ROS2 PlanSys2 and Task Planning for Multiple Robots:

Preliminary Research Toward DEL-based Task Planning with EPDDL for Multi-Agent Robotics in Uncertain Environments

Preliminary Research Document

Preliminary research in the context of epistemic multi-agent planning, grounded in Dynamic Epistemic Logic and the EPDDL framework

Research Area: Autonomous Robotics & Epistemic Planning
Keywords: ROS2, PlanSys2, PDDL, DEL, EPDDL, Multi-Robot Systems
Document Type: Preliminary Research / State of the Art
Version: 1.1

Haniel Ulises Vásquez Morales

This document is a preliminary academic survey compiled in preparation for research on DEL-based epistemic task planning using EPDDL for multi-agent robotic systems operating under partial observability. It is not a finished publication.

Abstract

The increasing deployment of autonomous robotic systems in complex, uncertain, and partially observable environments demands planning formalisms that transcend the capabilities of classical automated planning. This document constitutes a preliminary research survey covering two major pillars of the target research agenda: (i) the state of the art in robot task planning using the Robot Operating System 2 (ROS2), with particular focus on the PlanSys2 framework and its PDDL-based planning infrastructure; and (ii) the theoretical and conceptual foundations of Dynamic Epistemic Logic (DEL) and the Epistemic Planning Domain Definition Language (EPDDL), as introduced by Burigana and Fabiano for IPC 2026. The ultimate motivation of this survey is to chart a research path toward a DEL-based multi-agent task planning system for heterogeneous robot fleets operating under epistemic uncertainty. We discuss the architecture of ROS2 multi-robot systems, the structure and limitations of existing PDDL-based planners in robotics, the semantics of epistemic states and actions in DEL, and the syntactic mechanisms of EPDDL. Special attention is given to recent applied work applying DEL to heterogeneous robotic fleets under communication restrictions [15–17], and to the foundational treatment of DEL-based planning problems [18]. We conclude with an analysis of the integration challenges and open research questions at the intersection of these fields.

Keywords: ROS2, PlanSys2, PDDL, task planning, multi-robot systems, Dynamic Epistemic Logic, epistemic planning, EPDDL, partial observability, multi-agent systems, active inference, theory of mind.

Contents

1 Introduction

Modern autonomous robotic systems are increasingly expected to operate in environments that are shared with other agents whether human operators, other autonomous robots, or dynamically changing physical conditions where the state of the world is not fully and reliably observable. A warehouse robot fleet, a team of search-and-rescue drones, or a collaborative assembly line all require their constituent agents to reason not only about what is true in the world, but also about what each agent *knows*, what each agent *believes*, and what each agent knows or believes about the epistemic states of others. Classical task planning formalisms, such as PDDL (Planning Domain Definition Language), remain the de facto standard in practical robotic systems and are supported by major middleware frameworks such as ROS2 through tools like PlanSys2. However, classical planning fundamentally assumes a fully observable, deterministic environment and a single omniscient planning agent assumptions that are routinely violated in realistic multi-robot deployments.

The objective of this document is to establish a comprehensive preliminary understanding of two converging research areas: the engineering and middleware infrastructure for task planning in ROS2 multi-robot systems, and the theoretical framework provided by Dynamic Epistemic Logic (DEL) and its associated language, EPDDL, for epistemic multi-agent planning. Together, these two pillars form the foundation upon which a future system can be built: one capable of reasoning about the knowledge and beliefs of individual robots, performing multi-agent coordination under partial observability, and expressing rich epistemic goals such as "robot A knows whether corridor c is blocked," or "it is common knowledge among all robots that the package has been delivered."

Section 2 introduces ROS2 as a middleware architecture, covering its communication model, multi-robot support, and lifecycle management. Section 3 examines PlanSys2, the primary task planning framework for ROS2, including its PDDL integration, executor architecture, and a concrete worked example using ROS2 Humble/Jazzy C++. Section 4 surveys the state of the art in multi-robot coordination and planning within ROS2. Section 5 provides an accessible but technically grounded introduction to Dynamic Epistemic Logic, including its application to robotics [15–18]. Section 6 presents EPDDL, its syntax, and action type libraries. Section 7 analyzes the integration challenges and proposes a conceptual architecture for a DEL-based ROS2 planning system. Section 8 presents empirical benchmark results from the companion planner implementation. Section 9 summarizes findings and outlines open research questions.

2 ROS2: Architecture and Multi-Robot Infrastructure

2.1 Overview and Design Principles

The Robot Operating System 2 (ROS2) is an open-source middleware framework designed to provide a flexible and modular infrastructure for robotic software development [1]. Unlike its predecessor ROS1, which relied on a centralized master node, ROS2 is built atop the Data Distribution Service (DDS) standard, providing a fully decentralized, quality-of-service-aware publish-subscribe communication layer. This architectural shift makes ROS2 fundamentally more suited to multi-robot deployments, real-time systems, and industrial applications.

The core communication abstractions in ROS2 are *nodes*, *topics*, *services*, and *actions*. Topics implement an asynchronous, many-to-many publish-subscribe pattern. Services implement synchronous request-reply communication. Actions extend services with long-running asynchronous goals, periodic feedback, and cancellation.

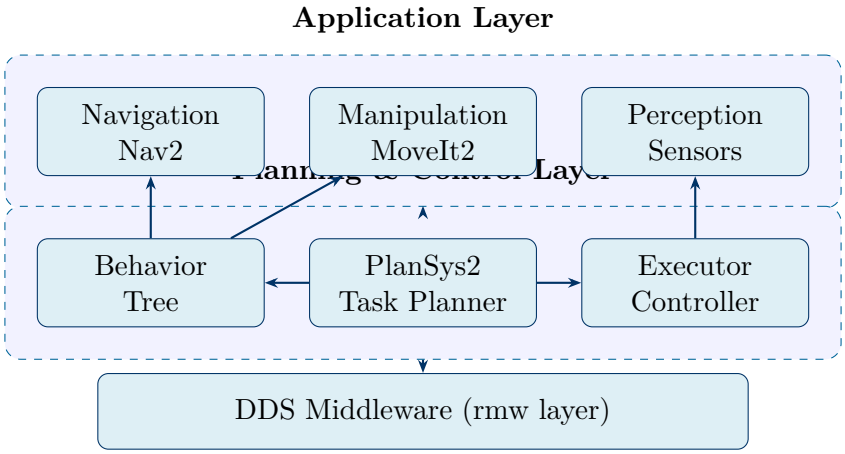

Figure 1: Simplified layered architecture of a ROS2 robot system. PlanSys2 operates within the Planning & Control Layer, interfacing with navigation, manipulation, and perception via behavior trees and action servers, all underpinned by DDS.

2.2 DDS and the Communication Model

DDS implements a data-centric publish-subscribe model where data flows between publishers and subscribers through a shared data space the *Global Data Space* without a centralized broker. Quality of Service (QoS) policies govern reliability, durability, deadline, and liveness. For multi-robot systems, DDS offers native distributed discovery: robots on the same network segment automatically discover each other without manual configuration, mediated through the *DDS Domain* abstraction that partitions communication into independent contexts.

2.3 Node Lifecycle and Managed Nodes

ROS2 introduced a formal node lifecycle management protocol with states (Unconfigured, Inactive, Active, Finalized) and defined transitions. From a task planning perspective, the lifecycle model provides a principled mechanism for activating and deactivating capabilities on demand, enabling a planning system to activate a navigation capability only when an action requiring it is dispatched.

2.4 Multi-Robot Namespacing and Coordination

ROS2 addresses multi-robot naming through a hierarchical namespace mechanism. Each robot is assigned a unique namespace prefix (e.g., `/robot1`, `/robot2`), and all its nodes, topics, services, and actions are scoped under that prefix. This allows multiple robot instances to coexist on the same DDS domain without topic collisions and enables a centralized planning node to address individual robots by namespace. Namespacing alone is, however, insufficient for true multi-robot coordination, which additionally requires shared state representation, inter-robot protocols, and synchronized action dispatch.

2.5 ROS2 Actions and Task Execution

The ROS2 action protocol is the primary mechanism for integrating task planning with robot execution. An action server accepts goal requests from action clients, provides periodic feedback, and returns a final result. This maps directly to the classical planning loop: the planner emits a sequence of actions, each dispatched as a goal to the corresponding action server, with execution monitored by the planner or a behavior executive. Action state management (ACCEPTED, EXECUTING, CANCELING, SUCCEEDED, ABORTED, CANCELED) is handled transparently by the ROS2 action infrastructure.

3 PlanSys2: Task Planning in ROS2

3.1 Architecture Overview

PlanSys2 is the primary open-source framework for PDDL-based task planning in ROS2 [2]. It provides a modular, multi-node architecture integrating domain and problem management, plan generation via external PDDL solvers, and plan execution via a behavior-tree-based action dispatcher.

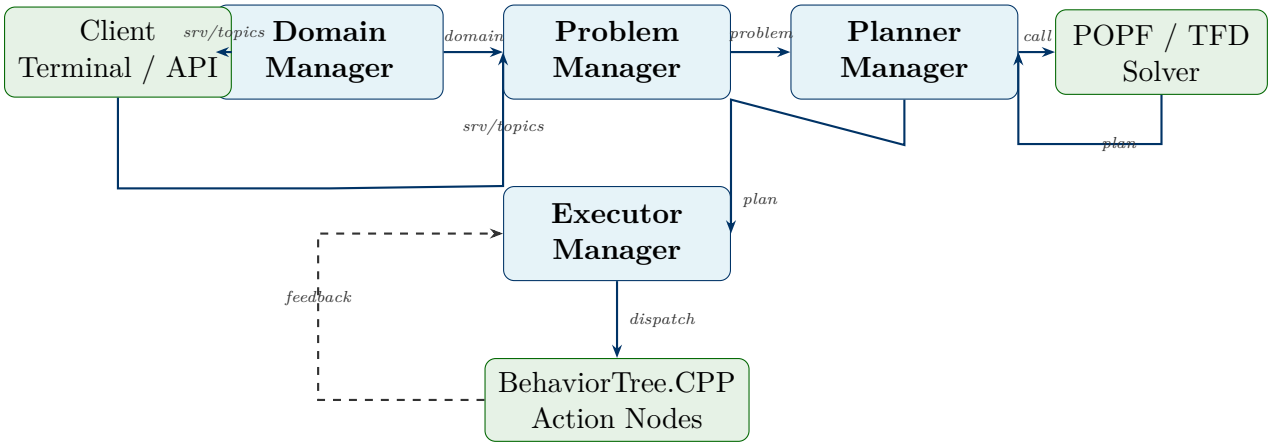

Figure 2: Internal architecture of PlanSys2. Domain Manager, Problem Manager, Planner Manager, and Executor Manager are separate ROS2 nodes. An external solver (POPF/TFD) is called by the Planner Manager, and execution is carried out through BehaviorTree.CPP action nodes.

