



**universidad  
de león**

TESIS DOCTORAL

**GENERACIÓN DE COMPORTAMIENTOS EN  
ROBOTS AUTÓNOMOS MEDIANTE UNA  
ARQUITECTURA COGNITIVA HÍBRIDA**

**Miguel Ángel González Santamarta**

Programa de Doctorado en Ingeniería de Producción y Computación

Tutor: Dr. Vicente Matellán Olivera

Directores:

*Dr. Francisco Javier Rodríguez Lera*

*Dr. Vicente Matellán Olivera*

León, 2023



*Dedicado a mis padres*



# Agradecimientos

En primer lugar, quiero agradecer al Ministerio de Universidades de España por otorgarme la beca para la Formación de Profesorado Universitario (FPU) con número de referencia FPU21/01438.

También quiero dar las gracias a mis directores de tesis, Fran y Vicente, por haber confiado en mí desde el principio y por su gran ayuda en la realización de este trabajo.

Agradezco también a Ángel por creer en mí cuando aún era un estudiante de grado y a Camino y a Lidia por la confianza para dar clase cuando era un estudiante pre-doctoral.

Agradezco al equipo del Grupo de Robótica de la Universidad de León por su apoyo y ayuda en los momentos buenos y, sobre todo, en los difíciles. Quiero destacar especialmente a mis compañeros doctores Claudia, Adrián, Alexis, Gonzalo y Virginia; y a mis compañeros doctorandos Irene y David.

Y, sobre todo, gracias a mi familia por su constante ánimo y por hacer todo lo posible para que pudiera estudiar sin preocupaciones.



# Resumen

En la inteligencia artificial y la robótica, uno de los objetivos que persisten en las últimas décadas es crear robots autónomos que puedan realizar tareas complejas en entornos dinámicos. En estos entornos, los robots deben no solo reaccionar en escenarios predefinidos o deterministas, sino también aprender y adaptarse en tiempo real, reflejando una flexibilidad cognitiva similar a la inteligencia humana. Sobreponerse a estos desafíos requiere el desarrollo de arquitecturas cognitivas que integren capacidades reactivas, deliberativas y emergentes de manera fluida. Se han explorado diversos enfoques, como sistemas deliberativos, arquitecturas de tres capas, modelos reactivos y sistemas emergentes, todos buscando generar comportamientos inteligentes en robots autónomos.

Así, este trabajo de investigación se centra en el diseño, desarrollo, despliegue y evaluación de una arquitectura cognitiva híbrida para generar, controlar, planificar y monitorizar comportamientos en robots autónomos. Esta arquitectura integra componentes reactivos, deliberativos y emergentes, con el objetivo de incrementar la capacidad para adaptarse a entornos dinámicos y tomar decisiones inteligentes en tiempo real, mejorando su autonomía y desempeño. En el panorama actual de la investigación en robótica, el desarrollo de arquitecturas híbridas ha surgido como un camino crítico para abordar las limitaciones de las arquitecturas puramente reactivas o puramente deliberativas. Los sistemas deliberativos son hábiles en el razonamiento simbólico y la planificación, pero tienen dificultades cuando se enfrentan a la naturaleza dinámica e impredecible de los entornos del mundo real. Por otro lado, los sistemas reactivos sobresalen en tareas que necesitan responder de forma rápida a estímulos producidos por los sensores en tiempo real, pero carecen de la capacidad para razonar a alto nivel y planificar. La mezcla de las

capacidades deliberativas, reactivas y emergentes junto a la organización en capas da como resultado la arquitectura que se presenta en este trabajo.

La revisión sistemática de la literatura llevada a cabo se ha centrado en los artículos de investigación recientes que tratan este tema. De esta manera, la revisión da como resultado las arquitecturas cognitivas recientes, las herramientas empleadas para generar comportamientos, los métodos y métricas utilizados para evaluar las arquitecturas. De esta forma, se han obtenido herramientas como las máquinas de estados, los árboles de comportamiento, el PDDL, los planificadores simbólicos, los grafos de conocimiento y diferentes tipos de componentes emergentes. También se han identificado diferentes formas para evaluar las arquitecturas cognitivas, entre las que destaca el uso de experimentos con interacción con humanos; y las métricas empleadas, como el tiempo de ejecución, la distancia recorrida y el rendimiento.

Con todo esto, se presenta una arquitectura cognitiva híbrida para robots autónomos integrada en ROS 2. La arquitectura cognitiva, llamada MERLIN2, consta de un sistema deliberativo, basada en una base de conocimiento y un planificador simbólico; un sistema de comportamientos, compuesto por componentes reactivos; y varios componentes emergentes. De esta manera, se tratan los principales aspectos cognitivos, como la percepción, la selección de acciones, la memoria, el aprendizaje, el razonamiento y la explicabilidad. Por último, se presenta la experimentación, que muestra que la arquitectura es una solución válida para robots autónomos.



# Abstract

In the field of artificial intelligence and robotics, one enduring objective is to create autonomous robots capable of performing complex tasks in dynamic environments. In these environments, robots not only need to react in predefined or deterministic scenarios but also learn and adapt in real-time, mirroring a cognitive flexibility akin to human intelligence. Achieving this autonomy entails developing cognitive architectures that seamlessly integrate reactive, deliberative, and emergent capabilities. Over the last few decades, various approaches—deliberative systems, three-layer architectures, reactive models, and emergent systems—have been explored, all aiming to generate intelligent behaviors in autonomous robots.

Thus, this research work focuses on the design, development, deployment and evaluation of a hybrid cognitive architecture to generate, control, plan, and monitor behaviors in autonomous robots. This architecture amalgamates reactive, deliberative, and emergent components, aiming to enhance adaptability in dynamic environments and make intelligent real-time decisions, thereby improving autonomy and performance. In the current landscape of robotics research, the development of hybrid architectures has emerged as a critical path to address limitations seen in purely reactive or purely deliberative architectures. Deliberative systems excel in symbolic reasoning and planning but face challenges when dealing with the dynamic and unpredictable nature of real-world environments. Conversely, reactive systems excel in tasks requiring rapid responses to real-time sensor stimuli but lack high-level reasoning and planning capabilities. The combination of deliberative, reactive, and emergent capacities, alongside layered organization, results in the architecture presented in this work.

The systematic literature review undertaken has focused on recent research

articles addressing this topic, resulting in insights into recent cognitive architectures, behavior-generating tools, methods, and metrics used to evaluate these architectures. Tools such as state machines, behavior trees, PDDL, symbolic planners, knowledge graphs, and various types of emergent components have been identified. Different methods for evaluating cognitive architectures, notably experiments involving human interaction, and metrics such as execution time, distance traveled, and performance, have also been highlighted.

In conclusion, a hybrid cognitive architecture integrated into ROS 2 for autonomous robots, termed MERLIN2, is presented. MERLIN2 comprises a deliberative system based on a knowledge base and a symbolic planner, a behavior system composed of reactive components, and several emergent components. It addresses core cognitive aspects like perception, action selection, memory, learning, reasoning, and explainability. Finally, the experimentation presented showcases the architecture as a valid solution for autonomous robots.

# Índice general

<b>1. Introducción</b>	<b>1</b>
1.1. Robots Autónomos . . . . .	2
1.2. La Inteligencia Artificial en la Robótica . . . . .	3
1.3. Arquitecturas Cognitivas Híbridas . . . . .	4
1.4. Motivación . . . . .	5
1.5. Planteamiento del trabajo de investigación . . . . .	6
1.5.1. Hipótesis de investigación . . . . .	6
1.5.2. Objetivos . . . . .	6
1.6. Estructura del trabajo de investigación . . . . .	7
<b>2. Revisión de la literatura</b>	<b>9</b>
2.1. Introducción . . . . .	9
2.2. Contexto . . . . .	10
2.2.1. Clasificación de arquitecturas de control . . . . .	10
2.2.2. Clasificación de arquitecturas cognitivas . . . . .	14
2.2.3. Preguntas de investigación . . . . .	18
2.3. Metodología . . . . .	20
2.3.1. Proceso de búsqueda . . . . .	20
2.3.2. Proceso de selección . . . . .	21
2.3.3. Proceso de evaluación de la calidad . . . . .	23
2.3.4. Procesos de extracción y síntesis de datos . . . . .	25
2.4. Resultados . . . . .	26
2.4.1. Procesos de búsqueda y selección . . . . .	26
2.4.2. Proceso de evaluación de la calidad . . . . .	27

2.4.3.	Extracción y síntesis de datos . . . . .	29
2.5.	Discusión . . . . .	36
2.5.1.	¿Qué arquitecturas cognitivas contempla el estado de la cuestión para robots autónomos? . . . . .	36
2.5.2.	¿Qué herramientas se utilizan en las arquitecturas cognitivas para generar comportamientos en robots autónomos? . . . . .	37
2.5.3.	¿Qué métodos se emplean para evaluar las arquitecturas cognitivas? . . . . .	40
2.5.4.	¿Qué métricas se utilizan para evaluar las arquitecturas cognitivas? . . . . .	41
2.6.	Conclusiones . . . . .	42
<b>3.</b>	<b>Arquitectura Cognitiva</b>	<b>45</b>
3.1.	Visión General . . . . .	45
3.1.1.	Aproximación Bioinspirada . . . . .	46
3.2.	Núcleo de la Arquitectura . . . . .	48
3.2.1.	Percepción . . . . .	48
3.2.2.	Selección de acciones . . . . .	49
3.2.3.	Memoria . . . . .	53
3.2.4.	Aprendizaje . . . . .	54
3.2.5.	Razonamiento . . . . .	54
3.2.6.	Explicabilidad . . . . .	55
3.3.	Sistema Deliberativo . . . . .	60
3.4.	Sistema de comportamientos . . . . .	60
3.5.	Conclusiones . . . . .	61
<b>4.</b>	<b>MERLIN</b>	<b>63</b>
4.1.	Visión General . . . . .	63
4.2.	Núcleo de MERLIN . . . . .	65
4.2.1.	Percepción . . . . .	66
4.2.2.	Selección de acciones . . . . .	66
4.2.3.	Memoria . . . . .	68

4.2.4. Aprendizaje . . . . .	68
4.2.5. Razonamiento . . . . .	68
4.2.6. Explicabilidad . . . . .	68
4.3. Sistema deliberativo . . . . .	68
4.3.1. Mission Layer . . . . .	69
4.3.2. Planning Layer . . . . .	69
4.4. Sistema de comportamientos . . . . .	70
4.4.1. Executive Layer . . . . .	71
4.4.2. Reactive Layer . . . . .	72
4.5. Limitaciones . . . . .	73
<b>5. MERLIN2</b>	<b>75</b>
5.1. Visión general . . . . .	75
5.2. Núcleo de MERLIN2 . . . . .	77
5.2.1. Percepción . . . . .	77
5.2.2. Selección de acciones . . . . .	79
5.2.3. Memoria . . . . .	83
5.2.4. Aprendizaje . . . . .	89
5.2.5. Razonamiento . . . . .	90
5.2.6. Explicabilidad . . . . .	93
5.3. Sistema deliberativo . . . . .	93
5.3.1. Mission Layer . . . . .	93
5.3.2. Planning Layer . . . . .	93
5.4. Sistema de comportamientos . . . . .	98
5.4.1. Executive Layer . . . . .	98
5.4.2. Reactive Layer . . . . .	99
5.5. Limitaciones . . . . .	99
<b>6. Evaluación de la Arquitectura Cognitiva</b>	<b>101</b>
6.1. Aplicación y despliegue . . . . .	102
6.1.1. MERLIN . . . . .	102
6.1.2. MERLIN2 . . . . .	105

