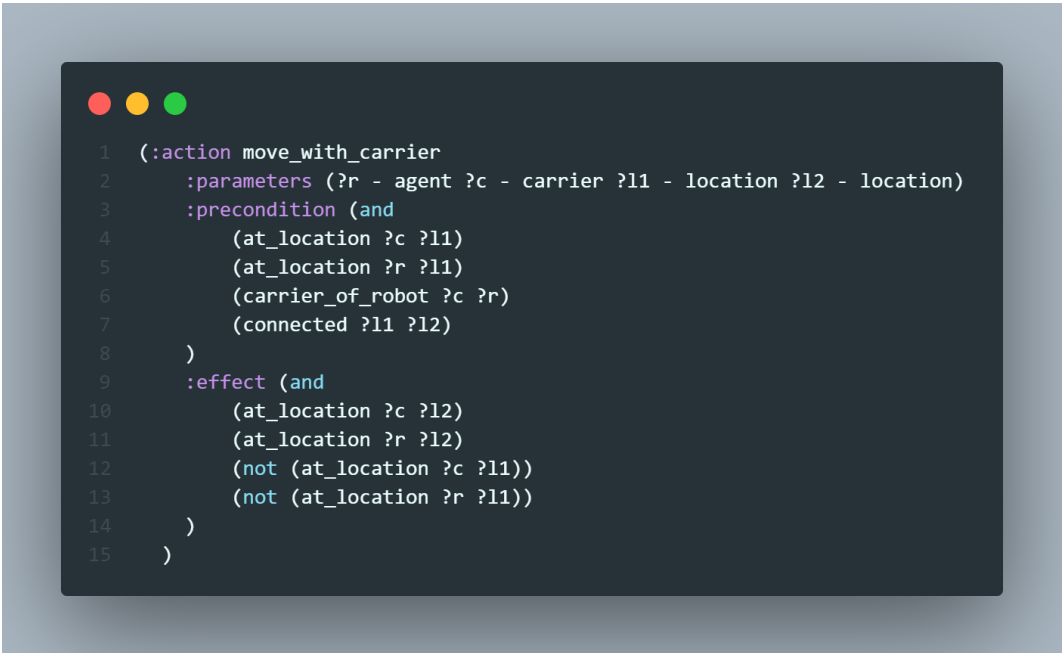
```
1 (:action empty_box
2   :parameters (?b - box ?c - supply ?r - agent ?l - location ?w - workstation)
3   :precondition (and
4     (content_in_box ?b ?c)
5     (at_location ?w ?l)
6     (at_location ?r ?l)
7     (is_carrying ?r ?b)
8     (not (free ?r))
9     (not (empty_box ?b))
10  )
11  :effect (and
12    (workstation_has_content ?w ?c)
13    (not (content_in_box ?b ?c))
14    (empty_box ?b)
15    (at_location ?b ?l)
16    (not (is_carrying ?r ?b))
17    (free ?r)
18  )
19 )
```

Figura 2.5: Azione *empty_box*

Questa azione consente all'agente ?r di svuotare la scatola ?b che contiene il materiale ?c per servire la workstation ?w. È necessario che l'agente e la workstation si trovino nella stessa location ?l, che il materiale ?c sia nella scatola ?b e che quindi la scatola non sia vuota, che la scatola ?b sia trasportata dall'agente ?r e che quindi l'agente non sia libero. Dopo aver eseguito l'azione, la workstation ?w avrà ricevuto il materiale ?c, la scatola ?b sarà vuota e si troverà nella location ?l e l'agente ?r sarà libero.

Azioni specifiche della seconda istanza

→ `move_with_carrier`:

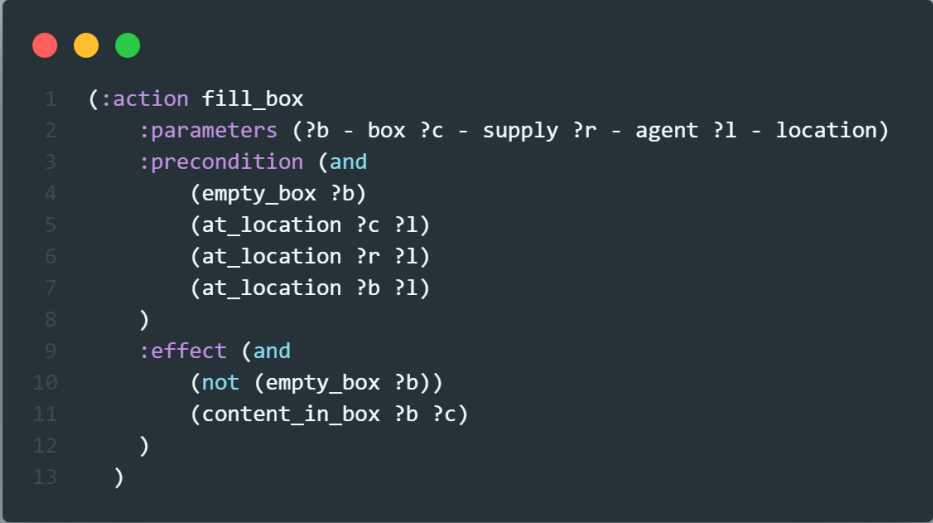


```
1  (:action move_with_carrier
2    :parameters (?r - agent ?c - carrier ?l1 - location ?l2 - location)
3    :precondition (and
4      (at_location ?c ?l1)
5      (at_location ?r ?l1)
6      (carrier_of_robot ?c ?r)
7      (connected ?l1 ?l2)
8    )
9    :effect (and
10     (at_location ?c ?l2)
11     (at_location ?r ?l2)
12     (not (at_location ?c ?l1))
13     (not (at_location ?r ?l1))
14   )
15 )
```

Figura 2.6: Azione *move_with_carrier*

Questa azione consente all'agente ?r e al suo trasportatore ?c di spostarsi dalla location ?l1 alla location ?l2. È necessario che l'agente e il suo trasportatore si trovino nella location ?l1 e che le due location siano connesse. Dopo aver eseguito l'azione, l'agente e il suo trasportatore si troveranno nella location ?l2 e non più nella location ?l1;

→ `fill_box`:

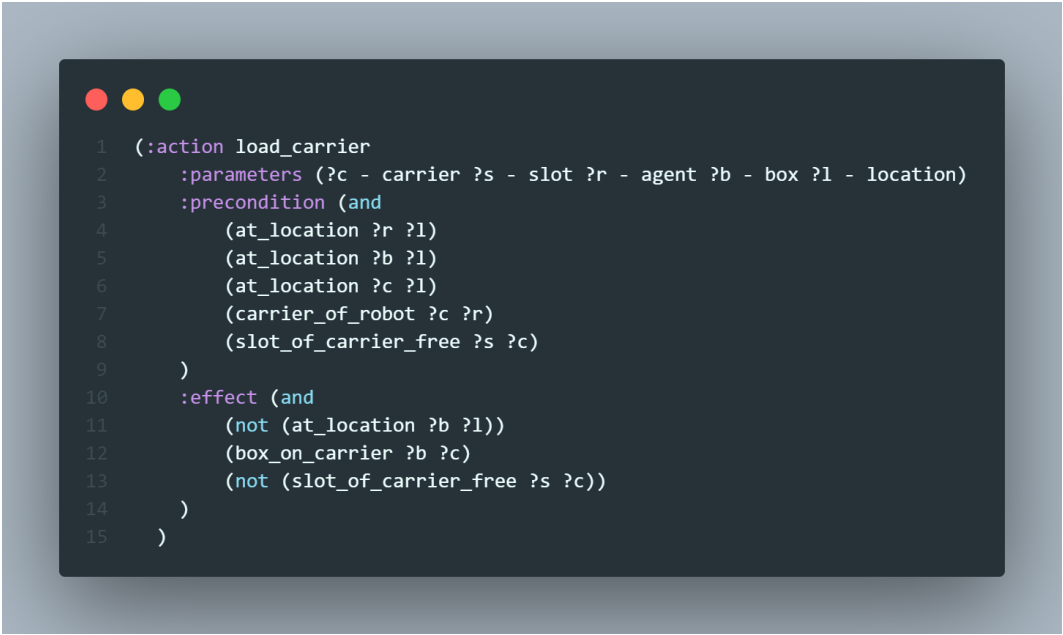


```
1  (:action fill_box
2    :parameters (?b - box ?c - supply ?r - agent ?l - location)
3    :precondition (and
4      (empty_box ?b)
5      (at_location ?c ?l)
6      (at_location ?r ?l)
7      (at_location ?b ?l)
8    )
9    :effect (and
10      (not (empty_box ?b))
11      (content_in_box ?b ?c)
12    )
13  )
```