PlanSys2 decouples domain management, problem management, planning, and execution into separate ROS2 nodes following the microservice principle. Each component can be independently replaced, updated, or scaled.

3.2 Domain and Problem Management

The Domain Manager maintains the PDDL domain specification. The Problem Manager maintains the dynamic planning problem the current set of ground predicates (the world state), instantiated objects, and the current goal specification and is the primary interface through which the robot system updates its world model as perception events are received. Both managers expose ROS2 services and topics, allowing any node to add/remove predicates, objects, and update goals.

3.3 Plan Generation and PDDL Solvers

PlanSys2 delegates planning to external PDDL planners via a plugin architecture. The most commonly used solvers are POPF [7] and TFD (Temporal Fast Downward), both supporting temporal PDDL with durative actions. The planning loop is triggered by a goal

update or world state change that renders the current plan invalid: the system serializes the domain and problem, calls the solver, receives the plan, and passes it to the Executor Manager.

3.4 Execution via Behavior Trees

Plan execution in PlanSys2 is mediated by the BehaviorTree.CPP library [8]. Each PDDL action is associated with a behavior tree node implementing execution logic, pre-condition verification, and post-condition assertion. The Executor Manager constructs a behavior tree from the plan and runs it, monitoring each action for success, failure, or timeout. The behavior tree is constructed statically from the plan; replanning must be explicitly triggered when execution deviates from expectations.

3.5 A Practical PlanSys2 Task: Package Transport in ROS2 Humble/Jazzy

To ground the preceding architectural description concretely, we walk through a complete PlanSys2 task package for ROS2 Humble (or Jazzy) targeting a logistics scenario: a single mobile robot must transport two packages between three named locations in a warehouse. We cover the scenario, the package structure, the PDDL domain and problem files, the C++ action executor implementations, the CMake build configuration, the launch file, and the expected execution behavior illustrated with diagrams.

3.5.1 Scenario Description

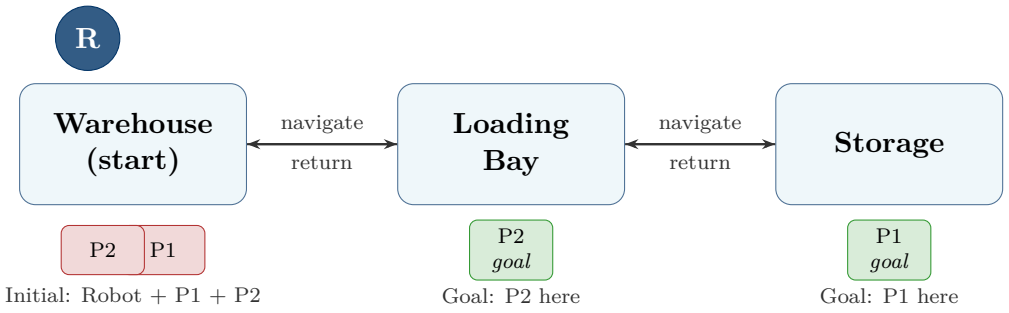

Figure 3: Warehouse scenario: the robot (R) starts at the Warehouse; P1 and P2 are both at the Warehouse. The goal is to deliver P1 to Storage and P2 to the Loading Bay.

The environment has three locations `warehouse`, `loading_bay`, and `storage` connected bidirectionally. The robot begins at the warehouse. Both packages begin at the warehouse. The plan requires picking up each package, navigating to its destination, dropping it off, and returning to the start.

3.5.2 Package Structure

The task is implemented as a standard ROS2 package (compatible with both Humble and Jazzy). The directory layout is as follows.

```
transport_task/
|-- CMakeLists.txt
|-- package.xml
|-- pddl/
|   |-- transport_domain.pddl
|   `-- transport_problem.pddl
|-- src/
|   |-- navigate_action_node.cpp
|   |-- pick_up_action_node.cpp
|   `-- drop_off_action_node.cpp
|-- launch/
|   `-- transport_task.launch.py
`-- config/
    `-- plansys2_params.yaml
```

Listing 1: Package directory structure for the transport task.

3.5.3 PDDL Domain

```
(define (domain transport)
  (:requirements :strips :typing :durative-actions :negative-
     preconditions)

  (:types
    robot location package
  )

  (:predicates
    (robot-at ?r - robot ?l - location)
    (package-at ?p - package ?l - location)
    (carrying ?r - robot ?p - package)
    (free ?r - robot)
    (connected ?from - location ?to - location)
  )

  (:durative-action navigate
    :parameters (?r - robot ?from - location ?to - location)
    :duration (= ?duration 10)
    :condition (and
      (at start (robot-at ?r ?from))
      (at start (connected ?from ?to))
      (over all (free ?r))
    )
```

```
    :effect (and
      (at start (not (robot-at ?r ?from)))
      (at end   (robot-at ?r ?to))
    )
  )

  (:durative-action pick-up
    :parameters (?r - robot ?p - package ?l - location)
    :duration (= ?duration 5)
    :condition (and
      (at start (robot-at ?r ?l))
      (at start (package-at ?p ?l))
      (at start (free ?r))
    )
    :effect (and
      (at start (not (package-at ?p ?l)))
      (at start (not (free ?r)))
      (at end   (carrying ?r ?p))
    )
  )

  (:durative-action drop-off
    :parameters (?r - robot ?p - package ?l - location)
    :duration (= ?duration 5)
    :condition (and
      (at start (robot-at ?r ?l))
      (at start (carrying ?r ?p))
    )
    :effect (and
      (at start (not (carrying ?r ?p)))
      (at end   (package-at ?p ?l))
      (at end   (free ?r))
    )
  )
)
```

Listing 2: PDDL domain for the warehouse transport task (`pddl/transport_domain.pddl`).

3.5.4 PDDL Problem

```
(define (problem transport-1)
  (:domain transport)

  (:objects
    robot1                          - robot
    warehouse loading-bay storage   - location
```

```
    parcel1 parcel2                    - package
  )

  (:init
    (robot-at robot1 warehouse)
    (package-at parcel1 warehouse)
    (package-at parcel2 warehouse)
    (free robot1)
    (connected warehouse loading-bay)
    (connected loading-bay warehouse)
    (connected loading-bay storage)
    (connected storage loading-bay)
  )

  (:goal (and
    (package-at parcel1 storage)
    (package-at parcel2 loading-bay)
    (robot-at robot1 warehouse)
  ))
)
```

Listing 3: PDDL problem instance (`pddl/transport_problem.pddl`).

3.5.5 C++ Action Executors

Each durative action in the PDDL domain corresponds to a C++ class implementing `plansys2::ActionExecutorClient`. The following listings show the three executors. All three follow the same pattern: on activation they simulate the action with a progress timer, and on success they call `finish(true, 1.0, "succeeded")`. In a physical deployment the timer would be replaced by actual hardware calls (Nav2 goals, gripper commands, etc.).

```cpp
#include <memory>
#include <string>
#include "plansys2_executor/ActionExecutorClient.hpp"
#include "rclcpp/rclcpp.hpp"
#include "rclcpp_lifecycle/lifecycle_node.hpp"

using namespace std::chrono_literals;

class NavigateAction : public plansys2::ActionExecutorClient
{
public:
  NavigateAction() : plansys2::ActionExecutorClient("navigate", 250ms)
  {
    progress_ = 0.0f;
```

```
  }

private:
  void do_work() override
  {
    if (progress_ < 1.0f) {
      progress_ += 0.05f;
      send_feedback(progress_, "navigating...");
    } else {
      finish(true, 1.0f, "navigation completed");
      progress_ = 0.0f;
    }
  }

  float progress_;
};

int main(int argc, char ** argv)
{
  rclcpp::init(argc, argv);
  auto node = std::make_shared<NavigateAction>();
  node->set_parameter(rclcpp::Parameter("action_name", "navigate"));
  node->trigger_transition(
    lifecycle_msgs::msg::Transition::TRANSITION_CONFIGURE);
  node->trigger_transition(
    lifecycle_msgs::msg::Transition::TRANSITION_ACTIVATE);
  rclcpp::spin(node->get_node_base_interface());
  rclcpp::shutdown();
  return 0;
}
```

Listing 4: Navigate action executor (`src/navigate_action_node.cpp`).

```
#include <memory>
#include <string>
#include "plansys2_executor/ActionExecutorClient.hpp"
#include "rclcpp/rclcpp.hpp"
#include "rclcpp_lifecycle/lifecycle_node.hpp"

using namespace std::chrono_literals;

class PickUpAction : public plansys2::ActionExecutorClient
{
public:
  PickUpAction() : plansys2::ActionExecutorClient("pick-up", 250ms)
  {
    progress_ = 0.0f;
  }
```

```

private:
  void do_work() override
  {
    if (progress_ < 1.0f) {
      progress_ += 0.10f;
      send_feedback(progress_, "picking up package...");
    } else {
      finish(true, 1.0f, "pick-up completed");
      progress_ = 0.0f;
    }
  }

  float progress_;
};

int main(int argc, char ** argv)
{
  rclcpp::init(argc, argv);
  auto node = std::make_shared<PickUpAction>();
  node->set_parameter(rclcpp::Parameter("action_name", "pick-up"));
  node->trigger_transition(
    lifecycle_msgs::msg::Transition::TRANSITION_CONFIGURE);
  node->trigger_transition(
    lifecycle_msgs::msg::Transition::TRANSITION_ACTIVATE);
  rclcpp::spin(node->get_node_base_interface());
  rclcpp::shutdown();
  return 0;
}
```

Listing 5: Pick-up action executor (`src/pick_up_action_node.cpp`).

```
#include <memory>
#include <string>
#include "plansys2_executor/ActionExecutorClient.hpp"
#include "rclcpp/rclcpp.hpp"
#include "rclcpp_lifecycle/lifecycle_node.hpp"

using namespace std::chrono_literals;

class DropOffAction : public plansys2::ActionExecutorClient
{
public:
  DropOffAction() : plansys2::ActionExecutorClient("drop-off", 250ms)
  {
    progress_ = 0.0f;
  }

```

```
private:
  void do_work() override
  {
    if (progress_ < 1.0f) {
      progress_ += 0.10f;
      send_feedback(progress_, "dropping off package...");
    } else {
      finish(true, 1.0f, "drop-off completed");
      progress_ = 0.0f;
    }
  }

  float progress_;
};

int main(int argc, char ** argv)
{
  rclcpp::init(argc, argv);
  auto node = std::make_shared<DropOffAction>();
  node->set_parameter(rclcpp::Parameter("action_name", "drop-off"));
  node->trigger_transition(
    lifecycle_msgs::msg::Transition::TRANSITION_CONFIGURE);
  node->trigger_transition(
    lifecycle_msgs::msg::Transition::TRANSITION_ACTIVATE);
  rclcpp::spin(node->get_node_base_interface());
  rclcpp::shutdown();
  return 0;
}
```

Listing 6: Drop-off action executor (`src/drop_off_action_node.cpp`).

