Interplanetary Museum Vault
Automated Planning: Theory and Practice
Academic Year 2025–2026

Jie Chen (mat. 256177)

University of Trento

jie.chen-2@studenti.unitn.it

1. Introduction

Automated planning is the process of using knowledge about the world, including possible actions and their results, to decide what to do and when in order to achieve an objective, before actually starting to do it. This capability is fundamental to enabling autonomous systems to operate effectively in complex, dynamic environments. From warehouse logistics to space exploration, planning systems must reason about actions, resources, and temporal constraints to generate feasible solutions. This report presents a comprehensive study of automated planning through the progressive implementation of planning models using PDDL (Planning Domain Definition Language), HDDL (Hierarchical Domain Definition Language), and their practical deployment within the ROS2-based PlanSys2 framework.

1.1. Report structure

This report is organized as follows: Section 2 describes the Interplanetary Museum Vault scenario, constraints, and assumptions. Sections 3 through 7 detail each problem's domain modeling, implementation, key design choices and results. Finally, Section 8 offers a discussion and conclusion for the project.

2. Interplanetary museum vault

2.1. Scenario description

The domain is a robotic artifact preservation system set in a subterranean museum vault on Mars. The environment consists of several specialized zones, each with distinct environmental properties and functional roles. Autonomous robotic curators are tasked with relocating fragile artifacts while adhering to physical, temporal, and resource-based constraints. The vault is structured into the following zones:

- Artifact Hall alpha: Stable display area.
- Artifact Hall beta: Temporarily unstable due to seismic activity; accessible only during safe windows.
- Cryo-Chamber: Low-temperature zone for temperature-sensitive artifacts.
- Anti-Vibration Pods (2 available): Used to transport delicate items safely.
- Maintenance Tunnel: Low-pressure area requiring robots to activate sealing mode.
- Stasis-Lab: Final secure storage with advanced preservation technology.

2.2. Constraints

*2.2.1. **Artifact Relocation Logic***

The following sequences define the required relocation paths for artifacts:

- Artifacts from Hall alpha: Hall alpha → Cryo-Chamber (with cooling) → Stasis-Lab.
- Artifacts from Hall beta: Hall beta → Anti-Vibration Pod → Stasis-Lab.

*2.2.2. **Physical & Environmental Constraints***

- Temperature limits: α artifacts moved to the Stasis-Lab must not exceed a temperature threshold.
- Seismic activity restricts access to Hall beta.
- Limited resources: Only two Pods are available.
- Carrying capacity: Robots can carry one artifact at a time (initially), later extended to multi-capacity.
- Movement restrictions: Robotic curator operates in sealed mode at all times while inside the tunnel

*2.2.3. **Temporal & Operational Constraints***

- Action durations (for temporal planning): Actions such as moving, picking, placing, cooling, and sealing have defined durations.
- Parallelism: Some actions cannot be executed concurrently by the same robot (e.g., picking and moving).
- Goal sequencing: Certain tasks must be performed in a specific order (e.g., artifacts from Hall beta must be placed in pods before transport).

2.3. Assumption

*2.3.1. **Modeling Assumptions***

- Single robot initially: Only one robotic curator is available in Problem 1, with extensions to multiple robots or varied capacities in later problems.
- Uniform artifact types: All artifacts are treated as identical in terms of physical handling; distinctions arise only from their origin location and required destination.
- Discrete time in temporal planning: In temporal planning (Problem 4), durative actions are modeled with fixed, predetermined durations rather than variable or continuous time.
- Deterministic actions: All actions have guaranteed outcomes with no probabilistic failures—movement always succeeds, artifacts never break, and environmental changes occur only as explicitly modeled.

- Seismic windows: Access to Hall beta is modeled as a boolean state (stable/unstable), rather than a continuous risk gradient or probabilistically varying threat level.

2.3.2. *Planning Assumptions*

- Conjunctive goals: Success requires all artifacts to reach their designated destinations; partial completion is not considered a valid solution.
- Renewable resources: Anti-Vibration Pods become available for reuse immediately after an artifact is delivered to the Stasis-Lab
- No breakdowns or failures: Robotic curators operate without malfunction; environmental conditions change only as modeled.

3. Problem 1

Problem 1 establishes the foundational planning scenario using a single robotic curator with a carrying capacity of one artifact. The domain models the core relocation workflows for two distinct artifact types under environmental constraints, using a classic STRIPS representation.

