Hospital Multi-Robot Medicine Delivery System: Consolidated PDDL Implementation Report

AINSTEIN Intelligent Systems Team
Elio Maria D'Alessandro, Cito Francesco, Yaekob Beyene

June 9, 2025

Abstract

This report presents a sophisticated Planning Domain Definition Language (PDDL) framework for a hospital multi-robot medicine delivery system, coordinating autonomous robots to deliver medications across multiple floors via a shared elevator. Four PDDL variants—PDDL 1.2 (STRIPS), PDDL 2.1 (numeric fluents), PDDL 2.1 with durative actions (temporal), and PDDL+ (processes and events)—model key operational constraints, including robot and elevator capacity, safety interlocks, and urgent medicine prioritization. By synthesizing these implementations, this report delivers a clear, actionable blueprint for optimizing hospital logistics, balancing planner compatibility, realism, and scalability. Performance results, challenges, and future enhancements are discussed to guide real-world deployment.

1 Introduction

Timely and efficient medication delivery is a cornerstone of modern hospital operations, directly impacting patient outcomes and staff efficiency. The hospital multi-robot delivery system addresses this challenge by coordinating four autonomous robots to deliver 16 medicines (four per floor) to patients across five floors, utilizing a shared elevator, and returning robots to a ground-floor base. The system enforces strict constraints, including robot and elevator capacity limits, floor-specific assignments, and safety protocols, with some models prioritizing urgent deliveries.

This report consolidates four PDDL implementations—PDDL 1.2 (STRIPS), PDDL 2.1, PDDL 2.1 Temporal, and PDDL+—each tailored to specific planner capabilities and operational needs. The objective is to minimize a total cost metric while ensuring safe, efficient delivery. This synthesis highlights the strengths, performance, and scalability of each approach, providing a robust foundation for hospital logistics automation.

2 System Overview

The hospital environment comprises:

- **Floors**: One ground floor (storage and robot rooms) and four patient floors, each with four patient rooms.
- **Robots**: Four robots, each assigned to a patient floor.
- **Medicines**: 16 medicines (four per patient floor), each designated for a specific patient.
- **Patients**: 16 patients (four per floor), located in respective rooms.
- **Elevator**: A shared elevator with a capacity of one or two robots, depending on the model.
- **Locations**: 18 total (two on ground floor, four per patient floor).

2.1 Initial state

At initial state :

- All robots are in floor 0 at robot room.
- All medicines are on floor 0 in the storage room.
- Elevator is in floor 0.
- All rooms in each floor are connected.
- All floors are connected.

2.2 Goal

The goal is to deliver all medicines to their designated patients and return robots to the robot room, minimizing a cost metric that accounts for movement, loading, delivery, and elevator actions. Constraints include:

- **Robot capacity:** Up to four medicines.
- **Elevator capacity:**Up to four robots.
- **Floor assignments:** Robots load and deliver medicines only for their assigned floors, except on the ground floor.
- **Safety:** Interlocks prevent unsafe movements (e.g., elevator conflicts).
- **Priority:** Urgent medicines prioritized in some models.

The general workflow of the scenario is shown in Figure 1

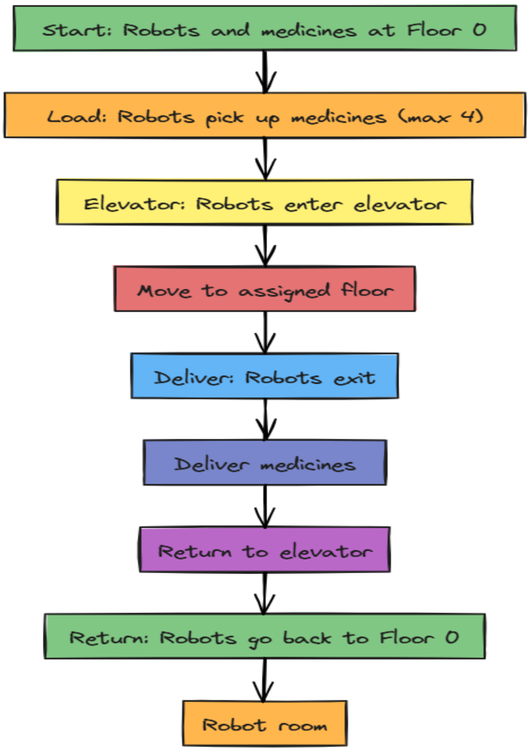

Figure 1: General Medicine Delivery Workflow

3 PDDL Implementations

The system is modeled using four PDDL variants, each addressing specific aspects of the problem. Below, we detail their domain structures, key features, and limitations.

