Sistemas de Planificación

Objetivo del Tema

1. Conoceremos qué es un Sistema de Planificación
2. Entenderemos como funciona por dentro un Sistema de Planificación
3. Aprenderemos cómo ejecutar planes en robots
4. Implementaremos acciones

¿Qué es un Sistema de Planificación?

Un Sistema de Planificación es un sistema que crea y ejecuta planes en robots

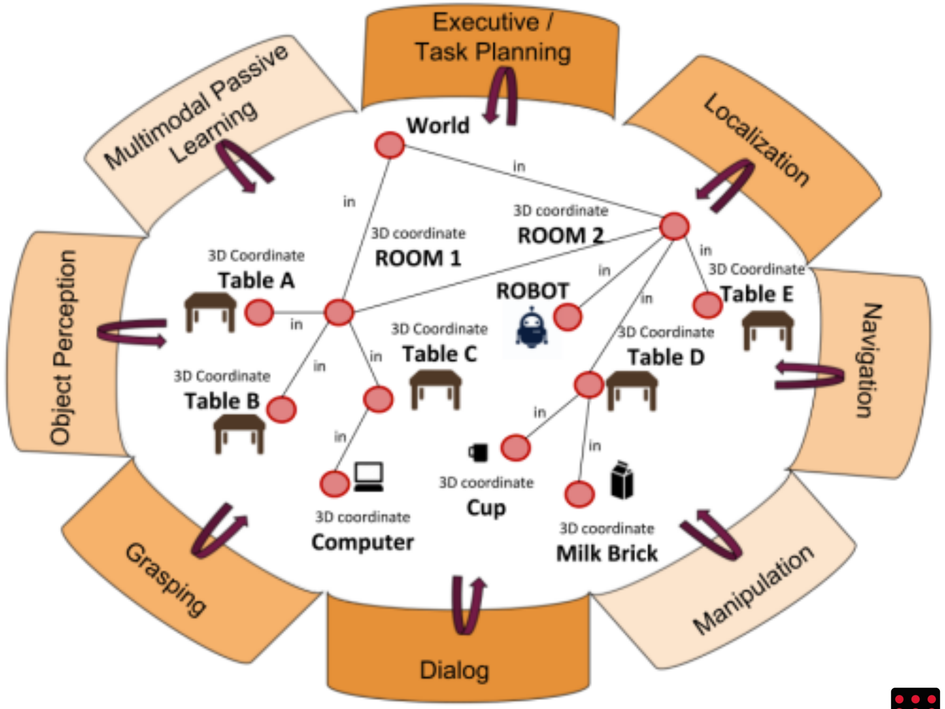
1. Leer uno o varios dominios PDDL
2. Gestionar el conocimiento (instancias, predicados, goals, funciones)
   a. Proporcionar una interfaz para añadir/eliminar/actualizar conocimiento
   b. Validación con el dominio
3. Proporciona mecanismos para implementar y ejecutar acciones
4. Verificar en tiempo ejecución los requisitos
5. Aplicar los efectos de las acciones

Ya no estamos en un mundo cerrado, y el conocimiento debe estar validado con la realidad

Ejemplos de sistemas que usan planificación

Cortex

- CORTEX es una arquitecta cognitiva que usa Planning
- Mantiene una representación del conocimiento basada en un grafo
- Los agentes acceden al grafo para cumplir sus tareas
  - Percepción
  - Actuación
  - Planning
- Un agente de Task Planning genera planes usando Metric-FF



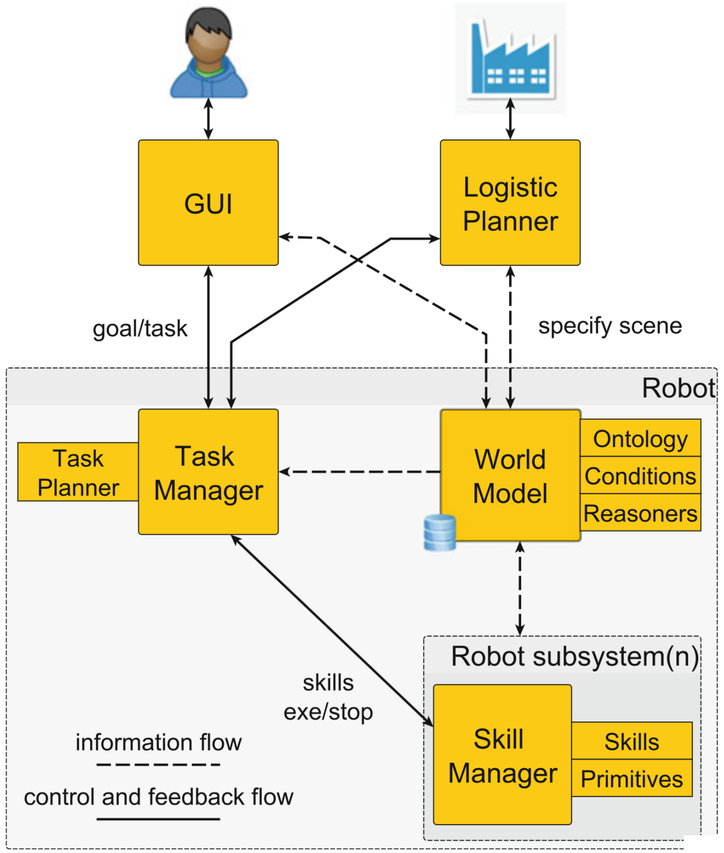
https://robolab.unex.es/wp-content/papercite-data/pdf/luis-pablo-cortex.pdf

Intelligent Robotics Lab

Ejemplos de sistemas que usan planificación

SKiROS2

- SkiROS2 es una plataforma para crear comportamientos robóticos complejos mediante la composición de skills (bloques de software modulares) en Behavior Trees.
- Disponible en ROS y ROS2
- El usuario proporciona Skills, una escena y un goal
- Usa razonamiento basado en ontologías OWL



https://github.com/RVMI/skiros2

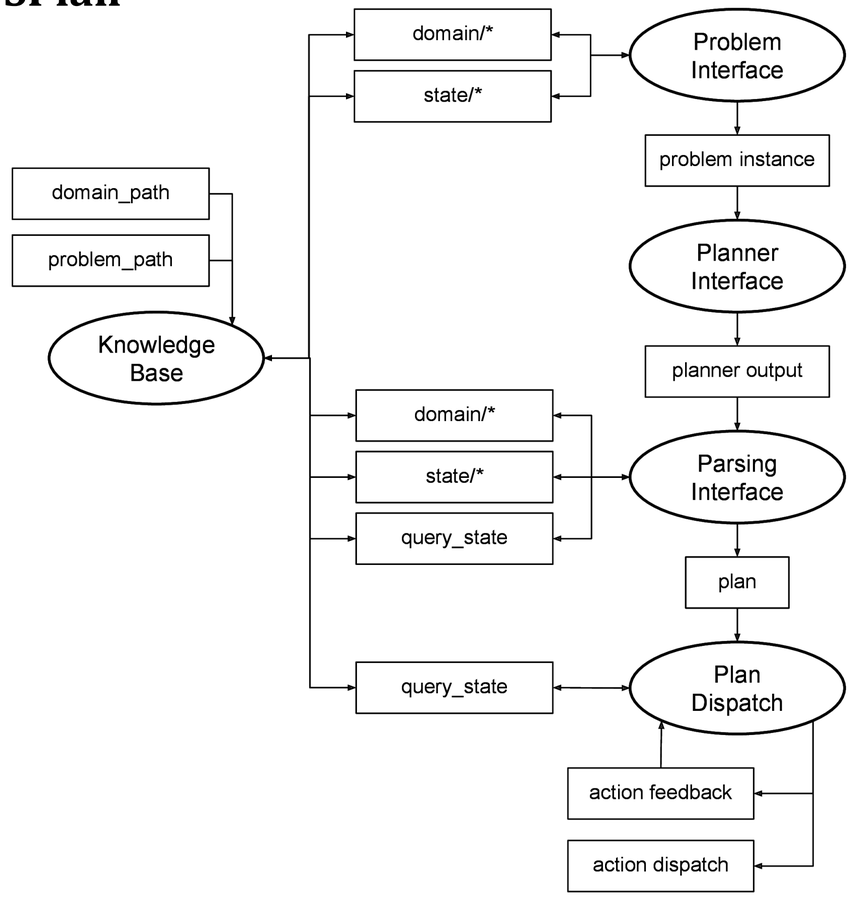
https://www.researchgate.net/publication/317753352_SkiROS-A_skill-based_robot_control_platform_on_top_of_ROS

