Interface for HDDL files and PANDA planner

Software Development Project

Team: Lou Lauter
Ayodeji Shittu

Supervisor: Aleksandar Mitrevski

Date: 9th March 2021

Scientific Part

PANDA Planning System

The PANDA planning system allows to solve different kinds of planning problems. The planning algorithm for all these problems is a hybrid planning approach, which fuses hierarchical planning with causal reasoning. It can solve the following classes of planning problems:

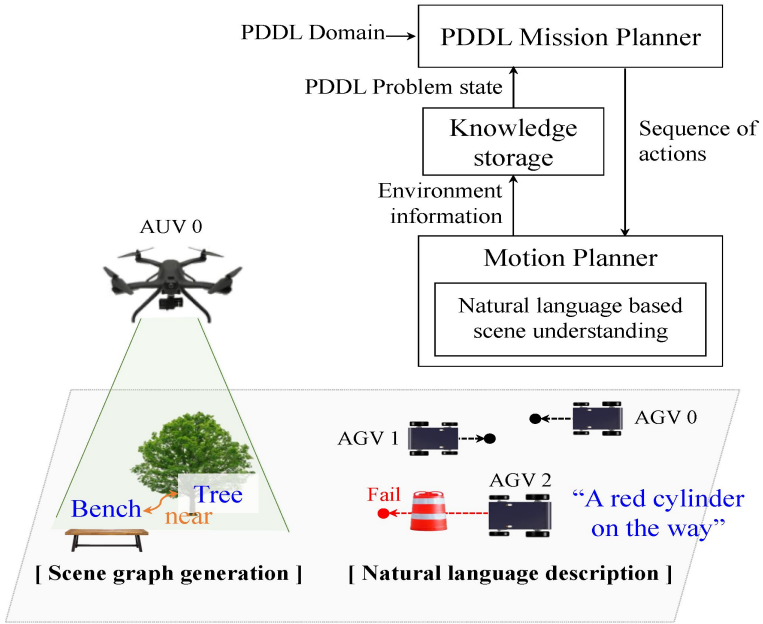
- hierarchical task network (HTN) problems, and
- hybrid planning problems

What is PDDL?

PDDL (Planning Domain Description Language) is a standard encoding language for “classical” planning.

The components of PDDL files are:

- **Requirements:** defining levels of abstraction in the language, e.g., "STRIPS", temporal, probabilistic effects etc.
- **Types:** sets of the things of interest in the world,
- **Objects:** instances of types,
- **Predicates:** Facts about objects that can be true or false,
- **Initial state** of the world: before starting the planning process,
- **Goal:** properties of the world true in goal states and achieved after the planning process,
- **Actions/Operators:** ways of changing states of the world and going from the initial state to goal states.



Domain Files

A **domain file** for requirements, types, predicates and actions.

```
(define (domain <domain name>)
    <PDDL code for requirements>
    <PDDL code for types>
    <PDDL code for predicates>
    <PDDL code for first action>
    [...]
    <PDDL code for last action>
)
```

Problem Files

A **problem file** for objects, initial state and goal specification.

```
(define (problem <problem name>)
  (:domain <domain name>)
  <PDDL code for objects>
  <PDDL code for initial state>
  <PDDL code for goal specification>
)
```