6.2. Evaluación de YASMIN . . . . .	107
6.2.1. Discusión . . . . .	109
6.3. Evaluación de KANT . . . . .	111
6.3.1. Discusión . . . . .	112
6.4. Evaluación de la arquitectura cognitiva . . . . .	113
6.4.1. Discusión . . . . .	116
6.4.2. Conclusiones . . . . .	118
<b>7. Conclusiones</b>	<b>121</b>
7.1. Validación de la hipótesis y objetivos . . . . .	121
7.2. Contribuciones . . . . .	124
7.3. Trabajo futuro . . . . .	126
<b>A. Diagramas de clases UML</b>	<b>143</b>
<b>B. Repositorios públicos</b>	<b>159</b>
<b>C. Prompts utilizados</b>	<b>161</b>

# Índice de figuras

2.1. Clasificación de arquitecturas de control . . . . .	11
2.2. Clasificación de arquitecturas cognitivas . . . . .	15
2.3. Ventajas e inconvenientes de los tipos de arquitecturas cognitivas.	17
2.4. Diagrama de los procesos de búsqueda y selección . . . . .	22
2.5. Diagrama PRISMA de la revisión sistemática . . . . .	26
3.1. Representación bioinspirada de la arquitectura cognitiva . . . . .	47
3.2. Dimensiones para un robot responsable. . . . .	58
4.1. Diagrama de MERLIN . . . . .	64
4.2. Diagrama de clases de las máquinas de estados de MERLIN . . . . .	66
4.3. Diagrama de la <i>Mission Layer</i> y el <i>Executor</i> de la <i>Planning Layer</i> . . . . .	69
4.4. Diagrama de clases de la <i>Executive Layer</i> . . . . .	70
4.5. Diagrama de la <i>Executive Layer</i> y ROSPlan de la <i>Executive Layer</i> . . . . .	72
4.6. Diagrama de la <i>Executive Layer</i> y la <i>Reactive Layer</i> . . . . .	73
5.1. Diagrama de MERLIN2 . . . . .	76
5.2. Ejemplo de YOLOv8 en RViz . . . . .	78
5.3. Ejemplo del visualizador de YASMIN . . . . .	82
5.4. KANT en una arquitectura genérica . . . . .	84
5.5. Arquitectura software de KANT . . . . .	87
5.6. Diagrama de clases de llama_ros . . . . .	92
5.7. Máquina de estados de la <i>Planning Layer</i> original . . . . .	95
5.8. Nueva máquina de estados para la <i>Planning Layer</i> con LLMs . . . . .	96
5.9. Ejemplo de grafo de conocimiento . . . . .	97

5.10. Ejemplo de árbol de comportamientos visualizado en Groot . . . . .	98
6.1. Robot RB1 . . . . .	101
6.2. Diagrama de flujo para utilizar MERLIN . . . . .	103
6.3. Ejemplo del SMACH Viewer . . . . .	105
6.4. Diagrama de flujo para utilizar MERLIN2 . . . . .	106
6.5. Ejemplo de máquina de estados que implementa la <i>Mission Layer</i> .	107
6.6. Vista aérea del almacén de AWS Robomaker . . . . .	108
6.7. Diagramas <i>BoxPlot</i> del experimento de YASMIN . . . . .	110
6.8. Vista aérea del apartamento simulado . . . . .	114
A.1. Diagrama de clases del paquete de ROS 2 <i>kant_knowledge_base</i> . .	143
A.2. Diagrama de clases del paquete de ROS 2 <i>simple_node</i> . . . . .	144
A.3. Diagrama de clases del paquete de ROS 2 <i>kant_dto</i> . . . . .	145
A.4. Diagrama de clases del paquete de ROS 2 <i>dao_factory</i> de KANT. .	146
A.5. Diagrama de clases del paquete de ROS 2 <i>mongo_dao</i> de KANT. .	147
A.6. Diagrama de clases del paquete de ROS 2 <i>ros2_dao</i> de KANT. . . .	148
A.7. Diagrama de clases de YASMIN. . . . .	149
A.8. Diagrama de clases de la <i>Mission Layer</i> de MERLIN2. . . . .	150
A.9. Diagrama de clases de la <i>Planning Layer</i> de MERLIN2. . . . .	151
A.10. Diagrama de clases del <i>PDDL Generator</i> de MERLIN2. . . . .	152
A.11. Diagrama de clases del <i>Planner</i> de MERLIN2. . . . .	153
A.12. Diagrama de clases del <i>Executor</i> de MERLIN2. . . . .	154
A.13. Diagrama de clases de la <i>Executive Layer</i> de MERLIN2. . . . .	155
A.14. Diagrama de clases de la <i>Reactive Layer</i> de MERLIN2. . . . .	156
A.15. Diagrama de clases de la demo de MERLIN2. . . . .	157
C.1. <i>Prompt</i> para planificar (I). . . . .	162
C.2. <i>Prompt</i> para planificar (II). . . . .	162
C.3. Respuesta al <i>prompt</i> para planificar. . . . .	163
C.4. <i>Prompt</i> para verificar el objetivo. . . . .	163



# Índice de tablas

2.1. Criterios de inclusión . . . . .	23
2.2. Criterios de exclusión . . . . .	23
2.3. Preguntas de la evaluación de la calidad . . . . .	24
2.4. Evaluación de la calidad . . . . .	27
2.5. Artículos finales . . . . .	28
2.6. Datos extraídos para las tres primeras preguntas . . . . .	29
2.7. Datos extraídos para la cuarta pregunta . . . . .	30
2.8. Datos extraídos para la quinta pregunta . . . . .	32
2.9. Datos extraídos para la sexta pregunta . . . . .	34
6.1. Resultados obtenidos durante la experimentación de YAMSIN . . .	110
6.2. Resultados obtenidos del experimento de KANT (I) . . . . .	112
6.3. Resultados obtenidos del experimento de KANT (II) . . . . .	113
6.4. Resultados obtenidos al evaluar la arquitectura cognitiva (I) . . . .	117
6.5. Resultados obtenidos al evaluar la arquitectura cognitiva (II) . . .	117
B.1. Repositorios públicos de GitHub . . . . .	159



# Capítulo 1

## Introducción

En el ámbito de la inteligencia artificial y la robótica, la búsqueda de crear robots autónomos capaces de realizar tareas complejas en entornos no estructurados y dinámicos continúa siendo una aspiración generalizada por la comunidad científica. Estos robots no solo deben reaccionar en escenarios predefinidos, sino también adaptarse y aprender en tiempo real, demostrando un nivel de flexibilidad cognitiva que se asemeje a la inteligencia humana. Lograr este nivel de autonomía requiere el desarrollo de arquitecturas de control que puedan combinar de manera fluida las capacidades reactivas y deliberativas.

Durante las últimas décadas se han diseñado y desarrollado varios enfoques en el campo de la generación y gestión de comportamientos en robots autónomos. Algunos de los paradigmas más populares han sido los sistemas deliberativos, las arquitecturas de tres capas, las arquitecturas reactivas y los sistemas emergentes. El objetivo de todos ellos es implementar un sistema inteligente para llevar a cabo la generación de comportamientos en robots autónomos.

De esta forma, este trabajo de investigación explora el diseño, la implementación y la evaluación de una arquitectura cognitiva híbrida para robots autónomos. En el panorama actual de la investigación en robótica, el desarrollo de sistemas híbridos ha surgido como un camino crítico para abordar las limitaciones de las arquitecturas puramente reactivas o puramente deliberativas. Si bien los sistemas deliberativos son hábiles en el razonamiento simbólico y la planificación, suelen tener dificultades cuando se enfrentan a la naturaleza dinámica e impredecible

de los entornos del mundo real. Por otro lado, los sistemas reactivos sobresalen en tareas que necesitan responder de forma rápida a estímulos producidos por los sensores en tiempo real, pero a menudo carecen de la capacidad para el razonamiento de alto nivel y la planificación. Así, la mezcla de las capacidades deliberativas, reactivas y emergentes junto a la organización en capas da como resultado la arquitectura que se presenta en este trabajo.

## 1.1. Robots Autónomos

Actualmente, los robots autónomos desempeñan un papel cada vez más crucial en diferentes sectores, desde la exploración espacial hasta la atención médica, la agricultura, la industria, la logística y los servicios. La capacidad de realizar tareas en entornos no estructurados o peligrosos, así como su potencial para aumentar la eficiencia y la seguridad en diversas industrias, ha impulsado un creciente interés en la investigación y desarrollo de sistemas robóticos autónomos. A pesar de los avances significativos, los robots autónomos se enfrentan a una serie de desafíos que limitan su desempeño en situaciones del mundo real. Estos desafíos incluyen la necesidad de navegar en entornos dinámicos y desconocidos, la toma de decisiones en tiempo real basada en datos sensoriales limitados y la adaptación a condiciones cambiantes y escenarios imprevistos. Estos obstáculos han llevado a la búsqueda constante de soluciones innovadoras que permitan a los robots autónomos operar de manera más efectiva y autónoma. La inteligencia artificial desempeña un papel fundamental en la habilitación de la autonomía de los robots. Los avances en el aprendizaje automático, la visión por computadora, la planificación y la toma de decisiones han revolucionado la forma en que los robots pueden interactuar con su entorno y ejecutar tareas. Las arquitecturas cognitivas, que modelan la estructura y el funcionamiento del pensamiento humano, proporcionan una base sólida para la implementación de la inteligencia en los robots, permitiéndoles generar comportamientos autónomos.

## 1.2. La Inteligencia Artificial en la Robótica

La sinergia entre la inteligencia artificial (IA) y la robótica [1] ha sido un factor clave en el avance de los sistemas robóticos, permitiendo que las máquinas adquieran la capacidad de realizar tareas complejas y adaptarse a entornos diversos. La robótica y la inteligencia artificial convergen en un punto donde la robótica impulsa la aplicación práctica de la IA y, a su vez, la IA mejora las capacidades de los robots. La capacidad de procesar datos y tomar decisiones basadas en los datos adquiridos de los sensores son características que han transformado la forma en que los robots interactúan con su entorno y realizan tareas específicas. A medida que la IA evolucionó, se incorporaron técnicas como la planificación de trayectorias, la visión por computadora, el aprendizaje automático y la toma de decisiones basada en reglas y conocimiento simbólico. Estas innovaciones han permitido a los robots funcionar de manera más autónoma y adaptativa. La IA ha influido en una amplia variedad de aplicaciones robóticas, desde la medicina, donde los robots asisten en cirugías complejas, hasta la exploración espacial, donde los robots autónomos exploran planetas lejanos.

Las arquitecturas cognitivas son parte de la investigación en IA en general, que comenzó en la década de 1950 con el objetivo de crear programas que pudieran razonar sobre problemas en diferentes dominios, desarrollar conocimientos, adaptarse a nuevas situaciones y reflexionar sobre sí mismos. Según Russell y Norvig [2], la IA se puede clasificar en cuatro grupos diferentes: sistemas que piensan como humanos, sistemas que piensan racionalmente, sistemas que actúan como humanos y sistemas que actúan racionalmente. Las arquitecturas cognitivas han explorado las cuatro posibilidades. Las arquitecturas cognitivas en la robótica representan una aplicación importante de la IA. Estas arquitecturas modelan la forma en que los seres humanos piensan y toman decisiones y se utilizan para generar comportamientos en robots. Las arquitecturas híbridas, que combinan componentes reactivos y deliberativos, son las más populares, ya que buscan un equilibrio entre la capacidad de respuesta en tiempo real y el razonamiento de alto nivel.