3.1. Domain

The domain uses STRIPS with typing and negative preconditions to represent the environment, robot capabilities, and procedural constraints for handling two distinct types of artifacts.

3.1.1. *Key Elements*

- Types: `robot`, `artifact`, `location`, `pod`
- Predicates:
  - Robot state: `(at ?r ?loc)`, `(carrying ?r ?a)`, `(empty-handed ?r)`, `(sealed ?r)`
  - Artifact state: `(at_artifact ?a ?loc)`, `(cooled ?a)`, `(antivibration_on ?a)`, `(ready_for_stasis ?a)`
  - Pod state: `(pod_at ?p ?loc)`, `(pod_free ?p)`, `(pod_occupied ?a ?p)`
- Environment: `(stable ?loc)`
- Artifact typing: `(is_alpha_art ?a)`, `(is_beta_art ?a)`

3.1.2. *Action Categories*

Movement & Sealing:

- `move`: Move the robot from a location to another location. Requires robot to be sealed and destination stable.
- `seal/unseal`: Toggle sealing mode.

Hall alpha workflow: pick_hall_alpha → drop_at_cryo → cool → pick_from_cryo

- `pick_hall_alpha`: pick the artifact from hall alpha
- `drop_at_cryo`: drop the artifact at cryo-chamber
- `cool`: cool the artifact for transportation
- `pick_from_cryo`: pick the artifact from cryo-chamber

Hall beta workflow: pick_hall_beta → drop_at_pod → activate_antivibration → pick_from_pod

- `pick_hall_beta`: pick the artifact from the hall beta
- `drop_at_pod`: drop the artifact to the pod area
- `activate_antivibration`: apply the antivibration to the artifact
- `pick_from_pod`: pick the artifact from the pod area

Final Delivery: `delive_to_stasis` deliver the artifact to the stasis-lab
Environmental Control: `stabilize` makes Hall beta accessible.

3.2. Problem

Two problem instances are defined:

- Problem 1 has a single robot curator with two artifacts: artifact_a (α-type), artifact_b (β-type). The goal is to deliver the artifact to the stasis-lab.
- Problem 2 has a single robot curator with six artifacts: 3 α-artifacts (artifact_a1, artifact_a2, artifact_a3), 3 β-artifacts (artifact_b1, artifact_b2, artifact_b3).

The goal is to deliver all the artifacts to the stasis-lab.

3.2.1. *Key Initial States*

- Robot starts `(at curator entrance)`, unsealed, empty-handed.
- All artifacts are in the respective halls.
- Hall beta is initially unstable.
- Pods are free and located in `pod_area`.
- No artifact is cooled, pod-protected, or ready for stasis.

3.3. Modeling Choices and Assumptions

- Single-carrying enforcement: The predicate `(empty-handed ?r)` ensures the robot can carry only one artifact at a time.
- Implicit tunnel representation: The Maintenance Tunnel is not modeled as a separate location. Movement between certain zones implicitly requires the robot to be sealed.
- Boolean stability for Hall beta: Accessibility is modeled as a discrete predicate `(stable hall_beta)`.
- Discrete cooling and anti-vibration: Cooling and anti-vibration are modeled as instant actions with no duration or continuous effects.
- Deterministic and sequential actions: No parallel execution or temporal durations are considered in this STRIPS formulation.

3.4. Results

3.4.1. *Problem 1*

This study (table 1) evaluates several classical planning configurations using the Fast Downward planning system. The planners tested include LAMA-first and LAMA-2011 (both portfolio-based planners combining landmark and FF heuristics), FF heuristic (used with eager greedy search), Additive heuristic (sum of costs), and LM-Cut (an optimal heuristic). Each configuration was applied to a small problem instance involving two artifacts to compare efficiency, plan quality, and search performance under unit-cost settings.

Key Observations:

Planner	Plan Length	Time (s)	Expanded States
LAMA-first	21	0.57	42
LAMA-2011	18	0.56	833
FF Greedy	18	0.54	27
Additive	19	0.55	32
LM-Cut	18	0.54	150

Table 1: *Problem 1_1: Single robot curator with two artifacts*

Planner	Plan Length	Time (s)	Expanded States
LAMA-first	53	0.58	209
FF Greedy	50	0.54	97
CG Heuristic	50	0.55	162

Table 2: *Problem 1_2: Single robot curator with six artifacts*

- FF greedy was the most efficient in terms of expansions and speed.
- LAMA-2011 successfully improved plans iteratively.
- LM-Cut confirmed the optimal plan length is 18 steps.
- All planners successfully solved the small problem under 0.6 seconds.