3.5.6 CMakeLists.txt

```
cmake_minimum_required(VERSION 3.8)
project(transport_task)

find_package(ament_cmake REQUIRED)
find_package(rclcpp REQUIRED)
find_package(rclcpp_lifecycle REQUIRED)
find_package(lifecycle_msgs REQUIRED)
find_package(plansys2_executor REQUIRED)

set(DEPS
  rclcpp
  rclcpp_lifecycle
  lifecycle_msgs
  plansys2_executor
```

```
)

add_executable(navigate_action_node src/navigate_action_node.cpp)
ament_target_dependencies(navigate_action_node ${DEPS})

add_executable(pick_up_action_node src/pick_up_action_node.cpp)
ament_target_dependencies(pick_up_action_node ${DEPS})

add_executable(drop_off_action_node src/drop_off_action_node.cpp)
ament_target_dependencies(drop_off_action_node ${DEPS})

install(TARGETS
  navigate_action_node
  pick_up_action_node
  drop_off_action_node
  DESTINATION lib/${PROJECT_NAME}
)

install(DIRECTORY pddl launch config
  DESTINATION share/${PROJECT_NAME}
)

ament_package()
```

Listing 7: CMakeLists.txt for the transport_task package.

3.5.7 Launch File

```
import os
from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch_ros.actions import Node
from plansys2_support_py.launch import generate_plansys2_launch

def generate_launch_description():
    pkg_dir = get_package_share_directory('transport_task')
    domain_file = os.path.join(pkg_dir, 'pddl', 'transport_domain.pddl')

    plansys2_nodes = generate_plansys2_launch(domain_file=domain_file)

    navigate_node = Node(
        package='transport_task',
        executable='navigate_action_node',
        name='navigate_action_node',
        output='screen',
    )

```

```
    pick_up_node = Node(
        package='transport_task',
        executable='pick_up_action_node',
        name='pick_up_action_node',
        output='screen',
    )

    drop_off_node = Node(
        package='transport_task',
        executable='drop_off_action_node',
        name='drop_off_action_node',
        output='screen',
    )

    return LaunchDescription(
        plansys2_nodes + [navigate_node, pick_up_node, drop_off_node]
    )
```

Listing 8: Python launch file that starts PlanSys2 core nodes, loads the PDDL domain, and launches the three action executors (`launch/transport_task.launch.py`).

3.5.8 Building and Running

After placing the package in a ROS2 workspace (`colcon` workspace), the build and activation sequence follows the standard ROS2 workflow. First, the workspace is built with `colcon build -symlink-install`, and the environment is sourced with `source install/setup.bash`. The system is then launched with `ros2 launch transport_task transport_task.launch.py`. Once running, the problem instance (objects, predicates, and goal) is loaded into the Problem Manager via the PlanSys2 terminal client or through programmatic ROS2 service calls, and planning is triggered.

3.5.9 Expected Execution Behavior

Figure 4 illustrates the expected action sequence produced by the POPF solver and dispatched by the PlanSys2 Executor for this problem instance. Because the robot can only carry one package at a time (`free` predicate prevents concurrent pick-up), the plan serializes the two delivery sub-tasks.

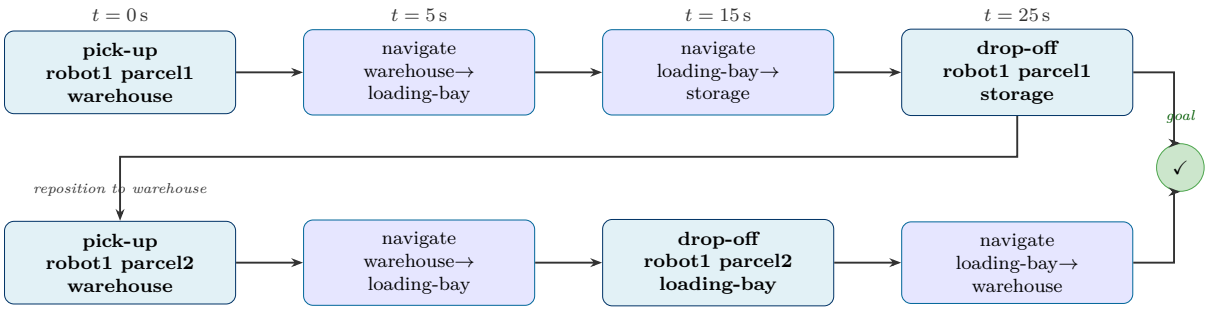

Figure 4: Expected plan execution sequence for the transport task. Row 1 handles parcel1 (deliver to storage); Row 2 handles parcel2 (deliver to loading-bay, return robot home). Durations: `pick-up`/`drop-off` = 5 s; `navigate` = 10 s. The tick mark indicates goal achievement.

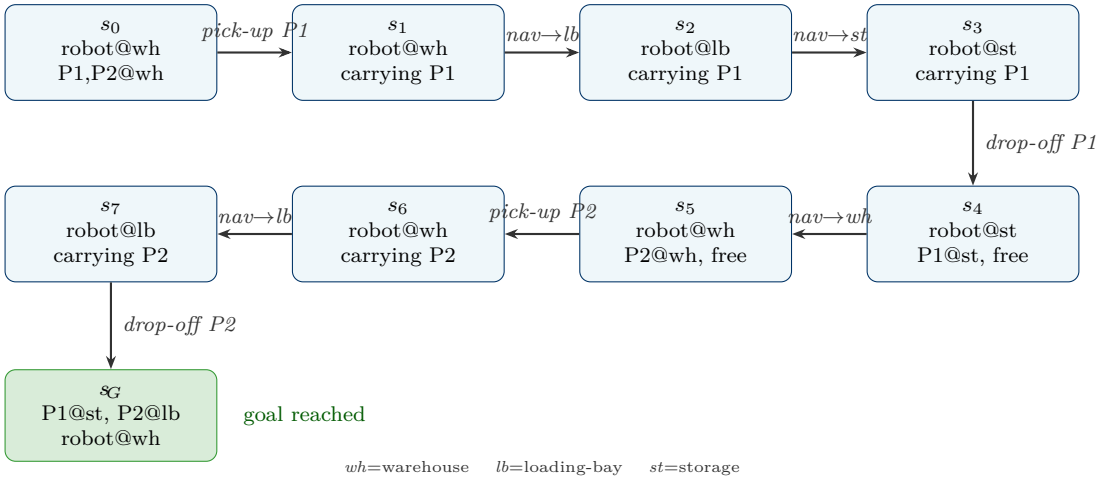

Figure 5: World state sequence corresponding to the transport plan. Each state transition corresponds to one PDDL action execution. The path snakes: top row left-to-right, then bottom row right-to-left, ending at the goal state s_G.

3.6 Known Limitations of PlanSys2

Despite its practical utility, PlanSys2 exhibits limitations that are directly relevant to this research:

Epistemic limitations. PlanSys2 is fundamentally a classical single-agent planning model. The world state is a set of ground predicates assumed to be fully known to the planner. There is no representation of agent-relative knowledge, no epistemic goal language, and no support for actions that change knowledge states without changing the physical world.

Partial observability. PlanSys2 does not natively handle partial observability. Sensor uncertainty and missing observations must be handled outside the planner before updating predicates. The planner always assumes predicates reflect the true world state.

Multi-agent coordination. PlanSys2 is designed for single-robot planning. While multiple instances can run in parallel, there is no native mechanism for cross-robot plan coordination, shared world state, or multi-agent goal management.

Solver expressivity. The solvers supported by PlanSys2 (POPF, TFD) implement classical or temporal planning with no support for epistemic or probabilistic reasoning. Extending PlanSys2 to use an epistemic planner would require a new solver plugin and a restructured problem representation.

These limitations motivate the core research objective: to integrate the engineering infrastructure of ROS2 and PlanSys2 with the expressive semantics of DEL-based epistemic planning.

4 Multi-Robot Systems in ROS2

4.1 Architectural Patterns for Multi-Robot Systems

Multi-robot systems in ROS2 are typically organized according to one of three architectural patterns: *centralized*, *decentralized*, or *hierarchical*. In a centralized architecture, a single master node maintains a global world model and generates plans for all robots. In a decentralized architecture, each robot operates as an independent planning agent, communicating with peers to resolve conflicts and share observations. A hierarchical architecture arguably the most practically relevant for the target research combines a fleet-level planner with robot-level planners, where the fleet planner decomposes high-level goals into sub-goals assigned to individual robots.

4.2 Multi-Robot Navigation and Coordination

Navigation in multi-robot ROS2 systems is typically managed by the Nav2 stack, supporting namespaced multi-robot deployments. Open-RMF (Open Robotics Middleware Framework) provides a standardized interface for multi-fleet, multi-robot task allocation and coordination in shared environments [9].