### 1.3. Arquitecturas Cognitivas Híbridas

La robótica autónoma se enfrenta a un conjunto diverso de escenarios en el mundo real, cada uno con sus propias demandas y requisitos. Los robots autónomos deben ser capaces de navegar por entornos dinámicos, tomar decisiones en tiempo real, adaptarse a situaciones imprevistas y aprender de sus experiencias. Las arquitecturas cognitivas son fundamentales para dotar a los robots de las capacidades necesarias para abordar estas situaciones complejas. Las arquitecturas puramente reactivas [3], aunque son efectivas en tareas de respuesta rápida, carecen de la capacidad de planificación a largo plazo y de la adaptación a situaciones no previstas. Por otro lado, las arquitecturas puramente deliberativas [4] son efectivas en la planificación y el razonamiento de alto nivel, pero son ineficientes en aplicaciones que requieren respuestas inmediatas. Estas limitaciones resaltan la necesidad de combinar ambos enfoques en una arquitectura híbrida.

Una arquitectura cognitiva híbrida [5] busca superar las limitaciones de los enfoques puramente reactivos o deliberativos. Al combinar componentes reactivos para la toma de decisiones en tiempo real con módulos deliberativos para la planificación y el razonamiento a largo plazo, se crea un sistema más equilibrado. Esto permite a los robots adaptarse a situaciones cambiantes mientras mantienen la capacidad de planificar y razonar, lo que resulta en una mayor autonomía y versatilidad. La necesidad de una arquitectura cognitiva híbrida se deriva de la complejidad de los desafíos que enfrentan los robots autónomos en el mundo real.

Por otro lado, en revisiones de la literatura más recientes como las de Kotseruba y Tsotsos [6] y Ye, Wang y Wang [7], es posible encontrar aproximaciones donde las arquitecturas cognitivas se clasifican en simbólicas, que son aquellas que usan conocimiento simbólico y sistemas deliberativos; emergentes, que se basan en el conexionismo y en modelos como redes neuronales; e híbridas, que mezclan conceptos de las dos anteriores. Esta clasificación se asemeja a las arquitecturas mencionadas anteriormente, donde las simbólicas equivalen a las deliberativas y las emergentes equivaldrían a las reactivas tradicionales pero usando redes neuronales y modelos bioinspirados.

## 1.4. Motivación

El uso de robots autónomos ha aumentado en los últimos años. Uno de los ámbitos en los que ha aumentado su uso es en el sector servicios, con el objetivo de mejorar la vida de las personas. De esta forma, hay en el mercado un amplio número de robots aspiradores y robots de cocina que realizan tareas repetitivas con muy buenos resultados. Por otro lado, también se tienen robots en otros sectores, como en la agricultura o en la exploración espacial. De esta forma, la motivación detrás de este trabajo de investigación se basa en la creciente importancia de la robótica autónoma en la sociedad contemporánea. Los robots autónomos desempeñan un papel cada vez más vital en una variedad de escenarios. Sin embargo, estos sistemas enfrentan desafíos significativos en términos de adaptación y autonomía, ya que a menudo operan en entornos dinámicos y no estructurados. La necesidad de desarrollar arquitecturas cognitivas avanzadas que permitan a los robots tomar decisiones inteligentes en tiempo real es el motor que impulsa esta investigación.

La generación del comportamiento en robots autónomos es un problema que se ha dado a lo largo de los años y aún no se tiene una solución definida. Se han dado diversos paradigmas para resolver este problema. Algunos de ellos son los sistemas deliberativos, las arquitecturas de tres capas [8] o las arquitecturas reactivas. Con todos estos paradigmas se han desarrollado arquitecturas cognitivas para gestionar los comportamientos de los robots. La creación de una arquitectura cognitiva híbrida que permita generar comportamientos en robots autónomos surge de la idea de que un enfoque equilibrado entre reactividad, deliberación y modelos emergentes puede resolver la forma en que los robots interactúan con su entorno y realizan tareas complejas. Esta investigación se inspira en el potencial de las arquitecturas híbridas, que pueden permitir a los robots afrontar de manera más eficaz las situaciones con las que se encuentren cuando realicen las tareas que tienen asignadas. La motivación principal radica en mejorar la autonomía, la versatilidad y la eficiencia de los robots autónomos, lo que, a su vez, tiene el potencial de generar un impacto positivo en campos tan diversos como la atención médica, la exploración espacial, la industria y los servicios.

## 1.5. Planteamiento del trabajo de investigación

El trabajo de investigación que se presenta tiene como objetivo desarrollar una arquitectura cognitiva híbrida para generar, controlar, planificar y monitorizar comportamientos en robots autónomos. Esta arquitectura busca abordar los desafíos inherentes a la robótica autónoma, equilibrando las capacidades deliberativas, las capacidades reactivas y las capacidades emergentes para mejorar la adaptación y la autonomía de los robots en entornos cambiantes. A lo largo de este trabajo, se explorarán los conceptos teóricos relacionados con las arquitecturas cognitivas, se describirá en detalle la arquitectura propuesta y se llevarán a cabo experimentos y evaluaciones para demostrar su efectividad en aplicaciones del mundo real.

### 1.5.1. Hipótesis de investigación

Basándose en lo expuesto en las secciones anteriores, este trabajo propone la siguiente hipótesis inicial:

**H<sub>0</sub>:** La implementación de una arquitectura cognitiva híbrida para robots autónomos, que integre componentes reactivos, deliberativos y emergentes, conducirá a un incremento significativo en su capacidad para adaptarse a entornos dinámicos y tomar decisiones inteligentes en tiempo real, lo que mejorará su autonomía y desempeño en varias tareas.

### 1.5.2. Objetivos

El objetivo principal de este trabajo es el desarrollo de una arquitectura cognitiva híbrida que permita generar, controlar, planificar y monitorizar comportamientos en robots autónomos. Para ello, se tienen los siguientes objetivos secundarios:

- O<sub>1</sub> Conocer las herramientas para generar comportamientos en robots autónomos que se utilizan actualmente en las arquitecturas cognitivas.
- O<sub>2</sub> Diseñar y desarrollar una nueva arquitectura cognitiva que permita generar comportamientos en robots autónomos.
- O<sub>3</sub> Evaluar la nueva arquitectura cognitiva desarrollada anteriormente con el objetivo de validar su utilidad y aplicabilidad en robots autónomos.



## **1.6. Estructura del trabajo de investigación**

El resto de este trabajo de investigación se organiza de la siguiente manera. En el Capítulo 2 se describe una revisión sistemática de la literatura en la que se identifican y analizan las arquitecturas cognitivas ya presentadas y aplicables a robots autónomos. Por otro lado, el Capítulo 3 se presenta formalmente la arquitectura cognitiva de este trabajo. Así, en el Capítulo 4 se presenta la primera versión de la arquitectura, mientras que en el Capítulo 5 se presenta la nueva versión de esta arquitectura, presentando y describiendo sus diferentes componentes. Posteriormente, en el Capítulo 6 se muestra la evaluación que se ha llevado a cabo para validar la arquitectura cognitiva. Por último, en el Capítulo 7 se presentan las conclusiones de este trabajo de investigación.



# Capítulo 2

## Revisión de la literatura

El objetivo de este capítulo es poner en contexto las arquitecturas cognitivas y realizar una revisión sistemática de la literatura que permita conocer el estado de estas arquitecturas usadas en robots autónomos. Para ello se siguen las recomendaciones expuestas en el trabajo de Kitchenham, Budgen y Brereton [9].

### 2.1. Introducción

Los robots autónomos representan un avance tecnológico de gran relevancia en la actualidad, capaces de operar de manera independiente en entornos diversos y complejos. Estos robots incorporan una variedad de sensores y actuadores que les permiten percibir su entorno, tomar decisiones y ejecutar tareas específicas sin intervención humana directa. Se sustentan en una variedad de sensores, como cámaras, LiDAR, micrófonos y sensores de proximidad, que les permiten capturar información del entorno. Esta información es procesada por las arquitecturas cognitivas para tomar decisiones y controlar los actuadores, como ruedas, brazos robóticos y sistemas de locomoción especializados.

Para alcanzar este nivel de autonomía, es esencial el desarrollo de arquitecturas cognitivas que permitan generar comportamientos en robots de forma autónoma. Para ello, es necesario que modelen la estructura y función de los procesos mentales humanos, permitiendo a los robots no solo reaccionar ante estímulos sensoriales, sino también planificar, aprender y adaptarse a medida que interactúan con su entorno. De esta forma, las arquitecturas cognitivas representan

una tecnología crucial en el ámbito de la robótica, siendo utilizadas para generar comportamientos en robots autónomos. Con todo esto, se puede concebir a las arquitecturas cognitivas como modelos computacionales compuestos por diversos elementos que buscan emular el razonamiento y el comportamiento humano.

Estas arquitecturas están enmarcadas en el campo de la Robótica Cognitiva. De acuerdo con Levesque y Lakemeyer [10], en su forma más general, la robótica cognitiva se refiere al estudio de cómo los robots autónomos enfrentan los desafíos de representar el conocimiento y razonar en un entorno dinámico y cambiante. Para lograrlo, es esencial comprender la interrelación entre la percepción, el conocimiento y las acciones del robot.

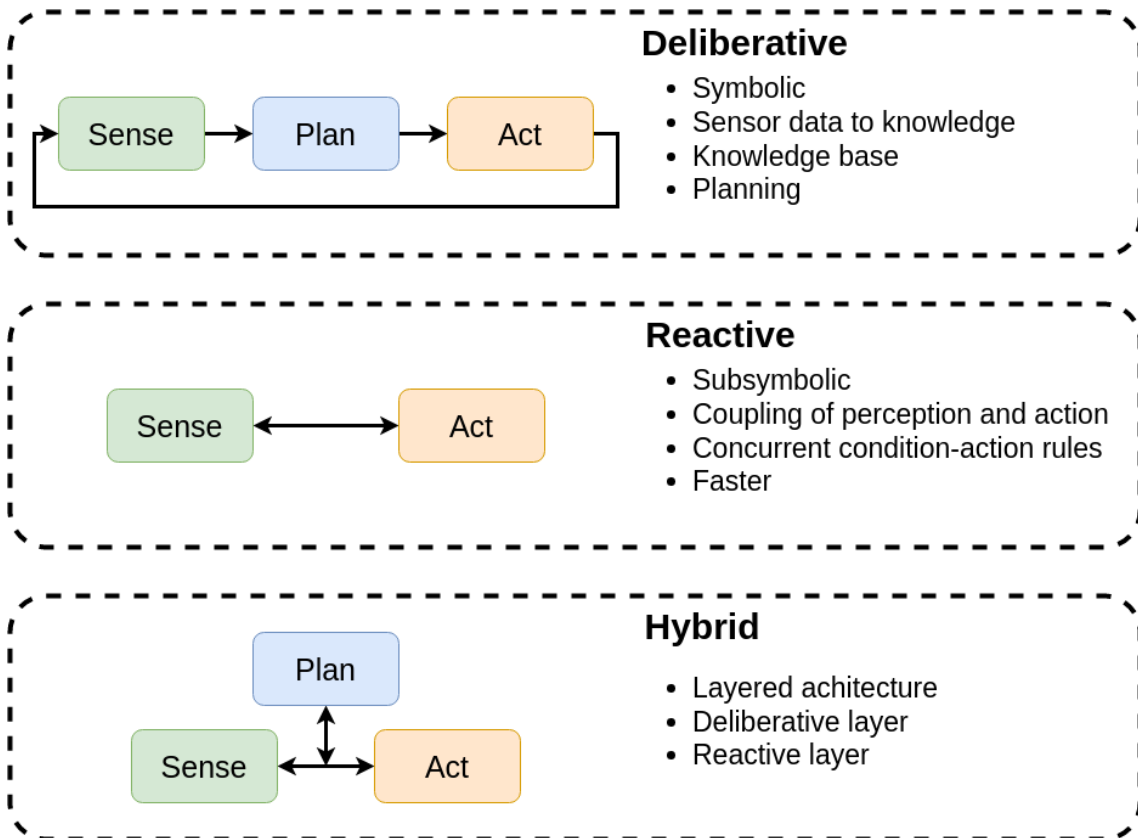
Este apartado introductorio da paso a una revisión sistemática de la literatura que tiene como objetivo explorar y analizar las arquitecturas cognitivas empleadas en robots autónomos, identificar tendencias y avances en el campo de la robótica cognitiva, y proporcionar una comprensión más profunda de cómo estas arquitecturas están contribuyendo al desarrollo de robots más inteligentes y versátiles.