HDDL

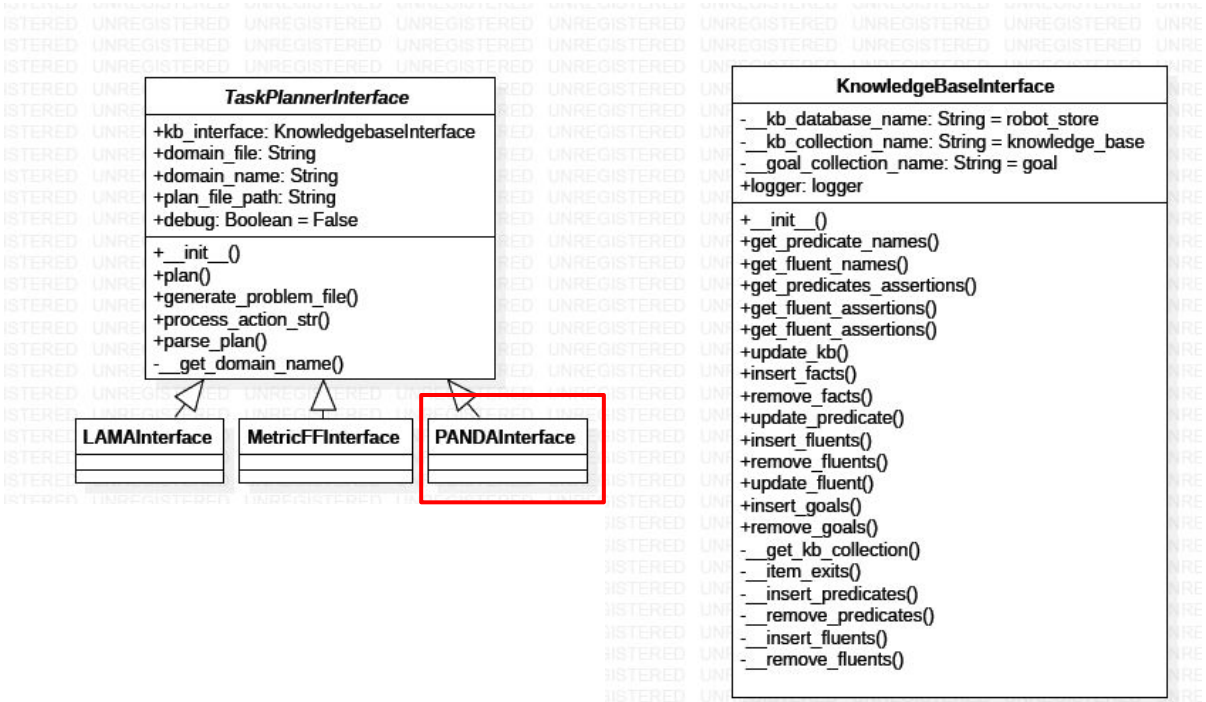
- HDDL is a hierarchical domain definition language.
- It is a common input language for hierarchical planning problems.
- It is widely based on the input language of PANDA, the framework underlying planning systems.

References

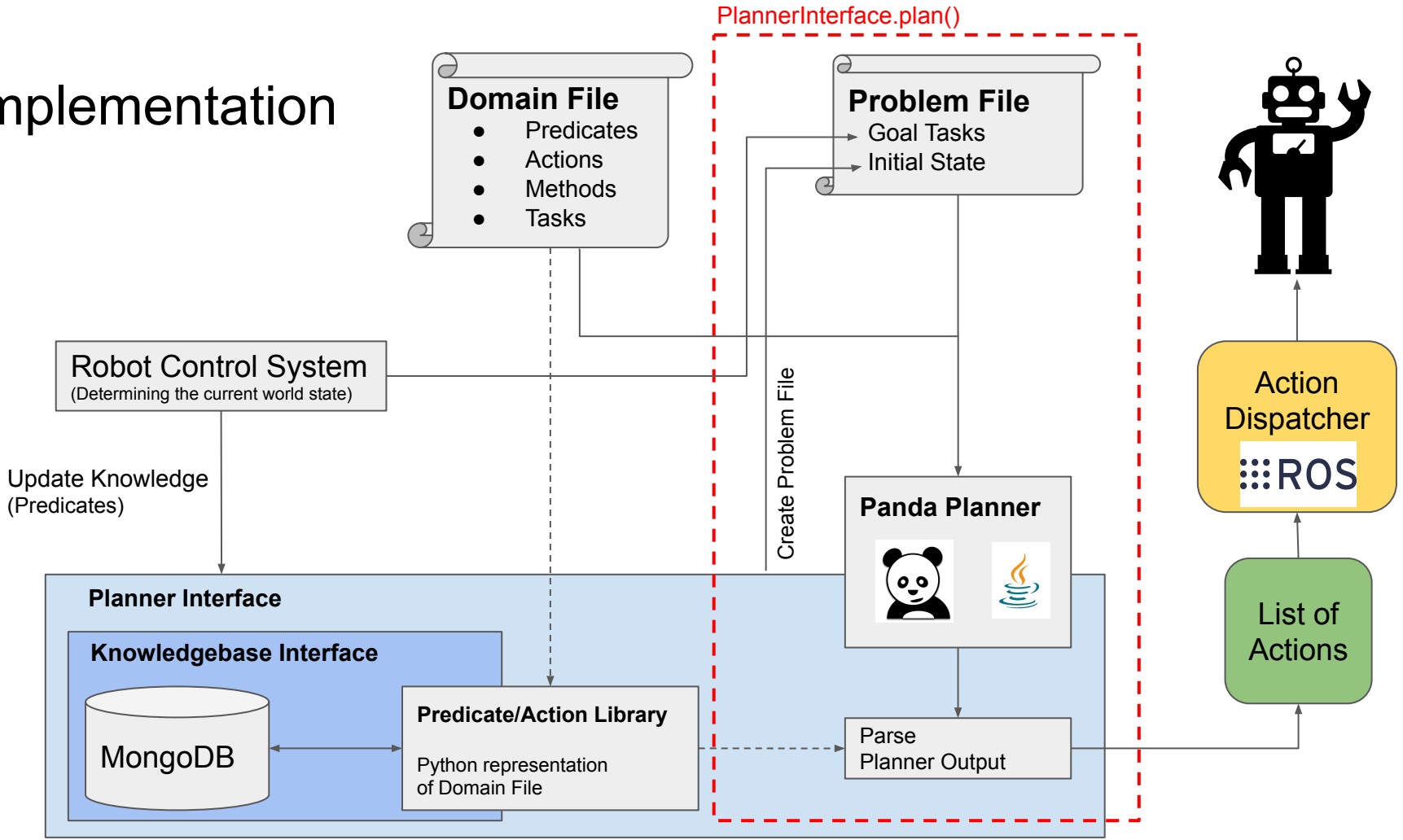
- https://www.cs.toronto.edu/~sheila/2542/s14/A1/introtopddl2.pdf
- https://algo2.iti.kit.edu/balyo/plan/files/getting-started-with-planning.pdf
- http://gki.informatik.uni-freiburg.de/papers/hoeller-etal-hplan19.pdf

