PlanOwl: Automated PDDL Files Generation from OWL Ontologies and Visual Language Models

Mark Adamik[1*], **Paolo Forte**[2*]

[1]Faculty of Science, Vrije Universiteit Amsterdam, Amsterdam, The Netherlands
[2]Centre for Applied Autonomous Sensor Systems, Örebro University, Örebro, Sweden

Abstract

Automated task planning traditionally relies on manually generated domain models, creating bottlenecks in scalability and requiring extensive domain expertise. This paper presents a novel framework to automate the process of generating Planning Domain Definition Language (PDDL) domains and problem files by integrating Web Ontology Language (OWL) ontologies with Visual Language Models (VLMs). Our approach leverages the rich semantic structure of OWL ontologies to systematically define domain classes, predicates, and actions, while VLMs ground abstract ontological concepts in concrete visual observations—automating the generation of instance-specific planning problems. The proposed framework transforms ontological knowledge into PDDL domain files through a mapping algorithm that preserves semantic relationships and logical constraints. The VLM performs visual scene analysis to identify relevant objects, attributes, and spatial configurations for generating initial states, while natural language instructions are used to derive goal states. We evaluate the framework across multiple planning domains, demonstrating that it generates syntactically correct and semantically coherent PDDL domain and problem files directly from OWL ontologies, camera images, and natural language inputs. The resulting files are comparable in quality to those manually generated by planning experts and outperform previous automated systems in terms of semantic fidelity and adaptability.

Introduction

Automated task planning remains one of the fundamental challenges in artificial intelligence, requiring systems to reason about complex environments and available actions to generate executable action sequences to reach desired goal states. Consider a simple task planning problem involving a robot arm, as in Figure 1: the robot must arrange objects on a table according to specific instructions, such as ”place all the fruits in the bowl.” Traditional approaches would require domain experts to manually define PDDL (Ghallab et al. 1998) domain files specifying object types (fruits, containers), predicates (on, in, clear), and actions (pick, place), along with problem files detailing the initial state and goal configuration. This process is not only time-consuming and error-prone, but also demands significant domain expertise. Furthermore, this process must be repeated for every new task or environment, presenting a substantial scalability challenge (Jiménez et al. 2012). Classical planning methods also assume closed-world semantics, where the initial and possible future states are explicitly defined as finite first-order interpretations. This requires a simplified and complete abstraction of the real world—an assumption that becomes unrealistic in more complex or dynamic environments where agents possess only partial knowledge.

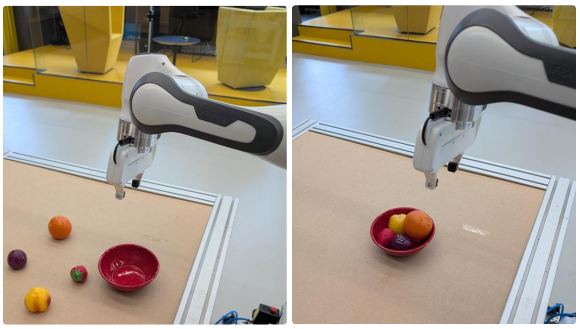


Figure 1: Robot used for the object arrangement scenario.

The integration of semantic knowledge representation through ontologies presents a promising approach to overcoming these limitations. Web Ontology Language (OWL) ontologies (Baader et al. 2017) provide rich, machine-readable representations of domain knowledge, capturing not only the structure of entities and their relationships but also semantic constraints and logical inference rules. In contrast to classical planning, ontologies offer a formalism for representing complex domains under open-world semantics, where the knowledge base is inherently incomplete and can be extended with new facts over time. This allows agents to reason more flexibly and robustly in uncertain or evolving scenarios.

Recent advances in visual language models (VLMs) have demonstrated remarkable capabilities in understanding and reasoning about visual scenes, bridging the gap between perception and symbolic representation for solving combined task and motion planning problems (Xie et al. 2023; Liu et al. 2023). These models can interpret complex visual in-

[*]These authors contributed equally.

formation and generate structured descriptions that capture spatial relationships, object properties, and contextual information—all essential components for effective task planning in real-world environments. However, these approaches rely heavily on manual knowledge engineering and expert input, making domain adaptation impractical. Moreover, they do not address long-term task planning, which becomes increasingly intractable as problem complexity grows (Chen et al. 2024). Despite significant individual progress, the integration of ontological knowledge representation with visual scene understanding for the automated generation of planning domain and problem files remains largely unexplored. Existing methods tend to operate in isolation: ontology-based planning systems typically depend on predefined semantic models without any visual grounding (Mayr, Rovida, and Krueger 2023), whereas vision-based approaches often lack the rich semantic structure and reasoning capabilities offered by ontologies (Aregbede et al. 2025).

To address this gap, we propose PlanOwl, a novel framework that integrates OWL ontologies with VLMs to automatically generate both planning domains and problem files for combined task and motion planning problems. PlanOwl exploits the semantic expressiveness of ontologies to define the structure of the planning domain and leverages VLMs to ground this abstract knowledge in real-world visual observations, enabling the generation of problem files that reflect the current environment. By translating semantic and visual inputs into PDDL representations, the framework bridges the gap between high-level knowledge and executable planning models. The key contribution of this work is PlanOwl, a unified framework that integrates a knowledge graph, a vision-language model, and a task planner within a behavior tree structure, with two main purposes: (1) to automatically generate planning domain files from ontologies and contextually grounded problem files from images, natural language instructions, and semantic knowledge; and (2) to use the generated files to compute an execution plan that achieves the desired goal states.

Related Work

LLM and VLM for Task Planning

Both Large Language Models (LLMs) and Visual Language models (VLMs) have shown remarkable capabilities in everyday tasks, increasing interest in their application in different fields, such as robotics. Several works have explored using LLMs as planners (Huang et al. 2022). ProgPrompt (Singh et al. 2023) uses programming-like structures to inform LLMs about available actions and environmental context, enabling the generation of executable task plans. Text2Motion (Lin et al. 2023) combines LLMs with learned skills and geometric reasoning for long-horizon manipulation tasks. LLM-Planner (Song et al. 2023) uses LLMs for few-shot planning grounded in physical environments. It combines LLMs with physical grounding to create and adjust plans using real-world environmental information. Despite these advances, LLM-based approaches remain fundamentally limited compared to classical planners. Their generative nature lacks the systematic search and reasoning capabilities required for reliable long-horizon planning (Hazra et al. 2024). As a result, LLM-generated plans often contain errors and fail to guarantee successful task execution or goal achievement.