## **2.2. Contexto**

Para poder analizar correctamente las arquitecturas cognitivas, en primer lugar es necesario identificar los tipos de arquitecturas cognitivas que se pueden encontrar. Para ello, a continuación se presenta una clasificación de arquitecturas cognitivas. Para llevar a cabo la clasificación de arquitecturas cognitivas, se ha decidido hacer una división entre la clasificación de las arquitecturas de control y la clasificación de arquitecturas cognitivas que se puede encontrar actualmente en la literatura.

### **2.2.1. Clasificación de arquitecturas de control**

Siguiendo un enfoque clásico, las arquitecturas cognitivas se pueden considerar arquitecturas de control. Siguiendo el estudio de Nakhaeinia et al. [11], las arquitecturas de control se dividen en tres tipos que se presentan en la Figura 2.1.



**Figura 2.1** – Clasificación de arquitecturas de control. Se tienen tres tipos de arquitecturas de control, que son los siguientes: arquitecturas deliberativas, que utilizan conocimiento simbólico y planificadores; arquitecturas reactivas, que utilizan conocimiento subsimbólico y tienen un alto grado de acoplamiento entre los sensores y los actuadores; y las híbridas, que mezclan los conceptos de las dos anteriores.

### Arquitecturas deliberativas

El primer tipo de arquitectura de control es la arquitectura deliberativa. Este tipo de arquitectura se basa en el uso de conocimiento simbólico para definir el mundo del robot. El conocimiento simbólico se almacena en una base de conocimiento que se puede usar para realizar procesos de razonamiento y planificación. Una de las formas más extendida en robótica para llevar a cabo este tipo de arquitecturas es mediante PDDL (*Planning Domain Definition Language*) de Fox y Long [12] que permite definir el mundo del robot y es compatible con múltiples planificadores.

Como presentan Ingrand y Ghallab [4], este tipo de arquitecturas están compuestas de tres fases principales: la fase de percepción, en la que el robot obtiene conocimiento simbólico a partir del procesamiento de los datos sensoriales; la fase de planificación, en la que el robot genera planes o secuencias de acciones para resolver unos objetivos concretos; y la fase de actuación, en la que el robot ejecuta las acciones que forman el plan generado anteriormente.

Para llevar a cabo los procesos de planificación y razonamiento, se pueden emplear diversas herramientas especializadas. Por ejemplo, para la planificación, es posible utilizar planificadores basados en PDDL, como POPF (*Partial Order Planning for PDDL*) [13]. En lo que respecta al razonamiento, se puede recurrir a la utilización de ontologías. Las ontologías son estructuras de conocimiento que representan de manera formal conceptos, relaciones y propiedades dentro de un dominio específico, proporcionando un marco sólido para la inferencia y el razonamiento en la toma de decisiones del robot. Estas ontologías permiten al robot comprender y procesar el conocimiento de manera más profunda y contextualizada, lo que es esencial para la toma de decisiones informadas y la interacción eficiente con su entorno.

Sin embargo, estas arquitecturas son más lentas que las arquitecturas reactivas, ya que necesitan un paso de planificación para generar la secuencia de acciones que resolverá los objetivos del robot. Además, es necesario un paso previo entre la percepción del mundo y la adquisición de conocimiento. Esto se conoce como el problema del *Symbolic Anchoring* presentado en [14] que trata los procesos necesarios para adquirir conocimiento simbólico a partir de los datos de los sensores y para mantener dicho conocimiento enlazado al mundo real.

### **Arquitecturas reactivas**

El segundo tipo de arquitecturas de control es la arquitectura reactiva. Este tipo de arquitecturas se caracterizan por su capacidad para tomar decisiones rápidas y específicas basadas en la información sensorial en tiempo real, lo que las hace adecuadas para tareas que requieren respuestas inmediatas y precisas.

Al tener un mayor acoplamiento entre los datos de los sensores y los actuadores,

se consigue tener una mayor velocidad a la hora de generar comportamientos. De esta forma, en vez de usar conocimiento simbólico como las arquitecturas deliberativas, las arquitecturas reactivas utilizan conocimiento subsimbólico, es decir, representaciones sin un significado directo para los humanos, como por ejemplo valores numéricos obtenidos de los sensores.

Las arquitecturas reactivas se basan en un conjunto de módulos de control, cada uno de los cuales se especializa en una tarea particular. Cada módulo toma datos de los sensores del robot y genera comandos para los actuadores en función de reglas o algoritmos predefinidos. Esto permite una ejecución eficiente de tareas específicas y una respuesta rápida a cambios en el entorno. A menudo, estas arquitecturas están diseñadas para operar en bucles de control cerrado, donde la información sensorial se actualiza constantemente y se toman decisiones en consecuencia.

Un ejemplo común de arquitectura reactiva es la *Subsumption Architecture* desarrollada por Brooks [15], que se basa en una jerarquía de comportamientos simples y reactivos que compiten por el control del robot en función de la relevancia y la urgencia de las situaciones. Este enfoque ha demostrado ser eficaz en aplicaciones de robótica móvil, como la navegación autónoma y la evasión de obstáculos.

Si bien las arquitecturas reactivas son ideales para tareas que requieren respuestas rápidas y específicas, a menudo se enfrentan a limitaciones en la capacidad de realizar tareas complejas que involucran planificación a largo plazo o razonamiento abstracto. Además, el uso de conocimiento subsimbólico hace que estas arquitecturas sean menos transparentes que las deliberativas.

### **Arquitecturas híbridas**

El tercer tipo de arquitecturas de control son las arquitecturas híbridas. Estas arquitecturas representan un enfoque que busca combinar lo mejor de dos mundos: la reactividad y la deliberación. Buscan superar las limitaciones inherentes de las arquitecturas puramente reactivas o puramente deliberativas, permitiendo a los robots autónomos operar de manera más eficiente y adaptativa en entornos complejos y cambiantes. Un ejemplo sería AuRA de Arkin y Balch [5].

Las arquitecturas reactivas, con su enfoque en la toma de decisiones inmediatas basadas en datos sensoriales, son ideales para tareas que requieren respuestas rápidas y específicas. Sin embargo, carecen de la capacidad de planificación a largo plazo y de la adaptación a situaciones no previstas. Por otro lado, las arquitecturas deliberativas son hábiles en el razonamiento abstracto y la planificación a largo plazo, pero pueden resultar ineficientes en situaciones donde se necesitan respuestas inmediatas.

Las arquitecturas híbridas buscan abordar estas limitaciones al combinar componentes reactivos y deliberativos en una única estructura. Los módulos reactivos permiten respuestas rápidas a estímulos sensoriales, mientras que los módulos deliberativos se encargan de la planificación a largo plazo y el razonamiento abstracto. Esta combinación ofrece la capacidad de adaptarse a situaciones cambiantes mientras se mantiene la capacidad de planificar y razonar. A la hora de organizar estos módulos se suele seguir un enfoque basado en capas como el que presenta Gat, Bonnasso, Murphy et al. [8]. De esta forma, se tendría una capa deliberativa y una capa reactiva.

Las arquitecturas híbridas han demostrado ser particularmente efectivas en aplicaciones de robótica autónoma, como la navegación en entornos complejos, la manipulación de objetos y la interacción con humanos. Al equilibrar la reactividad con la deliberación, estas arquitecturas permiten a los robots adaptarse a un amplio rango de desafíos en tiempo real, lo que las convierte en una elección prometedora en el desarrollo de robots más inteligentes y versátiles.

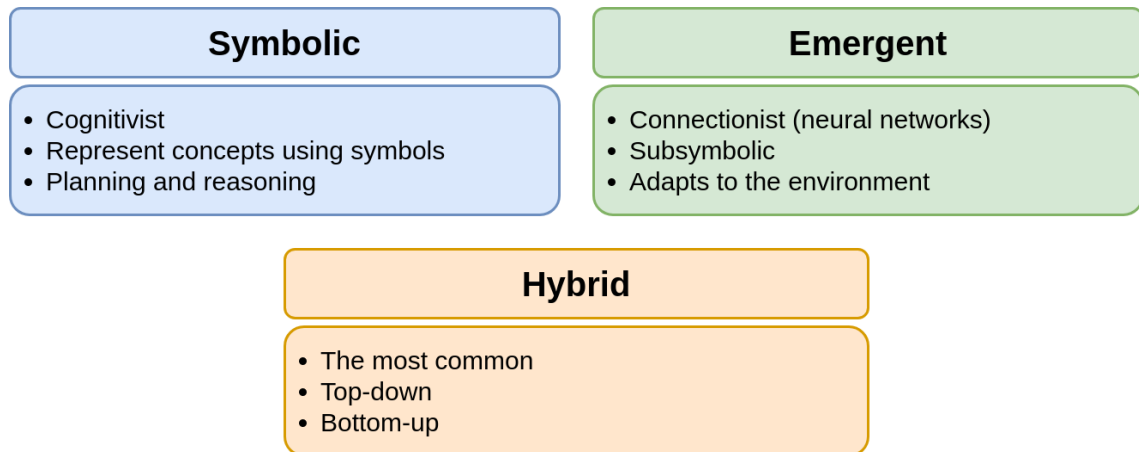
### 2.2.2. Clasificación de arquitecturas cognitivas

En revisiones de la literatura recientes ([6] y [7]), se suelen clasificar las arquitecturas cognitivas en tres categorías similares a las de las arquitecturas de control. Esta clasificación se presenta en la Figura 2.2.

#### **Arquitecturas simbólicas**

El primer tipo de arquitecturas cognitivas son las arquitecturas simbólicas. Estas arquitecturas son similares a las arquitecturas deliberativas presentadas anteriormente. Ambas se centran en el procesamiento de información basado





**Figura 2.2** – Clasificación de arquitecturas cognitivas. Se tiene tres tipos de arquitecturas cognitivas, que son los siguientes: arquitecturas simbólicas, que utilizan conocimiento simbólico, planificación y razonamiento; arquitecturas emergentes, que utilizan conocimiento subsimbólico y redes neuronales; y arquitecturas híbridas, que mezclan los conceptos de las dos anteriores.

en conocimiento simbólico. Las arquitecturas simbólicas están estrechamente relacionadas con el cognoscitivismo, un enfoque en la psicología y la inteligencia artificial que se basa en la idea de que la mente humana procesa la información a través de representaciones simbólicas y reglas lógicas. El cognoscitivismo sostiene que el pensamiento humano se puede entender mediante la manipulación de símbolos, y estas ideas se han aplicado en la creación de arquitecturas cognitivas simbólicas en inteligencia artificial, incluyendo aquellas utilizadas en la robótica.

Las arquitecturas simbólicas se basan en la idea de que la cognición y el razonamiento humano se pueden modelar a través de representaciones simbólicas y reglas lógicas. Estas arquitecturas utilizan lenguajes formales para representar información y reglas que gobiernan la toma de decisiones. A menudo, operan con una estructura jerárquica, donde se pueden establecer objetivos de alto nivel que guían la planificación y la ejecución de tareas.

En estas arquitecturas, los robots pueden razonar y tomar decisiones basadas en información simbólica y reglas lógicas. Esto las hace adecuadas para tareas que requieren un alto nivel de abstracción y planificación a largo plazo, como la planificación de rutas en entornos complejos, el procesamiento de lenguaje

natural, la inferencia lógica y la toma de decisiones en situaciones bien definidas.