Figura 2.7: Azione *fill_box*

Questa azione consente all'agente ?r di riempire la scatola ?b con il materiale ?c. È necessario che agente, scatola e materiale si trovino tutti nella stessa location ?l e che la scatola sia vuota. Dopo aver eseguito l'azione, la scatola non sarà più vuota, ma riempita dall'agente ?r del materiale ?c;

→ `load_carrier`:



```
1 (:action load_carrier
2   :parameters (?c - carrier ?s - slot ?r - agent ?b - box ?l - location)
3   :precondition (and
4     (at_location ?r ?l)
5     (at_location ?b ?l)
6     (at_location ?c ?l)
7     (carrier_of_robot ?c ?r)
8     (slot_of_carrier_free ?s ?c)
9   )
10  :effect (and
11    (not (at_location ?b ?l))
12    (box_on_carrier ?b ?c)
13    (not (slot_of_carrier_free ?s ?c))
14  )
15 )
```

Figura 2.8: Azione *load_carrier*

Questa azione consente all'agente ?r di caricare la scatola ?b sul suo trasportatore ?c, precisamente sullo slot ?s. È necessario che agente, scatola e trasportatore si trovino nella stessa location ?l e che lo slot ?s sia libero. Dopo aver eseguito l'azione, il trasportatore ?c starà portando la scatola ?b e lo slot ?s non sarà più vuoto; inoltre la scatola ?b non sarà più nella location ?l perché verrà trasportata altrove;

→ `empty_box`:

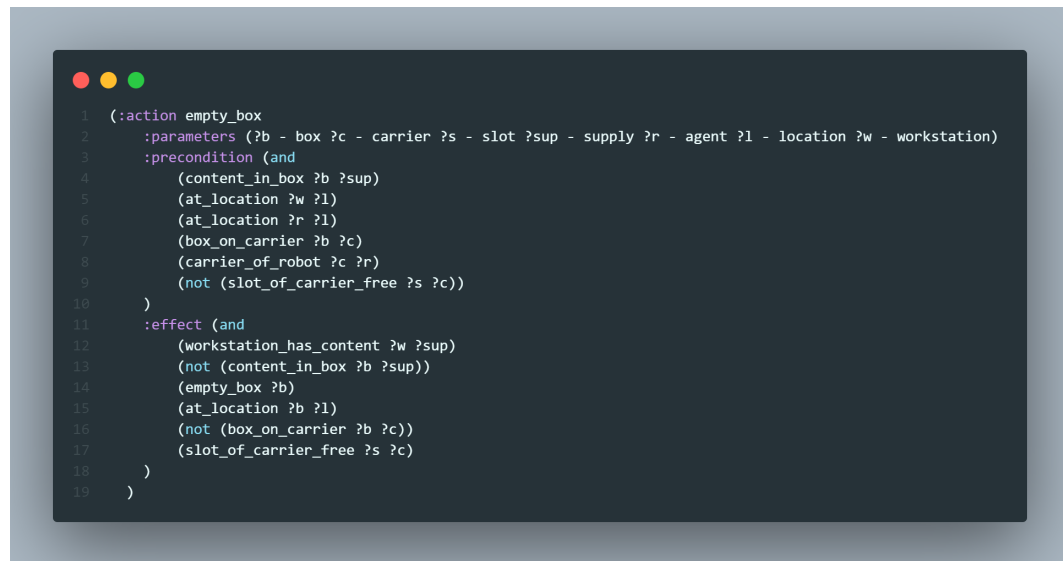


Figura 2.9: Azione *empty_box*

Questa azione consente all'agente ?r di svuotare la scatola ?b che contiene il materiale ?sup per servire la workstation ?w. È necessario che l'agente e la workstation si trovino nella stessa location ?l, che il materiale ?sup sia nella scatola ?b, che la scatola si trovi sul trasportatore ?c nello slot occupato ?s. Dopo aver eseguito l'azione, la workstation ?w avrà ricevuto il materiale ?c, la scatola ?b sarà vuota e si troverà nella location ?l e lo slot ?s del trasportatore ?c sarà libero.

Capitolo 3

Classical Planning

3.1 Istanza 1

L'istanza relativa al problema *Instance 1* implementa il primo scenario del dominio descritto nel Capitolo 2.1.

In questo problema troviamo i seguenti oggetti:

- **agent**: robot1, robot2;
- **box**: box1, box2, box3, box4;
- **supply**: bolt, valve, tool, wrench, screw, pickaxe, axe;
- **workstation**: workstation1, workstation2, workstation3, workstation4;
- **location**: warehouse, location1, location2, location3, location4;

Inizialmente tutti gli agenti, le scatole vuote e i materiali si trovano nella warehouse. Le workstation sono posizionate come segue:

- workstation1 in location1;
- workstation2 in location2;
- workstation3 in location3;
- workstation4 in location4;

Le location saranno collegate come segue (la freccia indica che i collegamenti sono fatti in entrambe le direzioni):

→ warehouse \leftrightarrow location1;

→ location1 \leftrightarrow location2;

→ location2 \leftrightarrow location3;

→ location3 \leftrightarrow location4;

L'obiettivo finale è descritto come segue:

→ workstation1 possiede bolt;

→ workstation2 possiede pickaxe;

→ workstation2 possiede screw;

→ workstation3 possiede tool;

→ robot1 e robot2 si trovano alla warehouse;

In figura 3.1 è possibile visualizzare il plan restituito da LPG-TD:

```

1 ; Version LPG-td-1.4
2 ; Seed 107495382
3 ; Command line: ./lpg-td -o ../instance_1/domain.pddl -f ../instance_1/problem.pddl -n 1
4 ; Problem ../instance 1
5 ; Actions having STRIPS duration
6 ; Time 0.08
7 ; Search time 0.07
8 ; Parsing time 0.01
9 ; Mutex time 0.00
10 ; NrActions 28
11
12 0: (FILL_BOX BOX4 SCREW ROBOT1 WAREHOUSE) [1]
13 0: (FILL_BOX BOX3 TOOL ROBOT2 WAREHOUSE) [1]
14 0: (FILL_BOX BOX1 PICKAXE ROBOT2 WAREHOUSE) [1]
15 1: (PICK_UP_BOX BOX4 ROBOT1 WAREHOUSE) [1]
16 1: (PICK_UP_BOX BOX3 ROBOT2 WAREHOUSE) [1]
17 2: (MOVE ROBOT1 WAREHOUSE LOCATION1) [1]
18 2: (MOVE ROBOT2 WAREHOUSE LOCATION1) [1]
19 3: (MOVE ROBOT1 LOCATION1 LOCATION2) [1]
20 3: (MOVE ROBOT2 LOCATION1 LOCATION2) [1]
21 4: (EMPTY_BOX BOX4 SCREW ROBOT1 LOCATION2 WORKSTATION2) [1]
22 4: (MOVE ROBOT2 LOCATION2 LOCATION3) [1]
23 5: (MOVE ROBOT1 LOCATION2 LOCATION1) [1]
24 5: (EMPTY_BOX BOX3 TOOL ROBOT2 LOCATION3 WORKSTATION3) [1]
25 6: (MOVE ROBOT1 LOCATION1 WAREHOUSE) [1]
26 6: (MOVE ROBOT2 LOCATION3 LOCATION2) [1]
27 7: (MOVE ROBOT2 LOCATION2 LOCATION1) [1]
28 8: (MOVE ROBOT2 LOCATION1 WAREHOUSE) [1]
29 9: (FILL_BOX BOX2 BOLT ROBOT2 WAREHOUSE) [1]
30 10: (PICK_UP_BOX BOX1 ROBOT2 WAREHOUSE) [1]
31 11: (MOVE ROBOT2 WAREHOUSE LOCATION1) [1]
32 12: (MOVE ROBOT2 LOCATION1 LOCATION2) [1]
33 13: (EMPTY_BOX BOX1 PICKAXE ROBOT2 LOCATION2 WORKSTATION2) [1]
34 14: (MOVE ROBOT2 LOCATION2 LOCATION1) [1]
35 15: (MOVE ROBOT2 LOCATION1 WAREHOUSE) [1]
36 16: (PICK_UP_BOX BOX2 ROBOT2 WAREHOUSE) [1]
37 17: (MOVE ROBOT2 WAREHOUSE LOCATION1) [1]
38 18: (EMPTY_BOX BOX2 BOLT ROBOT2 LOCATION1 WORKSTATION1) [1]
39 19: (MOVE ROBOT2 LOCATION1 WAREHOUSE) [1]

```

Figura 3.1: Plan relativo all'istanza 1

3.2 Istanza 2

L'istanza relativa al problema *Instance 2* implementa il secondo scenario del dominio descritto nel Capitolo 2.1.