Rather than relying on LLMs as standalone planners, a promising alternative is to use them for generating planning goals (Xie et al. 2023) or complete task planning inputs—such as domain and problem description files—which are then passed to classical planners. Traditionally, these files, which define actions, objects, initial states, and goals, are manually authored by planning experts—a process that is time-consuming, error-prone, and difficult to scale. To address this, DELTA (Liu et al. 2025) integrates LLMs, scene graphs, and automated planners by translating natural language and visual inputs into PDDL problem and domain files. DELTA uses LLMs to decompose complex goals into subgoals, reducing planning complexity. However, it depends on pre-constructed scene graphs for object retrieval and assumes natural language input for domain generation. Similarly, LLM+P (Liu et al. 2023) converts natural language descriptions of scenes into PDDL, solves the resulting problem with a classical planner, and outputs the plan in natural language. Nonetheless, it assumes that the entire scene—including objects and initial state—is fully described in natural language, which limits its applicability in real-world scenarios. ViLaIn (Shirai et al. 2024) introduces a modular pipeline that handles object detection, initial state estimation, and goal generation using separate LLM components refined via planner feedback. However, it requires a known object list and multiple LLM queries per task. ViPlan (Aregbede et al. 2025) integrates vision-language models with classical planners to automatically generate problem files from images and natural language instructions. Embedded within a behavior tree, ViPlan orchestrates object detection, problem generation, planning, and execution with built-in failure recovery. Despite these strengths, ViPlan still presents two key limitations: (1) domain files must be manually provided; and (2) problem prompts are static and must be manually adapted for each planning domain.

Knowledge Graph and Task Planning

Combining semantic technologies and classical planning has been an active domain of research for decades (Jiming Liu, Chi Kuen Wong, and Kin Hang Tsang 2005) and has been used from manufacturing systems (Al-Safi and Vyatkin 2007) through configuring automated web services (Ďurčík and Paralič 2011) to industrial kitting and assembly scenarios involving robotic agents (Balakirsky et al. 2013; Kootbally et al. 2015). Several frameworks have explored this integration by combining high-level reasoning with execution-level reactivity. In recent years, ontology-mediated planning has emerged as a structured interface between OWL-DL and PDDL, facilitating an integration between these languages while keeping them separated (John and Koopmann 2024). PlanOnto (Muppasani et al. 2024) has been developed as an ontology that formalizes and stores both domain and planning problems in OWL, and supports reasoning about planner suitability for specific problems via SPARQL queries. However, a key limitation of this approach is its inability

to explicitly and semantically represent the parameters, preconditions, and effects of domain actions. These components are treated as a single undifferentiated entity, which restricts the ontology's expressiveness and its utility for structured domain reconstruction. SkiROS2 (Mayr, Rovida, and Krueger 2023) is a ROS-based, open-source control platform that integrates symbolic task planning with reactive execution via extended behavior trees. It uses a skill model with semantic pre-, hold-, and post-conditions to support reasoning, parameter inference, and modular skill composition. The architecture includes a shared ontology-based world model (OWL/RDF) and automated PDDL generation. However, the list of objects in each scene is retrieved from a world model that maintains a persistent, explicitly structured representation of the environment. As this list is not generated directly from sensor data, the system is less adaptable to previously unseen objects.

Preliminary Concepts

The proposed framework adopts PDDL 2.1 (Fox and Long 2003) to model planning problems, which are specified through two main components: a domain file and a problem file. The ontology is defined using the OWL DL language (Bechhofer 2009), providing a formal and expressive framework for representing domain knowledge, including classes, properties, and constraints relevant to the planning environment.

PDDL

The domain file defines the structure of a planning problem through two fundamental components: predicates, which capture object properties and relationships, and actions, which describe the possible operations along with their preconditions and effects. Formally, a PDDL domain $\mathcal{D}$ is defined as a tuple $\mathcal{D}= \langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$ is the set of predicates and $\mathcal{A}$ is the set of actions.

The problem file defines a specific planning instance by specifying the objects involved, the initial state of the world, and the goal conditions to be achieved. It provides the context in which the domain actions are applied, enabling the planner to generate a valid sequence of actions that transforms the initial state into the goal state. Formally, a PDDL problem $\mathcal{P}= \langle \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$, where $\mathcal{O}$ is the set of objects, $\mathcal{I}$ is the initial state, formed by applying predicates to arguments (e.g., *(on green_block purple_block)*), and $\mathcal{G}$ is the goal state, expressed as a first-order logic formula (e.g., *(on yellow_block green_block) (on green_block red_block)*), meaning the green block should be placed on top of the red block and the yellow block on top of the green block.

Knowledge Graph

A knowledge graph (Hogan et al. 2020) is a directed, labelled graph $\mathcal{K} = (\mathcal{V}, \mathcal{J}, \mathcal{E})$, whose nodes $\mathcal{V}$ represent entities (classes, individuals, literals), $\mathcal{J}$ represents a set of properties and whose edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{J} \times \mathcal{V}$ encode relationships. Each edge $(v_1, p, v_2) \in \mathcal{E}$ encodes that node v_1 is connected to v_2 via property p. An OWL Knowledge Graph is a structured representation of entities (nodes) and their relationships (edges), grounded in the W3C's Web Ontology Language (OWL) (Noy and McGuinness 2001). OWL enables users to: (1) define classes (e.g., :Person), properties (e.g., :livesIn), and constraints (e.g., disjointness, cardinality); (2) instantiate individuals (e.g., :Alice rdf:type :Person) with associated data or object properties; and (3) perform automated reasoning, allowing inference of new facts, classification of instances, and detection of logical inconsistencies based on OWL's formal semantics. To construct an OWL knowledge graph, an ontology is written using formats such as Turtle, RDF/XML, or functional syntax, then loaded into a triple store (e.g., GraphDB, Stardog), and queried using SPARQL to access both asserted and inferred knowledge. These graphs support semantic search, data integration, and intelligent reasoning, making them ideal for applications that require understanding complex domain rules.