Ejemplos de arquitecturas simbólicas incluyen el sistema Soar de Laird, Newell y Rosenbloom [16], que se utiliza en aplicaciones de alto nivel como la toma de decisiones en robótica autónoma y sistemas expertos, y los sistemas basados en lógica formal que se emplean para la planificación y la resolución de problemas en robótica.

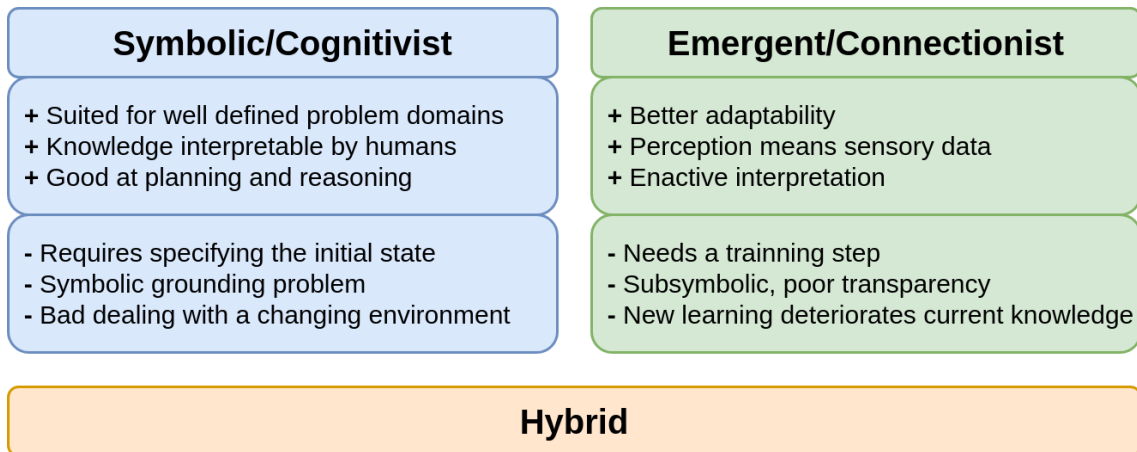
### **Arquitecturas emergentes**

El segundo tipo de arquitecturas cognitivas son las arquitecturas emergentes. Estas no son exactamente las mismas que las arquitecturas reactivas, aunque comparten algunas similitudes en su enfoque en la toma de decisiones basada en la percepción y la adaptación a entornos cambiantes. Las arquitecturas emergentes se basan en la idea de que los comportamientos y las capacidades complejas pueden emerger a partir de la interacción de múltiples componentes simples, sin necesidad de representaciones simbólicas o reglas lógicas.

A diferencia de las arquitecturas reactivas, que a menudo se basan en un conjunto predefinido de reglas y comportamientos específicos para situaciones particulares, las arquitecturas emergentes utilizan redes neuronales y otras técnicas de aprendizaje automático para generar los comportamientos en robots. Además, suelen tener una interpretación enactiva, es decir, estas redes neuronales, a menudo, aprenden y se adaptan a partir de la experiencia y de las retroalimentaciones recibidas del entorno, lo que les permite ajustar su comportamiento a medida que se enfrentan a nuevas situaciones o datos. Sin embargo, esto puede deteriorar el entrenamiento previamente realizado.

### **Arquitecturas híbridas**

El tercer tipo de arquitecturas cognitivas son las arquitecturas híbridas. Al igual que en el caso de las arquitecturas híbridas de la clasificación anterior, estas arquitecturas tratan de combinar lo mejor de dos mundos: lo simbólico y lo emergente. Buscan superar los problemas de las arquitecturas puras, que se presentan en la Figura 2.3, permitiendo a los robots autónomos acceder a las capacidades de planificación y razonamiento a la vez que a diferentes sistemas emergentes.



**Figura 2.3** – Ventajas e inconvenientes de los tipos de arquitecturas cognitivas. Las arquitecturas simbólicas se adaptan bien a los problemas con dominios bien definidos, su conocimiento es interpretable por humanos y son buenas planificando y razonando; pero necesitan generar conocimiento simbólico a partir de los sensores y un estado inicial, y son malas en entornos cambiantes. Las arquitecturas emergentes se adaptan mejor a los entornos pero como necesitan un paso de entrenamiento y reentrenar implica empeorar los modelos.

Las arquitecturas simbólicas tienen una gran capacidad para llevar a cabo tareas de planificación y razonamiento y tienen un alto grado de explicabilidad, pero necesitan de un estado inicial y tienen problemas con los entornos cambiantes. Por otro lado, las arquitecturas emergentes se adaptan mejor a los entornos y no necesitan un paso intermedio después de la obtención de datos de los sensores, sin embargo, necesitan una fase de entrenamiento, no son tan transparentes debido al uso de conocimiento subsimbólico y llevar a cabo nuevos aprendizajes puede deteriorar el conocimiento actual.

Como resultado, las arquitecturas híbridas tratan de combinar ambos componentes en una única estructura. De esta forma, las arquitecturas resultantes pueden tener un núcleo formado por un planificador simbólico que se encarga de la planificación a largo plazo y varios componentes emergentes que permiten realizar acciones con una mayor capacidad de adaptarse al entorno del robot. En la literatura es posible encontrarse con diferentes tipos de hibridación.

### 2.2.3. Preguntas de investigación

La estrategia PICOC [17] ofrece una estructura fundamental para formular preguntas de investigación. Esta metodología evoluciona desde la tradicional estrategia PICO, tal como fue presentada por Santos, Pimenta y Nobre [18] en 2007. De esta forma, la estrategia PICOC se enfoca en identificar los siguientes elementos clave en un trabajo de investigación:

- **Población:** ¿Quiénes son los sujetos de estudio o cuál es el problema que se investiga?
- **Intervención:** ¿Qué intervención, tratamiento o exposición se está examinando?
- **Comparación:** ¿Existe algún grupo de comparación o alternativa a la intervención que se está evaluando?
- **Resultados:** ¿Cuáles son los resultados esperados o las medidas que se utilizarán para evaluar la intervención?
- **Contexto:** ¿Cuál es el contexto en el que se lleva a cabo la investigación?

En este trabajo de investigación, la población se identifica en las arquitecturas cognitivas y el contexto es la robótica. La intervención se corresponde con la herramienta y componentes empleados para generar comportamientos en robots. Los resultados buscados son las arquitecturas cognitivas actuales y las herramientas que se utilizan para generar comportamientos en robots autónomos. No se buscan comparaciones concretas.

Basándonos en los conceptos identificados anteriormente, se construyen las siguientes preguntas de investigación:

RQ1 ¿Qué arquitecturas cognitivas contempla el estado de la cuestión para robots autónomos?

La primera de las preguntas de investigación se centra en conocer el estado de la cuestión en materia de generación de comportamientos autónomos en robots a través de arquitecturas cognitivas. El objetivo de esta pregunta es conocer qué arquitecturas existen actualmente y qué características tienen.

RQ2 ¿Qué herramientas se utilizan en las arquitecturas cognitivas para generar comportamientos en robots autónomos?

La segunda de las preguntas de investigación se centra en identificar la herramienta empleada por las arquitecturas cognitivas para generar comportamientos en robots. El objetivo de esta pregunta consisten en conocer las herramientas actuales aprovechadas por las arquitecturas cognitivas con el fin de valorar su inclusión en la arquitectura cognitiva que se va a desarrollar en este trabajo de investigación.

Se ha decidido incluir preguntas secundarias relacionadas con la evaluación de las arquitecturas cognitivas. Puesto que no es el objetivo principal de este trabajo, estas preguntas no se han incluido con las anteriores. Sin embargo, dada la naturaleza altamente diversa y compleja de las arquitecturas cognitivas en robótica, así como por las variaciones en las plataformas robóticas utilizadas, la inclusión de estas preguntas permite abordar estas diferencias y desafíos particulares de manera sistemática. De esta forma, se tienen las siguientes preguntas:

RQ3 ¿Qué métodos se emplean para evaluar las arquitecturas cognitivas?

La tercera de las preguntas de investigación se centra en conocer los métodos y experimentos realizados para evaluar las arquitecturas cognitivas. Puesto que las arquitecturas se utilizan en diferentes plataformas robóticas y con objetivos diferentes, su evaluación será diferente. El objetivo de esta pregunta es conocer qué métodos de evaluación existen en las arquitecturas cognitivas actuales.

RQ4 ¿Qué métricas se utilizan para evaluar las arquitecturas cognitivas?

La cuarta de las preguntas de investigación se centra en identificar las métricas empleadas en la evaluación de las arquitecturas cognitivas. El objetivo de esta pregunta consisten en conocer las métricas empleadas en las arquitecturas actuales durante las fases de experimentación y evaluación.

## 2.3. Metodología

Para la realización de esta revisión sistemática de la literatura se ha aplicado la metodología propuesta por Kitchenham, Budgen y Brereton [9] que aborda la realización de este tipo de trabajos en el ámbito de la ingeniería del software. El objetivo de esta revisión es obtener una perspectiva general del estado actual de las arquitecturas cognitivas utilizadas para generar comportamientos en robots autónomos. De esta forma, la revisión se centra en identificar y analizar la literatura en revistas que aborde el uso de arquitecturas cognitivas en robots autónomos.

### 2.3.1. Proceso de búsqueda

La búsqueda de información se lleva a cabo de manera automatizada utilizando diversas bases de datos digitales de renombre, que son la ACM Digital Library, IEEE Xplore Digital Library, Web of Science, ScienceDirect, Scopus y Springer Link. El objetivo principal de esta búsqueda es identificar y recopilar artículos publicados hasta agosto de 2023, que estuvieran relacionados con el ámbito de interés, es decir, con el uso de arquitecturas cognitivas para generar comportamientos en robots autónomos. Se pone un énfasis especial en examinar los campos de título, resumen y palabras clave de los artículos, y se excluyen deliberadamente aquellos escritos en idiomas distintos al inglés, con el fin de mantener la coherencia y la uniformidad en el análisis de la información.

Además, es importante destacar que se lleva a cabo un proceso adicional para construir una cadena de búsqueda precisa y específica. Esto se realiza mediante la extracción de palabras clave relevantes, las cuales se obtienen de las preguntas de investigación planteadas en el estudio. Para este propósito, se utilizó una estructura previamente definida conocida como PICOC, que engloba los elementos esenciales de la investigación. Estos campos del PICOC son consistentes con los mencionados previamente. Sin embargo, en el campo de intervención se incorporó la consideración de la utilización de middlewares, además de las herramientas para la generación de comportamientos en robots. Entre los middlewares más destacados en el campo de la robótica se encuentran ROS (*Robot Operating System*) [19], YARP (*Yet Another Robot Platform*) [20], NAOQi [21] y RoboComp [21].

Esto es de particular importancia, ya que las arquitecturas cognitivas en robótica requieren de un middleware para establecer la comunicación con la plataforma robótica en la que se implementarán.