Intelligent Robotics Lab

Ejemplos de sistemas que usan planificación

ROSPlan

- Es el sistema de planificación de referencia en ROS
- Disponible en ROS

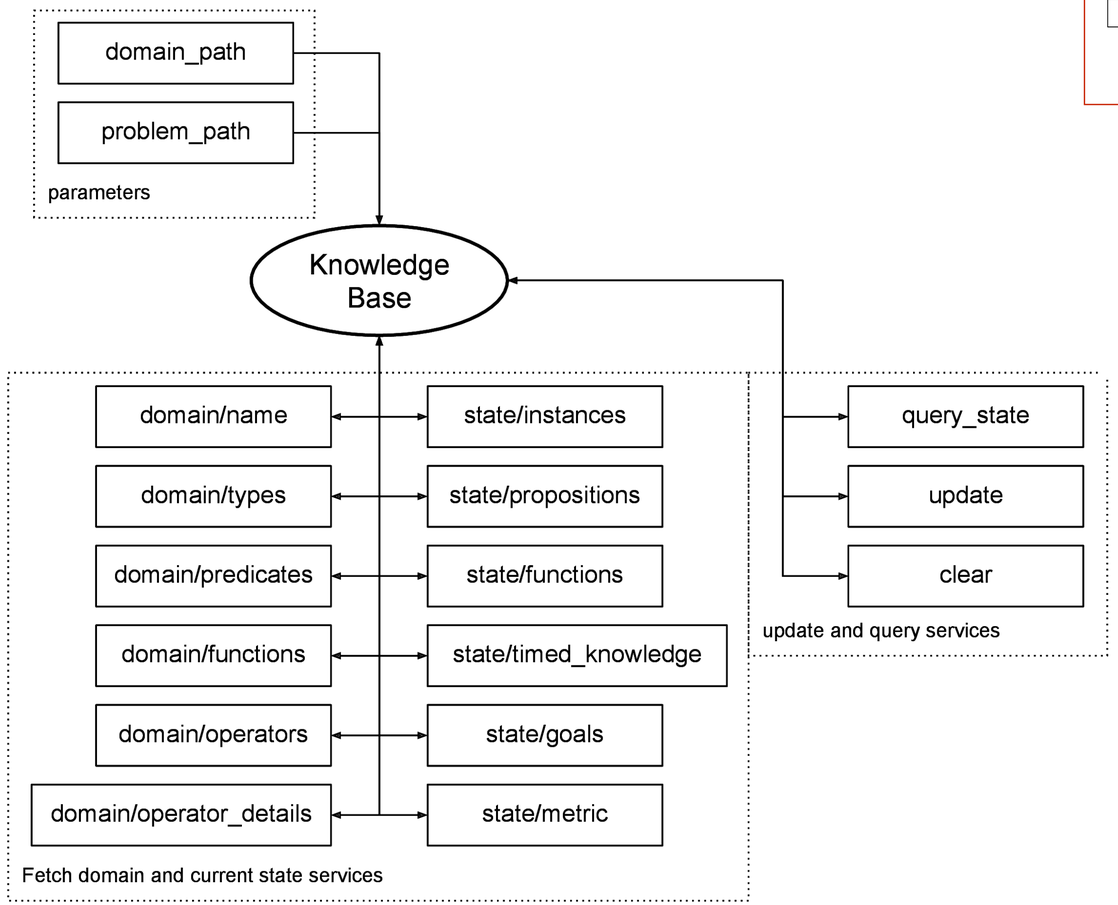


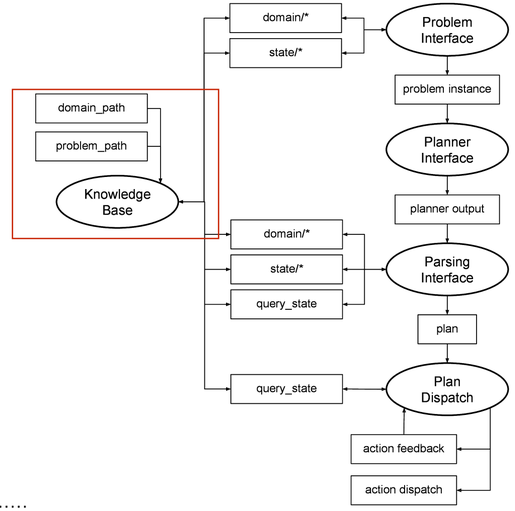
https://github.com/KCL-Planning/ROSPlan
https://kcl-planning.github.io/ROSPlan/
https://www.aaai.org/ocs/index.php/ICAPS/ICAPS15/paper/download/10619/10379