3.4.2. ***Problem 2***

The same planners (table 2) were tested on a larger problem instance involving six artifacts to evaluate scalability. All configurations successfully generated valid plans, though with notable differences in plan length and computational effort. The results indicate that while portfolio methods like LAMA remain robust, simpler greedy search with the FF heuristic maintains superior efficiency even as problem size increases. Key Observations:

- FF greedy remained the most efficient planner for the larger problem.
- LAMA-first produced a slightly longer plan but still solved quickly.
- The Causal Graph heuristic yielded a plan of equal length to FF but required more expansions.
- All planners scaled successfully to the 6-artifact problem, completing in under 0.6 seconds.

4. Problem 2

Problem 2 extends the basic scenario by introducing multiple robotic curators with distinct capabilities and capacities, moving beyond a single robot with a fixed carrying limit. The domain is enhanced with capacity tracking, typed robots, and resource-limited pods to model more realistic multi-agent planning.

4.1. Domain

New Requirements `:equality` is added to support robot type distinctions.

4.1.1. ***Key Extensions***

Robot Typing and Capacity:

Types: `alpha_robot` and `beta_robot`.

Capacity predicates:

- `capacity_level_0`, `capacity_level_1`, `capacity_level_2`, `capacity_level_3` track how many artifacts a robot is carrying.
- `max_capacity_1`, `max_capacity_2`, `max_capacity_3` define the robot's maximum load.
- `has_capacity` indicates whether the robot can pick up another artifact.

Pod Resource Management:

- `pod_slot_available` ensures no more than two pods are occupied simultaneously.
- `pods_used_0`, `pods_used_1`, `pods_used_2` track the number of pods in use.

4.1.2. ***Action Updates***

`pick_hall_alpha` and `pick_hall_beta` are restricted to `alpha_robot` and `beta_robot` respectively. All pick/-drop/deliver actions now update capacity levels and pod usage via conditional effects (when clauses).

4.2. Problem

Two problem instances are defined:

- Problem 1 has two different types of robot with different capacities: `alpha_bot_small` (α-robot, capacity = 1), `beta_bot_medium` (β-robot, capacity = 2)
- Problem 2 has two different types of robot with same capacities: `alpha_bot_large` (α-robot, capacity = 3), `beta_bot_large` (β-robot, capacity = 3)

Both problems have three artifact of each types: 3 α-artifacts, 3 β-artifacts. The goal is to deliver all artifacts to stasis-lab.

4.2.1. ***Key Initial States***

- Both robots start at entrance, unsealed.
- Capacity levels set to 0; `has_capacity` true.
- Pods are free and available.
- Hall beta is unstable initially.

4.3. Modeling Choices and Assumptions

- Robot specialization: `alpha-robots` can only pick α-artifacts; `beta-robots` only β-artifacts. This enforces role separation and reduces action branching.
- Discrete capacity levels: Capacity is modeled as discrete levels (0–3) rather than numeric variables. `has_capacity` abstracts whether a robot can pick another item.
- Pod counting via state predicates: Pod usage is tracked with `pods_used_*` predicates, ensuring 2 pods are occupied.
- Conditional effects for state updates: Capacity and pod states are updated using `when` to avoid enumerating all cases in separate actions.
- Sealing required for movement: The sealed precondition remains in move, though tunnel traversal is implicit.

Planner	Plan Length	Time (s)	Expanded States
LAMA-first	51	0.60	263
FF Greedy	52	0.59	78

Table 3: *Problem 2_1: Small capacity robots (α=1, β=2)*

Planner	Plan Length	Time (s)	Expanded States
LAMA-first	44	0.62	173
FF Greedy	43	0.57	102
CG Heuristic	51	0.58	410

Table 4: *Problem 2_2: Large capacity robots (α=3, β=3)*

4.4. Results

This study (table 3 and table 4) evaluates classical planning configurations in a multi-robot capacity domain using the Fast Downward planning system. The planners tested include LAMA-first (a portfolio-based planner combining landmark and FF heuristics) and FF heuristic (used with eager greedy search). Each configuration is applied to two problem variants with different robot carrying capacities. An additional Causal Graph (CG) heuristic is also tested in the large capacity scenario. The comparison assesses how heuristic selection and robot capacity influence coordination efficiency, plan length, and search performance in a collaborative multi-agent environment. Key Observations:

- Capacity matters: Larger robot capacities reduced plan length significantly (51→43 steps).
- FF greedy consistently showed low expansion counts and competitive plan quality.
- LAMA-first produced comparable plans but required more expansions.
- CG heuristic performed poorly with high expansions and longer plans.
- All planners successfully coordinated multiple robots with different capacities.