Problem Statement

The system input $\mathcal{S}$ is defined as a tuple $\langle \mathcal{L}, \mathcal{C} \rangle$, where $\mathcal{L}$ denotes a linguistic instruction (i.e., a natural language description of the tasks) and $\mathcal{C}$ represents a scene observation (an RGB image capturing the environment's state). The architecture of the proposed framework is shown in Figure 2.

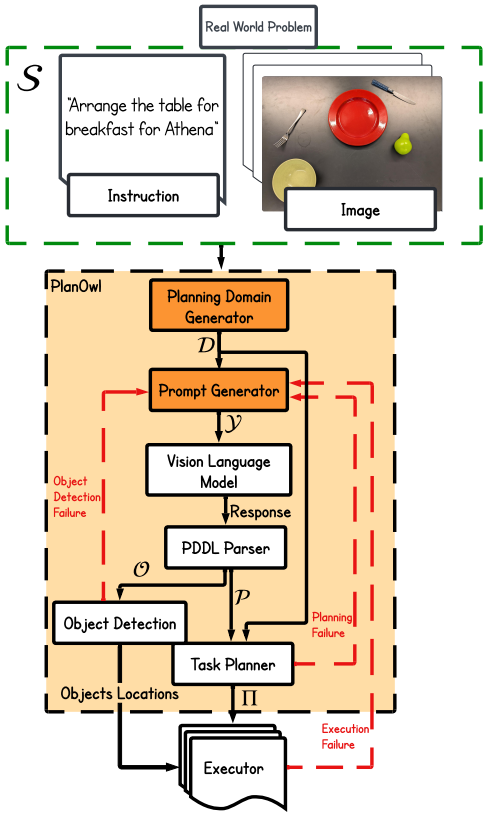


Figure 2: Architecture of PlanOwl. Contributions are highlighted in orange. White boxes represent modules retained from (Aregbede et al. 2025). In case of planning, object detection, or execution failure, the VLM system is reprompted with a new prompt to generate a new PDDL problem file.

The system produces three main outputs, generated sequentially as follows:

1. **Planning domain file** $\mathcal{D}$: This file defines the domain-specific components of the planning problem, including the set of predicates, types, and actions. It encodes the symbolic representation of the robot's capabilities and the rules governing the environment's dynamics.
2. **Planning problem file** $\mathcal{P}$: This file specifies the planning instance to be solved. It contains the list of objects $\mathcal{O}$ present in the scene, the initial state $\mathcal{I}$, and the goal conditions $\mathcal{G}$.
3. **Execution plan** $\Pi = \langle a_1, \ldots, a_n \rangle$: This is an ordered sequence of symbolic actions, where each action $a_i \in \mathcal{A}$, representing the steps the robot must follow to transition from the initial state to the desired goal state. This plan is generated by a classical planner using the generated domain and problem files.

Unifying the Planning Ontology

To tackle the challenge of automatic planning domain generation, we integrate two complementary systems: (1) the Planning Ontology (PO) (Muppasani et al. 2024), which enables the semantic modeling of planning domains and problems in OWL, and (2) the Unified Planning (UP) library (Micheli et al. 2025), a Python library that provides tools for formulating and manipulating planning problems. While PO offers a rich, high-level representation, it lacks the detailed structure needed for seamless integration with UP. To bridge this gap, we extend PO with additional classes tailored for UP compatibility. This approach enables bidirectional conversion—allowing OWL files to be generated from planning problems and planning problems to be reconstructed from semantic descriptions—thus combining the strengths of both frameworks. This section details the changes made to PO to make the integration possible, while the practical functionality and utility are described in the following section.

Extended Classes and Properties

To accurately capture the nested and compositional structure of action effects and preconditions, we introduce new classes and properties that extend the planning ontology. While PO lacks native support for representing complex logical formulae, such expressiveness is essential for interoperability with unified planning and for faithfully modeling planning domains. Our extension introduces a general `FormulaNode` class, which is specialized by two subclasses: `LogicalConnective` (for representing `And`, `Or`, and `Not` operators) and `AtomicFormula` (for atomic predicates with arguments). The `hasRootNode` object property links `ActionEffect` and `ActionPrecondition` instances to their corresponding formula trees. Furthermore, action effects in UP are represented as value assignments to domain predicates, taking the form `handfull(robot) := false`. To support this, we introduce an `AssignmentEffect` class, which connects the effect to the affected domain predicate via the `assignsFluent` object property and specifies the assigned value (such as a boolean) using the `assignsValue` data property.

For our purposes—retrieving, constructing, and modifying domain files automatically—we also introduce a `belongsToDomain` predicate. This property makes the connection between each domain component and its corresponding domain explicit, significantly speeding up the reconstruction and retrieval process. This design enables the ontology to capture arbitrarily complex and nested logical conditions, as well as explicit value assignments, facilitating full round-trip conversion between OWL and UP. An overview of these extensions and their integration with PO is shown in Figure 3.

Additional Considerations

One of the key challenges in translating between planning representations is the preservation of argument order for predicates and action effects. For example, in predicates such as `smaller(?x, ?y)`, simply associating arguments via a generic `hasArgument` property is insufficient, as the positional information is essential for correct interpretation and therefore reconstruction. To address this, we introduce a `hasArgumentIndex` data property, which assigns a non-negative integer to each `DomainPredicateParameter` instance. This ensures that the ordering of parameters is explicitly recorded within the ontology, enabling lossless and unambiguous mapping between OWL and PDDL/UP representations. In addition, one of the main benefits of using a knowledge representation formalism such as OWL is the ability to express rich metadata and annotations. To leverage this, we enable each domain node to be enriched with a textual description through the `hasDomainDescription` annotation property. This not only aids in the semantic differentiation between domains but also supports documentation and search functionalities. Furthermore, while the raw syntax of PDDL and the corresponding UP components may differ in representation, we explicitly preserve the original PDDL formulation using the `hasSyntax` property. This approach ensures that the native syntax is always available for parsing tools and for maintaining compatibility with existing planning software.