4.3 Task Allocation and Multi-Agent Planning

Task allocation assigns tasks to robots to optimize objectives such as makespan, coverage, or load balance. Classical approaches include market-based auction methods and integer programming formulations. In the context of PlanSys2, one approach is to model multiple robot agents as PDDL objects in a single planning instance, leveraging MA-PDDL or MASTRIPS extensions. However, such approaches remain within the classical planning paradigm and inherit all its epistemic limitations.

4.4 Shared World Modeling

A fundamental challenge is maintaining a shared, consistent world model across all agents. From an epistemic planning perspective, the problem is more subtle: it is not always desirable or possible to maintain a single globally consistent model. Different robots may have genuinely different beliefs about the state of the world, and those differences are semantically meaningful. An epistemic planner must represent and reason about individual perspectives, plan actions that resolve epistemic differences, and formulate goals referring explicitly to agent knowledge states.

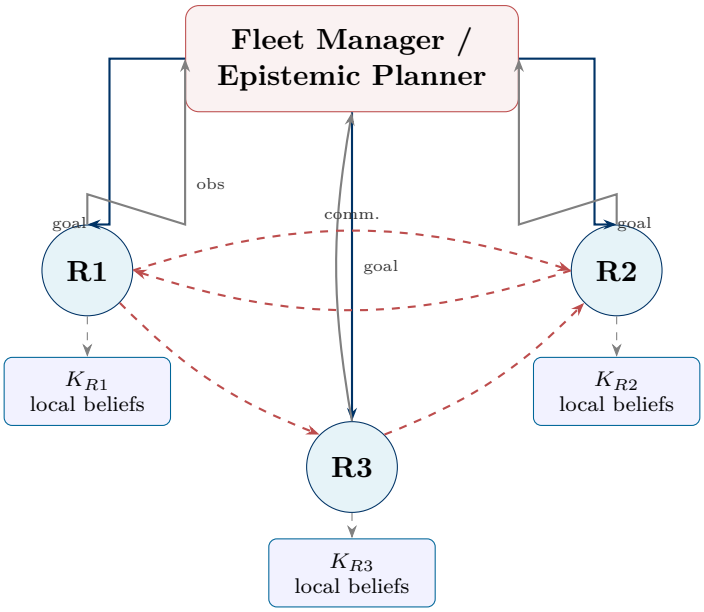

Figure 6: Multi-robot system with a fleet-level epistemic planner and robot-level local knowledge bases. Robots send observations to the fleet planner and engage in peer-to-peer communication, giving rise to heterogeneous epistemic states.

5 Dynamic Epistemic Logic: Foundations for Epistemic Planning

5.1 Motivation: From Classical to Epistemic Planning

Classical planning operates over a deterministic, fully observable world. It cannot represent the knowledge states of individual agents, uncertainty about the current state, or the epistemic effects of communication. Dynamic Epistemic Logic (DEL) [3, 4] provides a rigorous formal framework for reasoning about knowledge, belief, and informational change in multi-agent systems, extending classical propositional logic with modal operators and a formal semantics for how truth values change when epistemic actions are applied.

5.2 Epistemic States and Kripke Semantics

Definition 5.1 (Multi-Agent Epistemic Model)**.** A multi-agent epistemic model is a triple $M = (W, R, L)$ where:

- $W \neq \emptyset$ is a finite set of *possible worlds*;
- $R : \text{Ag} \to 2^{W \times W}$ assigns to each agent i an *accessibility relation* $R_i \subseteq W \times W$; and
- $L : W \to 2^P$ assigns to each world a *label* the set of atoms true there.

An epistemic state is a pair $s = (M, W_d)$ where $W_d \subseteq W$ is a non-empty set of *designated worlds*.

The accessibility relation wR_iv means that in world w, agent i considers v possible given its current observations. A formula $\Box_i\varphi$ ("agent i knows φ") holds in w iff φ holds in all R_i-accessible worlds:

$$(M, w) \models \Box_i\varphi \iff \forall v \in W : wR_iv \Rightarrow (M, v) \models \varphi$$

The *knowing-whether* modality $\text{Kw}_i\varphi = \Box_i\varphi \vee \Box_i\neg\varphi$ is especially important in robotics, where sensors resolve binary queries.

5.3 Common Knowledge

For a group $G \subseteq \text{Ag}$, *common knowledge* $C_G\varphi$ requires that all agents know φ, all agents know that all know it, and so on ad infinitum:

$$C_G\varphi = \bigwedge_{k>0} \Box_G^k\varphi, \quad \Box_G\varphi = \bigwedge_{i \in G} \Box_i\varphi$$

Common knowledge is essential for robotic coordination: a team that must act in concert requires not only that each robot knows the plan, but that it is common knowledge among all robots that they know the plan otherwise a robot might fail to act because it is

uncertain whether its teammates are aware of their roles.

5.4 Epistemic Actions and the Product Update

Definition 5.2 (Event Model and Epistemic Action)**.** An event model is a quadruple $A = (E, Q, \text{pre}, \text{post})$ where $E \neq \emptyset$ is a finite set of *events*; $Q : \text{Ag} \to 2^{E \times E}$ assigns each agent an accessibility relation over events; $\text{pre} : E \to \mathcal{L}^C_{P,\text{Ag}}$ assigns preconditions; and $\text{post} : E \times P \to \mathcal{L}^C_{P,\text{Ag}}$ assigns postconditions. An epistemic action is a pair $a = (A, E_d)$ where $E_d \subseteq E$ are the *designated events*.

Definition 5.3 (Product Update)**.** Let $a = ((E, Q, \text{pre}, \text{post}), E_d)$ be applicable in state $s = ((W, R, L), W_d)$. The product update $s \otimes a = ((W', R', L'), W'_d)$ is:

$$
\begin{aligned}
W' &= \{(w, e) \in W \times E \mid (M, w) \models \text{pre}(e)\} \\
R'_i &= \{((w, e), (v, f)) \mid wR_iv \text{ and } eQ_if\} \\
L'(w, e) &= \{p \in P \mid (M, w) \models \text{post}(e, p)\} \\
W'_d &= \{(w, e) \in W' \mid w \in W_d,\ e \in E_d\}
\end{aligned}
$$

The product update simultaneously captures physical effects (through postconditions) and epistemic effects (through combined world and event accessibility relations). An agent's new beliefs are determined by what it knew before combined with what it can observe about the action.

5.5 Types of Epistemic Actions

A *public announcement* $\text{ann}(\varphi)$ communicates a formula to all agents simultaneously with a reflexive accessibility relation. It is the simplest DEL action and forms the basis of the Public Announcement Logic (PAL) fragment.

A *private ontic action* is one where the actor moves a physical object while other agents are oblivious modeled with a "null event" linked to the real event via non-actor accessibility.

A *sensing action* allows agents to observe a binary property, with two designated events for each outcome. After sensing, a fully observant agent knows whether the property holds.

A *quasi-private sensing action* combines private and semi-private observability: some agents are fully observant, some know only that sensing occurred, and some are oblivious.

5.6 Epistemic Planning Tasks

Definition 5.4 (Epistemic Planning Task)**.** An epistemic planning task is a triple $T = (s_0, \text{Act}, \varphi_g)$ where s_0 is an initial epistemic state, Act is a finite set of epistemic actions, and φ_g is a goal formula. A solution is a sequence $\pi = a_1, \ldots, a_l$ such that π is applicable

in s_0 and $s_0 \otimes \pi \models \varphi_g$.

Crucially, φ_g may be an epistemic formula: for example, $\Box_{R_1}\text{packageDelivered} \wedge \text{Kw}_{R_2}\text{corridorBlocked}$ specifies knowledge states as goals not expressible in classical PDDL.

5.7 DEL in Robotics: From Theory to Application

The transfer of DEL from its origins in philosophical logic and game theory to practical multi-robot systems has been one of the most active research directions of the past decade. While the theoretical framework of DEL was originally developed for studying abstract epistemic puzzles [3, 4], the modeling machinery it provides Kripke models, accessibility relations, product updates maps naturally onto the challenge of reasoning about a distributed robot fleet in which agents have heterogeneous, potentially inconsistent observations of a shared environment.

The foundational insight is that a robot's *epistemic state* is not merely a record of what it has sensed, but a structured representation of what it knows, what it does not know, and crucially in a multi-robot setting what it knows or believes about the epistemic states of its teammates. This higher-order structure, formalized through nested modal operators $\Box_i \Box_j \varphi$ and common knowledge $C_G \varphi$, has no counterpart in classical PDDL-based planning and therefore cannot be captured by conventional ROS2 task planners such as PlanSys2.

An early and influential formalization of DEL for autonomous agents is provided by Löwe, Pacuit, and Witzel [18], who define the *DEL planning problem* as the task of finding a sequence of event models σ such that $M \otimes \sigma \models \varphi_g$, where (M, w) is the initial pointed model and φ_g is an epistemic goal formula. Their analysis of the *Thief: The Dark Project* scenario demonstrates that a knowledge module maintaining the Kripke model throughout the execution of a plan can support rich, belief-aware scripted and autonomous behaviors without requiring explicit re-engineering of the agent's control logic every time epistemic assumptions change. They also establish structural properties *self-absorption* and *commutativity* of propositional almost-mutex event models that bound the growth of the Kripke model under iterated updates. Concretely, for the event library $\mathcal{E} = \{N_t, N_g, B_t, B_g, F\}$ (noise and facing actions in the stealth game scenario), the bisimulation contraction of $I \otimes \sigma$ has size at most 6 for any legal sequence σ, regardless of length. This is an important tractability result: it shows that for suitably restricted event libraries, the Kripke model does not grow without bound and can be maintained in real time.

The framework of Löwe et al. also formalizes three planning problem variants of increasing difficulty. In the *absolute* problem, no agent assignment is specified and the planner searches for any legal sequence achieving the goal. In the *single-agent* problem, all events must lie within the power set $\text{power}(i)$ of a designated agent i. In the *adversarial* problem

most relevant to adversarial environments or unreliable communication the planner must find a partial strategy $\hat{\sigma}$ such that the goal is achieved under *all* legal completions, forming a game-theoretic worst-case guarantee. These distinctions directly inform the design of epistemic planning modules for robotic systems, where the choice between cooperative and adversarial problem formulations determines the computational complexity of planning and the robustness guarantees that can be provided.