3.1 PDDL 1.2 (STRIPS)

- **Requirements**: `:strips`, `:typing`, `:negative-preconditions`, `:equality`.
- **Types**: Robot, medicine, patient, elevator, location, floor.
- **Predicates**: `robot_at`, `medicine_at`, `medicine_for_floor`, `assigned_to_floor`, `carrying`, `can_move`, `in_elevator`, `elevator_at`, `robot_load_0` to `robot_load_4`.
- **Actions**: `move`, `enter_elevator`, `exit_elevator`, `load_med`, `deliver_med`, `move_elevator`, with priority actions (`load_priority_med`, `deliver_priority_med`).

  Load Medicine Action

  ```
  (:action load_med
    :parameters (?r - robot ?m - medicine ?l - location ?f - floor)
    :precondition (and
       (robot_at ?r ?l) (medicine_at ?m ?l) (medicine_for_floor ?m ?f)
        (assigned_to_floor ?r ?f)
        (or (robot_load_0 ?r) (robot_load_1 ?r) (robot_load_2 ?r)
          (robot_load_3 ?r)))
    :effect (and
        (not (medicine_at ?m ?l)) (carrying ?r ?m)
        (when (robot_load_0 ?r) (and (not (robot_load_0 ?r)) (robot_load_1 ?r)
              (not (can_move ?r))))
        (when (robot_load_1 ?r) (and (not (robot_load_1 ?r)) (robot_load_2 ?r)))
        (when (robot_load_2 ?r) (and (not (robot_load_2 ?r)) (robot_load_3 ?r)))
        (when (robot_load_3 ?r) (and (not (robot_load_3 ?r)) (robot_load_4 ?r)
              (can_move ?r)))))
  ```

- **Features**:
  - Uses one-hot encoding to track robot load (0–4 medicines), ensuring compatibility with all STRIPS planners.
  - Prioritizes urgent medicines via predicates (`medicine_priority`, `has_priority_med`, `priority_handled`), enforcing mutual exclusion (one urgent or up to four regular medicines).
- **Planner**: The planner used is **LAMA**. It excels on classic STRIPS domains because its landmark-based heuristics quickly find high-quality plans. Optimized grounding and preprocessing steps ensure fast search even on large Boolean problems. It is both reliable and efficient for evaluating pure PDDL 1.x implementations.[1]
- **Limitations**: Instantaneous actions lack temporal realism; one-hot encoding increases predicate count.
- **Priority-Handling Extension (Brief)**: Certain medications must preempt routine deliveries. We introduce three Boolean predicates—`medicine_priority` (flags urgent drugs), `has_priority_med` (robot carries urgent item), and `priority_handled` (urgent delivery completed)—and two specialized actions (`load_priority_med`, `deliver_priority_med`) that override the standard loading/delivering when an urgent drug is present. This scenario enforces an 'urgent-first' policy without relying on numeric fluents, improving responsiveness in time-critical situations.

3.2 PDDL 2.1 (Numeric Fluents)

- **Requirements**: :strips, :typing, :negative-preconditions, :equality, :numeric-fluents
- **Types**: Robot, location, medicine, patient, elevator, floor.
- **Predicates**: robot_at, belongs_to, robot_ready_for_delivery, robot_returning, entered_from.
- **Fluents**: load-count (0–4), elevator-load.
- **Actions**: load_medicine, enter_elevator, exit_elevator.