Both problem instances demonstrated that heuristic choice (FF vs. landmark-based) and robot capacity significantly impact plan efficiency and search performance in multi-robot coordination tasks.

5. Problem 3

Problem 3 reframes the artifact relocation scenario using Hierarchical Task Network (HTN) planning, shifting from flat action sequences to structured task decomposition. The domain introduces abstract tasks and decomposition methods, enabling more intuitive modeling of procedural workflows and enforcing structured execution order through hierarchical constraints.

5.1. Domain

Requirements `:typing`, `:hierachie` (hierarchical planning support)

*5.1.1. **Key Extensions***

Abstract Task Definitions:
High-level tasks:

- Root goal task: `achieve_all_artifacts_in_stasis`.
- Category-level tasks: `process_alpha_artifacts` / `process_beta_artifacts`.
- Per-artifact tasks: `process_alpha_artifact` / `process_beta_artifact`.

Tasks have parameters and preconditions but no direct effects; they are realized through method decomposition.

Method-Based Decomposition:

- `main_method`: Decomposes root task into parallel alpha and beta processing.
- `process_all_alpha` / `process_all_beta`: Sequence artifact tasks with ordering constraints.
- `process_single_alpha` / `process_single_beta`: Decompose per-artifact tasks into concrete action sequences with strict ordering.

Action Layer:

- Same low-level actions as Problem 2 (pick, drop, cool, move, etc.), but without capacity tracking.
- Actions are invoked as primitive tasks within methods.

Ordering Constraints:

- Strict linear ordering within each artifact workflow (e.g., seal → move → pick → move → drop → ...).
- Inter-task ordering: β-artifact processing requires prior stabilization of Hall beta.

5.2. Problem

Two types of robots with two artifacts for each artifact types:

- Robots: `alpha_bot` (α-robot), `beta_bot` (β-robot).
- Artifacts: 2 α-artifacts (artifact_a1, artifact_a2), 2 β-artifacts (artifact_b1, artifact_b2).

The goal is to deliver all the artifacts in the stasis lab. HTN goal: (`achieve_all_artifacts_in_stasis`).

*5.2.1. **Key Initial States***

- Both robots start at entrance.
- Artifacts in respective halls.
- Hall beta is initially unstable.
- Pods are available (`pod_slot_available`).

5.3. Modeling Choices and Assumptions

- Structured workflow enforcement: Tasks are decomposed in a fixed sequence, reducing planning search space.
- Robot specialization preserved: `alpha_robot` and `beta_robot` types restrict which robot can perform certain tasks. Specialized methods (`process_single_alpha` and `process_single_beta`) enforce role assignment.
- No capacity modeling: HTN domain omits capacity predicates (`empty-handed`, `capacity_level_*`), assuming implicit single-carrying per method.
- Deterministic task ordering: All subtasks are totally ordered within methods; no parallel execution is modeled. Inter-method ordering: β-processing requires prior stabilize.
- Simplified pod management: Pod usage is tracked; method preconditions ensure pod availability.

Phase	Artifacts Processed	Actions	Robot Used
Setup	-	1	-
α-Processing	a1, a2	20	alpha_bot
β-Processing	b1, b2	22	beta_bot
Transitions	-	6	Both
TOTAL	4 artifacts	49	

Table 5: *Problem 3 HTN: Plan structure*

Planner	Plan Len	Time (s)	Exp. States	Search Nodes
PANDA	49	0.013	549	243

Table 6: *Problem 3 HTN: Panda results*

5.4. Results

Problem 3 employs HTN planning using the PANDA planner, shifting from flat action sequences to structured task decomposition. See plan structure and result in the respective table 5 and table 6. Planning configuration:

- Planner: PANDA (Planning and Acting in a Network Decomposition Architecture)
- Search Algorithm: AStarActionsType (2.0)
- Heuristic: hhRC(hFF) - HTN heuristic with relaxed composition and hFF
- Abstract task selection: Random strategy