While a complete integration of the two frameworks is still a work in progress, the changes to PO described above already enable a seamless integration with the UP framework, resulting in a semantically rich and interoperable knowledge representation.

Proposed System

Our approach builds upon two prior systems: PlanOnto (Muppasani et al. 2024), which we extend and integrate with the Unified Planning library, and ViPlan (Aregbede et al. 2025), which employs a behavior tree structure for modular and reactive task planning and execution. By embedding the enhanced ontology into ViPlan, we enable more advanced knowledge-based planning and reasoning capabilities.

Our method introduces three key innovations. First, we expand PlanOnto by introducing new ontology classes that

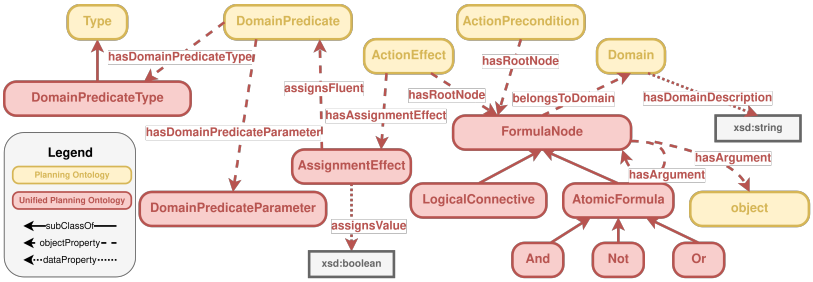


Figure 3: The main classes and object properties extending the Planning Ontology of (Muppasani et al. 2024).

enable the semantic reconstruction of planning domains and facilitate integration with the Unified Planning library. Second, unlike ViPlan, where planning domain files are manually defined, our framework automatically generates them from a knowledge graph. Third, we replace ViPlan's static planning prompts with dynamically generated ones, automatically adapted to the current context. This design enables our system to generalize across diverse domains without manual reconfiguration.

Our approach integrates both the generation of PDDL files and the planning process within a unified behavior tree framework (see Figure 4), resulting in a system that emphasizes efficiency, modularity, flexibility, and reactivity.

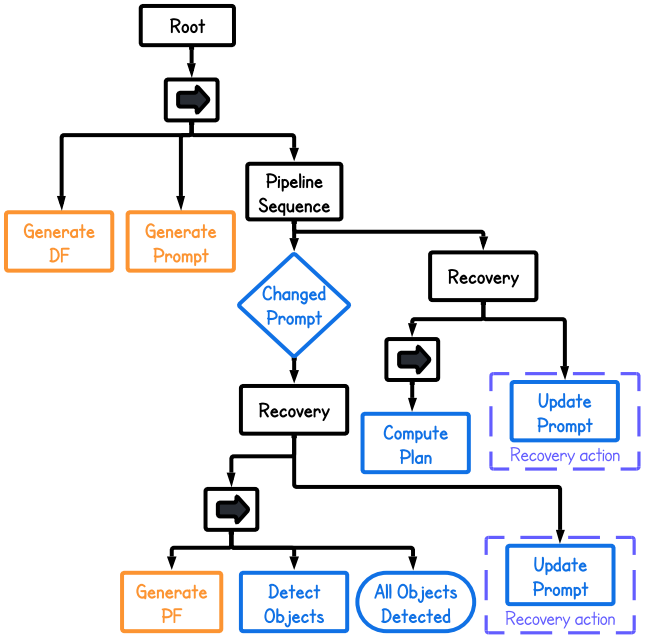


Figure 4: The proposed behavior tree framework, following the formalism of (Colledanchise and Ögren 2017). Novel nodes are highlighted in orange. Recovery and Pipeline Sequence nodes are as in (Macenski et al. 2020), while nodes highlighted in blue are based on (Aregbede et al. 2025).

A summary of the custom behavior tree nodes used in the system is provided in Table 1. The framework supports interchangeable components, allowing for the selection of different planners to compute execution plans and various VLMs to extract information from the current scene. To reduce redundant computation and support context-aware operation, we adopt the *Changed Prompt* decorator node introduced in (Aregbede et al. 2025), which triggers re-computation of the planning problem file only when the prompt provided to the VLM has changed.

Generate DF (A)	Generate a PDDL domain file.
Generate Prompt (A)	Generate the prompt for the VLM.
Generate PF (A)	Generate a PDDL problem file.

Table 1: Novel action (A) behavior tree nodes.

The workflow of the proposed system is divided into 4 steps:

Step 1: Domain File Generation

The first step of PlanOwl is to generate a planning domain file $\mathcal{D}$ using the OWL ontology. To generate actions $\mathcal{A}$ and predicates $\mathcal{T}$, the system must first determine which planning domain is most appropriate for the user's needs. To achieve this, the system takes the linguistic instruction $\mathcal{L}$, and searches for the corresponding domain in the ontology to construct $\mathcal{D}$. Specifically, each domain individual in the ontology is annotated with a `hasDomainDescription` data property, which stores a natural language description of the domain. Both this description and the $\mathcal{L}$ are embedded using a SentenceBERT model (Reimers and Gurevych 2019). The system then performs a similarity search and selects the domain whose description embedding is closest to the input embedding. Once the most relevant domain is identified, the corresponding planning domain is reconstructed using Unified Planning's (UP) native methods. An outline of this system component can be seen in Figure 5.

Step 2: VLM Prompt Generation

The second step involves generating a prompt $\mathcal{Y}$ for the VLM. As shown in Figure 6, the prompt consists of three main components: (1) a fixed template for generating PDDL planning problems, (2) the planning domain file $\mathcal{D}$ produced in the previous step, and (3) the natural language instruction $\mathcal{L}$ that describes the task to be accomplished.