Intelligent Robotics Lab

Ejemplos de sistemas que usan planificación

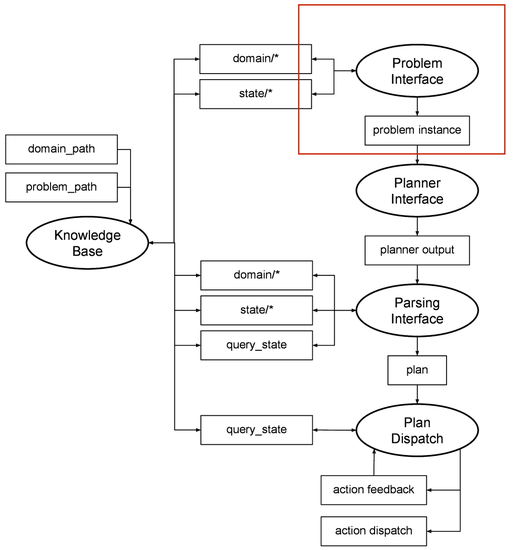
ROSPlan

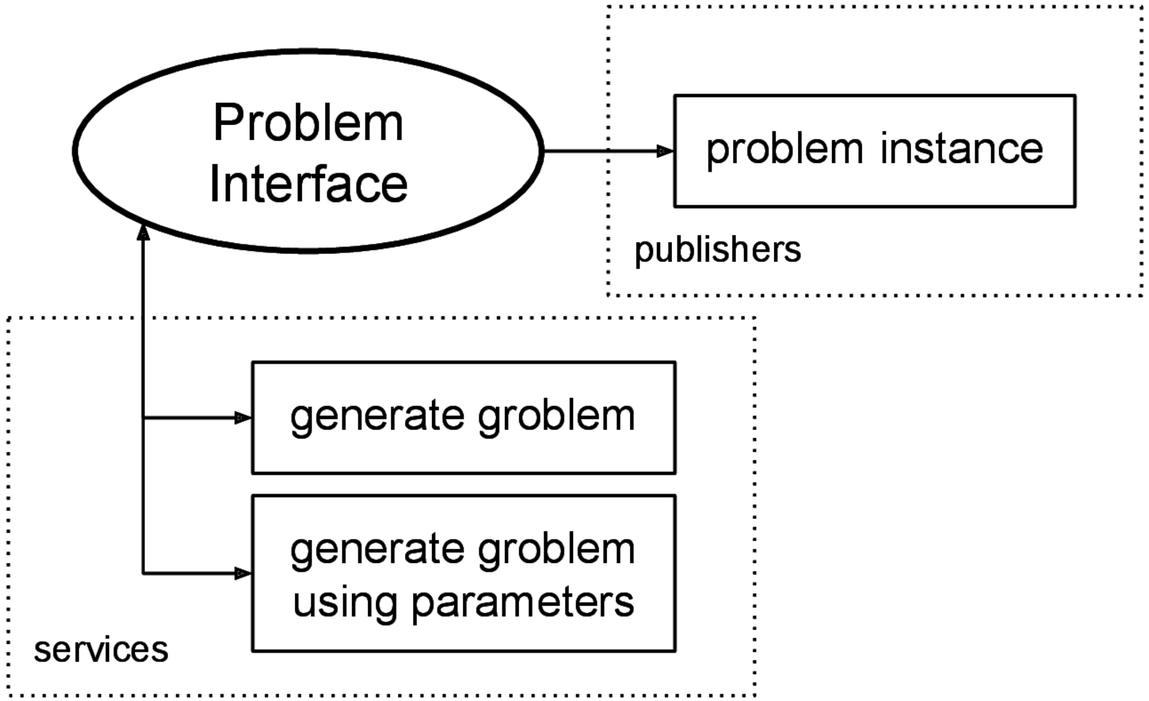




Ejemplos de sistemas que usan planificación

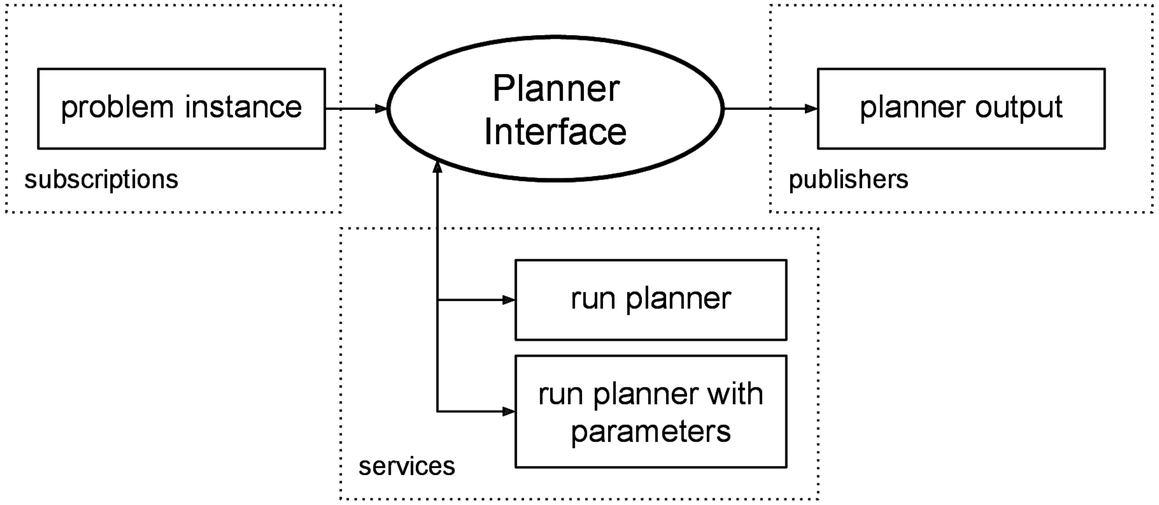
ROSPlan

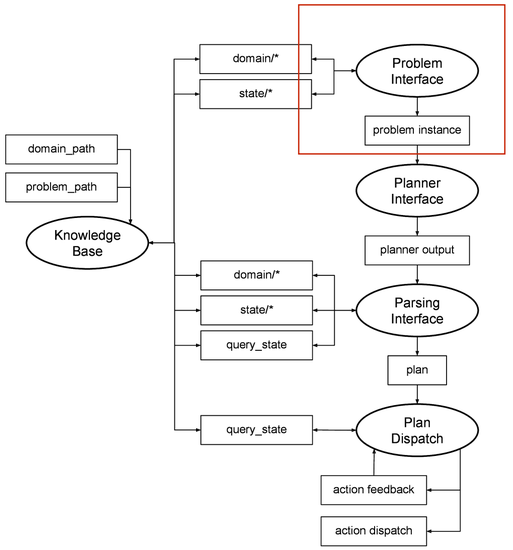




Ejemplos de sistemas que usan planificación

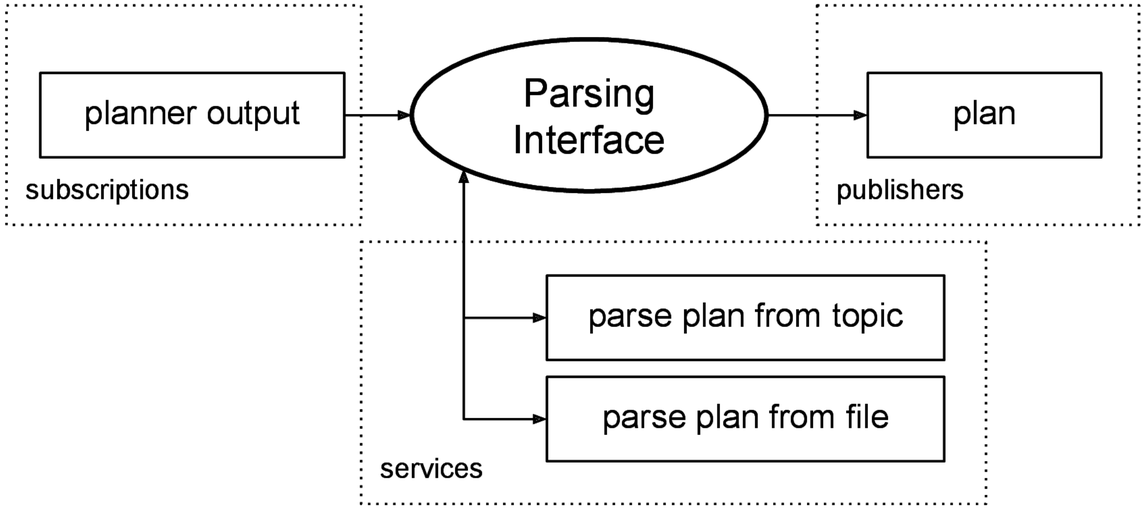
ROSPlan

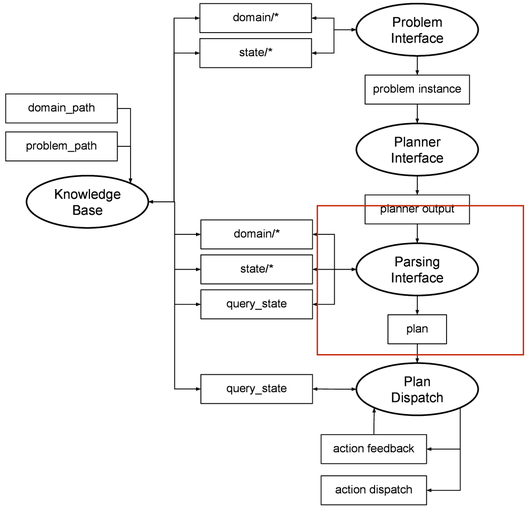


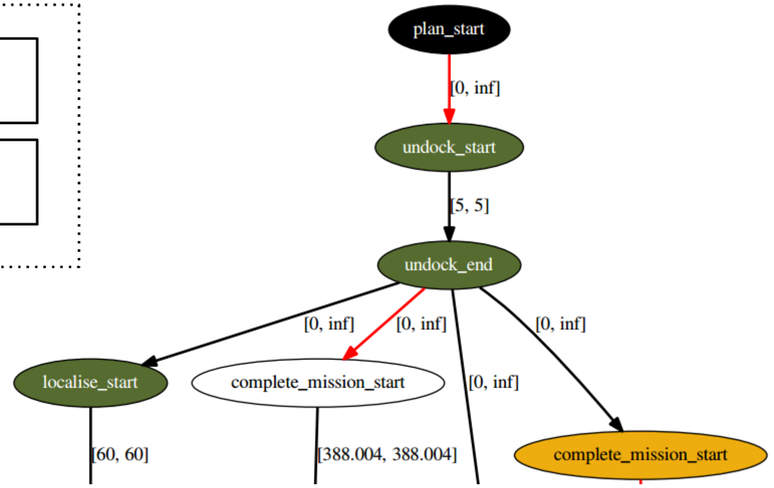