5.8 Belief and Empathy States in Multi-Robot DEL

A central design choice in applying DEL to robotics is how to represent epistemic states at runtime in a way that is both formally grounded and computationally tractable. Bramblett and Bezzo [16] propose an elegant particle-based approximation in which each robot i maintains a finite set of *belief particles* $\mathcal{P}_i = \{p_{ij,k} \mid j \in \mathcal{A}, k \in \mathcal{A}\}$, where $p_{ij,k}$ represents robot i's second-order belief its belief about robot j's belief about robot k's state. The designated subset $C_i \subseteq \mathcal{P}_i$, called the *common belief set*, corresponds to the last globally shared epistemic state; it is propagated forward in time using each robot's known dynamics and status propositions when communication is unavailable.

This formulation instantiates the DEL epistemic state (W, R_i, V, W_d) with a finite, tractable world set: worlds $w \in W$ are indexed by combinations of possible robot statuses $\sigma_i^t \in \{\text{exploring, gossiping, performing task, going home}\}$ for all $i \in \mathcal{A}$. The accessibility relation R_i then encodes robot i's uncertainty about which status combination is currently true. Epistemic updates follow from two action types in the library $\mathcal{A} = \{\text{perceive}(\varphi), \text{announce}(\varphi)\}$: a *perceive* action updates a robot's local belief when it directly observes a proposition φ (e.g., task discovered or teammate absent from expected location), and an *announce* action a DEL public announcement disseminates a state description to all locally connected robots, establishing common knowledge of the announced formula among them.

The product update semantics govern rational belief revision in each case. When all robots are connected and robot i announces the full system disposition $\Omega = (\Omega_j)_{j \in \mathcal{A}}$, the epistemic state from robot i's perspective evolves as:

$$s_{t-1}^i \otimes \text{announce}(\Omega) = s_t^i \models K_i \sigma_t^i \wedge \bigwedge_{j \in \mathcal{A}} K_i K_j \sigma_t^j,$$

establishing mutual knowledge of all robots' statuses the condition for a true world w_t^* to exist in the Kripke model. When robots are disconnected, the perceive action produces a local, asymmetric belief update: if robot i observes that robot j is absent from its expected location, the world w corresponding to j's originally predicted status is removed from R_i's accessible set, and the next most probable particle is activated. This selective pruning of the world set corresponds precisely to the DEL product update in the special case where

the event model has two events "j is here" and "j is not here" and the precondition of the former is not satisfied.

The companion paper by the same authors [15] extends this framework to heterogeneous robot teams (mixed UGV/UAV fleets) and introduces the concept of *empathy states*: a robot i maintains, alongside its belief particles about others, a set of particles representing what other robots might currently believe about i itself. Formally, $\mathcal{P}_i^e = \{p_{ii,b} \mid b \in B\}$ are robot i's empathy particles, and they are equivalent to robot j's belief particles about robot i under the assumption that all robots follow the same particle-propagation policy. This symmetry property enables a robot to proactively position itself so that at least one of its teammates' belief particles correctly predicts its location, ensuring future communicability without explicit signaling.

The epistemic task allocation problem in this setting is formalized as a non-linear integer program over the epistemic state s_t^i, where each robot i optimizes a joint utility function incorporating expected arrival times computed from the belief particles, capability constraints, and precedence constraints over task orderings. The solution, computed via a genetic algorithm with the initial population seeded by a greedy feasible-solution generator, produces a policy π as a concatenation of gossip bundles (which robot communicates with whom) and task bundles (who performs what). A gossip protocol then propagates task assignments through the network of disconnected robots, exploiting the particle-based belief model to route messages efficiently through predicted teammate locations [15, 16].

Simulation experiments over 15 randomized environments with two to eight robots demonstrated that the proposed DEL-based method achieves mission times close to an ideal fully-connected baseline and substantially outperforms a connectivity-constrained flock method across all team sizes. This provides strong empirical evidence that the particle-based DEL approximation is not merely a theoretical tool but a practically viable approach to runtime epistemic reasoning in robotic systems.

5.9 Theory of Mind, Higher-Order Reasoning, and Active Epistemic Inference

A significant limitation of first-order epistemic planning where each robot reasons only about its own beliefs about the world is that it cannot resolve ambiguities arising from simultaneous actions of multiple agents without communication. If two robots observe each other moving toward the same task, neither can determine from its own first-order observations alone who is actually committed to that task, leading to convergence failures and duplicated effort. The epistemic solution is to elevate the depth of reasoning to second and third order: a robot i reasons not only about what others know ($\Box_j\varphi$) but about what others know about what i knows ($\Box_j\Box_i\varphi$), capturing the "I know that you know that I know" structure of Theory of Mind (ToM) [17].

Bramblett, Reasoner, and Bezzo formalize this in an *active epistemic inference* framework that combines DEL-based higher-order belief maintenance with variational free energy minimization from active inference theory [17]. The epistemic state is represented as $s = (W, (R_i)_{i \in \mathcal{R}}, L, w)$, where the accessibility relation R_i encodes robot i's run-time uncertainty over world configurations each world $w \in W$ being an assignment of robots to task subsets. Composite accessibility relations $R_i \circ R_j$ encode second-order knowledge: $s \models K_i K_j \varphi$ iff φ holds in all worlds jointly reachable through both R_i and R_j. Plans for each robot are sequences $\pi_i = \langle a_{i1}, \ldots, a_{ip} \rangle$ leading from an initial epistemic state s_i^0 to a goal state $s_i^* \in \Gamma_i$, where the planning function incorporates the nested belief structure: $\pi_i = \text{Plan}(s_i^0, \Gamma_i, \mathbf{B}_i)$, with $\mathbf{B}_i$ encoding robot i's beliefs about the beliefs of all other robots.

The active inference layer converts epistemic beliefs into motion policies by minimizing the *variational free energy* (VFE):

$$\mathcal{F}(x_i(t), u_i(t), \mathcal{G}, y_i(t), \omega_i) = H[q(\mathcal{G})] + D_{\text{KL}}[q(\mathcal{G}) \,\|\, P(x_i(t) \mid y_i(t), \omega_i)],$$

where $q(\mathcal{G})$ is the robot's approximate posterior over valid task-assignment configurations $\mathcal{G}$, and ω_i is its sensor configuration. The likelihood function that drives belief updates is computed using a hierarchical *salience* function indexed by reasoning order k:

$$e_i^{(k)} \leftarrow \upsilon_i^{(k)}(y_i, \Omega, \mathcal{G}) = \exp\Big(-\tfrac{1}{\eta}\, h_i^{(k)}(y_i, \Omega, \mathcal{G})\Big)\,,$$

where zero- and first-order evidence use each robot's own direct observations; second-order evidence abstracts the perspective of another robot by using its known sensor model ω_j; and third-order evidence models what robot j believes robot k observes. The joint likelihood over all orders is shown to reduce the variance of the posterior belief distribution compared to first-order reasoning alone, by averaging out bounded, mean-zero observation errors across multiple reasoning layers [17].

In practice, robots execute a *perception-policy loop*: update the posterior $q(\mathcal{G})$ from observations, compute the expected free energy for a set of candidate movement policies, and select the policy minimizing EFE. Policies that actively disambiguate goal assignments by taking "exaggerated" movements that are informative to observers are preferred over purely task-directed paths because they minimize the epistemic term of the EFE. This produces implicit coordination: robots signal intent through motion without any explicit communication, resolving assignment conflicts that would stalemate first-order reasoners. Experiments with Husarion ROSbot 2.0 UGVs and Crazyflie UAVs confirm that higher-order reasoning (up to third order) achieves substantially higher mission success rates across rendezvous, task allocation, and multi-task scenarios, with the advantage growing markedly as team size exceeds three robots [17].

5.10 DEL Planning Problem Variants and Structural Properties

The formal study of DEL-based planning admits several natural variants depending on the degree of agent autonomy and the nature of the environment. Following the framework of Löwe et al. [18], one distinguishes:

- **Absolute DEL planning:** Find a sequence $\sigma \in T \subseteq LS$ of event models such that $M \otimes \sigma \models \varphi$, where T is a subtree of legal sequences and φ is an epistemic goal formula. This formulation is agent-agnostic.
- **Single-agent DEL planning:** As above, but additionally requiring $\sigma \in \text{power}(i)$, where the function $\text{power} : \mathcal{A} \to \wp(\mathcal{E})$ maps each agent to the set of event models it can instantiate. This corresponds to the classical planning assumption of a single omniscient planner.
- **Adversarial DEL planning:** Find a partial strategy function $\hat{\sigma} : \{0, \ldots, N\} \to \mathcal{E}$ such that (i) $\hat{\sigma} \in \text{power}(i)$, (ii) there exists a completion $\sigma \in T$ with $\hat{\sigma} \subseteq \sigma$, and (iii) for *all* such completions $M \otimes \sigma \models \varphi$. This is a worst-case game-theoretic problem in which the planning agent must achieve its epistemic goal regardless of the actions of opponents or environmental nondeterminism.

The adversarial formulation is the most directly relevant to multi-robot systems operating with unreliable communication channels or nondeterministic physical outcomes. Critically, the goal formula φ in all three variants is drawn from the full epistemic language and may therefore specify goals such as $\Box_{R_1}\text{taskComplete} \wedge C_{\mathcal{A}}\text{rendezvousAgreed}$ that cannot be reduced to classical propositional planning goals.

Two structural properties identified by Löwe et al. are directly relevant to the runtime tractability of DEL planning in robotics. An event model $\mathcal{E}$ is *self-absorbing* if $M \otimes \mathcal{E} \otimes \mathcal{E} \simeq M \otimes \mathcal{E}$ for all models M (applying the same event twice has the same effect as applying it once). Two event models $\mathcal{E}, \mathcal{E}'$ are *commutative* if $M \otimes \mathcal{E} \otimes \mathcal{E}' \simeq M \otimes \mathcal{E}' \otimes \mathcal{E}$ for all M. Propositional event models are always commutative (Lemma 1 in [18]), and almost-mutex event models with transitive accessibility relations are self-absorbing (Lemma 2 in [18]). When the event library $\mathcal{E}$ consists entirely of such event models, the Kripke model remains bounded in size under any sequence of updates, and the bisimulation contraction can be computed in linear time. This guarantees that the World Model Manager in the proposed architecture (Section 7) remains computationally tractable for suitably restricted event libraries a property that should be treated as a design criterion when specifying EPDDL action schemas for concrete robotic domains.