In questo problema troviamo i seguenti oggetti:

- **agent:** robot1, robot2;
- **carrier:** c, c1;
- **slot:** s1, s2, s3, s4;
- **box:** box1, box2, box3, box4;
- **supply:** bolt, valve, tool, wrench, screw;

- **workstation:** workstation1, workstation2, workstation3;
- **location:** warehouse, location1, location2, location3;

Inizialmente tutti gli agenti, le scatole vuote e i materiali si trovano nella warehouse. Il trasportatore c viene assegnato all'agente robot1, mentre c1 viene assegnato a robot2. Gli slot s1 ed s2 appartengono al trasportatore c, mentre s2 ed s4 appartengono a c1. Le workstation sono posizionate come segue:

- workstation1 in location1;
- workstation2 in location2;
- workstation3 in location3;

Le location saranno collegate come segue (la freccia indica che i collegamenti sono fatti in entrambe le direzioni):

- warehouse \leftrightarrow location1;
- location1 \leftrightarrow location2;
- location2 \leftrightarrow location3;

L'obiettivo finale è descritto come segue:

- workstation1 possiede bolt;
- workstation2 possiede valve;
- workstation2 possiede screw;
- workstation3 possiede tool;
- robot1 e robot2 si trovano alla warehouse;

In figura 3.2 è possibile visualizzare il plan restituito da LPG-TD:

```
; Version LPG-td-1.4
; Seed 42710136
; Command line: ./lpg-td -o ../instance_2/domain2.pddl -f ../instance_2/problem2.pddl -n 1
; Problem ../instance_2
; Actions having STRIPS duration
; Time 0.03
; Search time 0.02
; Parsing time 0.01
; Mutex time 0.00
; NrActions 24

0: (FILL_BOX BOX4 SCREW ROBOT2 WAREHOUSE) [1]
0: (FILL_BOX BOX2 VALVE ROBOT1 WAREHOUSE) [1]
0: (FILL_BOX BOX1 BOLT ROBOT2 WAREHOUSE) [1]
0: (FILL_BOX BOX3 TOOL ROBOT2 WAREHOUSE) [1]
1: (LOAD_CARRIER C S1 ROBOT1 BOX4 WAREHOUSE) [1]
1: (LOAD_CARRIER C S2 ROBOT1 BOX2 WAREHOUSE) [1]
1: (LOAD_CARRIER C1 S4 ROBOT2 BOX3 WAREHOUSE) [1]
2: (MOVE_WITH_CARRIER ROBOT1 C WAREHOUSE LOCATION1) [1]
2: (MOVE_WITH_CARRIER ROBOT2 C1 WAREHOUSE LOCATION1) [1]
3: (MOVE_WITH_CARRIER ROBOT1 C LOCATION1 LOCATION2) [1]
3: (MOVE_WITH_CARRIER ROBOT2 C1 LOCATION1 LOCATION2) [1]
4: (EMPTY_BOX BOX4 C S4 SCREW ROBOT1 LOCATION2 WORKSTATION2) [1]
4: (EMPTY_BOX BOX2 C S3 VALVE ROBOT1 LOCATION2 WORKSTATION2) [1]
4: (MOVE_WITH_CARRIER ROBOT2 C1 LOCATION2 LOCATION3) [1]
5: (MOVE_WITH_CARRIER ROBOT1 C LOCATION2 LOCATION1) [1]
5: (EMPTY_BOX BOX3 C1 S4 TOOL ROBOT2 LOCATION3 WORKSTATION3) [1]
6: (MOVE_WITH_CARRIER ROBOT1 C LOCATION1 WAREHOUSE) [1]
6: (MOVE_WITH_CARRIER ROBOT2 C1 LOCATION3 LOCATION2) [1]
7: (LOAD_CARRIER C S4 ROBOT1 BOX1 WAREHOUSE) [1]
7: (MOVE_WITH_CARRIER ROBOT2 C1 LOCATION2 LOCATION1) [1]
8: (MOVE_WITH_CARRIER ROBOT1 C WAREHOUSE LOCATION1) [1]
8: (MOVE_WITH_CARRIER ROBOT2 C1 LOCATION1 WAREHOUSE) [1]
9: (EMPTY_BOX BOX1 C S1 BOLT ROBOT1 LOCATION1 WORKSTATION1) [1]
10: (MOVE_WITH_CARRIER ROBOT1 C LOCATION1 WAREHOUSE) [1]
```

Figura 3.2: Plan relativo all'istanza 2

3.3 Planner personalizzato

Per la realizzazione del planner si è utilizzato PDDL4J, una libreria Java per la definizione e la soluzione di problemi di pianificazione utilizzando il linguaggio di descrizione della pianificazione (PDDL). Dopo un'analisi iniziale, è stato deciso di applicare la seguente strategia: l'algoritmo di ricerca utilizzato è A* leggermente modificato, mentre le euristiche utilizzate sono Fast Forward e Adjusted Sum.

3.3.1 Algoritmo di ricerca

L'idea principale di A^* è quella di trovare un percorso tra un punto di partenza e una destinazione in un grafo ponderato, minimizzando la somma dei costi dei percorsi parziali attraverso nodi intermedi. L'algoritmo tiene traccia di due valori per ciascun nodo visitato:

- $g(n)$: il costo effettivo del percorso dal punto di partenza al nodo corrente;
- $h(n)$: una stima del costo dal nodo corrente alla destinazione. Questa è una funzione euristica che stima quanto sia costoso raggiungere la destinazione dal nodo corrente.

A^* calcola una funzione di costo totale per ciascun nodo visitato: $f(n) = g(n) + h(n)$. L'algoritmo esplora i nodi in base al valore $f(n)$, selezionando prima i nodi con il valore $f(n)$ più basso. Questo fa sì che A^* sia una combinazione di ricerca basata su costo ($g(n)$) e una stima euristica ($h(n)$). L'efficacia di A^* dipende dalla scelta di una buona funzione euristica $h(n)$. Se $h(n)$ è ammissibile (cioè, non sovrastima mai il costo reale di raggiungere la destinazione), allora A^* garantisce di trovare il percorso più breve. Tuttavia, se la funzione euristica non è ammissibile, A^* potrebbe non garantire la soluzione ottimale.

La modifica principale apportata riguarda l'utilizzo di un'hash map al posto di un hash set per memorizzare gli stati chiusi e il rispettivo valore della funzione di costo: il primo miglioramento riscontrato risulta essere evidente nella memoria utilizzata. La scelta di utilizzare l'hash map ci ha consentito di diminuire il numero di stati esplorati: questo perché è stato inserito un controllo sul costo di uno stato ogni volta che viene esplorato nuovamente. Nel codice di default un nodo viene etichettato come esplorato, ma non viene effettuato un controllo sul costo nel caso in cui venga esplorato nuovamente. La modifica apportata al codice ci ha consentito di riprendere in considerazione solo gli stati che abbiano un costo minore di quello attualmente memorizzato nell'hash map.

```

1 public Plan aStar(Problem problem) throws ProblemNotSupportedException {
2     final StateHeuristic heuristic = StateHeuristic.getInstance(this.getHeuristic(), problem);
3     final State init = new State(problem.getInitialState());
4     final Node root = new Node(init, null, -1, 0, heuristic.estimate(init, problem.getGoal()));
5
6     final Map<State, Double> closed = new HashMap<>(); // Usa una HashMap per memorizzare i costi minimi degli stati
7     final double weight = this.getHeuristicWeight();
8     final PriorityQueue<Node> open = new PriorityQueue<>(100,
9         (n1, n2) -> Double.compare(weight * n1.getHeuristic() + n1.getCost(), weight * n2.getHeuristic() + n2.getCost())
10    );
11
12    open.add(root);
13
14    final int timeout = this.getTimeout() * 1000;
15    long time = System.currentTimeMillis();
16
17    while (!open.isEmpty() && (System.currentTimeMillis() - time) < timeout) {
18        final Node current = open.poll();
19        this.numOfStatesEvaluated++;
20
21        if (current.satisfy(problem.getGoal())) {
22            return extractPlan(current, problem);
23        }
24
25        // Se lo stato è già stato visitato con un costo inferiore, salta questo nodo
26        if (closed.containsKey(current) && closed.get(current) <= current.getCost()) {
27            continue;
28        }
29
30        closed.put(current, current.getCost()); // Aggiorna il costo minore per lo stato corrente
31
32        for (Action a : problem.getActions()) {
33            if (a.isApplicable(current)) {
34                Node next = new Node(current);
35
36                for (ConditionalEffect ce : a.getConditionalEffects()) {
37                    if (current.satisfy(ce.getCondition())) {
38                        next.apply(ce.getEffect());
39                    }
40                }
41
42                next.setCost(current.getCost() + 1);
43                next.setParent(current);
44                next.setAction(problem.getActions().indexOf(a));
45                next.setHeuristic(heuristic.estimate(next, problem.getGoal()));
46
47                if (!closed.containsKey(next) || closed.get(next) > next.getCost()) {
48                    open.add(next);
49                }
50            }
51        }
52    }
53
54    return null; // Nessun piano trovato entro il timeout
55 }