Ejemplos de sistemas que usan planificación

ROSPlan



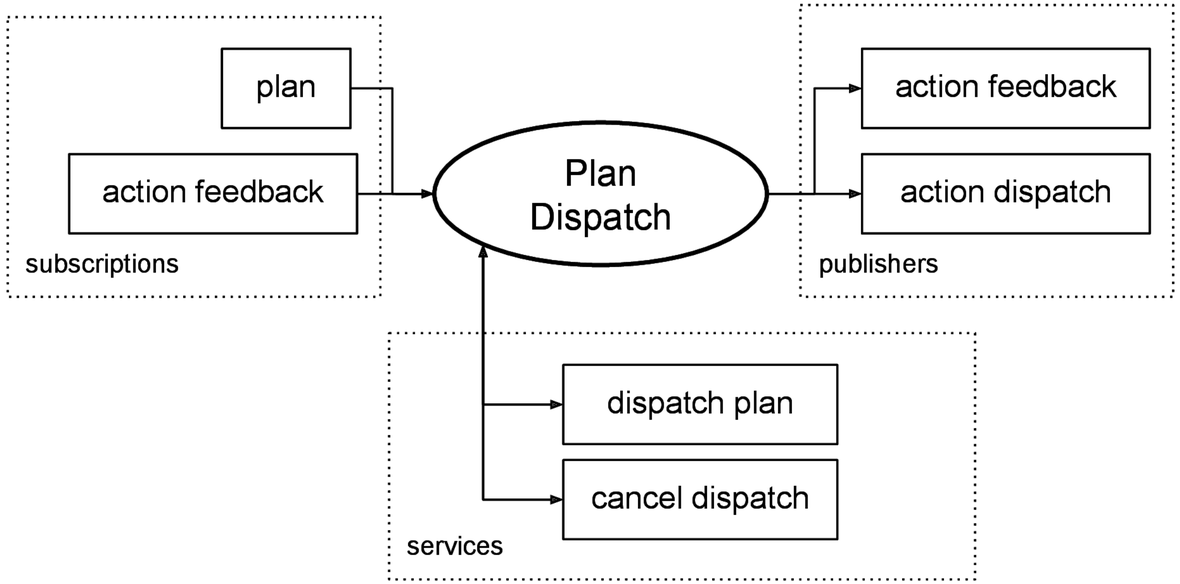


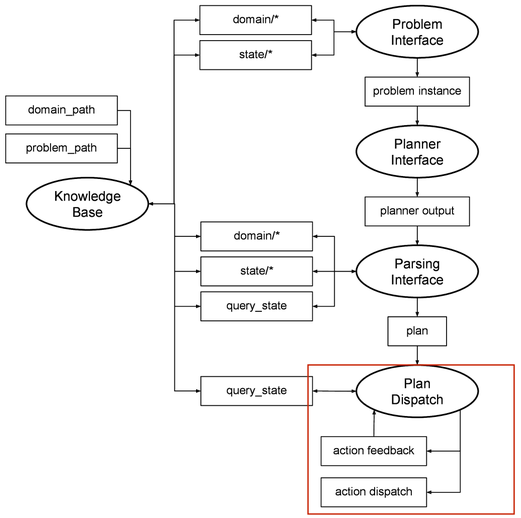


Intelligent Robotics Lab

Ejemplos de sistemas que usan planificación

ROSPlan

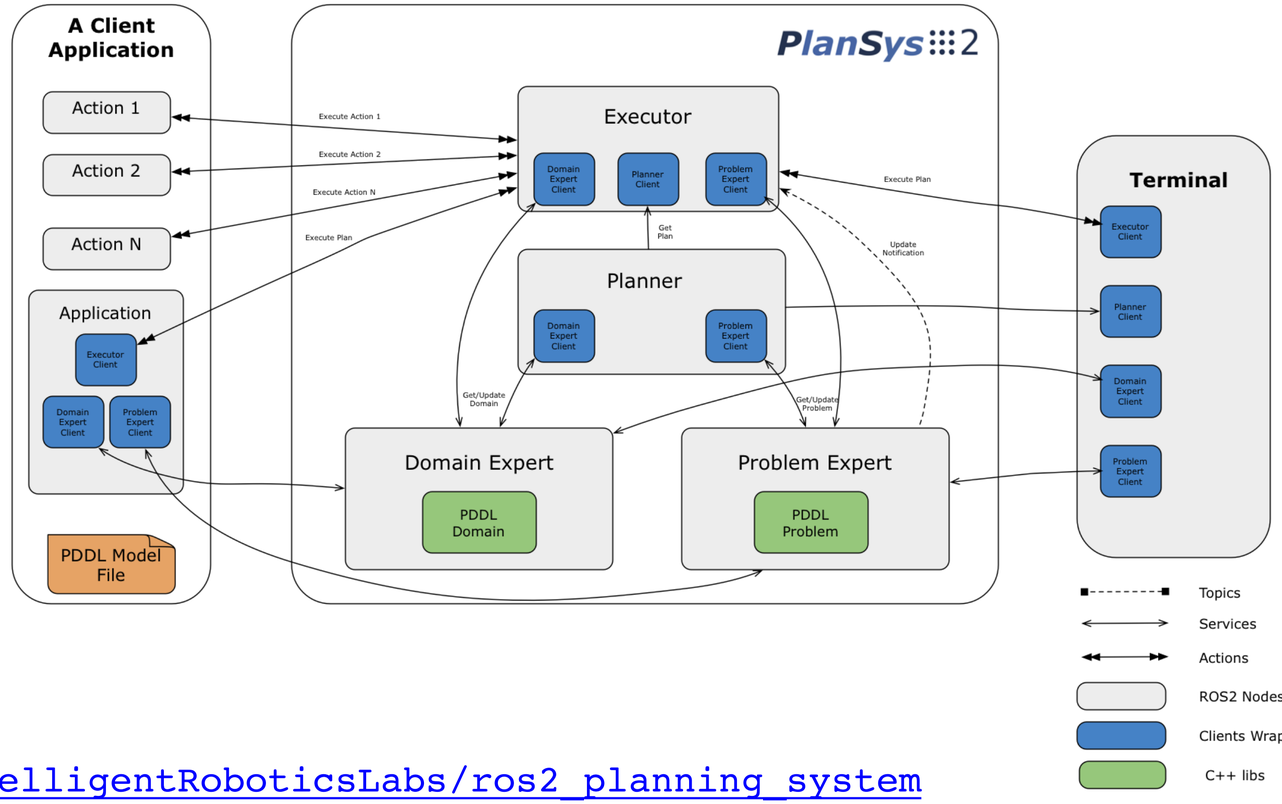




Ejemplos de sistemas que usan planificación

PlanSys2

- Inspirado en ROSPlan
- Disponible en ROS2
- Eficiente, predecible, seguro
- Multi-robot



https://github.com/IntelligentRoboticsLabs/ros2_planning_system