6 EPDDL: The Epistemic Planning Domain Definition Language

6.1 Motivation and Design Philosophy

EPDDL, introduced by Burigana and Fabiano [5] as the official language for the Epistemic Planning Track at IPC 2026, addresses the absence of a standardized language for DEL-based epistemic planning problems. It is designed for: (1) compatibility with PDDL syntax; (2) formal grounding in full DEL semantics with abstract event models; and (3) a modular action type library system enabling specification and comparison of DEL fragments.

6.2 Language Structure

An EPDDL specification consists of three components: **problems** (instance-specific: objects, agents, initial epistemic state, goal); **domains** (instance-independent: type declarations, predicates, event definitions, action schemas); and **action type libraries** (abstract frames defining observability patterns for different DEL fragments).

6.3 EPDDL Syntax: A Concrete Example

We present a simplified domain and problem for a two-robot corridor-sensing scenario: robots `r1` and `r2` are adjacent to corridor `c1`, whose blocked status is initially unknown to both. Robot `r1` can sense the corridor and then announce its finding. The goal is that all agents know whether the corridor is blocked.

6.3.1 EPDDL Domain

```
(define (domain corridor-sensing)
  (:requirements :typing :sensing :announcements :group-modalities)

  (:types robot corridor)

  (:predicates
    (blocked ?c - corridor)
    (adjacent ?r - robot ?c - corridor)
  )

  (:action sense-corridor
    :type sensing
    :actor ?r - robot
    :parameters (?r - robot ?c - corridor)
    :observability (
      (full-obs ?r)
      (no-obs All \ {?r})
    )
    :precondition (adjacent ?r ?c)
```

```
    :sense-formula (blocked ?c)
  )

  (:action announce-blocked
    :type public-announcement
    :actor ?r - robot
    :parameters (?r - robot ?c - corridor)
    :precondition (and
      (adjacent ?r ?c)
      ([?r] (blocked ?c))
    )
    :announced-formula (blocked ?c)
  )

  (:action announce-clear
    :type public-announcement
    :actor ?r - robot
    :parameters (?r - robot ?c - corridor)
    :precondition (and
      (adjacent ?r ?c)
      ([?r] (not (blocked ?c)))
    )
    :announced-formula (not (blocked ?c))
  )
)
```

Listing 9: EPDDL domain for a two-robot sensing and announcement scenario (`corridor_sensing.epddl`).

6.3.2 EPDDL Problem

```
(define (problem corridor-task-1)
  (:domain corridor-sensing)

  (:objects
    r1 r2 - robot
    c1 - corridor
  )

  (:agents r1 r2)

  (:init
    (:static
      (adjacent r1 c1)
      (adjacent r2 c1)
    )
    (:epistemic
```

```
      (C All (Kw r1 (blocked c1)) false)
      (C All (Kw r2 (blocked c1)) false)
    )
  )

  (:goal (and
    ([Kw. All] (blocked c1))
  ))
)
```

Listing 10: EPDDL problem instance using a finitary S5-theory for the initial epistemic state (`corridor_task.epddl`).

The `:epistemic` block encodes a finitary S5-theory: `(C All (Kw r1 (blocked c1)) false)` asserts that it is common knowledge among all agents that `r1` does not know whether the corridor is blocked. The goal `([Kw. All] (blocked c1))` uses the group knowing-whether modality to require all agents to resolve the uncertainty. Key syntactic constructs are summarized in Table 1.

EPDDL Construct	Logical Meaning	Context
`([i] phi)`	$\Box_i\varphi$ agent i knows φ	preconditions, goals
`(<i> phi)`	$\Diamond_i\varphi$ agent i considers possible	preconditions
`([Kw. i] phi)`	$\mathrm{Kw}_i\varphi$ knows whether φ	goals, init
`([Kw. All] phi)`	$\mathrm{Kw}_{\mathrm{All}}\varphi$ all know whether	goals
`(C All phi)`	$C_{\mathrm{All}}\varphi$ common knowledge	init, goals
`:type sensing`	sensing action type	action schema
`:type public-announcement`	public announcement type	action schema
`:sense-formula`	formula whose truth is observed	sensing actions
`:announced-formula`	formula publicly communicated	announcement actions
`:observability`	per-type observability assignment	action schema

Table 1: Key EPDDL syntactic constructs with logical counterparts and usage contexts.

6.4 Modal Operators in EPDDL

EPDDL Syntax	Logical Formula	Reading
`([i] phi)`	$\Box_i\varphi$	Agent i knows/believes φ
`(<i> phi)`	$\Diamond_i\varphi$	Agent i considers φ possible
`([Kw. i] phi)`	$\mathrm{Kw}_i\varphi$	Agent i knows whether φ
`(<Kw. i> phi)`	$\widehat{\mathrm{Kw}}_i\varphi$	Agent i does not know whether φ
`([C. G] phi)`	$C_G\varphi$	φ is common knowledge in G
`(<C. G> phi)`	$\hat{C}_G\varphi$	Dual of common knowledge

Table 2: Modal operators in EPDDL and their logical counterparts.

Modality indices may be single agent terms, agent group lists, or the reserved keyword `All`. Group modalities require the `:group-modalities` requirement flag.

6.5 Initial State Representation

EPDDL supports two methods for specifying the initial epistemic state. An *explicit definition* directly encodes worlds, accessibility relations, world labels, and designated worlds. A *finitary S5-theory* [11] specifies the state through commonly-known epistemic formulas of restricted form, consisting of propositional atoms describing the actual world, $C_{\mathrm{Ag}}(\Box_i\varphi)$ formulas for commonly-known truths, $C_{\mathrm{Ag}}(\mathrm{Kw}_i\varphi)$ formulas for agent certainties, and $C_{\mathrm{Ag}}(\widehat{\mathrm{Kw}}_i\varphi)$ formulas for agent uncertainties. The `plank` toolkit automatically constructs the Kripke model from a finitary S5-theory.

6.6 DEL Fragments via Action Type Libraries

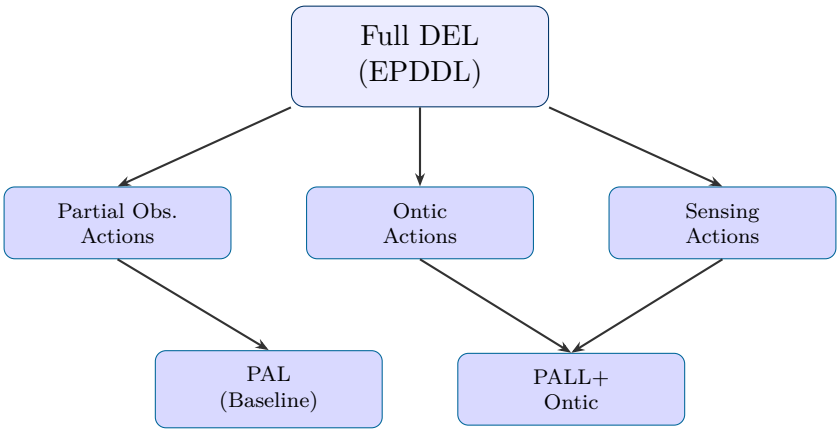

Figure 7: Hierarchy of DEL fragments expressible through EPDDL action type libraries. The PAL baseline supports only public announcements. Richer fragments add ontic actions, partial observability, and sensing actions.

Action type libraries allow benchmark designers to create problem sets solvable by planners of different capabilities. Any epistemic planning formalism translatable into a DEL fragment can be represented in EPDDL via an appropriate library, enabling cross-formalism comparison.

6.7 Abstract Event Models and Action Types

The key theoretical innovation is the *abstract epistemic action*, which decouples the structural pattern of observability (action type) from specific preconditions, effects, and agent assignments (domain action).

Definition 6.1 (Abstract Event Model)**.** An abstract event model on ObsTypes is a tuple $(E, Q, \mathrm{pre}, \mathrm{post}, \mathrm{obs})$ where (E, Q) is an abstract frame, pre and post are as in a standard

event model, and obs : Ag $\to$ (ObsTypes $\to \mathcal{L}^{C}_{P,\mathrm{Ag}}$) assigns to each agent an observability function satisfying mutual inconsistency and coverage of the logical space.

7 Integration: Toward a DEL-Based Planning System for ROS2

7.1 Conceptual Architecture

Integrating DEL-based epistemic planning with a ROS2 multi-robot system requires bridging three conceptual gaps: (i) between the physical robot state and the epistemic state representation; (ii) between EPDDL action schemas and ROS2 action server implementations; and (iii) between the planner's output a sequence of abstract epistemic actions and the robot execution infrastructure.

We propose a layered conceptual architecture composed of three interacting strata: a *Cognitive Layer*, an *Execution Layer*, and a *Perception Layer*. This separation clarifies the interaction between symbolic epistemic reasoning and physical multi-robot control.

Cognitive Layer. This layer maintains and manipulates the global epistemic state (M, W_d) of the fleet. It contains:

- **Epistemic World Model Manager:** Maintains the Kripke model and applies product update semantics (or suitable approximations) upon perception events or action completion. In the spirit of the knowledge module proposed by Löwe et al. [18], this component acts as a persistent epistemic engine that any behavior module can query for truth values of epistemic formulas.
- **EPDDL Planner Interface:** Generates EPDDL problem instances from the current epistemic state, invokes an external epistemic planner, and translates the resulting abstract plan into dispatchable actions.
- **Epistemic Action Dispatcher:** Instantiates abstract epistemic actions as physical or communication events, dispatches them to ROS2, and triggers the corresponding epistemic product update.

Execution Layer. Implements low-level robot behaviors using standard ROS2 action servers (navigation, manipulation, sensing, communication). This layer is responsible for physical actuation and execution monitoring.