```

Figura 3.3: Codice dell'algoritmo A*

3.3.2 Euristica

Le funzioni euristiche usate per risolvere le istanze precedentemente esposte sono:

- **Fast-Forward**: spesso abbreviata come FF, è un'euristica basata sui piani rilassati. È stata introdotta da J. Hoffmann e deriva dall'algoritmo Graphplan, il quale provvede alla costruzione di un grafo a livelli alternati di proposizioni e azioni. Tale grafo serve a rappresentare le possibili evoluzioni di uno stato iniziale verso uno stato finale;

- **AdjustedSum**: è un'euristica additiva, proposta da Nguye e Kambampati, che tenta di migliorare la stima sommando i costi delle singole sotto-attività, ma con aggiustamenti per tenere conto delle interazioni tra le attività.

3.4 Risultati ottenuti

3.4.1 Istanza 1: A* e Fast-Forward peso 3

```

00: (          fill_box box2 bolt robot2 warehouse) [0]
01: (          pick_up_box box2 robot1 warehouse) [0]
02: (          move robot1 warehouse location1) [0]
03: (    empty_box box2 bolt robot1 location1 workstation1) [0]
04: (          fill_box box4 screw robot2 warehouse) [0]
05: (          fill_box box1 pickaxe robot2 warehouse) [0]
06: (          fill_box box3 tool robot2 warehouse) [0]
07: (          pick_up_box box2 robot1 location1) [0]
08: (          pick_up_box box1 robot2 warehouse) [0]
09: (          move robot1 location1 warehouse) [0]
10: (          move robot2 warehouse location1) [0]
11: (          move robot2 location1 location2) [0]
12: (empty_box box1 pickaxe robot2 location2 workstation2) [0]
13: (          move robot2 location2 location1) [0]
14: (          move robot2 location1 warehouse) [0]
15: (          pick_up_box box4 robot2 warehouse) [0]
16: (          move robot2 warehouse location1) [0]
17: (          move robot2 location1 location2) [0]
18: (    empty_box box4 screw robot2 location2 workstation2) [0]
19: (          move robot2 location2 location1) [0]
20: (          move robot2 location1 warehouse) [0]
21: (          pick_up_box box3 robot2 warehouse) [0]
22: (          move robot2 warehouse location1) [0]
23: (          move robot2 location1 location2) [0]
24: (          move robot2 location2 location3) [0]
25: (    empty_box box3 tool robot2 location3 workstation3) [0]
26: (          move robot2 location3 location2) [0]
27: (          move robot2 location2 location1) [0]
28: (          move robot2 location1 warehouse) [0]

```

Figura 3.4: Istanza 1: A* e Fast-Forward peso 3

Come si può notare dalla figura 3.4, il plan restituito risulta essere efficiente nella prima fase in cui lavorano entrambi i robot in sinergia, per poi passare ad una seconda fase che si protrae fino alla fine del plan in cui è solamente il robot2 a svolgere tutto il lavoro.

3.4.2 Istanza 1: A* e AdjustedSum peso 7

```

00: (          fill_box box2 screw robot1 warehouse) [0]
01: (          pick_up_box box2 robot1 warehouse) [0]
02: (          move robot1 warehouse location1) [0]
03: (          fill_box box1 bolt robot2 warehouse) [0]
04: (          pick_up_box box1 robot2 warehouse) [0]
05: (          move robot2 warehouse location1) [0]
06: (    empty_box box1 bolt robot2 location1 workstation1) [0]
07: (          move robot2 location1 warehouse) [0]
08: (          fill_box box3 pickaxe robot2 warehouse) [0]
09: (          pick_up_box box3 robot2 warehouse) [0]
10: (          move robot2 warehouse location1) [0]
11: (          move robot1 location1 location2) [0]
12: (    empty_box box2 screw robot1 location2 workstation2) [0]
13: (          move robot1 location2 location1) [0]
14: (          move robot2 location1 location2) [0]
15: (empty_box box3 pickaxe robot2 location2 workstation2) [0]
16: (          move robot2 location2 location1) [0]
17: (          move robot1 location1 warehouse) [0]
18: (          fill_box box4 tool robot1 warehouse) [0]
19: (          move robot2 location1 warehouse) [0]
20: (          pick_up_box box4 robot1 warehouse) [0]
21: (          move robot1 warehouse location1) [0]
22: (          move robot1 location1 location2) [0]
23: (          move robot1 location2 location3) [0]
24: (    empty_box box4 tool robot1 location3 workstation3) [0]
25: (          move robot1 location3 location2) [0]
26: (          move robot1 location2 location1) [0]
27: (          move robot1 location1 warehouse) [0]

```

Figura 3.5: Istanza 1: A* e AdjustedSum peso 7

Come si può notare dalla figura 3.5, il plan restituito risulta essere molto più efficiente in termini di suddivisione dei compiti da parte dei robot. Un altro dettaglio da riportare è il fatto che il numero di azioni è leggermente inferiore rispetto al plan precedente 3.4.

3.4.3 Istanza 2: A* e Fast-Forward peso 7

```

00: (          fill_box box1 bolt robot1 warehouse) [0]
01: (          load_carrier c s1 robot1 box1 warehouse) [0]
02: (          move_with_carrier robot1 c warehouse location1) [0]
03: ( empty_box box1 c s4 bolt robot1 location1 workstation1) [0]
04: (          fill_box box4 valve robot2 warehouse) [0]
05: (          fill_box box2 tool robot2 warehouse) [0]
06: (          fill_box box3 screw robot2 warehouse) [0]
07: (          move_with_carrier robot1 c location1 warehouse) [0]
08: (          load_carrier c s2 robot1 box2 warehouse) [0]
09: (          load_carrier c1 s3 robot2 box4 warehouse) [0]
10: (          load_carrier c s4 robot1 box3 warehouse) [0]
11: (          move_with_carrier robot1 c warehouse location1) [0]
12: (          move_with_carrier robot1 c location1 location2) [0]
13: ( empty_box box3 c s1 screw robot1 location2 workstation2) [0]
14: (          move_with_carrier robot1 c location2 location3) [0]
15: ( empty_box box2 c s2 tool robot1 location3 workstation3) [0]
16: (          move_with_carrier robot1 c location3 location2) [0]
17: (          move_with_carrier robot1 c location2 location1) [0]
18: (          move_with_carrier robot1 c location1 warehouse) [0]
19: (          move_with_carrier robot2 c1 warehouse location1) [0]
20: (          move_with_carrier robot2 c1 location1 location2) [0]
21: (empty_box box4 c1 s1 valve robot2 location2 workstation2) [0]
22: (          move_with_carrier robot2 c1 location2 location1) [0]
23: (          move_with_carrier robot2 c1 location1 warehouse) [0]

```

Figura 3.6: Istanza 2: A* e Fast-Forward peso 7

Come si può notare dalla figura 3.6, il plan restituito mostra una suddivisione particolare del lavoro da parte dei robot: inizialmente, il robot2 si occupa di riempire la maggior parte delle scatole mentre il robot1 si occupa di trasportarle, sfruttando quasi sempre la capacità massima del suo trasportatore. Un dettaglio da sottolineare è il numero di azioni molto basso.