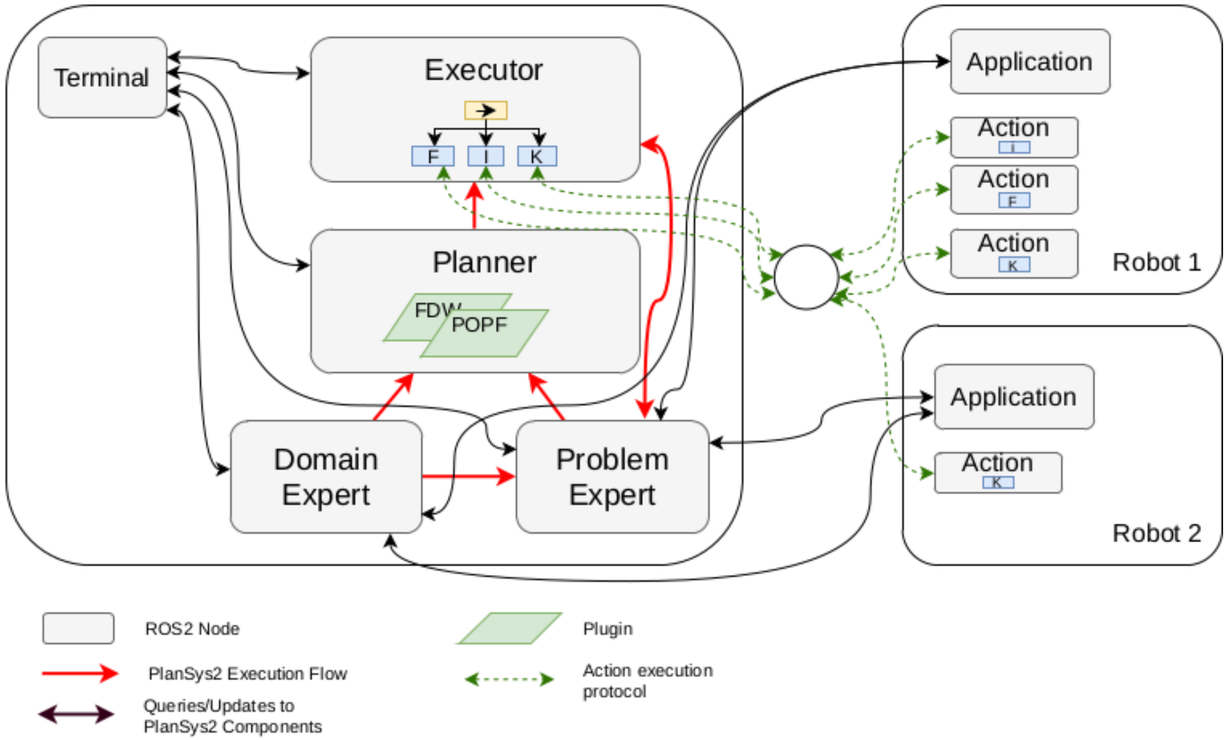
https://intelligentroboticslab.gsyc.urjc.es/ros2_planning_system.github.io/

Intelligent Robotics Lab

Ejemplos de sistemas que usan planificación

PlanSys2

- Inspirado en ROSPlan
- Disponible en ROS2
- Eficiente, predecible, seguro
- Multi-robot

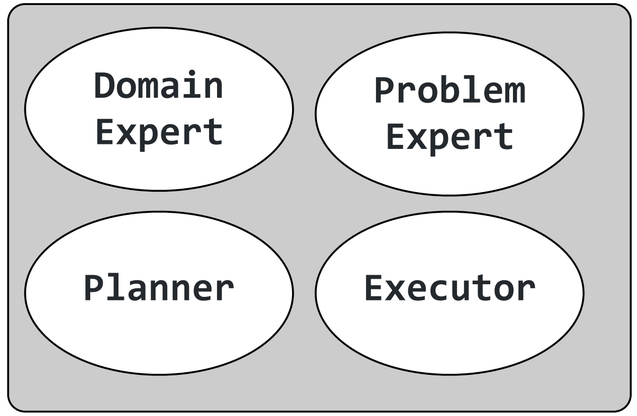


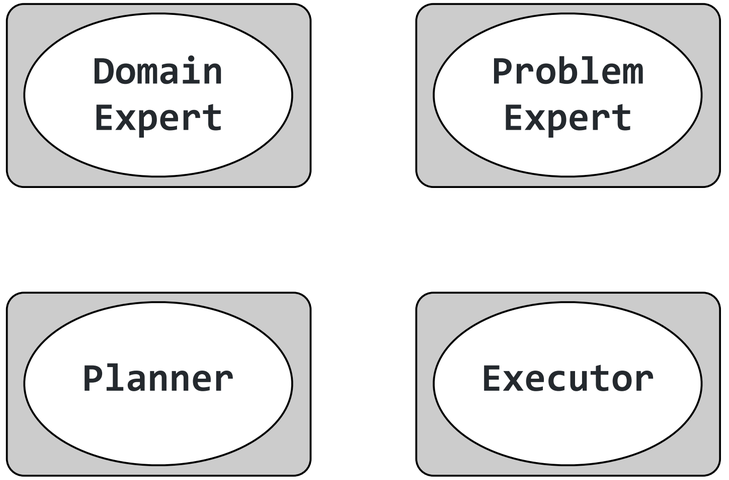
https://github.com/IntelligentRoboticsLabs/ros2_planning_system