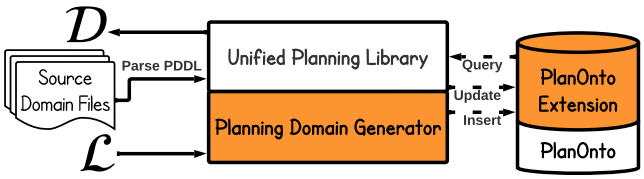


Figure 5: An outline of the planning domain generator elements. Given a set of domain files contained in the knowledge graph, the generator returns and reconstructs the closest match to the system input $\mathcal{L}$.

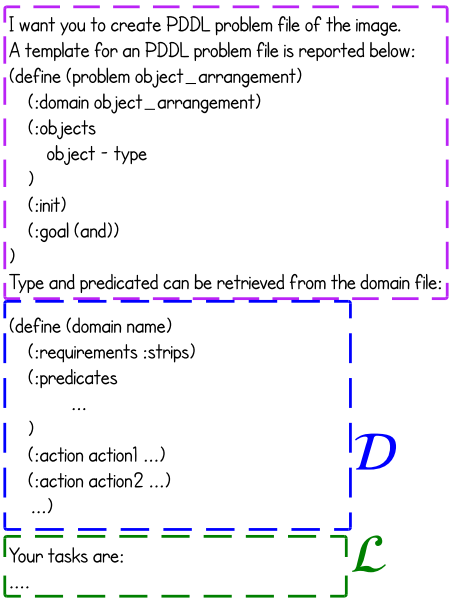


Figure 6: The structure of the prompt to the VLM. Fixed part is highlighted in purple, the domain file $\mathcal{D}$ in blue, and the natural language instruction $\mathcal{L}$ in green.

Step 3.1: Problem File Generation

The third step involves generating a PDDL planning problem file $\mathcal{P}$ using a VLM and the prompt assembled in the previous step. Instead of depending on a specific VLM, the framework offers a standardized plugin interface, enabling users to integrate their preferred model. The RGB image serves as visual input, providing contextual information about the current environment, is used to infer the `:init` section of the problem file, while the natural language prompt specifies the desired task or goal, such as object arrangement or navigation to a target location, and is used to generate the `:goal` section. The predicate syntax is directly derived from the previously generated planning domain file $\mathcal{D}$, ensuring syntactic and semantic consistency across the domain and problem representations.

Step 3.2: Object Detection

The object detection module serves two purposes: (1) verifying that the objects specified in the PDDL file correspond to those present in the input image; if a mismatch is detected, the system return to step 2 and generate a new prompt (see Figure 7); (2) estimating the positions of the detected objects, used during the execution of the plan, by leveraging GroundedSAM2 (Ren et al. 2024).

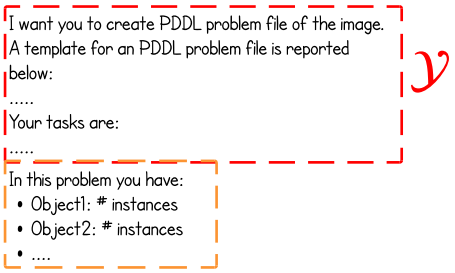


Figure 7: Structure of the updated prompt to the VLM. The original prompt $\mathcal{Y}$ is highlighted in red, while the added part is highlighted in orange.

Step 4: Execution Plan Computation

The fourth step of PlanOwl is computing an execution plan. The PDDL problem file $\mathcal{P}$ produced in Step 3.1, along with the domain file $\mathcal{D}$ generated in Step 1, are passed to a task planner. The planner first verifies the syntactic validity of both files. If they are valid, it proceeds to generate an execution plan based on the provided specifications. In case planning fails, the system enters a recovery loop: it returns to Step 2 and re-prompts the VLM, appending a new predicate that captures the cause of the failure. This process is repeated for a configurable number of iterations, defined as a system parameter. If a valid plan cannot be generated within this limit, the system requests user intervention to help generate correct planning problem and domain files. Otherwise, it advances to the execution phase using the computed plan.

Experimental Validation

Benchmark Problems

We considered three robot planning domains[1]: Hanoi, Block-world, and Object Arrangement (Figure 8). A brief summary of each is provided below.

Block-World Problem: This domain involves rearranging a set of blocks to reach a specified goal configuration (Gupta and Nau 1992). Blocks can either be stacked on top of one another or placed directly on the table.

Hanoi Problem: This domain requires moving a stack of disks from a source peg to a target peg, subject to two constraints: only one disk may be moved at a time, and a disk can only be placed on an empty peg or atop a larger disk (Hinz et al. 2013).

Object Arrangement Problem: This domain focuses on arranging objects in their designated locations, such as placing food items into containers.

[1]https://github.com/PaoloForte95/planning_domains

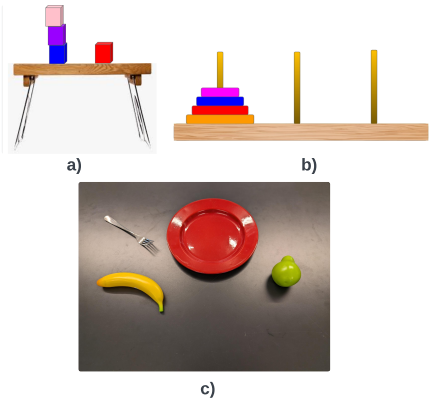


Figure 8: Example of the blockworld (a), hanoi (b), and object rearrangement (c) planning problems.

Setup

We employ the model Gemini 2.5 Pro[2] to generate the planning problem files and the Fast Downward planner (Helmert 2006) to create the execution plan. Real-world experiments are carried out using the robot arm shown in Figure 1 for the object arrangement scenario, while simulation experiments are used to validate the system in the Blockworld and Hanoi planning domains. The proposed system[3] is implemented in Python and C++ and runs on an Intel Core i9-10885H CPU 2.40GHz × 16 processor with 32 GiB of memory.

Metrics

We evaluate our framework by assessing the quality of the generated domain and problem files, as well as the resulting execution plan. To measure the solution quality, we adapt and extend the evaluation metrics proposed in (Aregbede et al. 2025) to better suit our system and use case.