Realization Part

Implementation



Implementation



Implementation

```
(:predicates
    (robotName ?Robot - Robot)
    (objectCategory ?Object0 - Object ?Object1 - Object)
    (robotAt ?Robot - Robot ?Waypoint - Waypoint)
    (doorAt ?Door - Door ?Waypoint - Waypoint)
```

```python
@staticmethod
def robotName(params: list, obj_types: dict) -> Tuple[list, dict]:
    param_order = {0: ('Robot', 'Robot')}
    return HDDLKnowledgeUtils.get_ordered_param_list(params, param_order, obj_types)

@staticmethod
def objectCategory(params: list, obj_types: dict) -> Tuple[list, dict]:
    param_order = {0: ('Object0', 'Object'), 1: ('Object1','Object')}
    return HDDLKnowledgeUtils.get_ordered_param_list(params, param_order, obj_types)

@staticmethod
def doorAt(params: list, obj_types: dict) -> Tuple[list, dict]:
    param_order = {0: ('Door', 'Door'), 1: ('Waypoint','Waypoint')}
    return HDDLKnowledgeUtils.get_ordered_param_list(params, param_order, obj_types)
```

Advantage of Python:
getattr(PredicateLibrary, "FUNCTION_NAME") → call function by its name

Implementation

```
(:action MoveBase
    :parameters (?Robot - Robot ?Location0 ?Location1 - Location)
    :precondition (and
        (robotAt ?Robot ?Location0)
    )
    :effect (and
        (not (robotAt ?Robot ?Location0))
        (robotAt ?Robot ?Location1)
    )
)

(:action Open
    :parameters (?Door - Door ?Robot - Robot ?Wayp
    :precondition (and
        (doorAt ?Door ?Waypoint)
        (robotAt ?Robot ?Waypoint)
    )
    :effect (and
        (doorOpen ?Door)
    )
)
```

Don’t care about preconditions and effects. The planner does the job!

```
@staticmethod
def MOVEBASE(action: Action, params: list) -> Action:
    action.parameter_order = ['Robot','Location0','Location1']
    return action

@staticmethod
def OPEN(action: Action, params: list) -> Action:
    action.parameter_order = ['Door','Robot','Waypoint']
    return action

@staticmethod
def PERCEIVEPLANE(action: Action, params: list) -> Action:
    action.parameter_order = ['Plane','Robot','Waypoint']
    return action
```

Implementation

```
(:action PickFromPlane
    :parameters (?Object - Object ?Plane - Plane ?Robot - Robot ?Waypoint - Waypoint)
    :precondition (and
        (robotAt ?Robot ?Waypoint)
        (planeAt ?Plane ?Waypoint)
        (explored ?Plane)
        (onTopOf ?Object ?Plane)
        (emptyGripper ?Robot)
    )
    :effect (and
        (not (onTopOf ?Object ?Plane))
        (not (emptyGripper ?Robot))
        (holding ?Robot ?Object)
    )
)
```