Load Medicine Action

```
(:action load_medicine
  :parameters (?r - robot ?m - medicine ?loc - location)
  :precondition (and
      (robot_at ?r ?loc)
      (medicine_at ?m ?loc)
      (belongs_to ?m ?r)
      (<= (load-count ?r) 3))
  :effect (and
      (not (medicine_at ?m ?loc))
      (carrying ?r ?m)
      (increase (load-count ?r) 1)
      (when (= (load-count ?r) 4)
        (robot_ready_for_delivery ?r))))
```

- **Features**:
  - Numeric fluents simplify load tracking.
  - Proposed elevator capacity of two robots, enforced via elevator-load.
  - Workflow control via robot_ready_for_delivery and entered_from.
- **Planner:** ENHSP is used to execute this implementation [3]
- **Limitations**: No temporal modeling and priority handling.

3.3 PDDL 2.1 Temporal

- **Requirements**: :strips, :typing, :negative-preconditions, :equality, :fluents, :durative-actions.
- **Types**: Robot, medicine, patient, elevator, location, floor.
- **Predicates**: robot_at, medicine_at, patient_at, connected, at_floor, elevator_at, robot_in_elevator, assigned_to_floor, medicine_for_floor, carrying, robot_available, elevator_available.
- **Fluents**: robot_load, elevator_load, total-cost, move_progress, elevator_progress, robot_speed, elevator_speed.
- **Actions**: Durative actions (e.g. load_med) with durations (e.g. 1 unit) and qualifiers (at start, over all, at end).

Load Medicine Action

```
(:durative-action load_med
  :parameters (?r - robot ?m - medicine ?l - location ?f - floor)
  :duration (= ?duration 1)
  :condition (and
      (over all (robot_at ?r ?l))              ; robot stays at source
      (at start (medicine_at ?m ?l))           ; medicine present at start
      (at start (medicine_for_floor ?m ?f))    ; correct floor assignment
      (at start (assigned_to_floor ?r ?f))     ; robot assigned to same floor
      (at start (<= (robot_load ?r) 3))        ; capacity check
      (at start (robot_available ?r)))         ; robot is free
  :effect (and
      (at start (not (medicine_at ?m ?l)))     ; remove from shelf
      (at end   (carrying ?r ?m))              ; robot carries med
      (at end   (increase (robot_load ?r) 1)) ; increment load
      (at end   (increase (total-cost) 1))     ; accumulate cost
  ))
```

- **Features**:
  - Models realistic timing and concurrency.
  - Optimizes `total-cost` (incremented by 1 per action).
- **Planner**: The LPG++ planner is used to execute this implementation[2].
- **Limitations**: No processes/events or explicit priority handling.

3.3.1 New scenario

We retain all previously defined elements and introduce a dedicated charging room on each floor. Whenever a robot's battery level drops below 25% during a delivery, the robot must navigate to the corresponding charging room on its assigned floor to recharge. This extension highlights a realistic operational contingency and improves the overall fidelity of the model.

3.4 PDDL+

- **Requirements**: Extends PDDL 2.1 with processes and events.
- **Types**: Robot, medicine, patient, elevator, location, floor.
- **Predicates**: `robot_at`, `medicine_at`, `elevator_at`, `assigned_to_floor`, `connected`.
- **Fluents**: `robot_load`, `elevator_load`, `total-cost`, `move_progress`, `elevator_progress`, `robot_speed` (0.2), `elevator_speed` (0.1).
- **Actions**: `start_move`, `start_elevator_move`, `load_med`, `deliver_med`, `enter_elevator`, `exit_elevator`.
- **Processes and Events**: Continuous movement (`move_progress`, `elevator_move_progress`) and safety checks (`robot_overload`, `elevator_overload`).
- **Features**:
  - Continuous movement (5 units for robots, 10 for elevator).
  - Enforcement of safety through events.
  - Cost optimization (`total-cost`: 0.5 for movement, 1 for others).

- **Planner:** `ENHSP` planner is used to execute the implementation [3]
- **Limitations**: Limited planner compatibility (e.g., ENHSP).

3.4.1 Battery-Aware Robot Navigation (Extension to PDDL+)

To enhance the realism of the PDDL+ implementation, we introduce **battery-aware navigation** for robots. This extension models continuous battery drain during movement and enforces recharging when levels fall below a specified battery level.

