AIPLAN 4EU

FONDAZIONE BRUNO KESSLER
FUTURE BUILT ON KNOWLEDGE

PSO
PLANNING SCHEDULING AND OPTIMIZATION

Automated Planning in Practice:

The Unified Planning Library

Andrea Micheli

Most of the material was prepared in collaboration with Gabriele Roger, Arthur Bit-Monnot and Sebastian Stock and presented at ICAPS 2023

The AIPlan4EU project is funded by the European Commission - H2020 research and innovation programme under grant agreement No 101016442

<insitutional-slides>

About me

My name is Andrea Micheli.

- PhD in Computer Science from the University of Trento and Fondazione Bruno Kessler (2016)
  - Visiting scientist at NASA Ames Research Center (Mountain View, California)
- Head of the Planning Scheduling and Optimization Unit at Fondazione Bruno Kessler (Trento, Italy)
  - 9 members and growing!
- Coordinator of the AIPlan4EU project
- European Research Council (ERC) Principal Investigator

Fondazione Bruno Kessler
About us

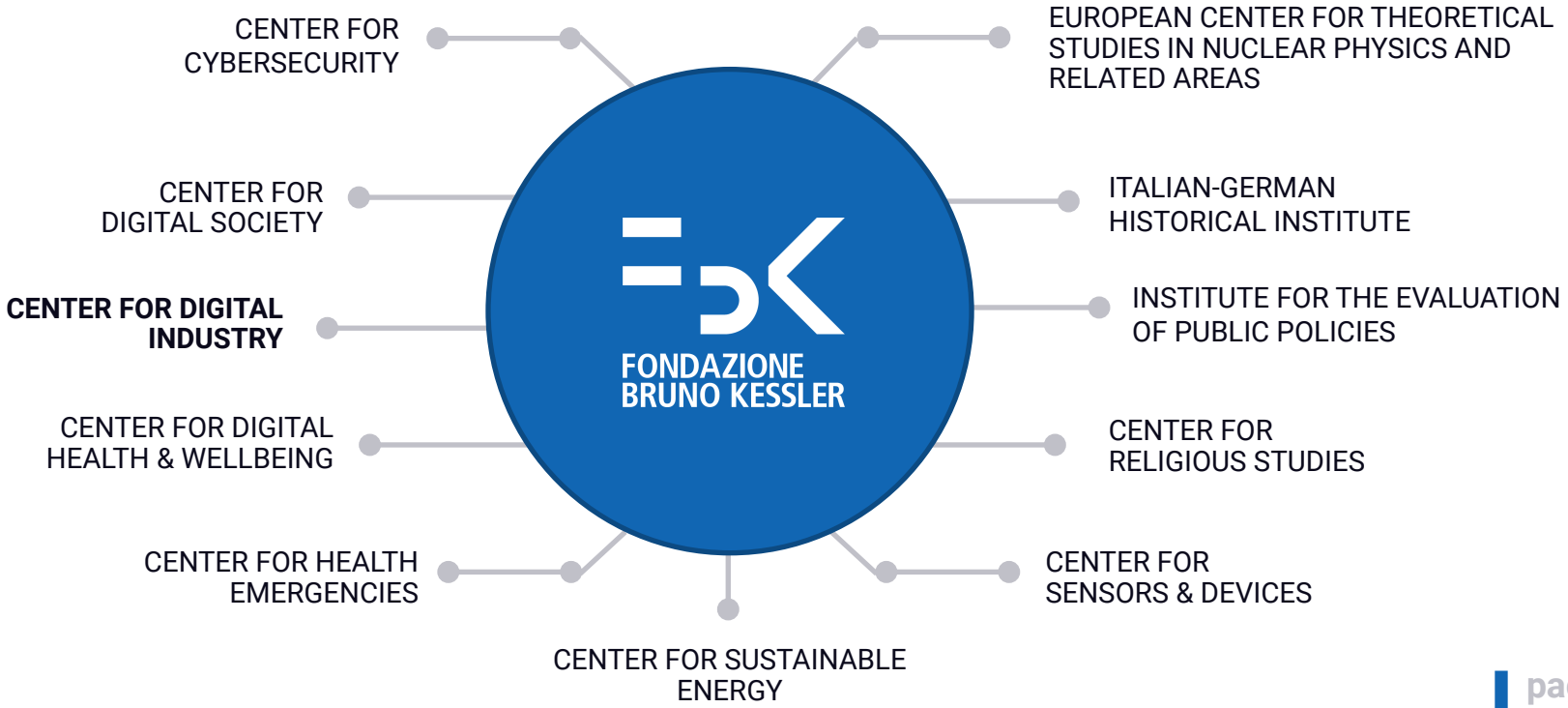
PROFILE

Fondazione Bruno Kessler (FBK) is a research not-for-profit public interest entity result of a history that is more than half a century old.

MISSION

FBK aims to excellence in science and technology with particular emphasis on interdisciplinary approaches and to the applicative dimension.

FBK
Research Centers



Digital Industry
At a glance

3D Optical Metrology

Technologies of Vision

Software Engineering

Machine Translation

Data Science for Industry and Physics

Formal Methods

Planning Scheduling and Optimization

Open IoT

CENTER FOR DIGITAL INDUSTRY
FONDAZIONE BRUNO KESSLER

People

- ✔ 121 (staff, postdocs, RA, PhD)
- ✔ 31 staff, 31 postdocs
- ✔ 24 research assistants
- ✔ 34 phd students

Scientific Staff

- ✔ 8 Associate Professors
- ✔ 6 Full Professors

Budget

- ✔ Costs: 6.1 M€
- ✔ Revenues 4.3 M€
- ✔ Self-funding: 70%

</insitutional-slides>

Seminar Agenda

1. Introduction: genesis of the Unified Planning library and scope
2. Architecture and design principles: Operation modes and API structure
3. How to model and solve problems
4. Advanced features (if time permits)
5. Applications and conclusions

Let's make this seminar interactive!

- Ask questions anytime!
- I will run the code live
- Try the notebooks on your laptop!
  - Feel free to edit the code!

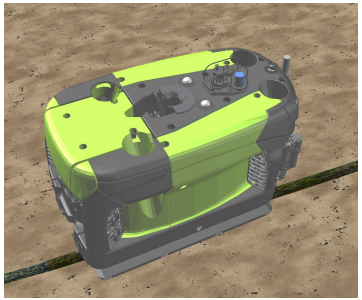
How to Participate

- Tutorial resources available at: **https://tinyurl.com/demoup**
- You can either run the examples of Google colab (Google account required, no installation needed) or on your local machine
- Basic installation
  - **apt install graphviz graphviz-dev** (for plotting)
  - **apt install openjdk-17-jre** (only needed if you want to use ENHSP)
  - **python3 -m pip install unified-planning[engines,plot]**

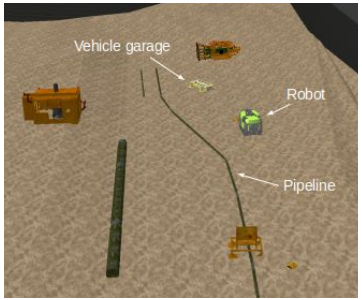
Introduction

Planning, Scheduling and Optimization

Given a **model of a system** and a **goal to be reached under constraints**, find a **course of actions/schedule** to drive the system to the goal.