Perception Layer. Transforms physical sensor data into symbolic assertions and epistemic events. A state estimation and belief update module translates observations into grounded propositions and applies updates to the global epistemic state.

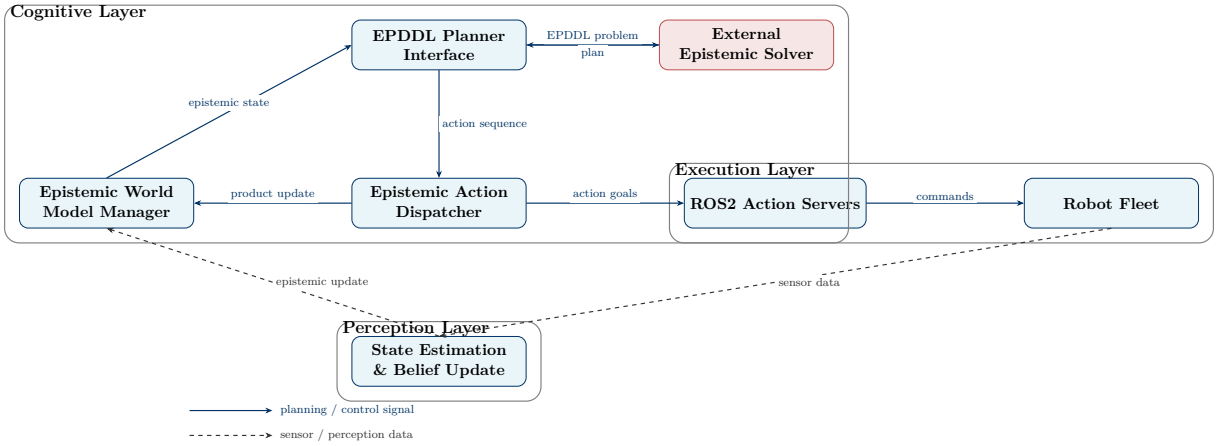

Figure 8: Layered conceptual architecture for DEL-based epistemic multi-robot planning. The Cognitive Layer maintains and updates the global Kripke model and generates epistemic plans. The Execution Layer implements physical action through ROS2 action servers. The Perception Layer translates sensor observations into symbolic epistemic updates applied via DEL product update semantics.

7.2 State Space and Computational Challenges

The number of possible worlds in a Kripke model can grow exponentially with the number of propositions and agents. Several approximation strategies have been explored: *bounded world representations* limit the number of worlds using heuristic pruning; the particle-based approximation of Bramblett and Bezzo [16] offers a computationally efficient alternative that preserves the essential DEL update structure while bounding world-set size; and *compiled classical planning approaches* [10] transform epistemic problems into classical ones by encoding epistemic state as additional propositional atoms. The structural tractability results of Löwe et al. [18] for self-absorbing and commutative event libraries provide a formal basis for bounding model growth and should inform the choice of EPDDL action schemas in a deployed system.

7.3 Perception-to-Epistemic-State Translation

A critical engineering challenge is translating robot sensor data into Kripke model updates. A robot's observation a camera image, a lidar scan is a physical signal that must be interpreted as an epistemic event. In the ROS2 context, this pipeline consists of a perception node (producing symbolic object detections or map assertions), a semantic interpretation layer (translating detections into PDDL-compatible ground atoms), and an epistemic update module (applying the product update to the World Model Manager's Kripke model).

7.4 Communication Actions as Epistemic Events

Inter-robot communication is itself an epistemic event: when robot R_1 broadcasts that corridor c_3 is blocked, this constitutes a public announcement of $\Box_{R_1}\text{corridorBlocked}(c_3)$, and the epistemic state changes via the DEL product update. In the applied systems of Bramblett et al. [15, 16], this is implemented via a gossip protocol in which robots route to the predicted locations of teammates and deliver task assignments, treating each successful rendezvous as a localized public announcement that updates both robots' belief states. In the proposed architecture, the Epistemic Action Dispatcher implements communication actions as ROS2 topic publications while simultaneously applying the corresponding product update to the World Model Manager's Kripke model.

7.5 Open Research Questions

Tractable epistemic state representations. What Kripke model representations are both expressive enough for EPDDL-level reasoning and compact enough for real-time multi-robot fleets? Are factored or compiled representations sufficient? The particle-based approach of [16] and the bisimulation bounding results of [18] offer complementary partial answers.

Online epistemic planning. What are the performance characteristics of available epistemic planners (EFP, mGPT, MCEPlan) in online replanning settings relevant to robotics?

EPDDL-to-ROS2 toolchain. How should the `plank` toolkit integrate with the ROS2 build and runtime systems? Can EPDDL problem files be auto-generated from ROS2 system state?

Higher-order reasoning at scale. The active epistemic inference results of [17] show that third-order reasoning significantly improves task allocation success rates, but the computational cost grows with the number of reasoning levels and robots. What is the optimal trade-off between reasoning depth and computational budget?

Failure handling. When epistemic plan actions fail, the epistemic state must be updated and a new plan generated. What recovery strategies are appropriate in a ROS2 runtime?

8 Planner Performance: Benchmark Comparison

The DEL-based planner described in the companion technical report [19] was evaluated on five benchmark instances spanning two domains: Box Task and Active Muddy Children. Table 3 and Figures 9–10 summarise search performance across all problems, algorithms, and heuristics. Results are drawn directly from the experimental evaluation reported in that work.

Table 3: Search performance summary: nodes expanded and plan depth by problem, algorithm, and heuristic. `wc` = world count; `ug` = unsatisfied goal conjuncts; `ed` = epistemic distance. GBFS was used for Box Task 1 (linear plan); AO* with iterative-deepening AND-OR search was used for all remaining instances (conditional plans).

Problem	**Worlds**	**Alg.**	**Heuristic**	**Expanded**	**Generated**	**Depth**
Box Task 1	128 (8 des.)	GBFS	wc	1	6	1
Box Task 1	128 (8 des.)	GBFS	ug	1	6	1
Box Task 1	128 (8 des.)	GBFS	ed	1	6	1
Box Task 2	64 (4 des.)	AO*	wc	163	177	3
Box Task 2	64 (4 des.)	AO*	ug	139	153	3
Box Task 2	64 (4 des.)	AO*	ed	139	153	3
Muddy-2	4 (1 des.)	AO*	wc	7	4	2
Muddy-2	4 (1 des.)	AO*	ug	7	4	2
Muddy-2	4 (1 des.)	AO*	ed	7	4	2
Muddy-3	8 (1 des.)	AO*	wc	19	15	3
Muddy-3	8 (1 des.)	AO*	ug	19	15	3
Muddy-3	8 (1 des.)	AO*	ed	19	15	3
Muddy-3-asym	8 (1 des.)	AO*	wc	19	15	3
Muddy-3-asym	8 (1 des.)	AO*	ug	19	15	3
Muddy-3-asym	8 (1 des.)	AO*	ed	19	15	3

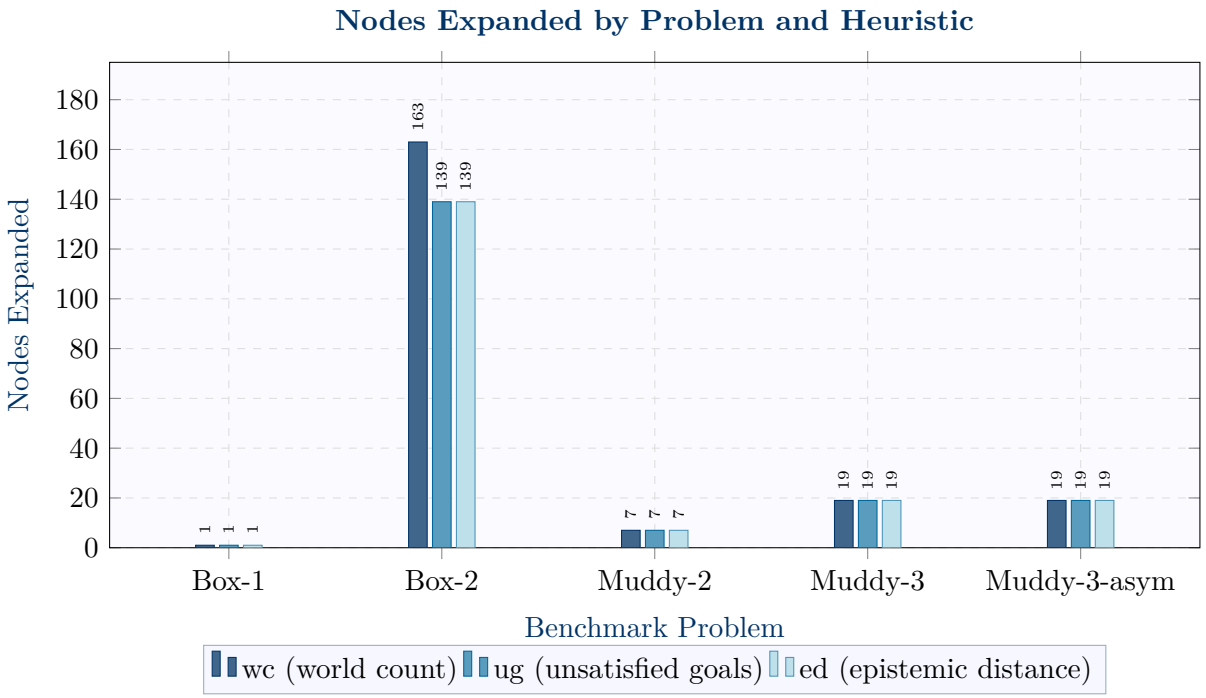

Figure 9: Grouped bar chart of nodes expanded for each benchmark problem and heuristic. Box Task 1 (GBFS) is trivially solved in one expansion regardless of heuristic. Box Task 2 (AO*) shows the largest search effort; goal-directed heuristics (`ug`, `ed`) reduce expansions by 15% relative to `wc` (163 → 139). Purely epistemic domains (Muddy Children) show no heuristic differentiation at these problem scales. Note the log-scale gap between Box Task 2 and the remaining instances.