3.4.4 Istanza 2: A* e AdjustedSum peso 7

```

00: (          fill_box box3 screw robot1 warehouse) [0]
01: (          load_carrier c1 s4 robot2 box3 warehouse) [0]
02: (    move_with_carrier robot2 c1 warehouse location1) [0]
03: (          fill_box box2 bolt robot1 warehouse) [0]
04: (          load_carrier c s2 robot1 box2 warehouse) [0]
05: (    move_with_carrier robot1 c warehouse location1) [0]
06: ( empty_box box2 c s2 bolt robot1 location1 workstation1) [0]
07: (    move_with_carrier robot1 c location1 warehouse) [0]
08: (          fill_box box1 valve robot1 warehouse) [0]
09: (          load_carrier c s1 robot1 box1 warehouse) [0]
10: (    move_with_carrier robot1 c warehouse location1) [0]
11: (    move_with_carrier robot1 c location1 location2) [0]
12: ( empty_box box1 c s1 valve robot1 location2 workstation2) [0]
13: (    move_with_carrier robot1 c location2 location1) [0]
14: (    move_with_carrier robot2 c1 location1 location2) [0]
15: (empty_box box3 c1 s1 screw robot2 location2 workstation2) [0]
16: (    move_with_carrier robot2 c1 location2 location1) [0]
17: (    move_with_carrier robot1 c location1 warehouse) [0]
18: (          fill_box box4 tool robot1 warehouse) [0]
19: (    move_with_carrier robot2 c1 location1 warehouse) [0]
20: (          load_carrier c1 s1 robot2 box4 warehouse) [0]
21: (    move_with_carrier robot2 c1 warehouse location1) [0]
22: (    move_with_carrier robot2 c1 location1 location2) [0]
23: (    move_with_carrier robot2 c1 location2 location3) [0]
24: ( empty_box box4 c1 s1 tool robot2 location3 workstation3) [0]
25: (    move_with_carrier robot2 c1 location3 location2) [0]
26: (    move_with_carrier robot2 c1 location2 location1) [0]
27: (    move_with_carrier robot2 c1 location1 warehouse) [0]

```

Figura 3.7: Istanza 2: A* e AdjustedSum peso 7

Come si può notare dalla figura 3.7, il plan restituito mostra una suddivisione abbastanza equa del lavoro da parte dei robot. Ciò comporta un plan più efficiente da questo punto di vista, ma un numero di azioni superiore rispetto al plan precedente 3.6.

3.4.5 Confronto tra i risultati ottenuti

Osservando i plan precedentemente riportati, possiamo notare che l'euristica Fast-Forward predilige soluzioni più veloci ma meno efficienti, al contrario di AdjustedSum che riesce a restituire soluzioni più efficienti in tempi comunque rispettabili. Di seguito sono riportati alcuni dati aggiuntivi riguardo l'esecuzione del planner con le varie euristiche, confrontando le statistiche ottenute con il peso euristico migliore trovato rispetto a quelle ottenute con un peso euristico inferiore:

A*+Euristica	Peso	Istanza	Tempo	Stati valutati	Memoria utilizzata
Fast-Forward	3	1	1,10s	3607	1,55 MB
Fast-Forward	2	1	35s	477.751	1,55 MB
AdjustedSum	7	1	0,50s	666	1,53 MB
AdjustedSum	2	1	30s	375.995	1,53 MB
Fast-Forward	7	2	0,50s	1094	2,84 MB
Fast-Forward	2	2	15s	65.258	2,84 MB
AdjustedSum	7	2	0,75s	711	2,85 MB
AdjustedSum	2	2	30s	179.466	2,85 MB

Tabella 3.1: Comparazione dei risultati ottenuti

La tabella 3.1 dimostra che il peso dell'euristica è un fattore determinante nell'esecuzione del planner: si può ampiamente notare il distacco che si verifica tra i dati ottenuti dall'euristica con peso migliore trovato rispetto alla stessa euristica con un peso inferiore.

Capitolo 4

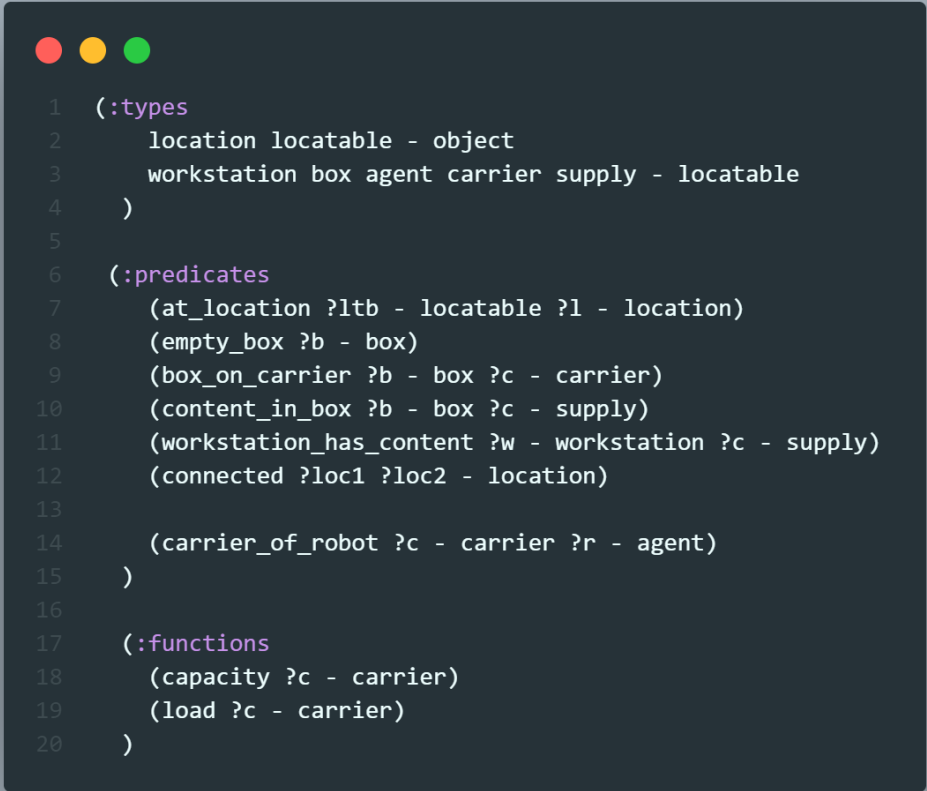
Temporal Planning

Per sviluppare il punto della traccia riguardante il *Temporal planning*, si è partiti dal dominio stilato per la seconda istanza e lo si è convertito per utilizzare le azioni durative. Le azioni durative differiscono da quelle classiche in quanto prevedono una durata, delle condizioni e degli effetti definiti in maniera differente: per ogni condizione ed effetto, si deve specificare se essi valgono solo all’inizio (*at start*), solo alla fine (*at end*) o per tutta la durata dell’azione (*over all*).

Un’altra differenza sostanziale con il dominio classico riguarda l’utilizzo delle *functions*: nel nostro caso specifico, esse sono state utilizzate per gestire la capacità dei trasportatori attraverso i *numeric-fluents*. È presente una funzione che gestisce la capacità massima del trasportatore e una che gestisce il carico attuale: in questo modo, risulta essere estremamente semplice controllare e manipolare la capienza del trasportatore durante le azioni di carico e scarico delle scatole.

Naturalmente, anche il problema presenta delle piccole modifiche rispetto a quello classico, in quanto bisogna inizializzare i valori delle funzioni. Per l’esecuzione si è scelto di utilizzare nuovamente il planner LPG-TD.

Di seguito vengono riportati i tipi, i predicati e le funzioni utilizzati:



```
1 (:types
2   location locatable - object
3   workstation box agent carrier supply - locatable
4 )
5
6 (:predicates
7   (at_location ?ltb - locatable ?l - location)
8   (empty_box ?b - box)
9   (box_on_carrier ?b - box ?c - carrier)
10  (content_in_box ?b - box ?c - supply)
11  (workstation_has_content ?w - workstation ?c - supply)
12  (connected ?loc1 ?loc2 - location)
13
14  (carrier_of_robot ?c - carrier ?r - agent)
15 )
16
17 (:functions
18   (capacity ?c - carrier)
19   (load ?c - carrier)
20 )
```

Figura 4.1: Tipi, predicati e funzioni del dominio temporale

La logica delle azioni è rimasta la medesima, ma la struttura è stata adattata per trasformarle in azioni durative e per utilizzare le funzioni. Di seguito sono riportate tutte le azioni durative, il problema durativo completo ed infine il plan generato da LPG-TD:


```

1  (:durative-action movewithcarrier
2    :parameters (?r - agent ?c - carrier ?l1 - location ?l2 - location)
3    :duration (= ?duration 3)
4    :condition (and
5      (at start (at_location ?c ?l1))
6      (at start (at_location ?r ?l1))
7      (over all (carrier_of_robot ?c ?r))
8      (over all (connected ?l1 ?l2))
9    )
10   :effect (and
11     (at start (not (at_location ?c ?l1)))
12     (at start (not (at_location ?r ?l1)))
13     (at end (at_location ?c ?l2))
14     (at end (at_location ?r ?l2))
15   )
16 )