System Specification



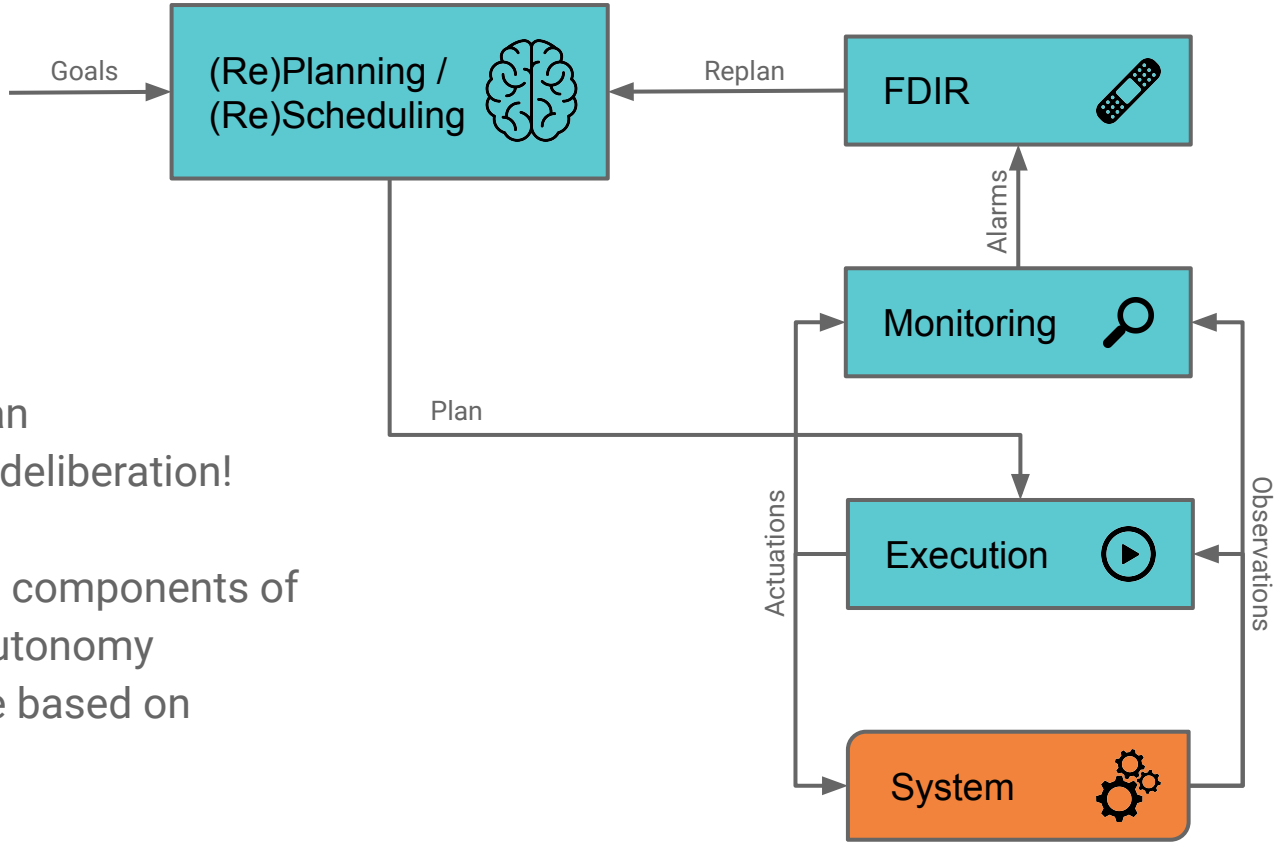
Initial Configuration and Objectives

PYSMT

Planner / Scheduler / Optimizer

(Optimal) Plan/Schedule

General system architecture



Not only plan generation/deliberation!

Support the components of a general autonomy architecture based on planning

What the Heck is “unified-planning”?

“Permissively **open-source**

python library

for the **modeling**,

manipulation

and **solving**

of **several kinds of planning** problems”

```python
from unified_planning.shortcuts import *

x = Fluent("x")

a = InstantaneousAction("a")
a.add_precondition(Not(x))
a.add_effect(x, True)

problem = Problem("basic")
problem.add_fluent(x)
problem.add_action(a)
problem.set_initial_value(x, False)
problem.add_goal(x)

with OneshotPlanner(problem_kind=problem.kind) as planner:
    result = planner.solve(problem)
    if result.plan:
        print(f"{planner.name} found a plan: {result.plan}")
    else:
        print("No plan found.")
```

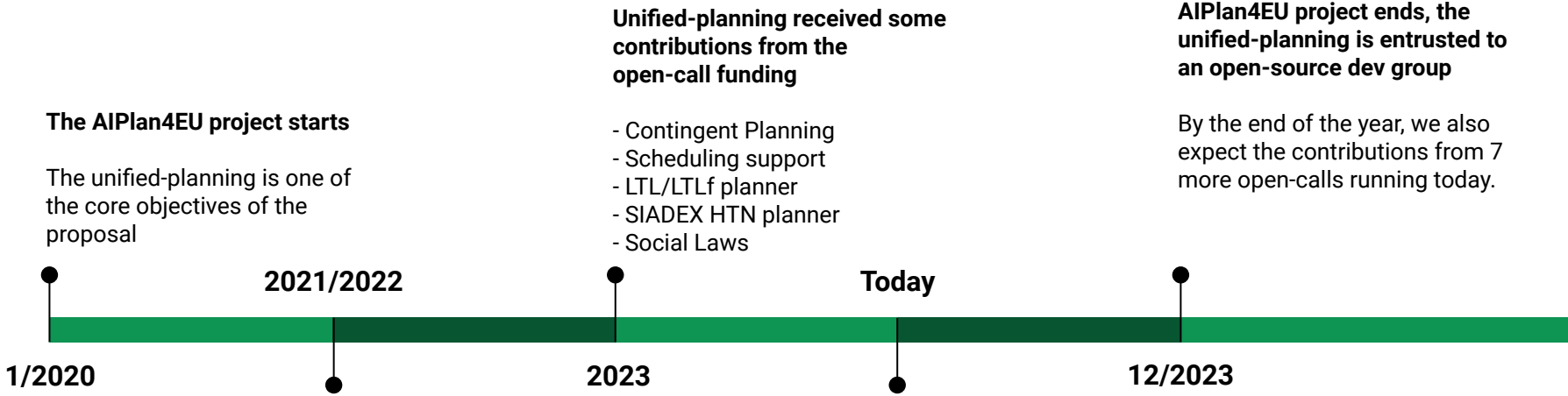
https://github.com/aiplan4eu/unified-planning

https://unified-planning.readthedocs.io

Key Features

- Diverse planning problem classes supported
  - Action-based (Classical, Numeric, Temporal); (Temporal) Hierarchical Task Networks; Multi-agent; Resource scheduling; Contingent planning
- Non only plan generation: Operation Modes
  - OneshotPlanner, PlanValidator, SequentialSimulator, Compiler, AnytimePlanner, Replanner, PlanRepairer, PortfolioSelector
- Automatic “requirements”: ProblemKind
- Different types of plans supported
  - Sequential, Partial-order, Time-triggered, Simple Temporal Network, Hierarchical
- Interoperability with formal languages and other libs
  - PDDL, ANML, Tarski, GRPC
- Advanced features
  - Meta-engines, simulated effects, custom heuristics