En consecuencia, se ha definido un PICO que abarca los siguientes elementos clave:

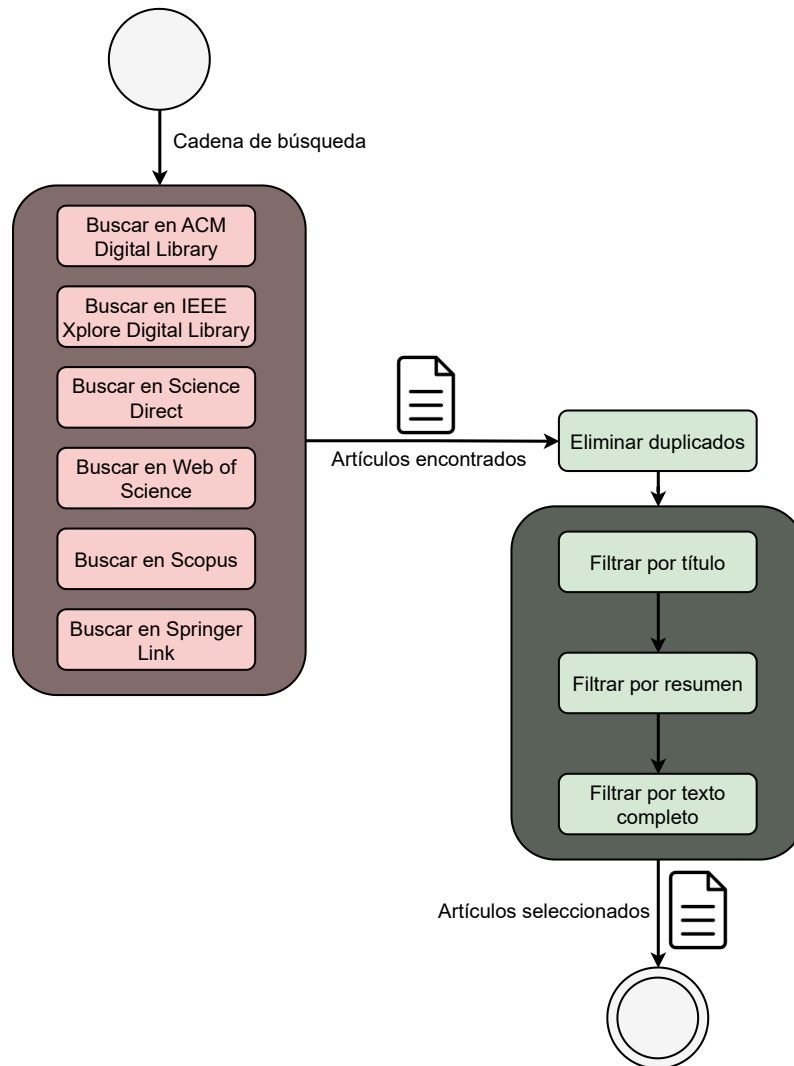
- **Population:** Arquitecturas Cognitivas
- **Intervention:** Herramientas para generar comportamientos y Middleware (ROS, YARP, NAOQi, RoboComp)
- **Comparison:** No aplica en este contexto
- **Outcome:** Identificación de las arquitecturas cognitivas actuales y de las herramientas empleadas para generar comportamientos en robots autónomos
- **Context:** Robótica

Utilizando estos componentes del PICO como punto de partida, se generan las palabras clave esenciales que constituyen la cadena de búsqueda resultante. Esta cadena se presenta a continuación:

QS1 (*“Cognitive Architecture”*) AND (*“Robotics”*) AND ((*“ROS”* OR *“Robot Operating System”*) OR (*“YARP”* OR *“Yet Another Robot Platform”*) OR *“Robocomp”* OR *“NAOqi”*)

### 2.3.2. Proceso de selección

El proceso completo de búsqueda y selección se representa en la Figura 2.4. La primera etapa consiste en la búsqueda de artículos usando la cadena previamente definida. La segunda etapa del proceso implica la eliminación de duplicados en los resultados de la búsqueda. Luego, se diseñó el proceso de selección en torno a tres fases: filtrado por título, filtrado por resumen y filtrado por texto completo. En cada una de estas etapas, se revisó la sección correspondiente del artículo o el texto completo y se aplicaron los criterios de inclusión y exclusión posteriormente definidos. Esto permitió llevar a cabo un filtrado gradual de los artículos que no eran relevantes para los temas de investigación planteados.



**Figura 2.4** – Diagrama de los procesos de búsqueda y selección. El proceso de búsqueda, marcado en rojo, utiliza la cadena de búsqueda para realizar la búsqueda de artículos en ACM Digital Library, IEEE Xplore Digital Library, Web of Science, ScienceDirect, Scopus y Springer Link. El proceso de selección, marcado en verde, se ha dividido en eliminación de duplicados y varios procesos de filtrado, por título, por resumen y por el texto completo.

Para llevar a cabo el proceso de selección y filtrado de artículos, se han definido un conjunto de criterios de inclusión y un conjunto de criterios de exclusión que se detallan en las Tablas 2.1 y 2.2 respectivamente. Para que un artículo pase el proceso, debe satisfacer todos los criterios de inclusión y no cumplir ninguno de los criterios de exclusión.



**Tabla 2.1** – Criterios de inclusión para el proceso de selección.

#	Descripción
IC1	Artículo de investigación de revistas
IC2	Totalidad del artículo disponible en inglés
IC3	La totalidad o parte del contenido del artículo debe responder directamente a alguna de las preguntas de investigación
IC4	Fecha de publicación posterior a 2017 no incluido

**Tabla 2.2** – Criterios de exclusión para el proceso de selección.

#	Descripción
EC1	Artículos no relacionados con la robótica o las arquitecturas cognitivas
EC2	Artículos que no presenten una arquitectura cognitiva
EC3	Artículos pertenecientes al autor de este trabajo de investigación Miguel Ángel González Santamarta

### 2.3.3. Proceso de evaluación de la calidad

Una vez que se ha completado la etapa de planificación en el proceso de selección, se lleva a cabo una evaluación de calidad de los artículos. En esta fase, se procede a asignar una puntuación a cada uno de los artículos previamente seleccionados, estableciendo un umbral mínimo que estos deben superar para ser considerados en la revisión.

La evaluación de calidad se realiza a través de la respuesta a una serie de preguntas relacionadas con cada uno de los artículos previamente seleccionados en la etapa anterior. Para este propósito, se ha adoptado el conjunto de preguntas propuesto por Dybå y Dingsøyr [22] y citado por Kitchenham, Budgen y Brereton [9] en la guía de referencia utilizada para la elaboración de esta revisión. Se han modificado las preguntas, eliminando aquellas que no eran pertinentes para esta revisión y añadiendo otras nuevas. De esta manera, se ha conformado el conjunto

final de preguntas, que se detalla en la Tabla 2.3. Estas preguntas cubren las cuatro áreas principales de un trabajo de investigación:

- **Comunicación:** Estas preguntas se centran en determinar la calidad de la presentación y comunicación del artículo.
- **Rigor:** Aquí se incluyen preguntas relacionadas con el diseño de la investigación llevada a cabo en el artículo.
- **Credibilidad:** Estas preguntas están destinadas a evaluar si los resultados de la investigación son válidos y significativos.
- **Relevancia:** Estas preguntas se enfocan en determinar la relevancia de la información presentada en el artículo.

**Tabla 2.3** – Preguntas de la evaluación de la calidad de los artículos. Se han clasificado en cuatro áreas, que son comunicación, rigor, credibilidad y relevancia.

Área	Pregunta
Comunicación	¿Existe una definición clara de los objetivos de la investigación?
Comunicación	¿Existe una definición clara de las conclusiones?
Comunicación	¿Contiene una descripción adecuada del contexto de la investigación?
Comunicación	¿Existe un diagrama claro de la arquitectura cognitiva?
Comunicación	¿Se describen los componentes de la arquitectura cognitiva?
Rigor	¿Es apropiado el diseño de la investigación?
Rigor	¿Se ha realizado una fase de validación o experimentación durante la investigación?
Credibilidad	¿Se ha probado la arquitectura cognitiva en robots reales?
Credibilidad	¿Existe una referencia clara a los resultados obtenidos?
Relevancia	¿Tiene valor el estudio para investigación o práctica?

Cada una de las preguntas previamente mencionadas ofrece cuatro posibles opciones de respuesta: “Sí”, “Parcialmente”, “No” y “No aplica”. A estas tres respuestas se les asignan puntuaciones respectivas de 1, 0.5 y 0 respectivamente.

Se ha establecido un valor mínimo de puntuación, denominado umbral, que los artículos deben superar para no ser excluidos del proceso de revisión. En este caso, el umbral elegido es de 7.5. La elección de este umbral en lugar de otros, como por ejemplo 5.0 que sería la mitad del máximo que se puede alcanzar, viene de la necesidad de garantizar que solo los artículos de alta calidad sean considerados en el proceso de revisión. Un umbral más alto indica que se requiere una calidad sustancial en múltiples aspectos evaluados en las preguntas. Esto ayuda a evitar que se acepten artículos que responden correctamente a preguntas genéricas, pero fallan en preguntas más específicas.

#### **2.3.4. Procesos de extracción y síntesis de datos**

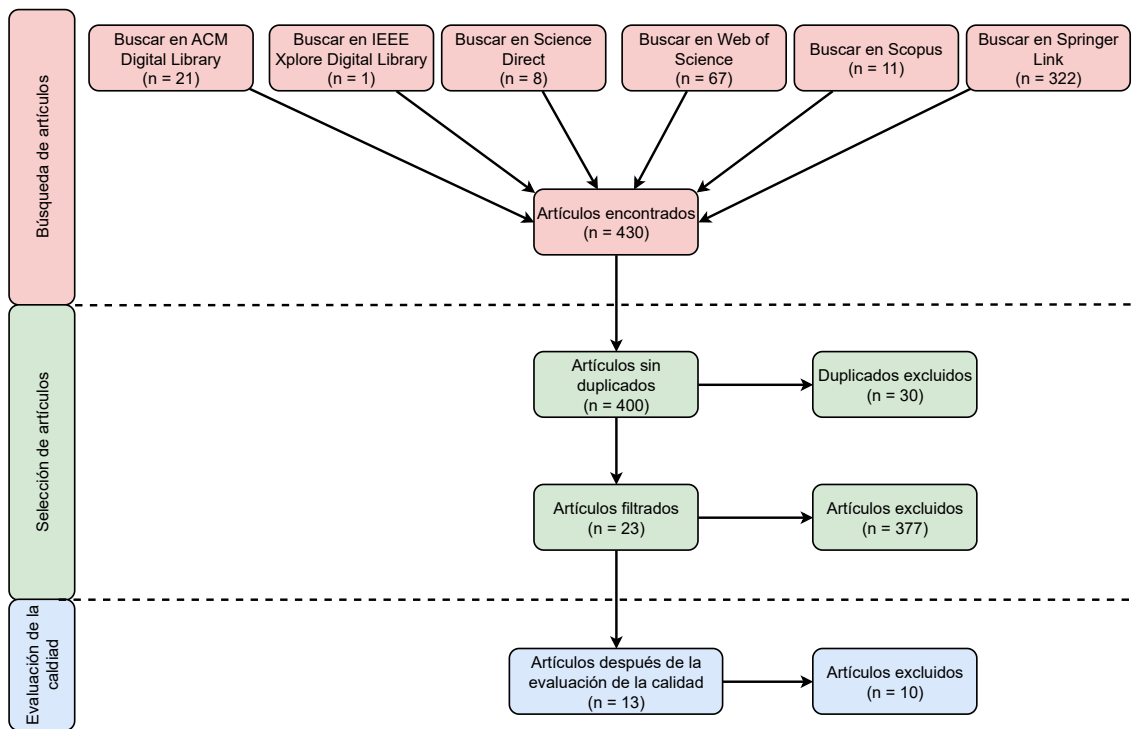
En la etapa de extracción de datos se plantea extraer datos directamente relacionados con la revisión y las preguntas de investigación, es decir, con las arquitecturas cognitivas, las herramientas para generar comportamientos y los métodos de evaluación. De esta forma, para extraer información se han definido las siguientes preguntas clave:

- ¿Qué middleware se utiliza en el artículo?
- ¿Qué tipo de arquitectura cognitiva se presenta en el artículo?
- ¿Es una arquitectura organizada en capas?
- ¿Qué herramientas de generación de comportamientos se utilizan?
- ¿Cómo se evalúa la arquitectura cognitiva?
- ¿Qué métricas se utilizan para evaluar la arquitectura cognitiva?