1. A correctness metric $F^{\text{plan}} \in \{0, 1\}$, which evaluates whether an execution plan can be successfully generated and validated by a task planner. This metric ensures that the plan achieves the specified goal state without introducing unintended actions. Specifically, $F^{\text{plan}} = 1$ if a valid and feasible plan exists and leads to the desired goal; otherwise, $F^{\text{plan}} = 0$ if the plan is incorrect, infeasible, or cannot be generated.
2. A syntax correctness metric $F^{\text{syn}} = 0.5 \cdot F_p^{\text{syn}} + 0.5 \cdot F_d^{\text{syn}}$, with $F^{\text{syn}} \in \{0, 1\}$, which assesses whether the generated PDDL problem and domain files can be successfully parsed by a task planner. Here, $F_{p,d}^{\text{syn}} \in \{0, 1\}$ denote the syntax validity of the problem and domain files, respectively. The metric $F^{\text{syn}} = 1$ only if both $F_p^{\text{syn}} = 1$ and $F_d^{\text{syn}} = 1$ have no syntax errors.
3. A balanced F-score as in (Aregbede et al. 2025):

$$F_1^{pd} = 2 * \frac{\text{p} * \text{r}}{\text{p} + \text{r}} \quad p = \frac{T_r}{T_{ta}} \quad r = \frac{T_r}{T_{tr}} \tag{1}$$

where T_r represents the total correctly identified instances (init, goal, object), T_{ta} refers to the number of instances extracted in the generated problem, and T_{tr} denotes the total ground truth items.

System		$\mathbf{F_1}^{pd}$			$\mathbf{F}^{plan}$	$\mathbf{F}^{syn}$
		$\mathcal{O}$	$\mathcal{I}$	$\mathcal{G}$		
ViPlan	**H**	100	95.6	100	80	100
	B	100	88.6	100	80	100
	O	98.0	98.0	98.0	80	100
	LO	93.0	93.0	93.0	0	100
	MO	87.8	87.8	87.8	0	100
	R	100	100	100	100	100
PlanOwl	**H**	95.6	97.4	98.4	80	100
	B	100	100	100	100	100
	O	100	100	100	100	100
	LO	100	100	100	100	100
	MO	98.8	98.8	98.4	0	100
	R	100	100	100	100	100

Table 2: Solution quality (%) for the Hanoi (H), Blockworld (B), and Object Arrangement (O) problems. For the latter, additional variants are considered: a large number of objects (LO), multiple objects of the same type (MO), and predefined relationships (R) between objects.

Results

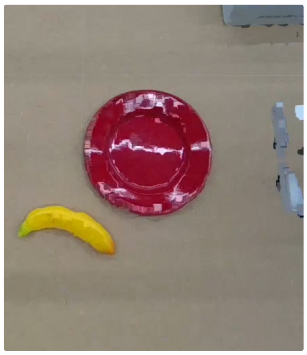
Table 2 report the solution quality metrics achieved by the proposed framework across the three domains introduced in the previous section. These results reflect the system's performance prior to the object detection stage (i.e., before Step 3.2), meaning no explicit object detection has yet been applied. Furthermore, since only three distinct domains were present in the knowledge base, the planning domain generator achieved a perfect accuracy, consistently returning the correct domain. As shown in the tables, our framework outperforms the baseline ViPlan (Aregbede et al. 2025) across all evaluated metrics. This improvement can be attributed not only to our architectural contributions but also to recent advancements in visual language models, which have demonstrated notable improvements in both visual understanding and language generation capabilities—resulting in more accurate and coherent domain and problem file generation. One limitation observed in the results is a metric degradation in one of the Hanoi tasks due to an object detection error, which affected the correctness of the generated plan. However, the system still demonstrated improved accuracy in generating initial conditions for planning, highlighting its robustness even in the presence of partial perception errors. A key contribution of this experiment is demonstrating PlanOwl's ability to automatically adapt its prompt and planning strategy to different domains. Unlike the baseline in (Aregbede et al. 2025), which requires manually switching between domain files and rewriting prompts for each scenario, our framework autonomously selects the appropriate domain and generates a context-aware prompt based on the scene and task description. This significantly enhances

[2]We use the most recent model as of August 2025

[3]https://github.com/PaoloForte95/athena

the generality and scalability of the approach, making it suitable for multi-domain robotic applications without manual intervention.

Figure 9 illustrates the online adaptation process of PlanOwl. In the initial generated problem file (Figure 9 b)) additional objects were incorrectly included. This mismatch is detected during object pose estimation using Grounded-SAM2. The detection triggers a recovery action, which updates the prompt to the VLM, prompting the system to regenerate the PDDL problem file (Figure 9 c)). In the updated prompt, the correct number of detected objects is explicitly included, ensuring that the regenerated file is consistent with the environment.



a)

```
(define (problem object_arrangement)
  (:domain object_arrangement)
  (:objects
    plate1 plate2 - container
    banana1 banana2 - food
    robot1 - robot
    wp1s wp2s wp3s wp4s home1
    wp1f wp2f - location)
  (:init (free robot1)
    (at robot1 home1)
    (at plate1 wp1s)
    (at plate2 wp2s)
    (at banana1 wp3s)
    (at banana2 wp4s))
  (:goal (and
      (at plate1 wp1f)
      (at plate2 wp2f)
      (on banana1 plate1)
      (on banana2 plate2))))
```

b)

```
(define (problem object_arrangement)
  (:domain object_arrangement)
  (:objects
    plate - container
    banana - food
    robot1 - robot
    wp1s wp2s home1 wp1f- location)
  (:init (free robot1)
    (at robot1 home1)
    (at plate wp1s)
    (at banana wp2s)
  (:goal (and
      (at plate wp1f)
      (on banana plate))))
```

c)

Figure 9: Example of the updated prompt recovery action in response to an object detection failure: (a) current scenario, (b) initial PDDL problem file, and (c) regenerated PDDL problem file after recovery.

Limitations and Future Work

As our current extensions to the Planning Ontology still rely on already existing domain files to be added to the Knowledge Base, future work will focus on several key advancements aimed at supporting fully automatic domain construction and greater flexibility.