Genesis and History of the Library



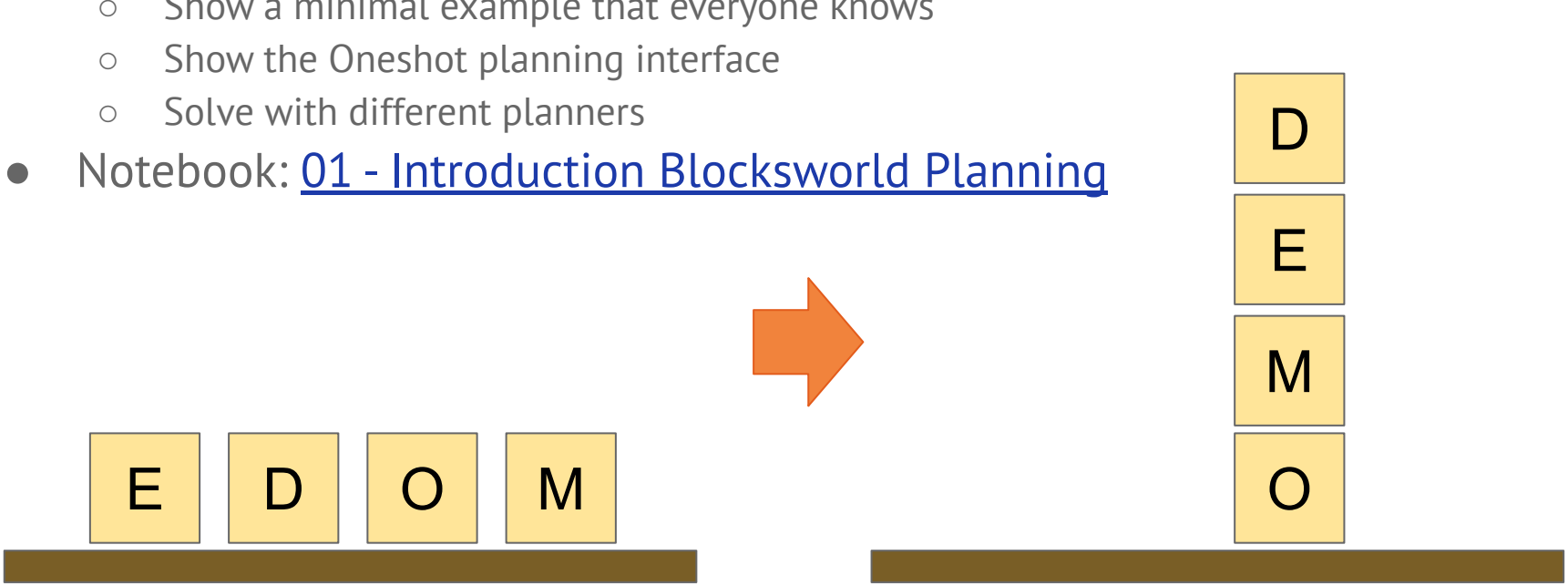
License Schema and Governance

Very last boring slide, I promise...

- The library itself is released under Apache License 2.0
  - Very permissive, also for commercial, non-free usages
- Linked planners (and their interfacing code) retain their own licenses
  - The unified-planning library by default prints credits and licensing information every time an engine is used
  - The use of a specific planner might be restricted for certain applications
- So far, the project has been governed by the AIPlan4EU consortium
  - We are now writing a set of governance rule to entrust the unified-planning to an open-source do-ocracy
  - Maintenance ensured by many partners
  - Open to new contributions / pull-requests / maintainers
  - We welcome feedback on the governance: https://github.com/aiplan4eu/unified-planning/pull/422

Basic Example: blocksworld

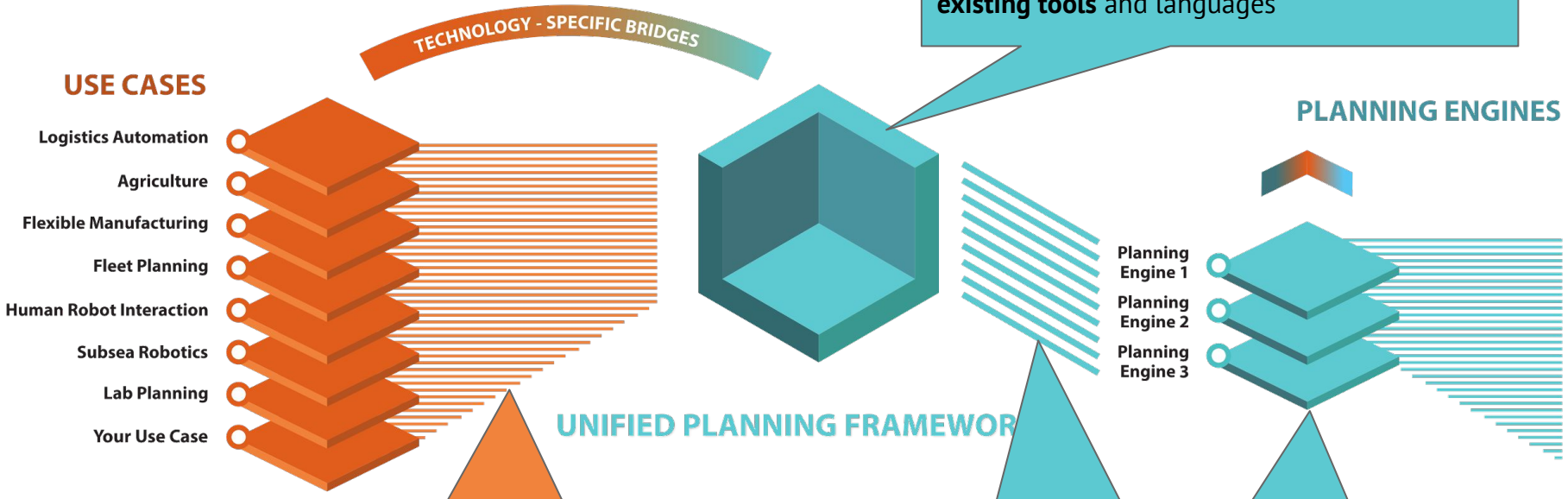
- Objectives:
  - Show a minimal example that everyone knows
  - Show the Oneshot planning interface
  - Solve with different planners
- Notebook: [01 - Introduction Blocksworld Planning]



Architecture

Global Vision

A **reusable, planner-agnostic** Python library, offering an **abstraction layer** for diverse planning engines and **interoperability with existing tools** and languages



The API offered by the UPF is integrated by a **collection of reusable components** that are specific to a certain technology (not use-case specific).
E.g. Integration in ROS, Integration in a WMS, ...

...e UPF has a notion of "**Operation Mode**" (OM):

- OM examples include "**OneshotPlanning**", "**PlanValidation**" and "**PlanRepair**"
- Each engine declares which operation modes it supports

...eneral word:
...dators,

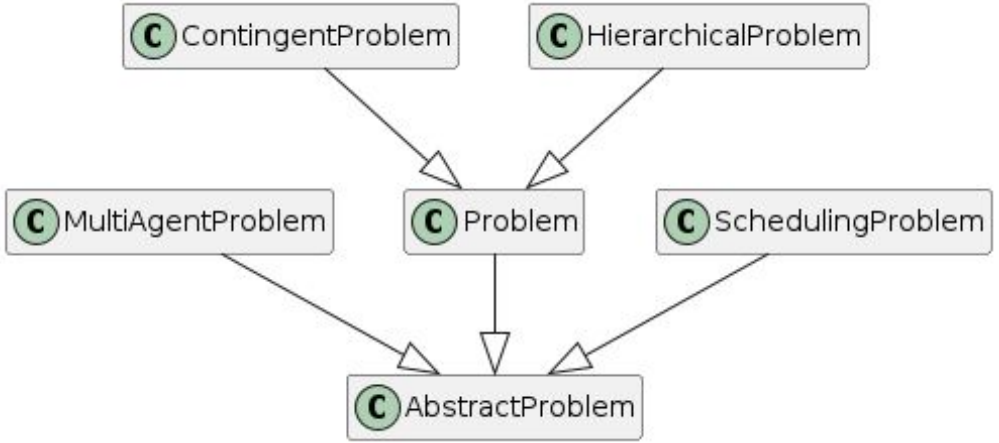
Library Scope

AIPLAN 4EU

- Prototype planning applications
  - Construct planning problems from data
  - Easily try multiple planners on the same problem
  - Explore multiple formulations
- Algorithms using planning as oracle
  - “Meta-planners” (more on this later)
- Combine multiple planners in a single solution
  - Ground with engine1 and solve with engine2
- Procedural modeling and solving (alpha)
  - Simulated effects
  - Custom heuristics

Problem Formulations

- 5 “classes” supported (for now)
- APIs “as shared as possible”
  - Actions, expressions, fluents…
- UP infrastructure works on `AbstractProblem` for full generality



