# Proyecto abierto: arquitectura Misión-Tarea-Capacidad con FSM y Behavior Trees

## Enunciado de la práctica con FSM y Behavior Trees

Francisco Martín Rico y Rodrigo Pérez Rodríguez

## Qué es esta práctica

- Esta práctica es un **proyecto abierto en grupos de cuatro** para diseñar e implementar un sistema robótico completo.
- El foco ya no está en una mini-teoría adicional, sino en **aplicar de forma integrada** la arquitectura software vista durante el curso.
- La arquitectura exigida es **Misión–Tarea–Capacidad**:
  - **Misión**: coordinación global mediante una FSM.
  - **Tareas**: comportamientos encapsulados como Behavior Trees.
  - **Capacidades**: navegación, percepción, diálogo, actuación, etc.
- El objetivo es justificar y materializar decisiones arquitectónicas reales, no sólo hacer que el robot "funcione".

# Objetivos de aprendizaje

- Diseñar una arquitectura robótica completa con separación clara entre misión, tareas y capacidades.
- Usar una **FSM** para coordinar estados de misión y transiciones globales.
- Modelar tareas complejas con **Behavior Trees** reutilizables y mantenibles.
- Integrar **lifecycle nodes** en ROS 2 para gestionar recursos y ejecución de tareas.
- Construir nodos genéricos y parametrizables que eviten duplicación de código.
- Argumentar las decisiones tomadas con criterios de arquitectura software, observabilidad y robustez.

# Organización del trabajo

- El proyecto se realiza en **grupos de cuatro estudiantes**.
- Cada grupo debe proponer una idea propia o adaptar una de las sugeridas en el enunciado.
- Antes de implementar, hay que entregar una **propuesta inicial** y obtener validación del profesorado.
- Esa propuesta debe incluir al menos:
  - descripción de la misión global,
  - estados y transiciones de la FSM,
  - tareas que serán BTs,
  - capacidades necesarias,
  - conexión explícita con conceptos del curso.
- No se debe arrancar el desarrollo fuerte sin esa validación previa.

# Arquitectura obligatoria del proyecto

- **Nivel de misión**: FSM con al menos **4 estados** y transiciones justificadas por eventos observables.
- **Nivel de tareas**: al menos **3 tareas** implementadas como Behavior Trees ejecutados en nodos lifecycle.
- **Nivel de capacidades**: nodos BT reutilizables para navegación, percepción, actuación, diálogo o capacidades equivalentes.
- **Integración FSM–Lifecycle**: la FSM debe gestionar transiciones `configure`, `activate`, `deactivate` y `cleanup`.
- **Nodo genérico parametrizable**: se pide reutilizar el patrón `BTTaskNode` para ejecutar distintos BT a partir de parámetros ROS.

## Requisitos técnicos mínimos

- Implementación completa en **ROS 2 Humble o superior**.
- Uso de **BehaviorTree.CPP v4.x** para las tareas.
- Los nodos de tarea deben heredar de `rclcpp_lifecycle::LifecycleNode`.
- El sistema debe ejecutarse con **MultiThreadedExecutor** para coordinar varios nodos lifecycle.
- Los BT deben definirse en **XML externos**, no embebidos en C++.
- Debe existir al menos **una capacidad real** integrada; el resto pueden simularse si tiene sentido.
- Se debe publicar el estado de misión en un topic ROS y disponer de un launch file parametrizable.

## Qué hay que documentar y entregar

- **Código fuente**: paquete ROS 2 completo, XML de BTs, launch files, `CMakeLists.txt` y `package.xml`.
- **Diagramas**: arquitectura Misión–Tarea–Capacidad, FSM de misión y BTs principales.
- **README técnico**: instalación, compilación, ejecución, dependencias y ejemplos de uso.
- **Justificación arquitectónica**: explicar por qué FSM para misión y BT para tareas, y qué alternativas se descartaron.
- **Entrega**: aceptad la invitación de GitHub Classroom en classroom.github.com/a/wGKnH06V.
- **Demostración**: vídeo de 3–5 minutos, presentación de 10 minutos y defensa técnica.

# Ejemplo de referencia que debéis estudiar

- El paquete `mission_task_example` sirve como **referencia arquitectónica obligatoria**.
- Su estructura resume exactamente el patrón pedido:
  - `TaskLifecycleNode`: clase base abstracta para tareas lifecycle.
  - `BTTaskNode`: nodo genérico que ejecuta un BT indicado por parámetro.
  - `MissionExecutor`: FSM que coordina estados y transiciones lifecycle.
  - `main`: instancia tareas, las conecta con la FSM y ejecuta todo con `MultiThreadedExecutor`.
- No se trata de copiarlo sin criterio, sino de reutilizar su organización y adaptarla al proyecto del grupo.

# Plan de desarrollo recomendado

1. **Semana 1**: definir misión, FSM, tareas, capacidades y validar la propuesta.
2. **Semana 2**: crear el paquete ROS 2 e implementar la base `TaskLifecycleNode`/`BTTaskNode`.
3. **Semana 3**: implementar la FSM, las transiciones lifecycle y el programa principal.
4. **Semana 4**: integrar una capacidad real y refinar timeouts, errores y recuperación.
5. **Semana 5**: cerrar documentación, diagramas, demo y presentación final.

## Idea práctica

Empezad con una versión mínima que compile, ejecute una misión simple y publique estado; luego añadid complejidad.

# Ideas de proyecto válidas

- **Robot asistente de oficina**: mensajería, recordatorios y entrega de objetos.
- **Robot guía de museo**: detección de visitante, selección de tour y guiado por salas.
- **Robot de mantenimiento de almacén**: inspección por zonas, captura de datos y reporte.
- **Robot de limpieza hospitalaria**: asignación de habitación, limpieza, desinfección y reporte.
- **Robot de servicio en restaurante**: espera, toma de pedido, servicio y limpieza de mesa.
- También podéis proponer una idea propia si mantiene la complejidad y la arquitectura exigidas.

## Cómo se evaluará

- **Arquitectura y diseño (40 %)**:
  - corrección del patrón Misión–Tarea–Capacidad,
  - separación de responsabilidades,
  - uso adecuado de FSM, BT y lifecycle.
- **Implementación técnica (30 %)**:
  - código limpio,
  - buen uso de ROS 2,
  - configuración externa,
  - robustez y capacidad real integrada.
- **Documentación y comunicación (20 %)** y **creatividad/complejidad (10 %)**.

## Cómo se evaluará: presentación final

- El día de corrección, que será el **último día de clase**, cada equipo hará una presentación pública del proyecto.
- Se contará con **10 minutos** para exponerlo.
- La presentación se hará con **transparencias en el proyector** y podrá incluir:
  - vídeos del sistema funcionando,
  - una demo en robot real, si se ha utilizado.
- Se valorará:
  - la calidad y claridad de la presentación,
  - lo balanceada que esté la intervención entre los 4 miembros,
  - y la solidez con la que se expliquen los puntos ya recogidos en la evaluación técnica y arquitectónica.

## Recomendaciones finales

- Empezad simple y validad componentes por separado antes de integrar todo el sistema.
- Simular capacidades complejas es válido si eso permite concentrarse en la arquitectura.
- Usad herramientas de observabilidad desde el principio: `ros2 topic echo`, `ros2 lifecycle list`, `rqt_graph`, RViz.
- Documentad conforme avanzáis; no dejéis diagramas y README para el último día.
- Consultad pronto dudas de arquitectura para no invertir semanas en una estructura equivocada.

**El objetivo final es demostrar criterio arquitectónico, no sólo completar una demo.**