https://intelligentroboticslab.gsyc.urjc.es/ros2_planning_system.github.io/

PlanSys2: Arranque

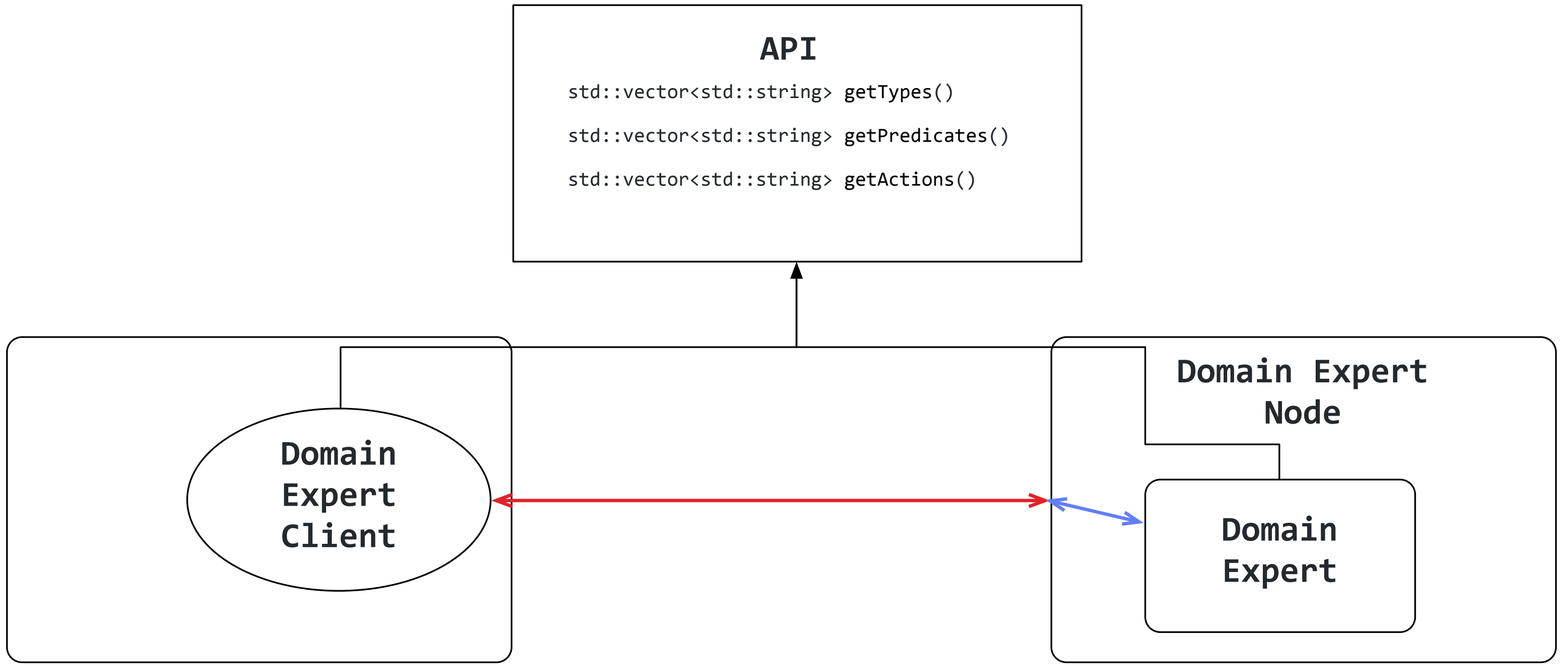
- El arranque del sistema lo lleva a cabo `plansys2_bringup` en dos modos:
  - Monolithic: Todos los componentes en un solo proceso
    - [+] Comunicaciones más rápidas (shared memory)
    - [+] Launcher menos complejo
    - [-] Un fallo en un componente hace fallar el proceso
  - Distributed: Cada componente en un proceso
    - [+] Permite arrancar y depurar componentes por separado
- Los componentes de PlanSys2 son LifeCycle Nodes
  - Orquestado por el el `plansy2_lifecycle_manager`
  - Primero arranca el domain y el problem expert
- Puedes tener varias instancias de PlanSys2 en diferentes namespaces





PlanSys2: Clientes

- La lógica del sistema de cada componente es independiente de ROS2
  - Permite depurar su funcionalidad por separado
  - Permite futuras migraciones en su estructura
- Tiene un interfaz que se reproduce en un cliente
- El nodo que recubre cada componente y el cliente ocultan la complejidad de las comunicaciones con servicios

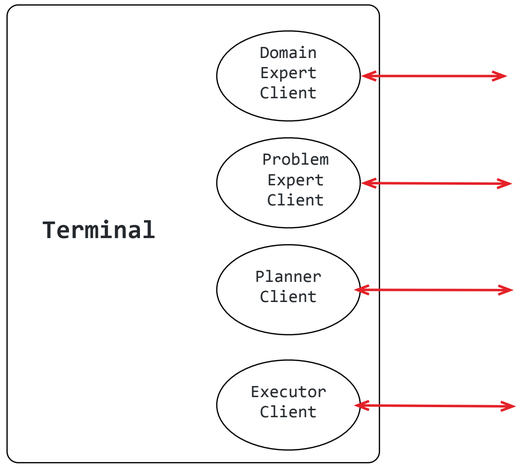


PlanSys2: Terminal

- Permite interactuar con PlanSys2 para gestionar/monitorizar su funcionamiento
- Interfaz shell con funcionalidades avanzadas

- get
  - model
    - types
    - predicates
    - functions
    - actions
    - predicate [predicate]
    - function [function]
    - action [action]
  - problem
    - instances
    - predicates
    - functions
    - goal
  - domain
  - plan

- set
  - instance [id] [type]
  - predicate [(predicate)]
  - function [(function)]
  - goal [(and(goal))]
- remove
  - instance [id]
  - predicate [(predicate)]
  - function [(function)]
  - goal [(and(goal))]
- run
- check
  - actors



PlanSys2

Ejercicio en clase

- Instalación
- Ejecución del ejemplo sensible
- Ejecución de la terminal de PlanSys2

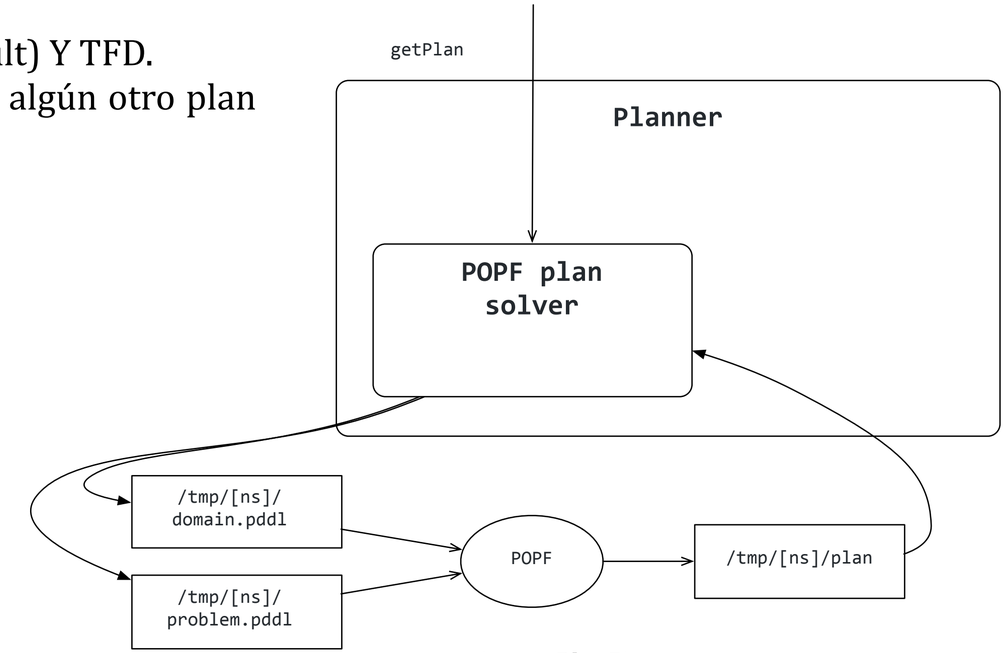
https://github.com/IntelligentRoboticsLabs/ros2_planning_system

https://github.com/PlanSys2/ros2_planning_system_examples.git

https://intelligentroboticslab.gsyc.urjc.es/ros2_planning_system.github.io/

PlanSys2: Planner

- El planner pide el dominio y el problema pddl, y llama al plan solver
- Cada uno es un plugin. Actualmente POPF (default) Y TFD.
- **Ejercicio propuesto**: Crea tu propio plugin con algún otro plan solver que analizaste.