Available Operation Modes

- OneshotPlanner
- PlanValidator
- SequentialSimulator
- Compiler
- AnytimePlanner
- Replanner
- PlanRepairer
- PortfolioSelector

ProblemKind

- Similar to PDDL `:requirements`
- Automatically calculated from a problem specification
  - Syntactic checks (some corner cases are over-approximated)
- Planning engines declare the `ProblemKind` they support
- Used by the UP to filter applicable engines when invoking OMs without specifying the engine name

```
print(problem.kind)
```

```
PROBLEM_CLASS: ['ACTION_BASED']
PROBLEM_TYPE: ['SIMPLE_NUMERIC_PLANNING']
NUMBERS: ['BOUNDED_TYPES', 'DISCRETE_NUMBERS']
CONDITIONS_KIND: ['NEGATIVE_CONDITIONS', 'EQUALITIES']
TYPING: ['FLAT_TYPING']
FLUENTS_TYPE: ['NUMERIC_FLUENTS', 'OBJECT_FLUENTS']
SIMULATED_ENTITIES: ['SIMULATED_EFFECTS']
```

Interoperability and Interfaces

- Software converters
  - tarski (https://github.com/aig-upf/tarski)
  - protobuf (more on this later)
- Formal language input/output
  - PDDL
    - PDDL 2.1 level 3 for classical numeric and temporal planning
    - HDDL for hierarchical
    - MA-PDDL for multi-agent
  - (fragments of) ANML

```
from unified_planning.io import PDDLReader

reader = PDDLReader()
domain_filename = ...
problem_filename = ...

# Reader used to parse PDDL files and return a
up.model.Problem
problem = reader.parse_problem(
    domain_filename,
    problem_filename
)

problem = ... # A new problem
writer = PDDLWriter(problem)
# Path to file where the PDDL domain will be printed.
domain_filename = ...
writer.write_domain(domain_filename)
# Path to file where the PDDL problem will be printed.
problem_filename = ...
writer.write_problem(problem_filename)
```

How to Model and Solve Problems

Modelling and Solving Problems

- classical and numeric problems

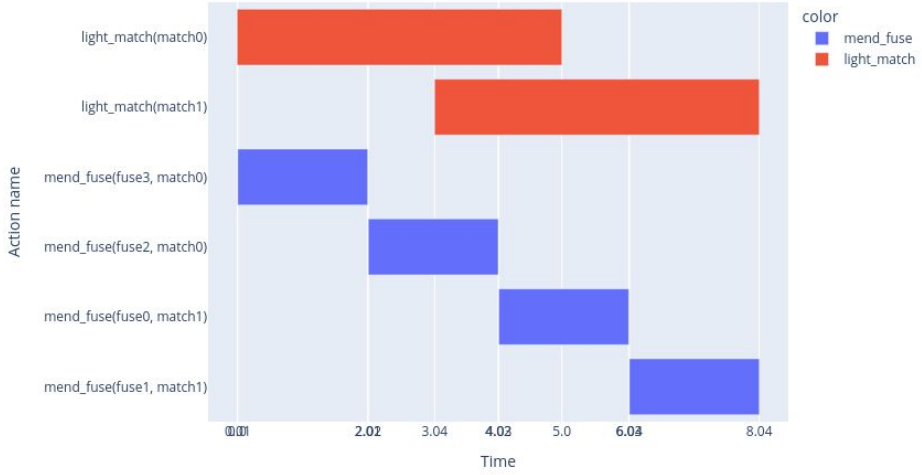
  Notebook: 03 - Modelling

- hierarchical problems

  Notebook: 04 - Hierarchical Planning

Temporal Planning Example

- Show how to mix PDDL parser and code (MatchCellar domain)
- Show some more OMs:
  - OneshotPlanner, Compiler and PlanValidator
- Glimpse of advanced features
  - custom heuristic
- Notebook: [02 -Temporal Planning]



Advanced Features

Simulated Effects

Simulated Effects allow to compute the effects of an action on a list of fluents via a call to a provided Python function

```
Location, Robot = UserType('Location'), UserType('Robot')
battery_charge = Fluent('battery_charge', IntType(0, 100), robot=Robot)

def battery_fun(problem, state, actual_params):
  value = state.get_value(battery_charge(actual_params.get(robot))).constant_value()
  return [Int(value - 10)]

move = InstantaneousAction('move', robot=Robot, l_from=Location, l_to=Location)
...
move.add_precondition(GE(battery_charge(robot), 10))
move.add_effect(at(robot), l_to)
move.set_simulated_effect(SimulatedEffect([battery_charge(robot)], battery_fun))
```

Dict: action's parameters/values

List of computed fluent values

List of fluents whose values are calculated

Notebook: [07 - Simulated Effects]

Custom Heuristics

- **Idea**: specify a planning heuristic in Python and ask an engine to use it
  - Heuristic is a function on a state and computes a heuristic value or `None` if it is a dead end

  Notebook: last part of [02 -Temporal Planning]

```
problem = self.problems["basic"].problem
x = problem.fluent("x")

def h(state):
      v = state.get_value(x()).bool_constant_value()
      return 0 if v else 1

with OneshotPlanner(name="tamer") as planner:
      final_report = planner.solve(problem, heuristic=h)
```

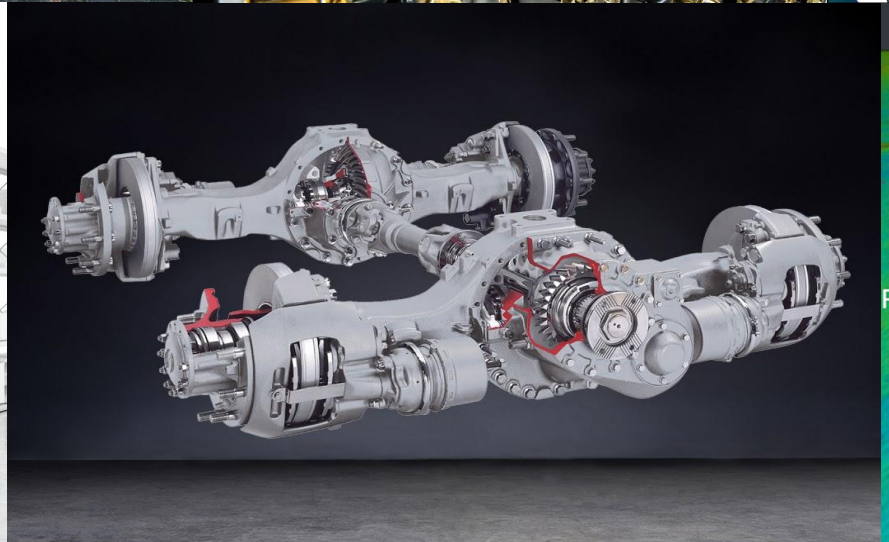
Meta-Engines

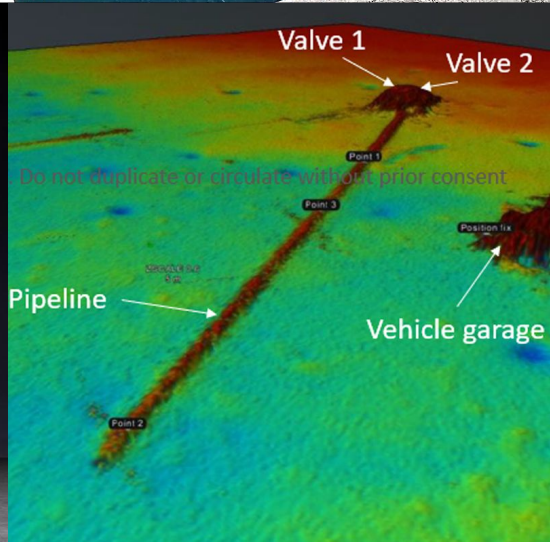
- **Idea**: solve a problem by calling to another (or the same) OperationMode