- **Robot:** 1 robot used; begins charging when battery $< 50\%$
- **Elevator:** 1 elevator for floor transitions
- **Floors:** 3 floors — Floor 0, Floor 1, Floor 2
- **Rooms:** 10 total — 2 on Floor 0, 4 each on Floors 1 & 2
- **Medicines:** 8 medicines delivered to 8 patients (1 per room on Floors 1 & 2)
- **Charging:** Charging station located at the robot room on Floor 0

4 Performance and Evaluation

The implementations were validated with compatible planners:

- **PDDL+**: ENHSP (`sat-had`) produced plans with costs of 50–100 units, accurately modeling movement times (5 units for robots, 10 for elevator).
- **PDDL 1.2**: Compatible with any STRIPS planner, successfully handled priority deliveries without temporal or cost metrics.
- **PDDL 2.1** Numeric: ENSHP generated planning time (msec): 1302 and grounding time: 119
- **PDDL 2.1 Temporal**: LPG++ generated plans in 2.39 seconds, supporting concurrency and cost optimization.

Table 1: Comparison of PDDL Implementations

Feature	PDDL+	PDDL 1.2	PDDL 2.1	PDDL 2.1 Temporal
Numeric Fluents	Yes	No (one-hot)	Yes	Yes
Temporal Modeling	Continuous (processes, events)	No	No	Durative actions
Priority Handling	No	Yes	No	No
Concurrency	Yes	No	No	Yes
Cost Optimization	Yes	No	No	Yes
Planner Compatibility	Low (ENHSP)	High (STRIPS)	Moderate	Moderate (LPG++)
Scalability	Moderate	High	Moderate	High
Realism	High	Low	Moderate	High

4.1 Analysis

- **PDDL+** offers superior realism with continuous dynamics, but is constrained by planner compatibility.
- **PDDL 1.2** excels in compatibility and priority handling, ideal for prototyping.
- **PDDL 2.1** simplifies load tracking but lacks timing and priorities.
- **PDDL 2.1 Temporal** balances realism, concurrency, and scalability, suitable for deployment.

5 Challenges

- **Planner Compatibility**: PDDL+'s advanced features limit planner options.
- **Scalability**: Large state spaces challenge planners, especially in PDDL+.
- **Priority Integration**: Only PDDL 1.2 addresses urgent deliveries, a critical hospital requirement.
- **Cost Tuning**: The metrics need to be adjusted to prioritize speed or urgency.

6 Future Directions

- **Unified Model**: Combine PDDL+'s continuous dynamics with PDDL 1.2's priority handling.
- **Dynamic Allocation**: Enable real-time task re-assignment for workload balancing.
- **Energy Modeling**: Incorporate battery constraints using PDDL+ processes.
- **Scalability**: Apply hierarchical planning to reduce state space.
- **Validation**: Test plans in simulated hospital environments to refine metrics.

7 Conclusion

This report consolidates four PDDL implementations for a hospital multi-robot medicine delivery system, each offering unique strengths: PDDL+ for continuous realism, PDDL 1.2 for compatibility and priority handling, PDDL 2.1 for simplified load tracking, and PDDL 2.1 Temporal for concurrency and scalability. Together, they provide a robust framework for automating hospital logistics.

8 Availability of Resources

To facilitate reproduction and experimentation, a comprehensive README file is included, detailing installation instructions for planners (LPG++, ENHSP, LAMA) and providing links to all relevant source code repositories. Future work should integrate priority handling into temporal or PDDL+ models and validate the generated plans in real world settings to ensure practical implementation.

References

[1] Planning.domains editor. https://editor.planning.domains/, n.d. Accessed: 2025-06-03.

[2] Matteo Carde. Lpg planner on github. https://github.com/matteocarde/unige-aai/tree/master/planners/LPG, n.d. Accessed: 2025-06-03.

[3] Enrico Scala. Enhsp planner on gitlab. https://gitlab.com/enricos83/ENHSP-Public, n.d. Accessed: 2025-06-03.