5.4.1. ***Comparison with Previous Approaches*** *(table 7)*

PANDA successfully solves the HTN planning problem in just 13 ms of search time, demonstrating the efficiency of hierarchical decomposition for structured domains. The resulting 49-step plan follows intuitive procedural workflows with guaranteed correctness through method constraints. However, the strict sequential execution and fixed decomposition patterns lead to inefficiencies compared to classical planning approaches:

- 49 steps vs potentially fewer in classical planning
- No parallelism despite two available robots
- Excessive repositioning (return to entrance after each artifact)

HTN planning excels when:

1. Domain knowledge dictates specific workflows
2. Correctness through structure is prioritized over optimality
3. Fast planning is critical
4. Human interpretability of plans is important

Plan Aspect	Problem 2 (Classical)	Problem 3 (HTN)
Paradigm	Flat state-space search	Hierarchical decomp.
Structure	Flexible, emergent	Structured, fixed
Parallelism	Possible if allowed	None (sequential)
Plan length	Typically shorter	Longer (49 vs $\tilde{3}$0-40)
Planning time	Depends on search	Very fast (13 ms)
Solution quality	Optimizes step count	Predefined workflow

Table 7: *Comparison table: Problem 2 vs Problem 3*

For this artifact relocation problem, HTN provides a correct but suboptimal solution that would benefit from incorporating more flexible decomposition methods or hybrid approaches that allow some parallelism while maintaining hierarchical structure.

6. Problem 4

Problem 4 extends the planning model to temporal planning with durative actions, introducing explicit action durations, timed preconditions and effects, and the possibility of parallel execution. The domain models realistic timing constraints for robotic operations while maintaining the artifact workflow structure, enabling the planner to reason about concurrency and total plan makespan.

6.1. Domain

Requirements `:strips`, `:typing`, `:durative-actions`

6.1.1. ***Key Extensions***

- Durative action: All actions are redefined as durative actions with explicit `:duration` clauses.
- Timed conditions and effects:Preconditions and effects are tagged with temporal qualifiers:
  - at start: must hold at action start.
  - at end: takes effect at action completion.
- Location typing forsSafety: Location-type predicates (`is_hall_alpha`, `is_cryo_chamber`, etc.) are introduced to enforce that actions occur in correct zones. Prevents logical errors like cooling outside the cryo chamber.
- Mutually exclusive state predicates: States like `sealed` / `unsealed`, `cooled` / `uncooled`, `stable` / `unstable` are modeled as complementary pairs to simplify temporal reasoning.
- Parallel execution support: Actions can overlap if their preconditions allow, enabling concurrent robot operations.

6.2. Problem

Two types of robots with three artifacts for each artifact types:

- Robots: `robot_alpha`, `robot_beta`.
- Artifacts: 3 α-artifacts (`artifact_a1-a3`), 3 β-artifacts (`artifact_b1-b3`).

The goal is to deliver all the artifacts in the stasis-lab aiming to minimize the total plan makespan (plan completion time).

6.2.1. ***Temporal Initial State***

- Robots start unsealed at entrance.
- All location types are declared.
- Hall beta is initially unstable.
- Pods are free (`pod_free`).
- Goal: All artifacts in `stasis_lab`.

Planner	Makespan	Steps	States Eval.	Time (s)
POPF	76.022	44	155	0.16
TDF	141.481	52	177	0.03
OPTIC	76.022	44	155	0.22
OPTIC-SAT	76.022	44	155	0.21
OPTIC-Graph	76.022	44	155	0.21

Table 8: *Problem 4 results*

6.3. Modeling Choices and Assumptions

- Durations: Fixed, reasonable duration with no variable durations or resource-dependent timing. Durations are arbitrary but realistic:
  - Sealing / stabilize: 2 units
  - Pick: 3 units
  - Movement: 4 units
  - Drop: 5 units
  - Cooling Anti-vibration activation: 6 units
- No capacity constraints: Capacity tracking from Problem 2 is omitted; each robot can carry only one artifact implicitly via carrying predicate.
- Parallelism: Robots can act concurrently if preconditions permit, but the planner may serialize if beneficial.