plansys2_params.yaml

```
planner:
  ros__parameters:
    plan_solver_plugins: ["POPF"]
    POPF:
      plugin: "plansys2/POPFPlanSolver"
    TFD:
      plugin: "plansys2::TFDPlanSolver"
```

```
struct PlanItem
{
  float time;
  std::string action;
  float duration;
};

typedef std::vector<PlanItem> Plan;
```

https://github.com/IntelligentRoboticsLabs/plansys2_tfd_plan_solver

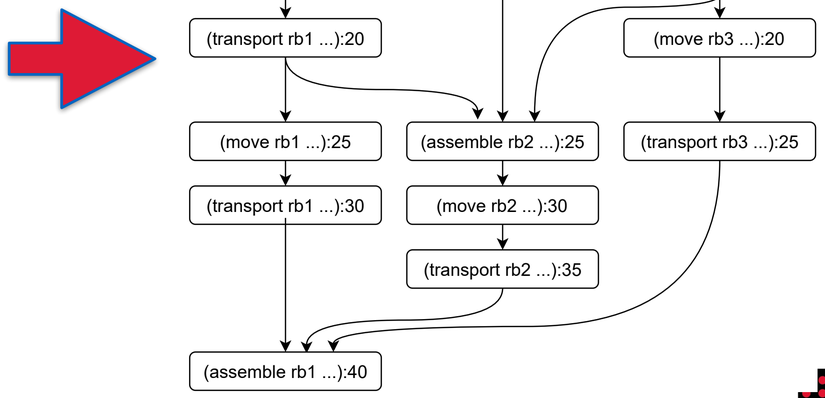
https://github.com/IntelligentRoboticsLabs/ros2_planning_system/tree/master/plansys2_popf_plan_solver

Intelligent Robotics Lab

PlanSys2: Executor

- El Executor es el más complejo de los componentes
- Transforma el plan a un grafo, y luego a un BT

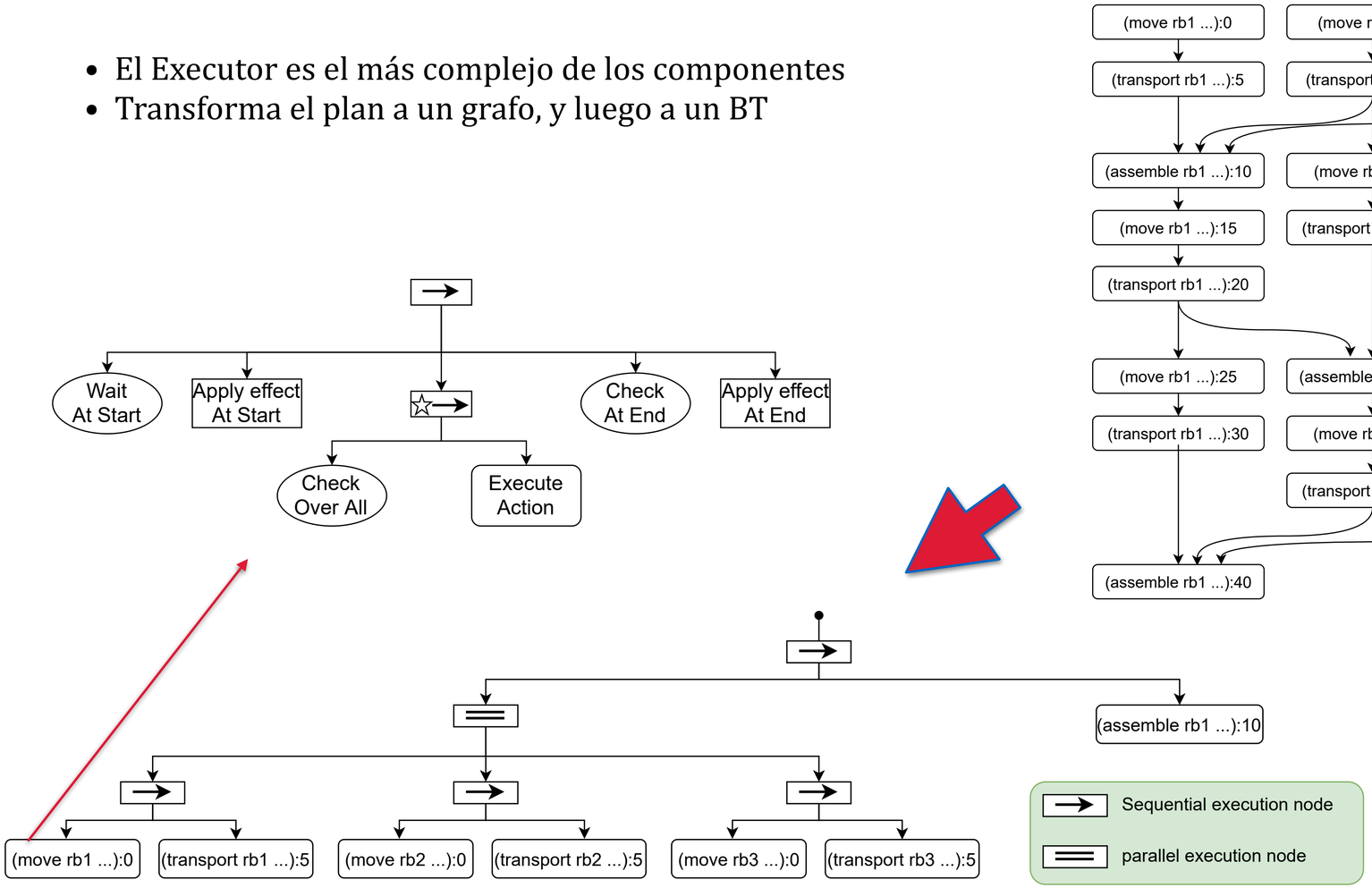
```
0       (move rb1 assembly_zone body_car_zone)
0       (move rb2 assembly_zone steerwheel_zone)
0       (move rb3 assembly_zone wheels_zone)
5.001   (transport rb1 bc_1 body_car_zone assembly_zone)
5.001   (transport rb2 stwhl_1 steerwheel_zone assembly_zone)
5.001   (transport rb3 whl_1 wheels_zone assembly_zone)
10.002  (assemble rb1 assembly_zone whl_1 bc_1 stwhl_1 car_1)
10.002  (move rb2 assembly_zone body_car_zone)
10.002  (move rb3 assembly_zone steerwheel_zone)
15.003  (move rb1 assembly_zone wheels_zone)
15.003  (transport rb2 bc_2 body_car_zone assembly_zone)
15.003  (transport rb3 stwhl_2 steerwheel_zone assembly_zone)
20.004  (transport rb1 whl_2 wheels_zone assembly_zone)
20.004  (move rb3 assembly_zone body_car_zone)
25.005  (assemble rb2 assembly_zone whl_2 bc_2 stwhl_2 car_2)
25.005  (move rb1 assembly_zone steerwheel_zone)
25.005  (transport rb3 bc_3 body_car_zone assembly_zone)
30.006  (move rb2 assembly_zone wheels_zone)
30.006  (transport rb1 stwhl_3 steerwheel_zone assembly_zone)
35.007  (transport rb2 whl_3 wheels_zone assembly_zone)
40.008  (assemble rb1 assembly_zone whl_3 bc_3 stwhl_3 car_3)
```