First, we aim to make the Planning Ontology more broadly compatible with Unified Planning (UP), allowing for the representation and reasoning over multiple planning problems within a single ontology. Future extensions will also focus on leveraging PO's native ability to encode information about the suitability of different planners for specific domains, a feature not currently exploited in our system. Finally, we intend to develop more autonomous methods for domain file construction by employing reusable action templates and discovering semantically similar predicates and actions across domains, standardizing common patterns (such as always using load for loading actions or `on` for spatial relationships). This will reduce manual modeling effort while enabling the creation of semantically rich and reusable planning knowledge.

Second, we plan to develop alignment modules for other robotics-oriented ontologies, such as (Beßler et al. 2020) and (Adamik et al. 2025), enabling the integration of user-specific preferences, richer reasoning over object properties and affordances, and explicit representation of robot capabilities. This will foster a more adaptive and context-aware construction of planning domains. For instance, distinct users may prefer different configurations for tasks like setting a breakfast table; our objective is to encode and dynamically apply such preferences during goal generation.

Third, we aim to explore automatic Knowledge Graph Construction methods (KGC) combined with Graph Retrieval Augmented Generation methods (Martorana et al. 2025) to explore possibilities of automatic domain file generation based on the extended planning ontology.

Conclusion

In this paper, we presented PlanOwl, a novel framework that bridges the gap between semantic knowledge representation and automated task planning and execution by integrating OWL ontologies with visual language models for automatic PDDL generation. Our approach systematically tackles the scalability bottleneck by automatically generating both domain and problem files from visual observations and natural language instructions, reducing the need for manual domain engineering while maintaining semantic consistency. The key contributions of this work include: (1) a systematic methodology to automatically generate, store and manipulate PDDL domain files from OWL ontologies by creating an integration of the Planning Ontology and the Unified Planning library, (2) an integrated framework that combines ontological knowledge with visual language models to create contextually grounded planning problems from images and natural language instructions, and (3) a reactive behavior tree architecture that enables dynamic adaptation and failure recovery during the planning process. The modular design of PlanOwl, built upon a behavior tree framework, provides flexibility in component selection and supports interchangeable planners and visual language models. Our experimental validation across three distinct planning domains—Blocksworld, Hanoi, and Object Arrangement—demonstrates the effectiveness of the proposed approach. The framework successfully generates syntactically correct and semantically meaningful PDDL files, achieving comparable performance to manually crafted domains while significantly reducing the manual effort required for domain modeling.

Overall, PlanOwl demonstrates how the fusion of ontological knowledge with neural models can advance automated planning, offering the community a practical path toward scalable, semantically grounded, and adaptive robot intelligence.

References

Adamik, M.; Pernisch, R.; Tiddi, I.; and Schlobach, S. 2025. ORKA: An Ontology for Robotic Knowledge Acquisition. In Alam, M.; Rospocher, M.; Van Erp, M.; Hollink, L.; and Gesese, G. A., eds., *Knowledge Engineering and Knowledge Management*, volume 15370, 309–327. Cham: Springer Nature Switzerland. ISBN 978-3-031-77791-2 978-3-031-77792-9. Series Title: Lecture Notes in Computer Science.

Al-Safi, Y.; and Vyatkin, V. 2007. An Ontology-Based Reconfiguration Agent for Intelligent Mechatronic Systems. In Mařík, V.; Vyatkin, V.; and Colombo, A. W., eds., *Holonic and Multi-Agent Systems for Manufacturing*, volume 4659, 114–126. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-540-74478-8 978-3-540-74481-8. ISSN: 0302-9743, 1611-3349 Series Title: Lecture Notes in Computer Science.

Aregbede, V.; Forte, P.; Gupta, H.; Andreasson, H.; Köckemann, U.; and Lilienthal, A. J. 2025. Here's your PDDL Problem File! On Using VLMs for Generating Symbolic PDDL Problem Files. *IEEE International Conference Robot. Automat. (ICRA)*.

Baader, F.; Horrocks, I.; Lutz, C.; and Sattler, U. 2017. *An Introduction to Description Logic*. Cambridge University Press.

Balakirsky, S.; Kootbally, Z.; Kramer, T.; Pietromartire, A.; Schlenoff, C.; and Gupta, S. 2013. Knowledge driven robotics for kitting applications. *Robotics and Autonomous Systems*, 61(11): 1205–1214.

Bechhofer, S. 2009. OWL: Web Ontology Language. In Liu, L.; and Özsu, M. T., eds., *Encyclopedia of Database Systems*, 2008–2009. Boston, MA: Springer US. ISBN 978-0-387-35544-3 978-0-387-39940-9.

Beßler, D.; Porzel, R.; Pomarlan, M.; Vyas, A.; Höffner, S.; Beetz, M.; Malaka, R.; and Bateman, J. 2020. Foundations of the Socio-physical Model of Activities (SOMA) for Autonomous Robotic Agents. Version Number: 1.

Chen, Y.; Arkin, J.; Dawson, C.; Zhang, Y.; Roy, N.; and Fan, C. 2024. AutoTAMP: Autoregressive Task and Motion Planning with LLMs as Translators and Checkers. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 6695–6702.

Colledanchise, M.; and Ögren, P. 2017. Behavior Trees in Robotics and AI: An Introduction. *ArXiv*, abs/1709.00084.

Fox, M.; and Long, D. 2003. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research*, 20: 61–124. ArXiv:1106.4561 [cs].

Ghallab, M.; Knoblock, C.; Wilkins, D.; Barrett, A.; Christianson, D.; Friedman, M.; Kwok, C.; Golden, K.; Penberthy, S.; Smith, D.; Sun, Y.; and Weld, D. 1998. PDDL - The Planning Domain Definition Language. *Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control*.

Gupta, N.; and Nau, D. 1992. On the Complexity of Blocks-World Planning. *Artif. Intell.*, 56: 223–254.

Hazra, R.; Venturato, G.; Martires, P. Z. D.; and Raedt, L. D. 2024. Can Large Language Models Reason? A Characterization via 3-SAT. ArXiv:2408.07215 [cs].