6.4. Results

This study (table 8) evaluates three temporal planners: POPF (Partial Order Planning Forward), TDF (Temporal Fast Downward), and OPTIC (Optimizing Preferences and Timing Constraints). Each employs different heuristic approaches to navigate the expanded state space created by temporal constraints. POPF and OPTIC utilize admissible temporal heuristics derived from relaxed problem graphs, while TDF employs causal graph heuristics with preferred operators. These heuristic methods guide search toward solutions that not only satisfy all logical constraints but also optimize temporal efficiency through parallel execution, balancing exploration of concurrent actions with the computational cost of reasoning about time.

*6.4.1. **Key Observations***

- Effective parallelism: POPF and OPTIC successfully exploit concurrency. Both robots act simultaneously where preconditions allow. Overlapping movements and operations reduce total makespan.
- Makespan ,inimization: Optimal makespan of 76.022 achieved. Represents the fastest possible completion time given action durations.
- TDF limitations: TDF produces a suboptimal sequential plan. Processes all β-artifacts before starting α-artifacts. Results in 85.7% longer makespan (141.481 vs 76.022). Demonstrates the importance of search strategy in temporal planning.

The temporal extension successfully models realistic robotic operations with timing constraints while maintaining the artifact workflow structure from previous problems. The planners correctly reason about concurrency, producing efficient schedules that minimize total completion time. Results how that for temporal planning problems where makespan minimization is critical, POPF or OPTIC with makespan optimization should be preferred over TDF due to their superior parallel execution capabilities.

7. Problem 5

Problem 5 integrates the temporal planning model into the PlanSys2 (Planning System 2) framework, a ROS2-based planning and execution infrastructure. This final step demonstrates the deployment of the museum vault planning domain in a realistic robotic execution environment, where abstract PDDL actions are mapped to executable ROS2 nodes with simulated execution.

7.1. Domain Adaptation for PlanSys2

*7.1.1. **Key Modifications from Problem 4***

- Enhanced requirements: added `:adl` and `:fluents` to support PlanSys2's advanced features.
- Explicit connectivity graph: Introduced `(connected ?from ?to)` predicate to explicitly model traversable paths between locations, replacing implicit movement assumptions.
- Temporal action refinements:
  - All durative actions maintained from Problem 4.
  - Modified effect ordering for PlanSys2 compatibility (start effects before end effects).
  - Added connected precondition to move action.

*7.1.2. **PlanSys2-Compatible Domain Features***

- No capacity constraints: Simplified to single-carrying without explicit capacity predicates.
- Location typing preserved: All `is_*` predicates retained for action safety.
- Complementary state pairs: `sealed` / `unsealed`, `cooled` / `uncooled`, etc., for clean state transitions.

7.2. ROS2 Package Implementation

Package Structure (problem5):

```
problem5/
|-- launch/
|   -- problem5_launch.py
|-- pddl/
|   -- domain.pddl
|-- src/
|   |-- move_action_node.cpp
|   |-- seal_action_node.cpp
|   |-- unseal_action_node.cpp
|   |-- pick_hall_alpha_action_node.cpp
|   |-- pick_hall_beta_action_node.cpp
|   |-- drop_at_cryo_action_node.cpp
|   |-- drop_at_pod_action_node.cpp
|   |-- cool_artifact_action_node.cpp
|   |-- activate_antivibration_action_node.cpp
|   |-- pick_from_cryo_action_node.cpp
```

```
|   |-- pick_from_pod_action_node.cpp
|   |-- deliver_to_stasis_action_node.cpp
|   |-- stabilize_action_node.cpp
|-- CMakeLists.txt
|-- package.xml
|-- tmux_session.sh
```

*7.2.1. **Key Implementation Components***

The PlanSys2 integration was implemented through a structured ROS2 package containing several key components. The launch configuration, defined in `problem5_launch.py`, initializes the PlanSys2 monolithic bringup with the PDDL domain and launches thirteen distinct action executor nodes—one for each action type—while configuring the namespace and debugging environment for robust execution. Each action executor node, such as `move_action_node.cpp`, extends the `plansys2::ActionExecutorClient` base class to implement simulated execution with real-time progress feedback, utilizing a 250ms update rate to emulate realistic operational timing. The build system, comprising `CMakeLists.txt` and `package.xml`, follows standard ROS2 `ament_cmake` conventions to compile all action nodes as separate executables and declares dependencies on essential PlanSys2 packages including `plansys2_msgs` and `plansys2_executor`. Finally, an automation script, `tmux_session.sh`, orchestrates the entire workflow by setting up a `tmux` session that simultaneously runs the PlanSys2 launcher and terminal, automating the build, sourcing, and launch processes to streamline testing and demonstration.