- Examples:
  - Oversubscription planning
    - by repeated solving os subsets of goals
    - Natively implemented in the library **`oversubscription[engineX]`**
    - Notebook: 08 - Oversubscription using MetaEngines
  - Contingent planning
    - By calling classical planning on different observations
    - **`CPORPlanning[engineX]`** (Credit: Shani Guy and Hila Kesem Hadad)

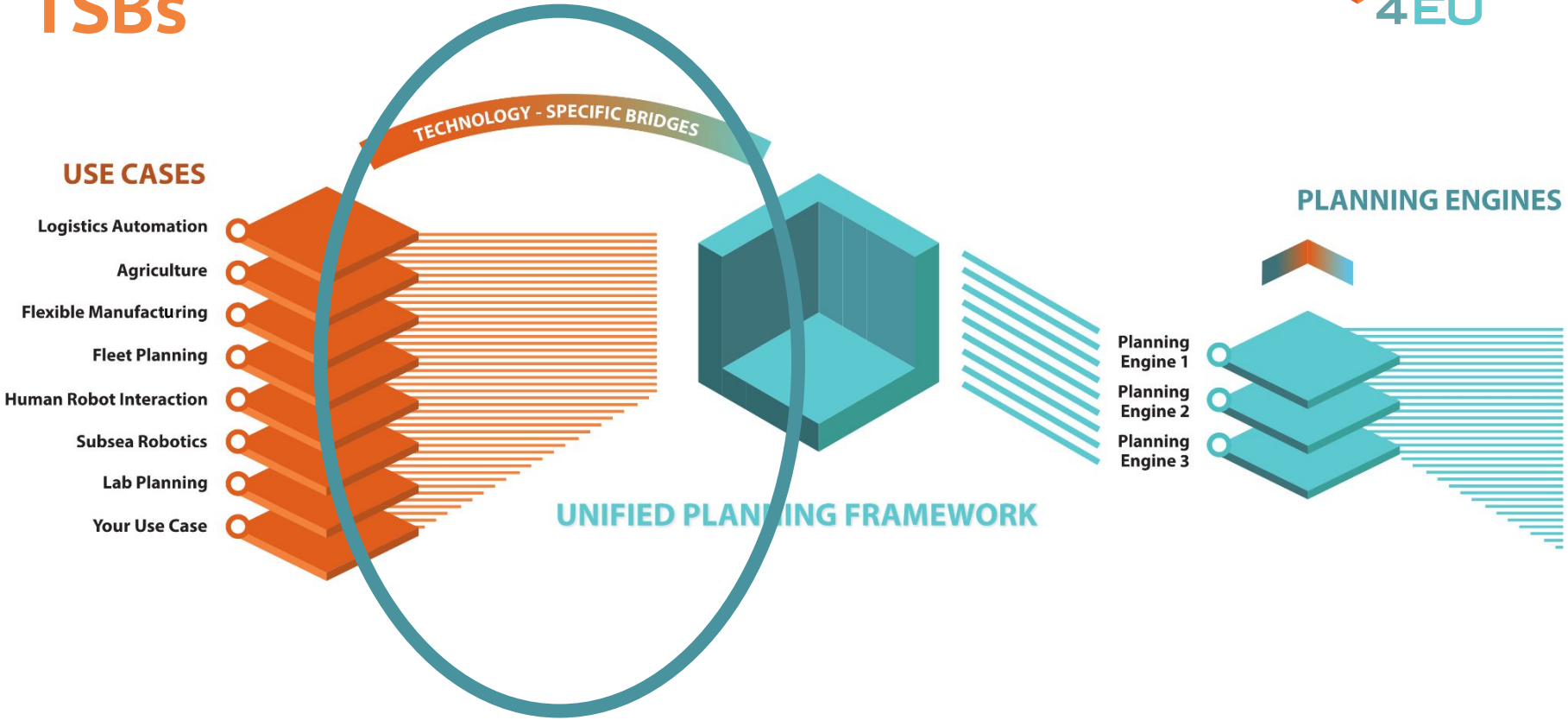
Applications

Applications





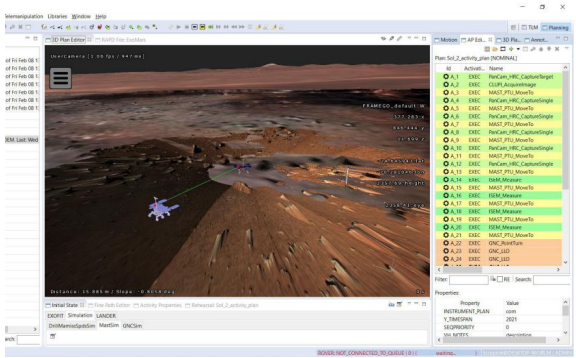
TSBs


TECHNOLOGY - SPECIFIC BRIDGES
USE CASES
Logistics Automation
Agriculture
Flexible Manufacturing
Fleet Planning
Human Robot Interaction
Subsea Robotics
Lab Planning
Your Use Case
UNIFIED PLANNING FRAMEWORK
PLANNING ENGINES
Planning Engine 1
Planning Engine 2
Planning Engine 3

Planning in Applications

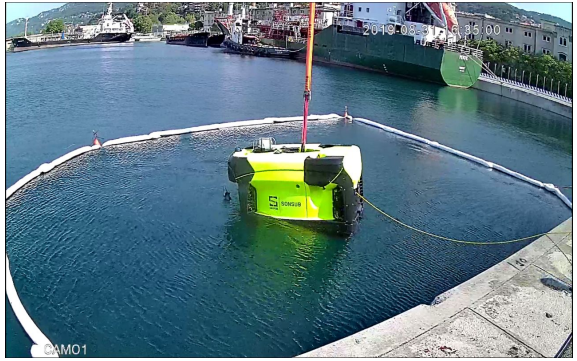
AIPLAN 4EU

Planning	Execution	Monitoring	Failure response and recovery	Replanning
(Semi-)Automatic synthesis of operations to achieve a certain objective/goal, keeping into account resources capabilities and optimality.	Transform high-level plans into executable instructions for machines. Interface with management systems (e.g. WMSs).	Continuously supervise the plan execution and provide state-estimations.	Identify when a monitored situation is problematic and raise alarms to users/subsystems. Automatically recover from problematic situations.	When goals or conditions change, we need to adapt the planning currently in execution to cope with the new contingencies.



For remote operation

For process automation



For autonomous robotics

Agriculture Example: Maize Harvesting

Agriculture campaign planning: silage maize harvesting

- Coordinate machines in their joint farming process
- Harvesting multiple fields, transport to the silo, compaction in the silo
- Efficient use of resources (machinery, silo, staff, time) is crucial

Harvesting Process Overview

General goal: Harvest all fields with the available resources (machines, silos) in a way that **minimizes the overall campaign duration**:

- Selection of ordering in which fields are harvesting
- Assignment of harvesters to fields
- Assignment of TVs to HVs
- Assignment of silos for yield unload

Goals (UP):