```

Figura 4.2: Azione durativa *move_with_carrier*

```

1  (:durative-action fillbox
2    :parameters (?b - box ?s - supply ?r - agent ?l - location)
3    :duration (= ?duration 2)
4    :condition (and
5      (at start (empty_box ?b))
6      (over all (at_location ?s ?l))
7      (over all (at_location ?r ?l))
8      (over all (at_location ?b ?l))
9    )
10   :effect (and
11     (at end (not (empty_box ?b)))
12     (at end (content_in_box ?b ?s))
13   )
14 )

```

Figura 4.3: Azione durativa *fill_box*

```

1  (:durative-action loadcarrier
2    :parameters (?c - carrier ?r - agent ?b - box ?l - location)
3    :duration (= ?duration 3)
4    :condition (and
5      (over all (at_location ?r ?l))
6      (at start (at_location ?b ?l))
7      (over all (at_location ?c ?l))
8      (over all (carrier_of_robot ?c ?r))
9      (over all (< (load ?c) (capacity ?c))))
10  )
11  :effect (and
12    (at start (not (at_location ?b ?l)))
13    (at end (box_on_carrier ?b ?c))
14    (at end (increase (load ?c) 1))
15  )
16  )

```

Figura 4.4: Azione durativa *load_carrier*

```

1  (:durative-action emptybox
2    :parameters (?b - box ?c - carrier ?s - supply ?r - agent ?l - location ?w - workstation)
3    :duration (= ?duration 3)
4    :condition (and
5      (at start (content_in_box ?b ?s))
6      (over all (at_location ?w ?l))
7      (over all (at_location ?r ?l))
8      (at start (box_on_carrier ?b ?c))
9      (over all (carrier_of_robot ?c ?r))
10   )
11   :effect (and
12     (at start (not (box_on_carrier ?b ?c)))
13     (at end (workstation_has_content ?w ?s))
14     (at end (not (content_in_box ?b ?s)))
15     (at end (empty_box ?b))
16     (at end (at_location ?b ?l))
17     (at end (decrease (load ?c) 1))
18   )
19   )

```

Figura 4.5: Azione durativa *empty_box*

```

1 (define (problem prob3)
2   (:domain durative_task)
3
4   (:objects
5     robot - agent
6     c - carrier
7     box1 box2 box3 - box
8     bolt valve tool - supply
9     workstation1 workstation2 workstation3 - workstation
10    warehouse location1 location2 - location
11  )
12
13  (:init
14    (at_location robot warehouse) (at_location c warehouse) (carrier_of_robot c robot)
15
16    (= (capacity c) 2)
17    (= (load c) 0)
18
19    (at_location box1 warehouse) (at_location box2 warehouse) (at_location box3 warehouse)
20
21    (empty_box box1) (empty_box box2) (empty_box box3)
22
23    (at_location bolt warehouse) (at_location valve warehouse) (at_location tool warehouse)
24
25    (at_location workstation1 location1) (at_location workstation2 location2) (at_location workstation3 location2)
26
27    (connected warehouse location1) (connected location1 location2)
28    (connected location1 warehouse) (connected location2 location1)
29  )
30
31  (:goal
32    (and
33      (workstation_has_content workstation1 bolt) (workstation_has_content workstation2 valve) (workstation_has_content workstation3 tool)
34      (at_location robot warehouse)
35    )
36  )
37 )

```

Figura 4.6: Problema durativo

```

; Version LPG-td-1.4
; Seed 42298968
; Command line: ./lpg-td -o ../temporal/domain.pddl -f ../temporal/problem.pddl -v off -out qualityplan -quality
; Problem ../temporal/problem.pddl
; Time 0.51
; Plan generation time 0.08
; Search time 0.08
; Parsing time 0.00
; Mutex time 0.00
; MetricValue 15.00

0.0003: (FILLBOX BOX3 TOOL ROBOT WAREHOUSE) [2.0000]
0.0005: (FILLBOX BOX2 VALVE ROBOT WAREHOUSE) [2.0000]
0.0010: (FILLBOX BOX1 BOLT ROBOT WAREHOUSE) [2.0000]
2.0008: (LOADCARRIER C ROBOT BOX2 WAREHOUSE) [3.0000]
5.0012: (LOADCARRIER C ROBOT BOX1 WAREHOUSE) [3.0000]
8.0015: (MOVEWITHCARRIER ROBOT C WAREHOUSE LOCATION1) [3.0000]
11.0017: (EMPTYBOX BOX1 C BOLT ROBOT LOCATION1 WORKSTATION1) [3.0000]
14.0020: (MOVEWITHCARRIER ROBOT C LOCATION1 WAREHOUSE) [3.0000]
17.0023: (LOADCARRIER C ROBOT BOX3 WAREHOUSE) [3.0000]
20.0025: (MOVEWITHCARRIER ROBOT C WAREHOUSE LOCATION1) [3.0000]
23.0028: (MOVEWITHCARRIER ROBOT C LOCATION1 LOCATION2) [3.0000]
26.0030: (EMPTYBOX BOX3 C TOOL ROBOT LOCATION2 WORKSTATION3) [3.0000]
26.0033: (EMPTYBOX BOX2 C VALVE ROBOT LOCATION2 WORKSTATION2) [3.0000]
29.0035: (MOVEWITHCARRIER ROBOT C LOCATION2 LOCATION1) [3.0000]
32.0037: (MOVEWITHCARRIER ROBOT C LOCATION1 WAREHOUSE) [3.0000]

```

Figura 4.7: Plan generato dal dominio e dal problema durativi

Capitolo 5

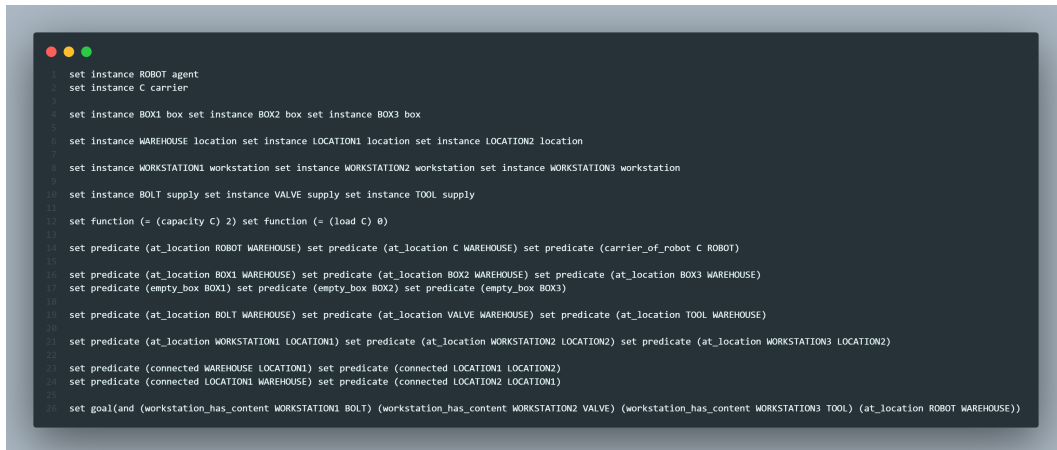
Robotics Planning

Per quanto concerne la fase di *Robotics Planning*, viene richiesto di implementare il problema risultante dalla fase precedente con l'utilizzo di PlanSys2 e mediante la definizione delle fake actions. La cartella presente nell'archivio del progetto va inserita all'interno della workspace di PlanSys2 per essere eseguita ed è così strutturata:

- Cartella *build*: contiene la build del progetto, che viene generata in seguito all'invocazione del comando “colcon build –symlink-install”;
- Cartella *install*: il file che ci interessa particolarmente è “setup.bash”, che viene eseguito ogni volta che vogliamo lanciare il progetto;
- Cartella *log*: contiene i file di log generati automaticamente;
- Cartella *pddl*: contiene il file di dominio descritto nel Capitolo 4 e il plan generato da LPG-TD, che è stato anche modificato per consentire il corretto funzionamento del progetto;
- Cartella *src*: contiene i file cpp in cui sono implementate le fake actions;
- Cartella *launch*: contiene il file “plansys2_rpl.py” necessario per avviare i nodi ROS responsabili dell'esecuzione del piano, ed il file “commands” che contiene tutte le specifiche riguardanti l'istanza del problema;
- File *CMakeLists.txt*: file necessario per la corretta esecuzione della build;

→ File *package.xml*: contiene tutte le dipendenze utilizzate dal progetto.