7.3. Execution Workflow

1. System Initialization

```
./tmux_session.sh
```

Builds ROS2 workspace and launches PlanSys2.

2. Problem Setup via PlanSys2 Terminal

```
# Set instances
set instance bot_alpha alpha_robot
set instance bot_beta beta_robot
...

# Set initial predicates
set predicate (at_robot bot_alpha entrance)
set predicate (connected entr hall_alpha)
...

# Set goal
set goal (and
   (at_artifact artifact_a1 stasis_lab)
   (at_artifact artifact_b1 stasis_lab))
```

3. Planning and Execution

```
get plan # Generate plan
run # Execute via action nodes
```

7.4. Modeling Choices and Assumptions

Several modeling choices were made to facilitate the PlanSys2 integration and ensure reliable execution. The domain was simplified for practical deployment by omitting explicit capacity tracking, implicitly enforcing single-carrying per robot, and not enforcing parallel execution, opting instead for sequential action processing to reduce complexity. Fake actions were implemented to simulate real robotic operations without requiring physical hardware. Explicit connectivity was introduced through dedicated connected predicates that define the traversable movement graph, with bidirectional connections manually declared in the initial state to ensure complete navigability. Action durations were simulated within ROS2 nodes using a progress-bar mechanism that increments from 0–100% over time, aligning simulated execution time with the durative values specified in the PDDL model. Finally, a centralized PlanSys2 control architecture was adopted, wherein the PlanSys2 executor monitors execution, manages predicate updates, and handles replanning if necessary, while individual action nodes report success or failure back to the executor to maintain state consistency.

7.5. Integration Challenges and Solutions

Several technical challenges arose during the PlanSys2 integration, each addressed with specific solutions. The mapping from PDDL actions to ROS2 components was resolved by implementing a one-to-one correspondence where each action type corresponds to a dedicated ROS2 node. State synchronization between the planning domain and runtime environment was managed by PlanSys2's executor, which centrally maintains predicate truth and updates the world model as actions complete. Execution monitoring was enabled through feedback mechanisms where each action node reports progress and completion status back to the PlanSys2 executor, allowing for real-time oversight and potential replanning. To manage launch complexity, a single comprehensive launch file was created to start all thirteen action nodes alongside the PlanSys2 core, simplifying deployment and ensuring consistent initialization. Finally, debugging and operational visibility were enhanced through a custom tmux script that provides separate terminal panes for the PlanSys2 launcher and the interactive terminal, facilitating concurrent monitoring and command input during execution.

7.6. Results

The PlanSys2 integration successfully generated and executed a temporal plan for relocating both α and β artifacts. Real-time progress visualization via terminal feedback validated the end-to-end workflow from planning to simulated execution, establishing a foundation for potential hardware deployment. Key results:

- Plan Generated: 18-action temporal plan created
- Plan Executed: All actions completed successfully
- Makespan: 30.008 simulated time units
- Coordination: Both robots worked where possible

The generated plan (Figure 1) shows efficient coordination:

- Parallel operations: Multiple actions execute concurrently (e.g., `robot_alpha` picks artifact while `robot_beta` moves)
- Constraint adherence: All workflow rules followed (α-artifacts cooled, β-artifacts use pods)
- Temporal accuracy: Action durations matched PDDL specifications exactly

```
> get plan
plan:
0:      (move robot_alpha entrance hall_alpha)  [4]
0:      (stabilize hall_beta)   [2]
2.001:  (move robot_beta entrance hall_beta)    [4]
4.001:  (pick_hall_alpha robot_alpha artifact_a hall_alpha)     [3]
4.002:  (move robot_alpha hall_alpha cryo_chamber)      [4]
6.002:  (pick_hall_beta robot_beta artifact_b hall_beta)        [3]
6.003:  (move robot_beta hall_beta pod_area)    [4]
8.003:  (drop_at_cryo robot_alpha artifact_a cryo_chamber)      [5]
10.004: (drop_at_pod robot_beta artifact_b pod1 pod_area)       [5]
13.004: (cool_artifact robot_alpha artifact_a cryo_chamber)     [6]
15.005: (activate_antivibration robot_beta artifact_b pod1 pod_area)    [6]
19.005: (pick_from_cryo robot_alpha artifact_a cryo_chamber)    [3]
19.006: (move robot_alpha cryo_chamber stasis_lab)      [4]
21.006: (pick_from_pod robot_beta artifact_b pod1 pod_area)     [3]
21.007: (move robot_beta pod_area stasis_lab)   [4]
23.007: (deliver_to_stasis robot_alpha artifact_a stasis_lab)   [5]
25.008: (deliver_to_stasis robot_beta artifact_b stasis_lab)    [5]
```