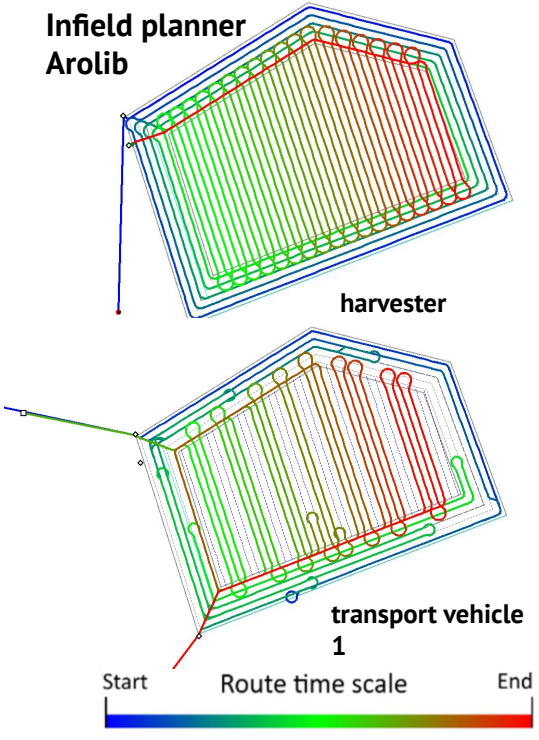
- All fields are fully harvested
- All HVs are ‘free’
- All TVs are empty and ‘free’
- All silo unloading locations are cleared
- Optimization goal: minimize overall_duration

Maize Harvesting Planning Problem

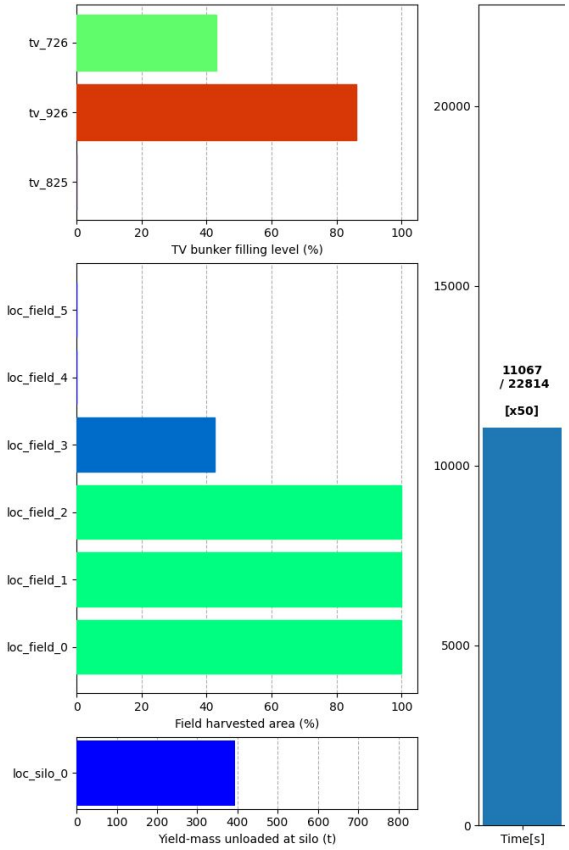
Goal: Minimize duration (leading to cost reduction)

Requirements / UP Features

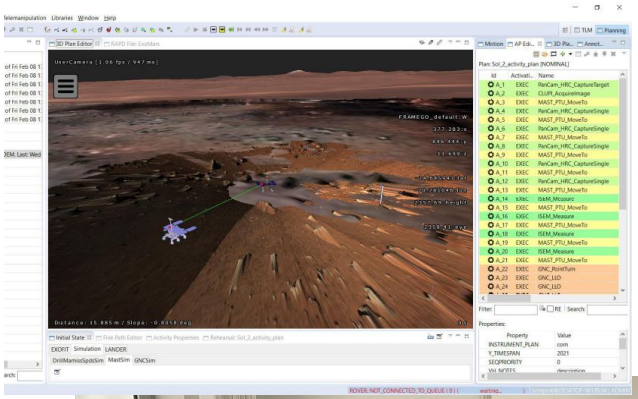
- Temporal planning
- Scheduling resources
- Optimization
- Simulated effects
- Heuristics

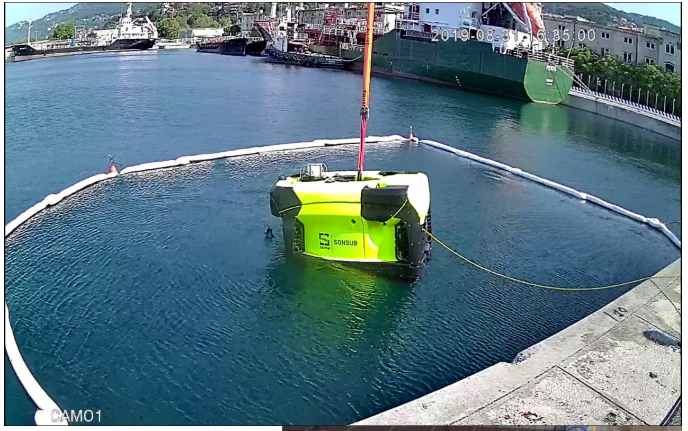


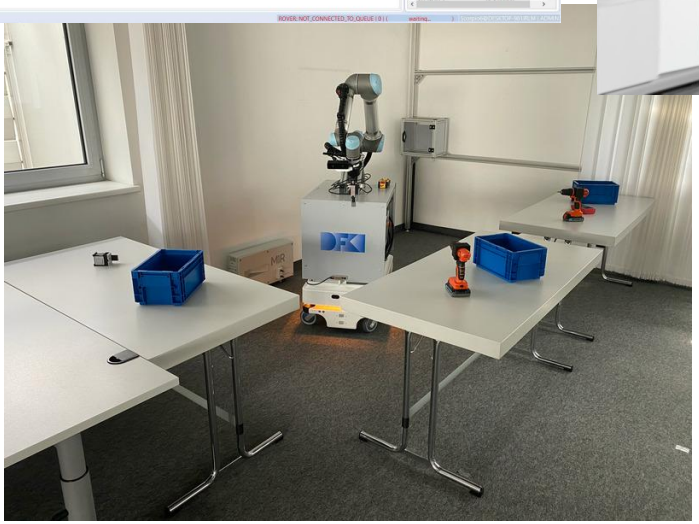
Harvesting Process: Plan Snapshot

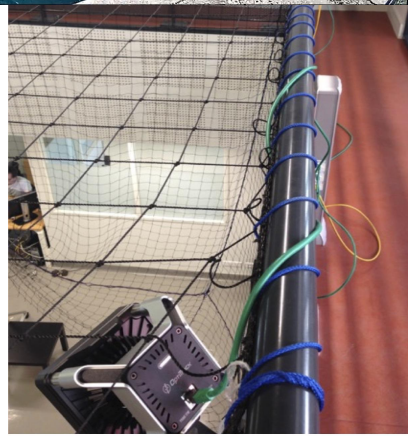


Robotics









Integrations with Robot Technologies

- Embedded Systems Bridge
- Robot Operating System (ROS/ROS2) wrappers: UP4ROS / UP4ROS2
- UP support in PlanSys2

Embedded Systems Bridge (ESB)

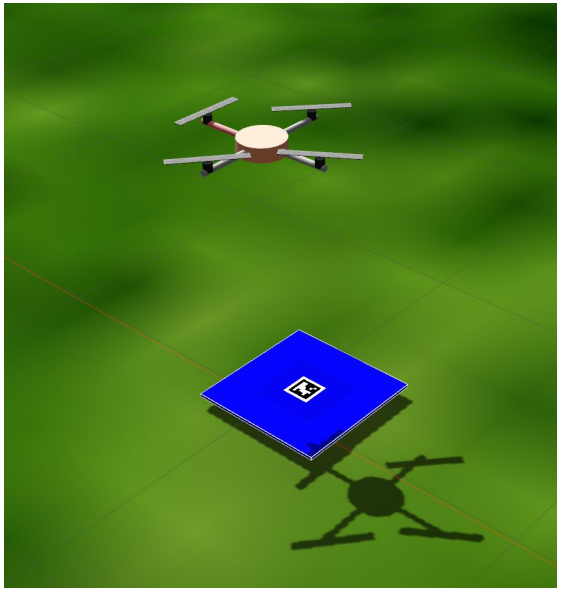
- A bridge from UP to Embedded Systems
  - Instead of wrapping UP, ESB aims to provide additional functionalities for robotic systems
  - Python library for using UP
  - Robot framework and middleware independent
- Simplifies creating planning problems for existing robot actions and fluent representations
- Plan execution and monitoring capabilities