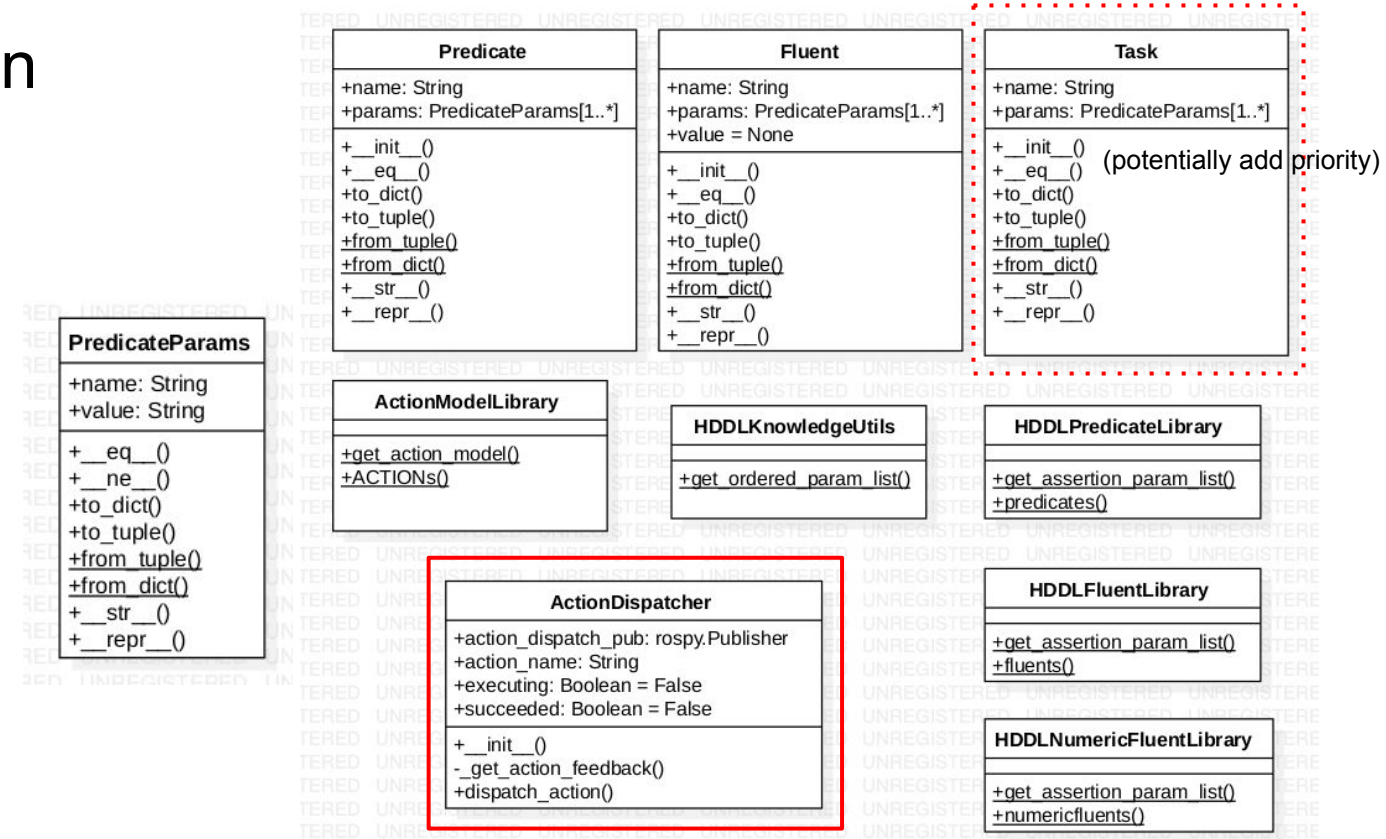
```python
class ActionModelLibrary(object):

    @staticmethod
    def PICKFROMFURNITURE(action: Action, params: list) -> Action:
        action.parameter_order = ['Object','Furniture','Robot','Waypoint']
        action.type = 'PICK'
        action.parameters['Context'] = 'pick_from_container'
        return action

    @staticmethod
    def PICKFROMPLANE(action: Action, params: list) -> Action:
        action.parameter_order = ['Object','Plane','Robot','Waypoint']
        action.type = 'PICK'
        action.parameters['Context'] = 'pick_from_plane'
        return action
```

Convert planner actions to robot actions

Implementation



Implementation

Show video?

THANK YOU