Figure 1: *Problem 5:* `get plan`

```
move in progress... [100%]
stabilize in progress... [100%]
pick_hall_alpha in progress... [100%]
move in progress... [100%]
move in progress... [100%]
pick_hall_beta in progress... [100%]
drop_at_cryo in progress... [100%]
move in progress... [100%]
cool_artifact in progress... [100%]
drop_at_pod in progress... [100%]
pick_from_cryo in progress... [100%]
activate_antivibration in progress... [100%]
move in progress... [100%]
pick_from_pod in progress... [100%]
deliver_to_stasis in progress... [100%]
move in progress... [100%]
deliver_to_stasis in progress... [100%]
[plansys2_node-1] [INFO] [1768322257.964625878] [executor]: Plan Succeeded
```

Figure 2: *Problem 5:* `run`

During execution (Figure 2), the system provided real-time feedback:

- Each action showed progress from 0% to 100%
- All 18 actions completed without errors
- Final confirmation: `[INFO] [executor]: Plan Succeeded`

8. Discussion and Conclusion

This project evaluated five distinct planning approaches for the Interplanetary Museum Vault scenario, each with different strengths and trade-offs.

- Classical Planning (Problems 1–2) proved effective and efficient, though its complexity increases with multi-robot coordination.
- HTN Planning (Problem 3) offered the fastest planning time and clear interpretability, but produced longer plans without leveraging available parallelism.
- Temporal Planning (Problem 4) enabled concurrent robot operations, reducing plan makespan, though this was dependent on the choice of planner (POPF/OPTIC).
- PlanSys2 Integration (Problem 5) successfully demonstrated that the planning models can be deployed in a robotic execution environment, achieving complete execution success.

Overall, the project successfully demonstrated a complete workflow from planning theory to a functional robotic simulation. The main findings are:

1. No single planner is universally optimal; each addresses different planning needs and constraints.
2. Transitioning from theoretical planning to practical execution often requires simplifying certain model features.
3. Careful selection of planners and domain modeling significantly influences performance and feasibility.

This work illustrates how automated planning can be applied to control robots in real-world tasks, such as the autonomous handling of artifacts within a museum vault.

9. References

[1] McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., & Wilkins, D. (1998). PDDL—The Planning Domain Definition Language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.

[2] Höller, D., Behnke, G., Bercher, P., Biundo, S., Fiorino, H., Pellier, D., & Alford, R. (2020). HDDL: An extension to PDDL for expressing hierarchical planning problems. *Proceedings of the AAAI Conference on Artificial Intelligence, 34*(06), 9883-9891.

[3] Helmert, M. (2006). The Fast Downward planning system. *Journal of Artificial Intelligence Research, 26*, 191-246.

[4] Richter, S., & Westphal, M. (2010). The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research, 39*, 127-177.

[5] Hoffmann, J., & Nebel, B. (2001). The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research, 14*, 253-302.

[6] Helmert, M., & Domshlak, C. (2009). Landmarks, critical paths and abstractions: What's the difference anyway? *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 162-169.

[7] Bercher, P., Keen, S., & Biundo, S. (2014). Hybrid planning heuristics based on task decomposition graphs. *Proceedings of the Seventh Annual Symposium on Combinatorial Search (SoCS)*, 35-43.

[8] Coles, A., Coles, A., Fox, M., & Long, D. (2010). Forward-chaining partial-order planning. *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 42-49.

[9] Eyerich, P., Mattmüller, R., & Röger, G. (2009). Using the context-enhanced additive heuristic for temporal and numeric planning. *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 130-137.

[10] Benton, J., Coles, A., & Coles, A. (2012). Temporal planning with preferences and time-dependent continuous costs. *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2-10.

[11] Macías, S., Borrajo, D., & Castillo, L. (2021). PlanSys2: A planning system framework for ROS2. *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 9742-9749.

[12] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., & Ng, A. Y. (2009). ROS: An open-source Robot Operating System. *ICRA Workshop on Open Source Software, 3*(3.2), 5.