Di seguito viene riportato il codice del file “commands”:



```

1 set instance ROBOT agent
2 set instance C carrier
3
4 set instance BOX1 box set instance BOX2 box set instance BOX3 box
5
6 set instance WAREHOUSE location set instance LOCATION1 location set instance LOCATION2 location
7
8 set instance WORKSTATION1 workstation set instance WORKSTATION2 workstation set instance WORKSTATION3 workstation
9
10 set instance BOLT supply set instance VALVE supply set instance TOOL supply
11
12 set function (= (capacity C) 2) set function (= (load C) 0)
13
14 set predicate (at_location ROBOT WAREHOUSE) set predicate (at_location C WAREHOUSE) set predicate (carrier_of_robot C ROBOT)
15
16 set predicate (at_location BOX1 WAREHOUSE) set predicate (at_location BOX2 WAREHOUSE) set predicate (at_location BOX3 WAREHOUSE)
17 set predicate (empty_box BOX1) set predicate (empty_box BOX2) set predicate (empty_box BOX3)
18
19 set predicate (at_location BOLT WAREHOUSE) set predicate (at_location VALVE WAREHOUSE) set predicate (at_location TOOL WAREHOUSE)
20
21 set predicate (at_location WORKSTATION1 LOCATION1) set predicate (at_location WORKSTATION2 LOCATION2) set predicate (at_location WORKSTATION3 LOCATION2)
22
23 set predicate (connected WAREHOUSE LOCATION1) set predicate (connected LOCATION1 LOCATION2)
24 set predicate (connected LOCATION1 WAREHOUSE) set predicate (connected LOCATION2 LOCATION1)
25
26 set goal(and (workstation_has_content WORKSTATION1 BOLT) (workstation_has_content WORKSTATION2 VALVE) (workstation_has_content WORKSTATION3 TOOL) (at_location ROBOT WAREHOUSE))

```

Figura 5.1: File “commands” per l’istanziamento del problema

La definizione delle fake actions risulta essere equivalente per tutte. I passaggi principali sono:

1. **Definizione della classe:** viene definita una classe con il nome della fake action che eredita dalla classe “plansys2::ActionExecutorClient”;
2. **Definizione del costruttore:** nel costruttore della classe viene inizializzato l’executor dell’azione, richiamando il costruttore di “ActionExecutorClient” e specificando il nome dell’azione ed il valore di timeout in millisecondi. Inoltre, viene inizializzata la variabile “progress_” a 0.0;
3. **Metodo “do_work”:** questo metodo controlla lo stato di avanzamento dell’azione. Essenzialmente si occupa di incrementare la variabile “progress_” in maniera tale che la durata della fake action sia pari alla durata stabilita nel dominio. Il metodo si occupa anche di stampare sul terminale lo stato di avanzamento e di terminazione della fake action;
4. **Metodo “main”:** questo metodo rappresenta la funzione principale del programma, in cui viene inizializzato ROS2 tramite “rclcpp::init”, viene creata un’istanza della classe precedentemente definita e viene inizializ-

zato la fake action tramite i metodi forniti da ROS2. Infine viene chiuso il sistema ROS2.

```

1  #include <memory>
2  #include <algorithm>
3  #include <vector>
4
5  #include "plansys2_executor/ActionExecutorClient.hpp"
6
7  #include "rclcpp/rclcpp.hpp"
8  #include "rclcpp_action/rclcpp_action.hpp"
9
10 using namespace std::chrono_literals;
11
12 class MoveWithCarrierAction : public plansys2::ActionExecutorClient
13 {
14 public:
15     MoveWithCarrierAction()
16     : plansys2::ActionExecutorClient("movewithcarrier", 250ms)
17     {
18         progress_ = 0.0;
19     }
20
21 private:
22     void do_work()
23     {
24         std::vector<std::string> arg = get_arguments();
25         if (progress_ < 1.0) {
26             progress_ += 0.2;
27             send_feedback(progress_, "Robot " + arg[0] + " is moving from location " + arg[2] + " to location " + arg[3] + " with carrier " + arg[1]);
28         } else {
29             finish(true, 1.0, "Robot " + arg[0] + " moved from location " + arg[2] + " to location " + arg[3] + " with carrier " + arg[1]);
30
31             std::cout << "Robot " + arg[0] + " moved from location " + arg[2] + " to location " + arg[3] + " with carrier " + arg[1] + " ... [100%]" << std::endl;
32             progress_ = 0.0;
33         }
34     }
35
36     float progress_;
37 };
38
39 int main(int argc, char ** argv)
40 {
41     rclcpp::init(argc, argv);
42     auto node = std::make_shared<MoveWithCarrierAction>();
43
44     node->set_parameter(rclcpp::Parameter("action_name", "MOVEWITHCARRIER"));
45     node->trigger_transition(lifecycle_msgs::msg::Transition::TRANSITION_CONFIGURE);
46
47     rclcpp::spin(node->get_node_base_interface());
48
49     rclcpp::shutdown();
50
51     return 0;
52 }

```

Figura 5.2: Esempio di fake action

Per eseguire l'intero sistema si utilizzano due terminali, in modo che uno funga da nodo "server" e l'altro da nodo "client". Di seguito sono riportati i passi da eseguire su entrambi i terminali.

Terminale "server":

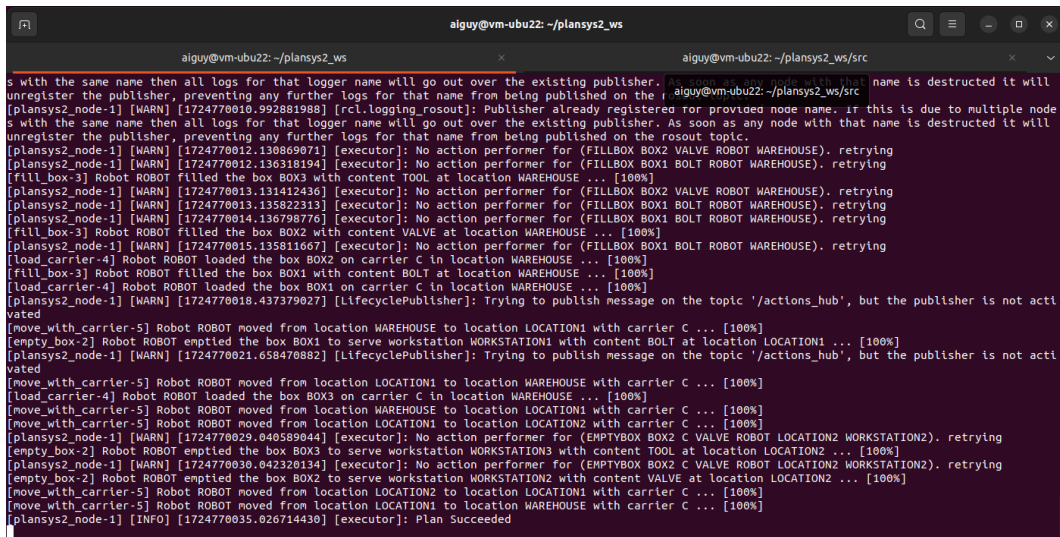
```
1 $ colcon build --symlink-install
2 $ source install/setup.bash
3 $ ros2 launch robotic_planning plansys2_rpl.py
```

Terminale "client":

```
1 $ source path/to/commandsFile/commands
2 $ run plan-file path/to/plan/myplanfile
```

Si noti che il file del piano deve essere precedentemente ripulito in modo da non contenga commenti e linee vuote ed in modo tale che segua la sintassi accettata da PlanSys2. Un altro importante dettaglio riguarda i percorsi dei file nel nodo "client", che devono essere tutti percorsi assoluti.