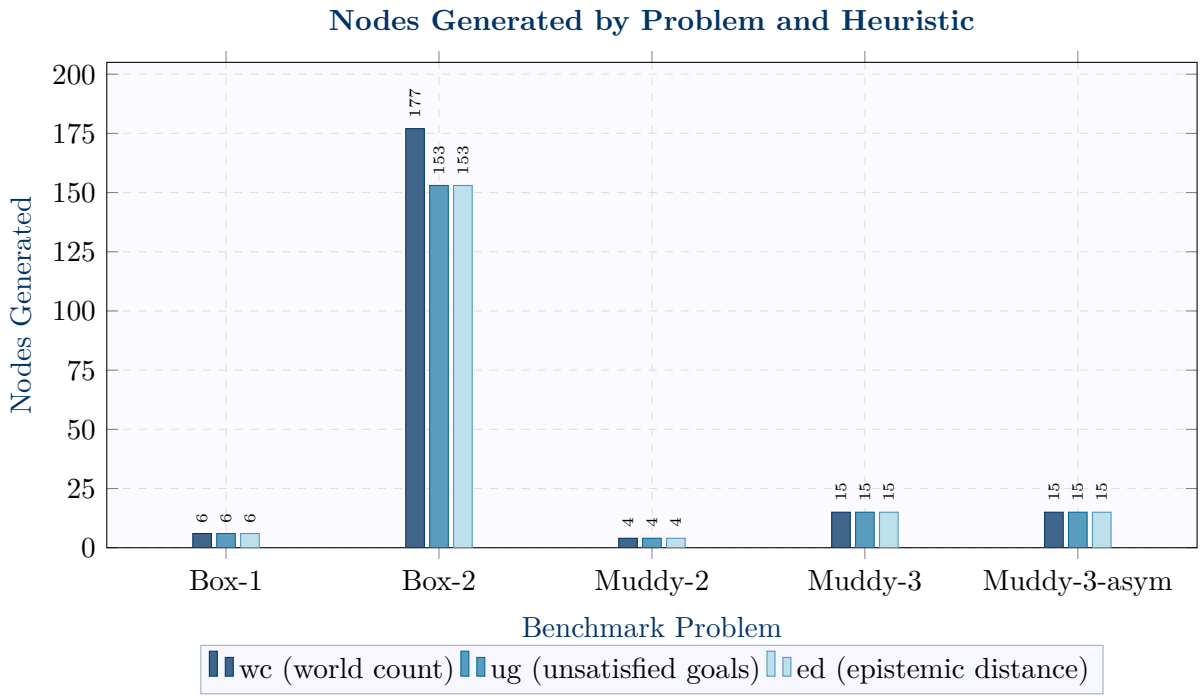

Figure 10: Nodes generated per benchmark problem and heuristic. The pattern mirrors the expansion counts: goal-directed heuristics offer meaningful reductions only on the mixed epistemic-ontic Box Task 2, where `ug` and `ed` generate 24 fewer nodes than `wc` (177 → 153). For GBFS on Box Task 1, generated nodes reflect the branching factor at the initial state rather than search effort.

Interpretation. The benchmark data reveals two distinct regimes of planner behaviour. In the *trivial regime* (Box Task 1, Muddy-2), the problem is solved at or near the root of the search tree and heuristic quality is irrelevant. In the *nontrivial regime* (Box Task 2), the search space is large enough for heuristic guidance to matter: the world-count heuristic (`wc`) generates 24 additional expansions compared to the goal-directed alternatives (`ug`, `ed`), corresponding to a 15% overhead. This is consistent with the known weakness of world-count as a heuristic: it measures global epistemic uncertainty rather than proximity to the specific goal formula, causing it to rank actions that reduce uncertainty in irrelevant dimensions equally to goal-relevant ones.

The equivalence of `ug` and `ed` on all tested instances suggests that for the domains examined, the finer-grained continuous measure provided by `ed` does not yet produce action orderings different from the coarser binary conjunct count of `ug`. This may change as problem complexity increases in particular, on problems with many epistemic conjuncts of varying difficulty or with large initial world sets where the normalized counterexample ratio varies meaningfully between actions. These are open directions for future benchmark development within the EPDDL framework, and directly inform the design of more discriminating test cases for the IPC 2026 Epistemic Planning Track [5].

On purely epistemic domains (Muddy Children, all sizes), all three heuristics yield identical

search profiles. This reflects the fact that in these domains the branching factor is small and every applicable action is equally relevant to the goal, so heuristic ranking produces no differentiation. The practical implication for the ROS2 integration architecture (Section 7) is that richer heuristics are most valuable in mixed epistemic-ontic domains precisely the setting most common in real robotic deployments while simpler heuristics suffice for purely epistemic communication and sensing tasks.

9 Conclusion

This document has surveyed the two pillars of a planned research program: the engineering infrastructure of ROS2 and PlanSys2 for task-level robot planning, and the theoretical framework of Dynamic Epistemic Logic and EPDDL for epistemic multi-agent planning. The limitations of classical PDDL-based planners no representation of agent knowledge, no epistemic goal language, no multi-agent coordination semantics motivate the adoption of DEL as the formal backbone of a next-generation planning system for heterogeneous robot fleets.

Recent applied work has demonstrated the practical viability of DEL in robotic systems. Bramblett and Bezzo [15, 16] show that a particle-based approximation of the Kripke model, combined with DEL product update semantics for belief revision, enables efficient multi-robot exploration and task allocation in communication-restricted environments, matching the performance of idealized fully-connected systems. Bramblett, Reasoner, and Bezzo [17] extend this to implicit coordination without any communication, using active inference over higher-order DEL belief structures to let robots signal intent through motion and resolve assignment ambiguities. Löwe, Pacuit, and Witzel [18] provide the formal planning problem definitions and structural tractability results (self-absorption, commutativity, bounded model growth) that ground these applied approaches in rigorous theory.

The benchmark evaluation of the companion DEL planner [19] provides concrete empirical grounding for the theoretical claims: goal-directed heuristics (`ug`, `ed`) reduce search effort by 15% on mixed epistemic-ontic domains relative to a world-count baseline, while all heuristics are equivalent on purely epistemic domains at the scales tested. These results inform the heuristic selection strategy for the EPDDL Planner Interface component of the proposed architecture.

The integration path identified in this survey a Cognitive Layer maintaining the Kripke model, interfacing with an EPDDL planner, and dispatching epistemic actions via ROS2 action servers provides a concrete architectural roadmap. The primary open challenges are the design of tractable epistemic state representations, the development of an EPDDL-to-ROS2 toolchain, and the scaling of higher-order reasoning to larger robot teams under real-time constraints.

References

[1] Macenski, S., Foote, T., Gerkey, B., Lalancette, C., & Woodall, W. (2022). *Robot Operating System 2: Design, architecture, and uses in the wild*. Science Robotics, 7(66).

[2] Martín, F., Mateos, J., Vargas, F., Moreno, R., Fernandez-Rodríguez, J. C., & Ginesi, A. (2021). *PlanSys2: A Planning System Framework for ROS2*. In Proceedings of the IEEE/RSJ IROS.

[3] Baltag, A., Moss, L. S., & Solecki, S. (1998). *The Logic of Public Announcements, Common Knowledge and Private Suspicions*. In Proceedings of TARK-98.

[4] van Ditmarsch, H., van der Hoek, W., & Kooi, B. P. (2007). *Dynamic Epistemic Logic*. Volume 337. Springer Netherlands.

[5] Burigana, A., & Fabiano, F. (2026). *The Epistemic Planning Domain Definition Language: Official Guideline*. Official Guideline Reference for the Epistemic Planning Track at IPC-26.

[6] Bolander, T., & Andersen, M. B. (2011). *Epistemic Planning for Single- and Multi-Agent Systems*. Journal of Applied Non-Classical Logics, 21(1), 9–34.

[7] Coles, A., Coles, A., Fox, M., & Long, D. (2010). *Forward-Chaining Partial-Order Planning*. In Proceedings of ICAPS.

[8] Faconti, D., & Colledanchise, M. (2018). *BehaviorTree.CPP: A Behavior Tree Library for C++*. Available: https://www.behaviortree.dev.

[9] Open Robotics (2021). *Open-RMF: Open Robotics Middleware Framework*. Available: https://www.open-rmf.org.

[10] Muise, C., Belle, V., Felli, P., McIlraith, S. A., Miller, T., Pearce, A. R., & Sonenberg, L. (2015). *Planning Over Multi-Agent Epistemic States: A Classical Planning Approach*. In Proceedings of AAAI 2015.

[11] Son, T. C., Pontelli, E., Baral, C., & Gelfond, G. (2014). *Finitary S5-Theories*. In Proceedings of JELIA 2014. LNCS vol. 8761, Springer.

[12] Kominis, F., & Geffner, H. (2015). *Beliefs in Multiagent Planning: From One Agent to Many*. In Proceedings of ICAPS 2015.

[13] Hintikka, J. (1962). *Knowledge and Belief: An Introduction to the Logic of the Two Notions*. Cornell University Press.

[14] Ghallab, M., Nau, D. S., & Traverso, P. (2004). *Automated Planning: Theory and*

Practice. Elsevier.

[15] Bramblett, L., Gao, S., & Bezzo, N. (2023). *Epistemic Planning for Heterogeneous Robotic Systems*. arXiv:2308.00579. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA).

[16] Bramblett, L., & Bezzo, N. (2023). *Epistemic Planning for Multi-Robot Systems in Communication-Restricted Environments*. Frontiers in Robotics and AI, 10:1149439. doi:10.3389/frobt.2023.1149439.

[17] Bramblett, L., Reasoner, J., & Bezzo, N. (2025). *Implicit Coordination using Active Epistemic Inference for Multi-Robot Systems*. arXiv:2501.03907.

[18] Löwe, B., Pacuit, E., & Witzel, A. (2010). *Planning Based on Dynamic Epistemic Logic*. In Proceedings of the Workshop on Logic and the Simulation of Interaction and Reasoning (LSIR), IJCAI 2009.

[19] Ulises, H. (2026). *A DEL-Based Epistemic Planner for Multi-Agent Robotic Systems*. Technical Report, IPC 2026 Epistemic Planning Track. Instituto Politécnico Nacional – ESCOM / Universidad Nacional Autónoma de México – FFyL.