https://github.com/aiplan4eu/embedded-systems-bridge

Robot Examples

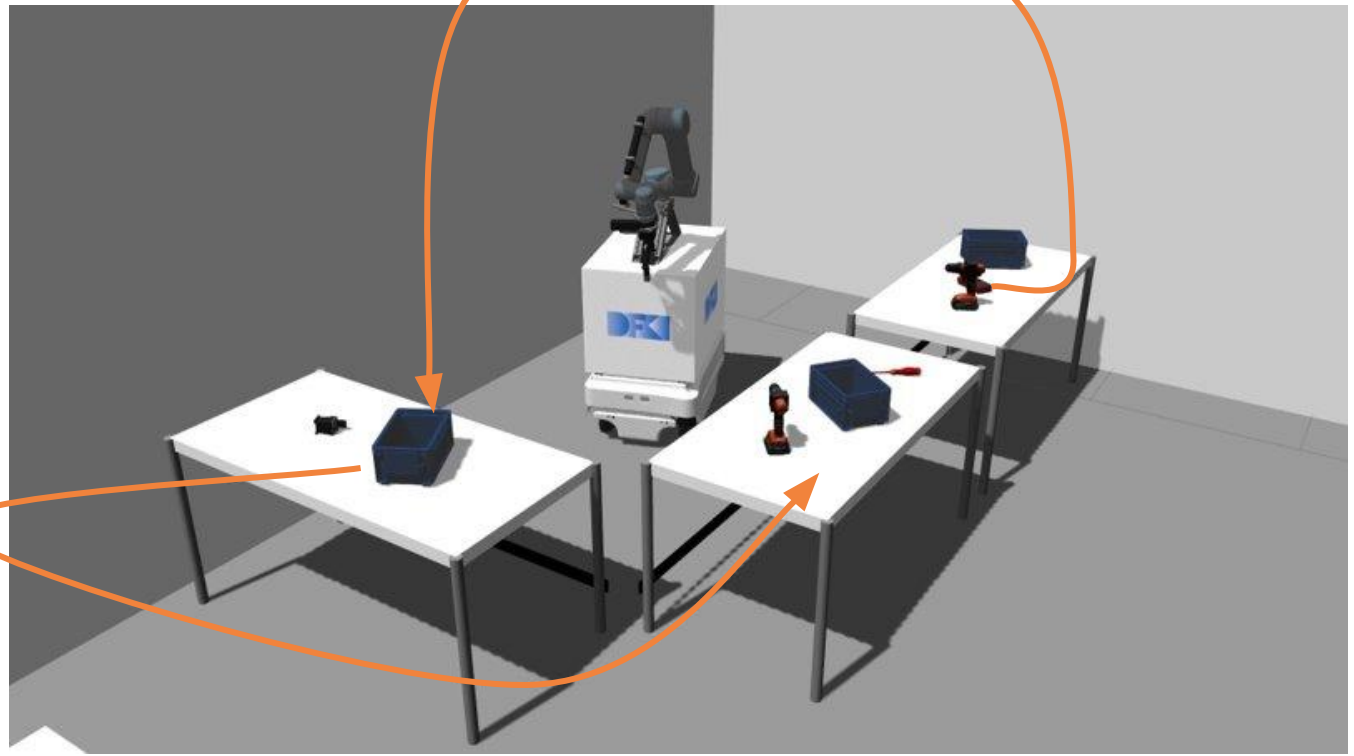
Service Robot: Mobipick

Drones



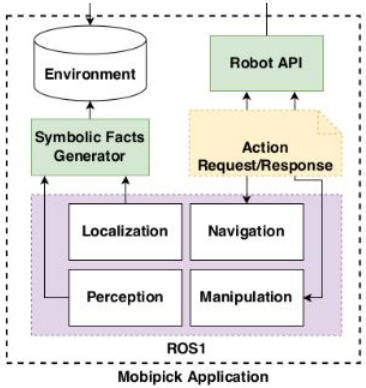
Mobipick

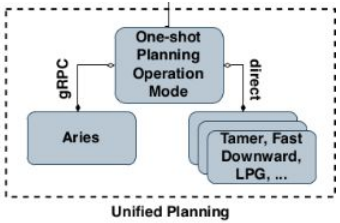
The multimeter shall be placed into the box



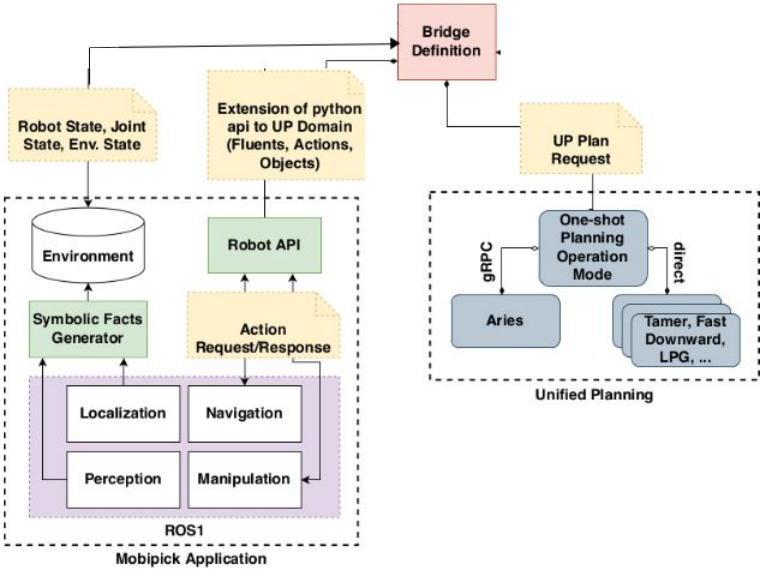
The box shall be placed onto the target table

Connecting UP to Robots





Connecting UP to Robots



Mapping between Robot and UP Representation

- We already have Python implementations for Mobipick's actions and computation of fluents
- Idea: Let's create UP definitions directly from those functions

```
def robot_at_fun(l: Location):
    """Check if the robot is at a location."""
    return Robot.location == l

bridge = Bridge()
bridge.create_types([Location, Area, Robot])
robot_at = bridge.create_fluent_from_function(robot_at_fun)
```

- ESB maintains mapping in both directions
  - e.g., call functions to get state

```
problem = bridge.define_problem()
bridge.set_initial_values(problem)
```

Mapping between Robot and UP Representation

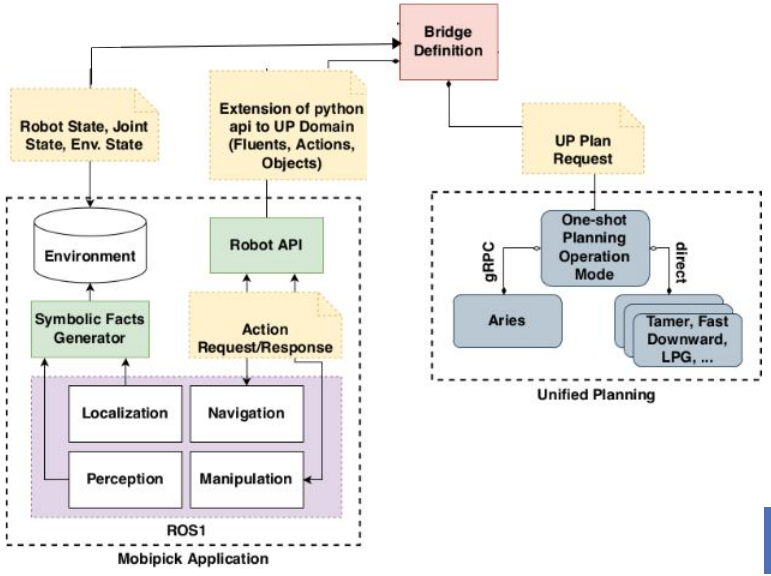
... same for actions

```
class Robot:

    @classmethod
    def move(cls, l_from: Location, l_to: Location):
        print(f"Moving from {l_from} to {l_to}")
        Robot.location = l_to
        return True


move, [l_from, l_to] = bridge.create_action(
    "move", _callable=Robot.move,
    l_from=Location, l_to=Location, duration=1
)
move.add_condition(StartTiming(), robot_at(l_from))
move.add_condition(StartTiming(), Not(robot_at(l_to)))
move.add_effect(StartTiming(), robot_at(l_from), False)
move.add_effect(EndTiming(), robot_at(l_to), True)
```