Helmert, M. 2006. The Fast Downward Planning System. *ArXiv*, abs/1109.6051.

Hinz, A. M.; Klavžar, S.; Milutinović, U.; and Petr, C. 2013. *The Tower of Hanoi - Myths and Maths*. Birkhäuser Basel.

Hogan, A.; Blomqvist, E.; Cochez, M.; d'Amato, C.; Melo, G. d.; Gutierrez, C.; Gayo, J. E. L.; Kirrane, S.; Neumaier, S.; Polleres, A.; Navigli, R.; Ngomo, A.-C. N.; Rashid, S. M.; Rula, A.; Schmelzeisen, L.; Sequeda, J.; Staab, S.; and Zimmermann, A. 2020. Knowledge Graphs. *ACM Computing Surveys (CSUR)*, 54: 1 – 37.

Huang, W.; Abbeel, P.; Pathak, D.; and Mordatch, I. 2022. Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents. In *Proceedings of the 39th International Conference on Machine Learning*, 9118–9147. PMLR. ISSN: 2640-3498.

Jiming Liu; Chi Kuen Wong; and Kin Hang Tsang. 2005. Ontoplan: semantic web based planning. In *Proceedings of the 2005 International Conference on Active Media Technology, 2005. (AMT 2005).*, 568–573. Takamatsu, Kagawa, Japan: IEEE. ISBN 978-0-7803-9035-5.

Jiménez, S.; De la Rosa, T.; Fernandez, S.; Fernández, F.; and Borrajo, D. 2012. A review of machine learning for automated planning. *The Knowledge Engineering Review*, 27.

John, T.; and Koopmann, P. 2024. Planning with OWL-DL Ontologies (Extended Version). Version Number: 1.

Kootbally, Z.; Schlenoff, C.; Lawler, C.; Kramer, T.; and Gupta, S. 2015. Towards robust assembly with knowledge representation for the planning domain definition language (PDDL). *Robotics and Computer-Integrated Manufacturing*, 33: 42–55.

Lin, K.; Agia, C.; Migimatsu, T.; Pavone, M.; and Bohg, J. 2023. Text2Motion: From Natural Language Instructions to Feasible Plans. *Autonomous Robots*, 47(8): 1345–1365. ArXiv:2303.12153 [cs].

Liu, B.; Jiang, Y.; Zhang, X.; Liu, Q.; Zhang, S.; Biswas, J.; and Stone, P. 2023. LLM+P: Empowering Large Language Models with Optimal Planning Proficiency. ArXiv:2304.11477 [cs].

Liu, Y.; Palmieri, L.; Koch, S.; Georgievski, I.; and Aiello, M. 2025. DELTA: Decomposed Efficient Long-Term Robot Task Planning using Large Language Models. ArXiv:2404.03275 [cs].

Macenski, S.; Mart'in, F. J. P.; White, R.; and Clavero, J. G. 2020. The Marathon 2: A Navigation System. *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2718–2725.

Martorana, M.; Urgese, F.; Adamik, M.; and Tiddi, I. 2025. Bridging Bots: from Perception to Action via Multimodal-LMs and Knowledge Graphs. Version Number: 1.

Mayr, M.; Rovida, F.; and Krueger, V. 2023. SkiROS2: A skill-based Robot Control Platform for ROS. ArXiv:2306.17030 [cs].

Micheli, A.; Bit-Monnot, A.; Röger, G.; Scala, E.; Valentini, A.; Framba, L.; Rovetta, A.; Trapasso, A.; Bonassi, L.; Gerevini, A. E.; Iocchi, L.; Ingrand, F.; Köckemann, U.; Patrizi, F.; Saetti, A.; Serina, I.; and Stock, S. 2025. Unified Planning: Modeling, manipulating and solving AI planning problems in Python. *SoftwareX*, 29: 102012.

Muppasani, B.; Pallagani, V.; Srivastava, B.; Mutharaju, R.; Huhns, M. N.; and Narayanan, V. 2024. A Planning Ontology to Represent and Exploit Planning Knowledge for Performance Efficiency. ArXiv:2307.13549 [cs].

Noy, N. F.; and McGuinness, D. L. 2001. Ontology Development 101: A Guide to Creating Your First Ontology. *Stanford Medical Informatics: Stanford University*.

Reimers, N.; and Gurevych, I. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. ArXiv:1908.10084 [cs].

Ren, T.; Liu, S.; Zeng, A.; Lin, J.; Li, K.; Cao, H.; Chen, J.; Huang, X.; Chen, Y.; Yan, F.; Zeng, Z.; Zhang, H.; Li, F.; Yang, J.; Li, H.; Jiang, Q.; and Zhang, L. 2024. Grounded SAM: Assembling Open-World Models for Diverse Visual Tasks. ArXiv:2401.14159 [cs].

Shirai, K.; Beltran-Hernandez, C. C.; Hamaya, M.; Hashimoto, A.; Tanaka, S.; Kawaharazuka, K.; Tanaka, K.; Ushiku, Y.; and Mori, S. 2024. Vision-Language Interpreter for Robot Task Planning. ArXiv:2311.00967 [cs].

Singh, I.; Blukis, V.; Mousavian, A.; Goyal, A.; Xu, D.; Tremblay, J.; Fox, D.; Thomason, J.; and Garg, A. 2023. ProgPrompt: Generating Situated Robot Task Plans using Large Language Models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 11523–11530.

Song, C. H.; Wu, J.; Washington, C.; Sadler, B. M.; Chao, W.-L.; and Su, Y. 2023. LLM-Planner: Few-Shot Grounded Planning for Embodied Agents with Large Language Models. ArXiv:2212.04088 [cs].

Xie, Y.; Yu, C.; Zhu, T.; Bai, J.; Gong, Z.; and Soh, H. 2023. Translating Natural Language to Planning Goals with Large-Language Models. ArXiv:2302.05128 [cs].

Ďurčík, Z.; and Paralič, J. 2011. Transformation of Ontological Represented Web Service Composition Problem into a Planning One. *Acta Electrotechnica et Informatica*, 11(2).