A partir de los datos extraídos, se llevará a cabo una síntesis de esta información en forma de tablas, donde se registrarán tanto la referencia como un identificador único para cada artículo. Además, se establecerán los datos que se recopilan para cada uno de los artículos en función de las preguntas de investigación a las que dan respuesta.

## 2.4. Resultados

Los resultados numéricos de esta revisión sistemática se presentan en el diagrama PRISMA (*Preferred Reporting Items for Systematic Reviews and Meta-Analysis*) de la Figura 2.5. A continuación se explican más detalladamente los resultados obtenidos en las fases de búsqueda de artículos, selección de artículos, evaluación de la calidad y síntesis de datos.



**Figura 2.5** – Diagrama PRISMA de los procesos de búsqueda de artículos, selección de artículos y evaluación de la calidad. Inicialmente, se han encontrado 430 artículos en la búsqueda (21 artículos de ACM Digital Library, 1 artículo de IEEE Xplore Digital Library, 8 artículos de Science Direct, 67 artículos de Web of Science, 11 artículos de Scopus y 322 artículos de Springer Link). Después de la selección, se han obtenido 23 artículos. Por último, después de la evaluación de la calidad, se han obtenido 13 artículos.

### 2.4.1. Procesos de búsqueda y selección

Los procesos de búsqueda y selección de artículos para el proceso de revisión se resumen en el diagrama PRISMA que se muestra en la Figura 2.5. En el proceso

de búsqueda de artículos se encontraron un total de 430 artículos divididos entre las diferentes fuentes de búsqueda como se detalla a continuación:

- ACM Digital Library: 21 artículos
- IEEE Xplore Digital Library: 1 artículo
- Science Direct: 8 artículos
- Web of Science: 67 artículos
- Scopus: 11 artículos
- Springer Link: 322 artículos

El proceso de selección comenzó con la eliminación de artículos duplicados, 30 en total, dejando el número de artículos candidatos en 400. Posteriormente, se filtraron los artículos aplicando los criterios de inclusión y exclusión, obteniendo un total de 23 artículos candidatos y excluyendo 377 artículos.

#### 2.4.2. Proceso de evaluación de la calidad

Después de realizar la búsqueda y selección de artículos, se llevó a cabo la evaluación de la calidad. La Tabla 2.4 resume los resultados de la evaluación de la calidad realizada a los 23 artículos seleccionados.

**Tabla 2.4** – Evaluación de la calidad de los artículos seleccionados presentando su referencia, su identificador y la puntuación obtenida en la evaluación.

Referencia	Identificador	Puntuación
[23]	S1	9,0
[24]	S2	8,0
[25]	S3	3,0
[26]	S4	8,0
[27]	S5	5,0
[28]	S6	9,0
[29]	S7	5,0
[30]	S8	6,5
[31]	S9	9,0

[32]	S10	8,0
[33]	S11	7,5
[34]	S12	8,0
[35]	S13	7,0
[36]	S14	8,0
[37]	S15	7,5
[38]	S16	5,0
[39]	S17	8,5
[40]	S18	9,5
[41]	S19	9,5
[42]	S20	7,5
[43]	S21	8,5
[44]	S22	7,0
[45]	S23	9,5

De los 23 artículos seleccionados previamente, 13 superan el proceso de evaluación de la calidad. Se quedan excluidos 10 artículos. En la Tabla 2.5 se muestran los 13 artículos finales a los que se les asigna un identificador para hacer referencia a ellos más adelante.

**Tabla 2.5** – Artículos finales que han pasado el proceso de evaluación. Cada artículo tiene un nuevo identificador y su referencia.

ID	Referencia
FS1	[23]
FS2	[24]
FS3	[26]
FS4	[28]
FS5	[31]
FS6	[32]
FS7	[34]
FS8	[36]

FS9	[39]
FS10	[40]
FS11	[41]
FS12	[43]
FS13	[45]

---

### 2.4.3. Extracción y síntesis de datos

En la etapa de extracción y síntesis de datos se han obtenido los datos relacionados con las preguntas presentadas anteriormente. De esta forma, las Tablas 2.6, 2.7, 2.8, 2.9 presentan la información extraída tras la lectura completa de los artículos que se han seleccionado después de la evaluación de la calidad.

**Tabla 2.6** – Datos extraídos de los artículos aceptados para las tres primeras preguntas de la extracción de datos. Para cada artículo se muestra su identificador, el middleware que utiliza, el tipo de arquitectura y si se utiliza una organización en capas.

ID	Middleware	Tipo de arquitectura	Organización en capas
FS1	ROS	híbrida	Sí
FS2	ROS	reactiva/emergent	Sí
FS3	ROS	híbrida	No
FS4	ROS	híbrida	Sí
FS5	RoboComp	híbrida	No
FS6	ROS	híbrida	No
FS7	ROS	reactiva/emergente	No
FS8	ROS	híbrida	No
FS9	ROS	híbrida	Sí
FS10	ROS 2	híbrida	Sí
FS11	ROS	híbrida	No
FS12	ROS	deliberativa/simbólica	No
FS13	ROS	reactiva/emergente	No

**Tabla 2.7** – Datos extraídos de los artículos aceptados para la pregunta sobre herramientas utilizadas para generar comportamientos. Para cada artículo se muestra su identificador y un párrafo resumiendo las herramientas empleadas para generar comportamientos.

ID	Herramientas para generar comportamientos
FS1	MoBAR es una arquitectura que utiliza planificación simbólica basada en PDDL [12] y máquinas de estados.
FS2	Esta arquitectura cognitiva colaborativa se basa principalmente en teleoperación y navegación en robots. También utiliza una máquina de estados para controlar cuándo se tiene que cambiar de teleoperación del usuario a navegación autónoma.
FS3	Esta arquitectura utiliza una máquina de estados para controlar su funcionamiento. Esta máquina de estados es la encargada de llamar al planificador, que está basado en PDDL, y de ejecutar las acciones. También contiene un sistema de detección y reconocimiento de objetos basado en el uso de nubes de puntos e histogramas. Por último, utiliza un sistema de memorias, una para almacenar datos de la detección de objetos y otra para almacenar los planes, los resultados de las acciones y la información del mundo del robot.
FS4	HiMoP se basa en el uso de planificación basada en PDDL junto a máquinas de estados para generar comportamientos. Además, presenta un sistema motivacional que guía los objetivos del robot.
FS5	CORTEX es una arquitectura cognitiva que se basa en el uso de un grafo que mantiene conocimiento simbólico e información geométrica del robot. La arquitectura dispone de un conjunto de agentes que utilizan este grafo. Algunos de estos agentes realizan las siguientes tareas: localización, navegación, diálogo, manipulación y detección de objetos. Además, es posible incluir otras herramientas como planificación basada en PDDL, máquinas de estados y árboles de comportamiento.



- FS6 Esta arquitectura emplea representaciones simbólicas de las acciones que el robot puede realizar. Además, se utiliza una red neuronal para controlar la activación de las acciones del robot. El robot tiene una memoria en la que se definen, de forma simbólica, las acciones del robot y sus dependencias. La red neuronal utiliza esta memoria.
- FS7 Esta arquitectura utiliza un sistema de atención para centrarse en ciertas personas, como por ejemplo la persona que esté hablando en ese momento.
- FS8 Esta arquitectura cognitiva se basa en el modelo de ACT-R de Anderson, Matessa y Lebiere [46]. También utiliza otros componentes como el reconocimiento del habla, la síntesis de voz y MoveIt [47] para controlar los manipuladores del robot.
- FS9 Esta arquitectura híbrida utiliza conocimiento simbólico basado en PDDL y organizado como un grafo de conocimiento para definir el estado del mundo del robot. El conocimiento simbólico se utiliza junto a ROSPlan [48], un sistema de planificación, para generar las acciones del robot que están implementadas como máquinas de estados. Estas máquinas de estados se utilizan en varias capas de la arquitectura.
- FS10 Esta arquitectura cognitiva utiliza un sistema deliberativo basado en PDDL y PlanSys [49], junto con árboles de comportamientos para controlar los comportamientos del robot.
- FS11 Esta arquitectura tiene un sistema de planificación y razonamiento y un sistema reactivo. La planificación y razonamiento se basan en el uso de ontologías. Por otro lado, el sistema reactivo contiene las acciones del robot y realiza procesamiento del lenguaje natural. Por último, ambos sistemas están comunicados mediante *Hierarchical Reinforcement Learning* (HRL) [50].
- FS12 Esta arquitectura deliberativa utiliza PDDL y ontologías. De esta forma, define el estado del mundo del robot de forma simbólica. Además, utiliza ROSPlan, un sistema de planificación basado en PDDL, para generar los comportamientos del robot.

FS13 Esta arquitectura se basa en utilizar diferentes componentes emergentes. Utiliza diferentes tipos de redes neuronales para llevar a cabo su funcionamiento. De esta forma, utiliza redes neuronales convolucionales (CNN) para reconocer las acciones del usuario humano a partir de las imágenes que obtiene de una cámara. También utiliza redes *Long Short-Term Memory* (LSTM) para predecir cuánto tardará el usuario en realizar una acción. Por último, contiene un sistema de memorias para almacenar datos predefinidos del usuario y las acciones, para comunicar los diferentes componentes, y para almacenar información episódica que se puede analizar posteriormente.

---

**Tabla 2.8** – Datos extraídos de los artículos aceptados para la pregunta sobre la evaluación de la arquitectura. Para cada artículo se muestra su identificador y un párrafo resumiendo cómo se ha realizado la evaluación.

---

ID	Evaluación de la arquitectura
FS1	Se han realizado tres experimentos con tres robots diferentes. El primero utiliza un brazo robótico para coger y colocar objetos. El segundo utiliza un rover simulado para adquirir imágenes. El tercero utiliza un robot de vigilancia también para adquirir imágenes, pero esta vez teniendo en cuenta la batería.
FS2	Para evaluar la arquitectura se ha diseñado un experimento en un entorno simulado en el que varios participantes tienen que teleoperar el robot desde un punto inicial hasta un punto final. En este experimento, se tienen dos grupos de participantes: los de altas capacidades cognitivas y los de bajas capacidades cognitivas.

- FS3 Se ha realizado tres tipos de experimentos para evaluar la arquitectura. El primero es una evaluación cuantitativa del sistema de reconocimiento de objetos. El segundo se basa en probar la característica de aprendizaje de objetos. El tercero es un análisis cualitativo del sistema completo en el que un usuario interactúa con el robot intentando enseñar las categorías de diferentes objetos.
- FS4 La experimentación se basa en un juego de preguntas. El robot tiene una serie de preguntas. Cuando el experimento comienza, el robot y una persona empiezan a jugar hasta que suena un timbre. Entonces, el robot tiene que cambiar la misión.
- FS5 Se han realizado tres experimentos en este artículo. El primer experimento se basa en generar comportamientos para un robot que realizará anuncios. El robot se aproximará a personas y las invitará a visitar un panel. El segundo experimento incluye evaluar el grado de autonomía de personas de edad avanzada. El tercer experimento se realiza en un apartamento en el que el robot tiene que atender las necesidades de una persona con autonomía limitada, como llevar objetos a dicha persona.
- FS6 Se han llevado a cabo tres experimentos. El primero se basa en probar la capacidad de esquivar obstáculos, el segundo se centra en la ejecución de tareas jerárquicas basadas en realizar entregas, y el tercero se basa en la ejecución de múltiples planes de forma concurrente.
- FS7 La evaluación consiste en utilizar un grupo de participantes que pueden interactuar con el robot.
- FS8 La evaluación de este artículo se basa en realizar pruebas en las que participa una persona. Empieza con el humano pidiendo al robot que coloque un utensilio en la mesa y termina cuando el robot completa la tarea.
- FS9 Para validar la arquitectura se utiliza un robot de servicio en una casa. El robot tiene que notificar a una de las personas de la casa que recibirá un paquete.