Intelligent Robotics Lab

PlanSys2: Executor

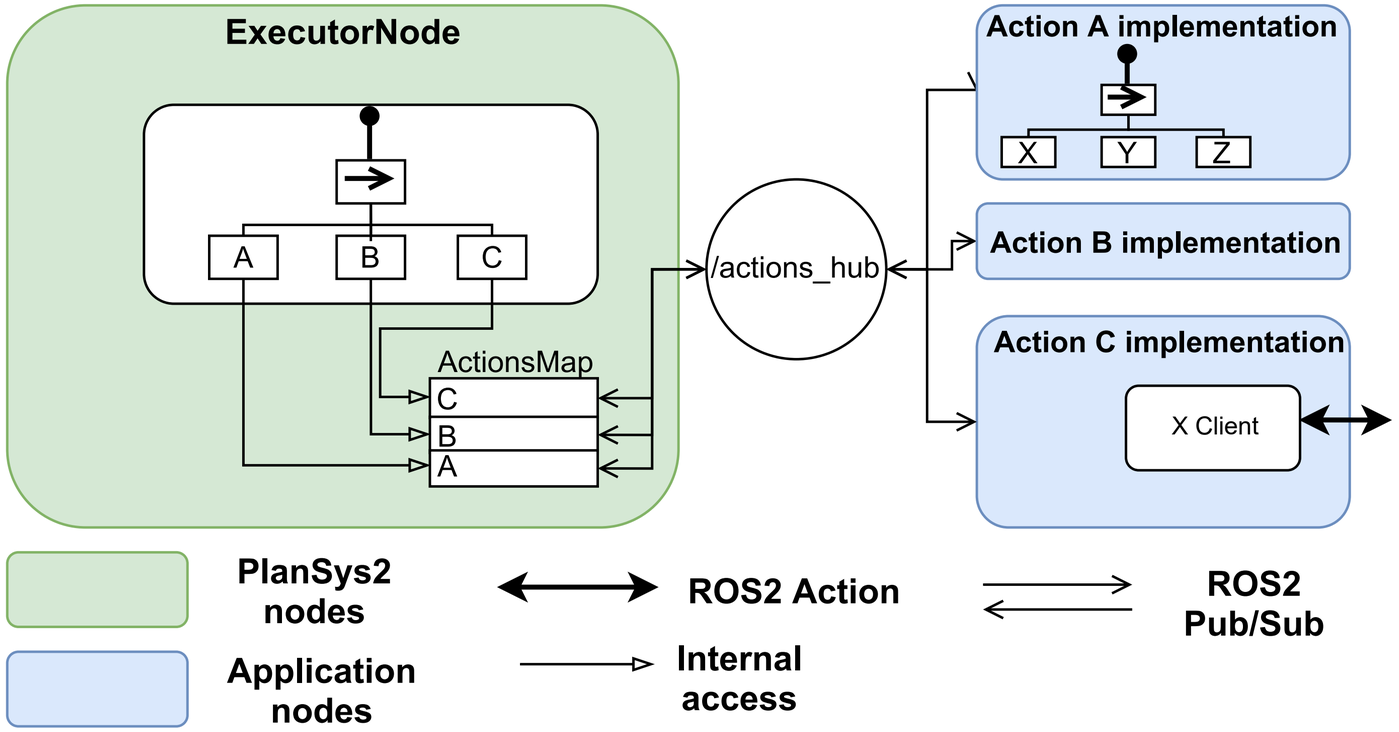
- El Executor es el más complejo de los componentes
- Transforma el plan a un grafo, y luego a un BT



Intelligent Robotics Lab

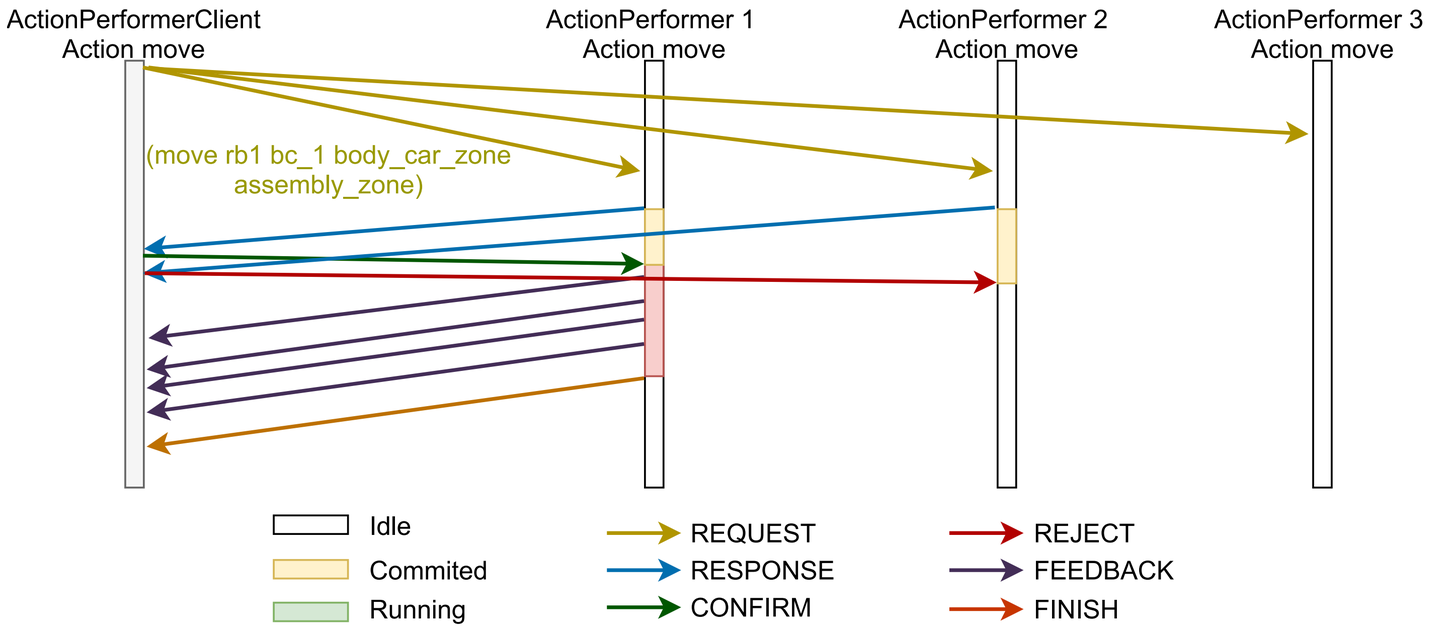
PlanSys2: Executor

- Cada acción debe tener una implementación a través de un actor (`ActionExecutorClient`)
- Las acciones a ejecutar se subastan
  - Puede haber más de un actor por acción
  - Actores especializados
  - Multi-robot



PlanSys2: Executor

- Cada acción debe tener una implementación a través de un actor (`ActionExecutorClient`)
- Las acciones a ejecutar se subastan
  - Puede haber más de un actor por acción
  - Actores especializados
  - Multi-robot



PlanSys2: Executor

- Los actores pueden estar implementados por BTs o no