Al termine dell'esecuzione di tutte le fake actions, il risultato ottenuto sui due terminali è il seguente:



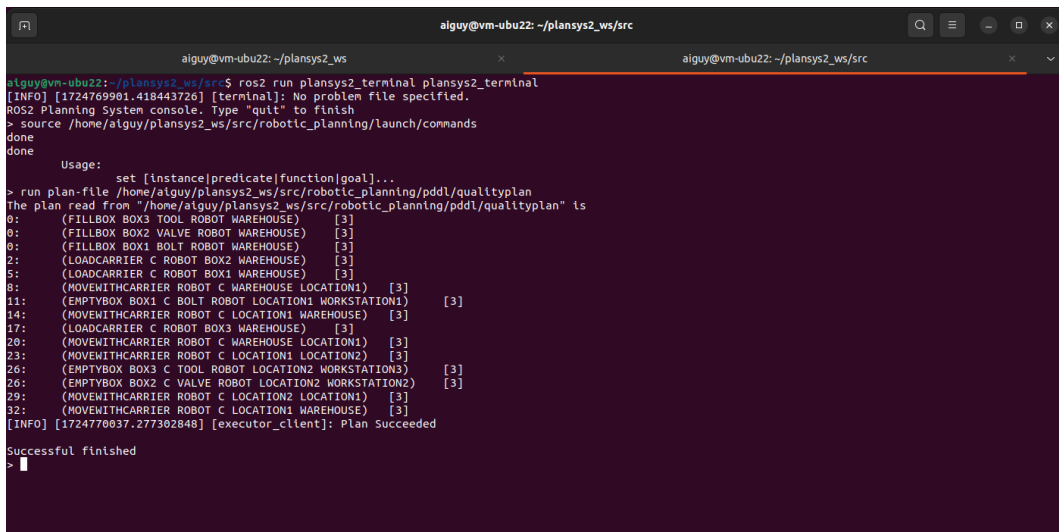
```

aiguy@vm-ubu22: ~/plansys2_ws
aiguy@vm-ubu22: ~/plansys2_ws/src

s with the same name then all logs for that logger name will go out over the existing publisher.
[plansys2_node-1] [WARN] [1724770010.992881988] [rcl_logging_rosout]: Publisher already registered for provided node name. If this is due to multiple node
s with the same name then all logs for that logger name will go out over the existing publisher. As soon as any node with that name is destructed it will
unregister the publisher, preventing any further logs for that name from being published on the rosout topic.
[plansys2_node-1] [WARN] [1724770012.130869071] [executor]: No action performer for (FILLBOX BOX2 VALVE ROBOT WAREHOUSE). retrying
[plansys2_node-1] [WARN] [1724770012.136318194] [executor]: No action performer for (FILLBOX BOX1 BOLT ROBOT WAREHOUSE). retrying
[fill_box-3] Robot ROBOT filled the box BOX3 with content TOOL at location WAREHOUSE ... [100%]
[plansys2_node-1] [WARN] [1724770013.131412436] [executor]: No action performer for (FILLBOX BOX2 VALVE ROBOT WAREHOUSE). retrying
[plansys2_node-1] [WARN] [1724770013.135822313] [executor]: No action performer for (FILLBOX BOX1 BOLT ROBOT WAREHOUSE). retrying
[plansys2_node-1] [WARN] [1724770014.136798776] [executor]: No action performer for (FILLBOX BOX1 BOLT ROBOT WAREHOUSE). retrying
[fill_box-3] Robot ROBOT filled the box BOX2 with content VALVE at location WAREHOUSE ... [100%]
[plansys2_node-1] [WARN] [1724770015.135811667] [executor]: No action performer for (FILLBOX BOX1 BOLT ROBOT WAREHOUSE). retrying
[load_carrier-4] Robot ROBOT loaded the box BOX2 on carrier C in location WAREHOUSE ... [100%]
[fill_box-3] Robot ROBOT filled the box BOX1 with content BOLT at location WAREHOUSE ... [100%]
[load_carrier-4] Robot ROBOT loaded the box BOX1 on carrier C in location WAREHOUSE ... [100%]
[plansys2_node-1] [WARN] [1724770018.437379027] [LifecyclePublisher]: Trying to publish message on the topic '/actions_hub', but the publisher is not acti
vated
[move_with_carrier-5] Robot ROBOT moved from location WAREHOUSE to location LOCATION1 with carrier C ... [100%]
[empty_box-2] Robot ROBOT emptied the box BOX1 to serve workstation WORKSTATION1 with content BOLT at location LOCATION1 ... [100%]
[plansys2_node-1] [WARN] [1724770021.658470882] [LifecyclePublisher]: Trying to publish message on the topic '/actions_hub', but the publisher is not acti
vated
[move_with_carrier-5] Robot ROBOT moved from location LOCATION1 to location WAREHOUSE with carrier C ... [100%]
[load_carrier-4] Robot ROBOT loaded the box BOX3 on carrier C in location WAREHOUSE ... [100%]
[move_with_carrier-5] Robot ROBOT moved from location WAREHOUSE to location LOCATION1 with carrier C ... [100%]
[move_with_carrier-5] Robot ROBOT moved from location LOCATION1 to location LOCATION2 with carrier C ... [100%]
[plansys2_node-1] [WARN] [1724770029.040589044] [executor]: No action performer for (EMPTYBOX BOX2 C VALVE ROBOT LOCATION2 WORKSTATION2). retrying
[empty_box-2] Robot ROBOT emptied the box BOX3 to serve workstation WORKSTATION3 with content TOOL at location LOCATION2 ... [100%]
[plansys2_node-1] [WARN] [1724770030.042281341] [executor]: No action performer for (EMPTYBOX BOX2 C VALVE ROBOT LOCATION2 WORKSTATION2). retrying
[empty_box-2] Robot ROBOT emptied the box BOX2 to serve workstation WORKSTATION2 with content VALVE at location LOCATION2 ... [100%]
[move_with_carrier-5] Robot ROBOT moved from location LOCATION2 to location LOCATION1 with carrier C ... [100%]
[move_with_carrier-5] Robot ROBOT moved from location LOCATION1 to location WAREHOUSE with carrier C ... [100%]
[plansys2_node-1] [INFO] [1724770035.026714430] [executor]: Plan Succeeded

```

Figura 5.3: Terminale “server”



```

aiguy@vm-ubu22: ~/plansys2_ws/src
aiguy@vm-ubu22: ~/plansys2_ws/src

aiguy@vm-ubu22:~/plansys2_ws/src$ ros2 run plansys2_terminal plansys2_terminal
[INFO] [1724769901.418443726] [terminal]: No problem file specified.
ROS2 Planning system console. Type "quit" to finish
> source /home/aiguy/plansys2_ws/src/robotic_planning/launch/commands
done
done
Usage:
  set [instance|predicate|function|goal]...
> run plan-file /home/aiguy/plansys2_ws/src/robotic_planning/pddl/qualityplan
The plan read from "/home/aiguy/plansys2_ws/src/robotic_planning/pddl/qualityplan" is
0: (FILLBOX BOX3 TOOL ROBOT WAREHOUSE) [3]
0: (FILLBOX BOX2 VALVE ROBOT WAREHOUSE) [3]
0: (FILLBOX BOX1 BOLT ROBOT WAREHOUSE) [3]
2: (LOADCARRIER C ROBOT BOX2 WAREHOUSE) [3]
5: (LOADCARRIER C ROBOT BOX1 WAREHOUSE) [3]
8: (MOVEWITHCARRIER ROBOT C WAREHOUSE LOCATION1) [3]
11: (EMPTYBOX BOX1 C BOLT ROBOT LOCATION1 WORKSTATION1) [3]
14: (MOVEWITHCARRIER ROBOT C LOCATION1 WAREHOUSE) [3]
17: (LOADCARRIER C ROBOT BOX3 WAREHOUSE) [3]
20: (MOVEWITHCARRIER ROBOT C WAREHOUSE LOCATION1) [3]
23: (MOVEWITHCARRIER ROBOT C LOCATION1 LOCATION2) [3]
26: (EMPTYBOX BOX3 C TOOL ROBOT LOCATION2 WORKSTATION3) [3]
26: (EMPTYBOX BOX2 C VALVE ROBOT LOCATION2 WORKSTATION2) [3]
29: (MOVEWITHCARRIER ROBOT C LOCATION2 LOCATION1) [3]
32: (MOVEWITHCARRIER ROBOT C LOCATION1 WAREHOUSE) [3]
[INFO] [1724770037.277302840] [executor_client]: Plan Succeeded
Successful finished
>

```

Figura 5.4: Terminale “client”

Capitolo 6

Organizzazione archivio

L'archivio consegnato è suddiviso nella maniera seguente:

- **Cartella *task 1***: contiene i due file di dominio corrispondenti alla prima e seconda istanza;
- **Cartella *task 2***: contiene tre sotto-cartelle: le prime due contengono i rispettivi file di dominio e problema, mentre la terza contiene tutti i file che compongono il planner personalizzato;
- **Cartella *task 3***: contiene due sotto-cartelle, una per ogni punto del task 3. La prima conterrà i file del dominio e problema durativi, mentre la seconda conterrà la cartella descritta nel Capitolo 5;
- **File *RelazioneIA_Fusto_Carida_Bongiovanni***: copia della relazione in formato PDF.
- **File *README***: contiene tutte le istruzioni necessarie all'esecuzione dei vari task.