Executing the Plan

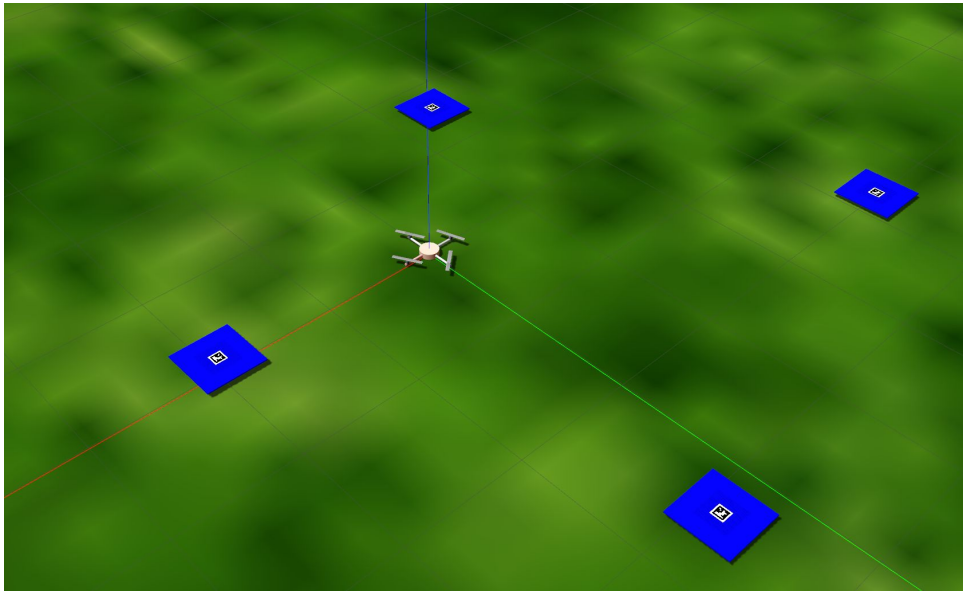
- ESB executes plans as dependency graphs
- Easy dispatching of actions via mapping

```
plan = bridge.solve(problem, planner_name="aries")
dependency_graph = bridge.get_executable_graph(plan)

dispatcher = PlanDispatcher()
dispatcher.execute_plan(plan, dependency_graph)
```



Drones Example

- Multiple Drones surveying an area
- Goal: Find plates in environment and inspect them
- Uses hierarchical planning (Aries)



https://github.com/franklinselva/genom3-experiment

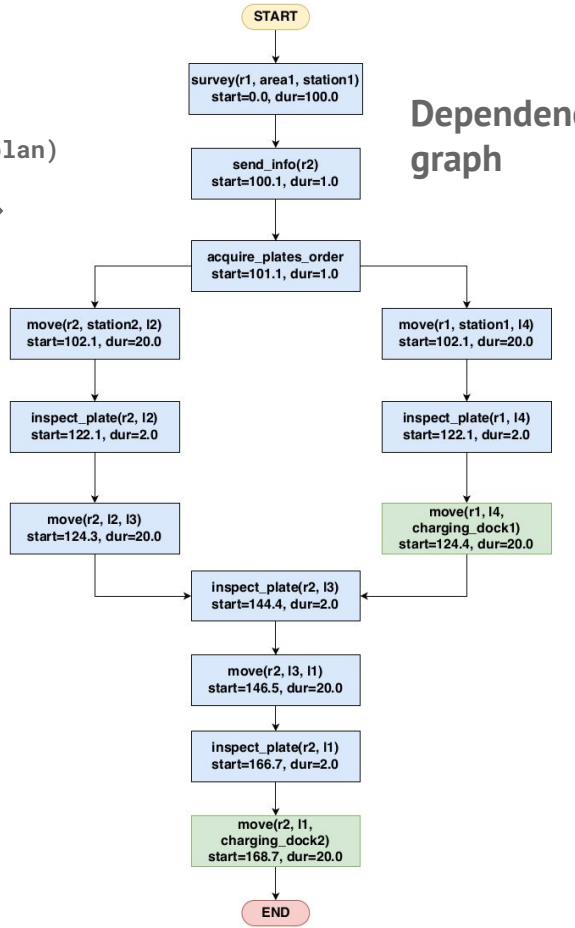
Drones: Plan and Dependency Graph

UP output: time-triggered plan

```
 # start    # action    # duration

    0.0 survey(r1, area, station1), 100.0
100.1 send_info(r2), 1.0
101.1 acquire_plates_order, 1.0
102.1 move(r2, station2, l2), 20.0
102.1 move(r1, station1, l4), 20.0
122.2 inspect_plate(r1, l4), 2.0
122.2 inspect_plate(r2, l2), 2.0
124.3 move(r2, l2, l3), 20.0
124.4 move(r1, l4, charging_dock1), 20.0
144.4 inspect_plate(r2, l3), 2.0
146.5 move(r2, l3, l1), 20.0
166.6 inspect_plate(r2, l1), 2.0
168.7 move(r2, l1, charging_dock2), 20.0
```

`bridge.get_executable_graph(plan)`

Dependency graph



UP4ROS & UP4ROS2

- ROS and ROS2 wrappers for UP
- Provide single ROS node that expose the UP API via ROS messages
- ROS Services allow to define and manage planning problems interactively,
  - e.g., AddAction.srv, AddFluent.srv, AddGoal.srv, AddObject.srv, GetProblem.srv
- ROS Actions for plan generation,
  - e.g., PlanOneShotAction, PDDLPlanOneShotAction
- Converters for UP Python objects to and from ROS messages

Credits: **Daniele Calisi** and **Guglielmo Gemignani**

https://github.com/aiplan4eu/UP4ROS https://github.com/aiplan4eu/UP4ROS2

UP4ROS Example Messages

```
# AddAction.srv
string problem_name
up_msgs/Action action
---
bool success
string message
```

```
# Action.msg
# Action name. e.g. "move"
string name

# Typed and named parameters of the action.
up_msgs/Parameter[] parameters

# If set, the action is durative. Otherwise it is instantaneous.
up_msgs/Duration[] duration

# Conjunction of conditions
up_msgs/Condition[] conditions

# Conjunction of effects as a result of applying this action.
up_msgs/Effect[] effects
```

UP4ROS Example Client

```
rospy.init_node("test_up4ros_client")
client = actionlib.SimpleActionClient(action_name, msgs.PlanOneShotAction)

pb_writer = ROSInterfaceWriter()
goal_msg = msgs.PlanOneShotGoal()
goal_msg.plan_request.problem = pb_writer.convert(
    get_example_problems()["robot"].problem    # UP problem
)

client.send_goal(goal_msg)
client.wait_for_result()
result = client.get_result()
```

Conclusions

Conclusions

- You can find many more interactive demos in the UP documentation
  - https://unified-planning.readthedocs.io
- We hope the UP library can be useful to some of you!
- Contributions are always welcome
  - New engines
  - New problems/feature operation modes
- **#ad**: FBK is always looking for talents!
  - PhD Students, Developers, Research Assistants, Internships
  - See jobs.fbk.eu
  - Contact me (amicheli@fbk.eu) for info!