- FS10 Se han llevado a cabo dos experimentos. En el primero se evaluó la navegación social del robot. Para ello el robot se aproxima a una persona. Después el humano cambia su orientación y el robot tiene que volver a acercarse a esa persona. En el segundo experimento, el robot se aproxima a la persona para hablar con ella y la persona le pide que navegue a ciertos puntos o que se acerque.
- FS11 La validación se basa en utilizar a un grupo de participantes que interactúan con el robot mediante el diálogo. La tarea consiste en el ejercicio de encontrar una palabra con conjuntos de palabras fáciles y difíciles.
- FS12 La validación presentada en este artículo se ha dividido en dos partes. El primer experimento trata el momento en el que salta la alarma de que hay que asistir a una persona. En el segundo experimento se evalúa la priorización de tareas en un escenario en el que se solicitan múltiples misiones al robot, como cerrar una válvula de gas, traer un libro, abrir las cortinas y monitorizar la casa.
- FS13 La evaluación llevada a cabo en este artículo tiene dos fases. En la primera se evalúa el funcionamiento de los diferentes modelos por separado, es decir, la CNN y la LSTM. En la segunda fase, se realizan dos experimentos diferentes. En el primero, el robot junto a un usuario tienen que montar una caja de madera. Por otro lado, en el segundo experimento, el usuario y el robot tienen que montar una pequeña silla de madera. Este experimento se realizó con tres usuarios distintos.

**Tabla 2.9** – Datos extraídos de los artículos aceptados para la pregunta sobre las métricas utilizadas en la evaluación. Para cada artículo se muestra su identificador y un párrafo resumiendo las métricas utilizadas en la evaluación.

ID	Métricas
FS1	Las métricas de este artículo son el tiempo gastado en el sistema deliberativo, el tiempo necesario para ejecutar las misiones y la distancia recorrida.

- FS2 Para medir los experimentos se realizaron medidas cualitativas, a través de cuestionarios que respondían los participantes; y medidas cuantitativas, como el tiempo total para completar los experimentos y el tiempo entre colisiones.
- FS3 Las métricas empleadas son el número de categorías aprendidas, el número de preguntas e interacciones para aprender las categorías, la media de instancias almacenadas por categoría y la precisión global de la clasificación.
- FS4 No hay métricas.
- FS5 Se tienen métricas cualitativas basadas en cuestionarios completados por los participantes. Como métrica cuantitativa se tiene el número de ejecuciones exitosas.
- FS6 Las métricas empleadas son el tiempo de ejecución, la precisión al esquivar obstáculos, las entregas realizadas y los planes erróneos.
- FS7 Se han utilizado métricas cualitativas mediante el uso de cuestionarios completados por los participantes.
- FS8 Se han medido diferentes conceptos durante los experimentos de esta arquitectura. Estos son la robustez, cuántos intentos necesita el robot para realizar la tarea; el tiempo empleado en el razonamiento; y la explicabilidad de la arquitectura, si es posible explicar las decisiones del robot.
- FS9 Las métricas empleadas son el tiempo de ejecución de las misiones, la distancia recorrida durante la misión, las llamadas al planificador y el número de acciones ejecutadas.
- FS12 La métrica empleada es el tiempo que se tarda en ejecutar los planes.
- FS10 Las métricas empleadas son la distancia mínima al humano durante la navegación, la distancia recorrida, el tiempo de navegación, y las intrusiones en el espacio personal de la persona. Además, se utilizaron también el uso de CPU, el uso de red y la memoria utilizada.

- FS11 Se han utilizado métricas cualitativas basadas en cuestionarios que evalúan la usabilidad, la duración de las interacciones, el número de turnos llevados a cabo durante los ejercicios y el número de errores en el diálogo.
- FS13 En el caso de la evaluación de los modelos por separado, se utilizaron métricas como matrices de confusión, error cuadrático medio raíz (RMSE) y error absoluto medio (MAE). En el caso de los experimentos con usuarios, las métricas empleadas fueron la media de tareas completadas y la media del tiempo que el usuario esperaba.
- 

## 2.5. Discusión

En esta sección se presenta la discusión de la revisión. Para ello, se responde a las preguntas de investigación con los datos obtenidos durante la revisión.

### 2.5.1. ¿Qué arquitecturas cognitivas contempla el estado de la cuestión para robots autónomos?

Como se ha presentado en la clasificación de arquitecturas cognitivas previamente detallada, se identifican tres tipos principales: las arquitecturas simbólicas, que hacen uso de planificadores y sistemas deliberativos para la toma de decisiones; las arquitecturas emergentes, que se basan en gran medida en el empleo de redes neuronales; y, finalmente, las arquitecturas híbridas, que buscan mezclar elementos de ambos paradigmas, simbólicos y emergentes. Dentro de esta tipología, se observa que las arquitecturas cognitivas de tipo híbrido prevalecen como la elección más común, ya que de un análisis detallado de los 13 artículos revisados, se constata que 9 de ellos implementan y presentan una arquitectura híbrida en sus sistemas robóticos. Por otra parte, se aprecia que la organización en capas, si bien no es la característica predominante, se encuentra presente en una proporción significativa, siendo adoptada por 5 de los 13 artículos. Es interesante señalar que algunos de los que no especifican de manera explícita una estructura en capas, a menudo organizan sus componentes de una manera que guarda similitudes notables con esta disposición estratificada.

### 2.5.2. ¿Qué herramientas se utilizan en las arquitecturas cognitivas para generar comportamientos en robots autónomos?

En los análisis de las arquitecturas cognitivas que se han explorado en los artículos seleccionados, es evidente que existen ciertos elementos comunes que unifican su enfoque y metodología. Dentro del marco del uso de conocimiento simbólico, es relevante destacar que varios de los artículos, específicamente FS1, FS3, FS4, FS5, FS9, FS10 y FS12, optan por el uso de PDDL como lenguaje para la representación del conocimiento del robot. Esta elección resalta la importancia de PDDL como una herramienta fundamental para representar el conocimiento y las tareas que un robot debe llevar a cabo. Además, se constata que estas mismas arquitecturas se apoyan en sistemas deliberativos basados en PDDL, como ROSPlan, para la generación de planes, secuencias de acciones, que permiten al robot llevar a cabo sus misiones. El conocimiento del robot se suele almacenar en bases de conocimiento, es decir, un tipo especial de base de datos que gestiona y mantiene el conocimiento del robot. Estas bases de datos se puede basar en bases de datos existentes o se pueden implementar nuevas bases de conocimiento. En el caso de varias arquitecturas presentadas, como en el caso de ROSPlan, se opta por una base de datos almacenada en memoria. También se pueden adoptar por grafos de conocimiento para representar y almacenar el conocimiento del robot. Este es el caso de FS5 y FS9, que utilizan grafos para mantener el conocimiento del robot, aunque después utilicen PDDL para planificar las misiones. Por otro lado, para llevar a cabo procesos de razonamiento se utilizan ontologías, definiciones formales los de tipos, las propiedades, y las relaciones entre entidades. Este es el caso de los artículos FS11 y FS12.

Cuando se analiza el empleo de soluciones reactivas en las arquitecturas cognitivas, es posible encontrar herramientas como las máquinas de estados y los árboles de comportamiento. Las máquinas de estados, en particular, han sido una elección recurrente en varios componentes de las arquitecturas cognitivas examinadas en este estudio. Este enfoque se hace evidente al observar las arquitecturas presentadas en los artículos FS1, FS2, FS3, FS4, FS5 y FS9. En estos artículos, las

máquinas de estados se erigen como una pieza fundamental para el control de las acciones del robot. Es interesante destacar, en el caso de las arquitecturas híbridas, que las máquinas de estados se han utilizado para implementar las acciones que conforman los planes generados por sistemas deliberativos, aunque también tienen un papel fundamental en la interacción entre componentes cognitivos a diferentes niveles. En contraste, el uso de árboles de comportamiento, aunque menos común, se presenta en las arquitecturas de los artículos FS5 y FS10. Estos árboles de comportamiento cumplen un papel similar al de las máquinas de estados, permitiendo implementar las acciones de los sistemas deliberativos. Esta diversidad en las elecciones de herramientas reactivas sugiere que los diseñadores de arquitecturas cognitivas evalúan y seleccionan enfoques que mejor se adaptan a las necesidades específicas de sus aplicaciones robóticas.

Cuando exploramos las arquitecturas emergentes, como las presentadas en los artículos FS7 y FS13, se evidencia un enfoque en el empleo predominante de redes neuronales. Tomando como ejemplo la arquitectura de FS13, se puede apreciar el uso de diversos tipos de redes neuronales, entre las que se incluyen las redes neuronales convolucionales (CNN) y las redes *Long Short-Term Memory* (LSTM). Esta diversidad de redes neuronales subraya el papel esencial que desempeñan en la percepción, el procesamiento de información y la toma de decisiones de los robots en contextos dinámicos. Por otro lado, en las arquitecturas híbridas, se observa una interesante convergencia entre componentes emergentes y simbólicos. Un ejemplo notable se presenta en el artículo FS5, en el que se podría emplear una red neuronal convolucional (CNN) para la detección de objetos. Un caso especial es el descrito en el artículo FS6, que se basa en la creación de una representación simbólica del estado y las acciones del robot, que luego se utiliza como entrada para una red neuronal, lo que ilustra la capacidad de fusionar elementos simbólicos y redes emergentes. Otro caso especial se da en el artículo FS11 en el que se usa el *Hierarchical Reinforcement Learning* (HRL) para facilitar la comunicación entre el sistema deliberativo y el sistema reactivo en arquitecturas híbridas.

Como se ha detallado previamente, las arquitecturas híbridas se caracterizan por su versatilidad en la integración de múltiples enfoques, incluyendo el uso



de conocimiento simbólico en diversas representaciones, como el PDDL y grafos de conocimiento. Este conocimiento simbólico es un componente fundamental en sistemas deliberativos que tienen la capacidad de generar planes detallados para guiar las acciones del robot. Estas acciones pueden implementarse de varias formas, ya sea mediante máquinas de estados o árboles de comportamiento. No obstante, es relevante subrayar que tanto las máquinas de estados como los árboles de comportamiento pueden desempeñar un papel más amplio en la arquitectura híbrida porque no se limitan únicamente a las acciones del robot. También se pueden usar para controlar otros elementos de la arquitectura. Un caso particularmente interesante se presenta en el artículo FS4, donde se introduce un sistema motivacional que posee la capacidad de afectar tanto a los componentes deliberativos como a los reactivos del robot. En lo que respecta al uso de componentes emergentes, es frecuente encontrarlos en arquitecturas híbridas en forma de habilidades específicas del robot, como el reconocimiento de objetos.

Por último, entre los componentes comunes que emergen con recurrencia en las arquitecturas cognitivas que han sido objeto de evaluación, se destacan una serie de elementos esenciales que permiten a los robots móviles desenvolverse de manera eficaz en su entorno y desempeñar tareas autónomas. Estos elementos incluyen la localización, la navegación y la teleoperación, todos fundamentales para la movilidad y el desplazamiento de los robots en su entorno. La localización y la navegación son cruciales para que los robots puedan determinar su posición precisa y trazar rutas óptimas hacia destinos específicos, lo que es particularmente relevante en aplicaciones de logística, transporte autónomo y de servicios. También se tienen componentes para controlar los manipuladores de los robots, como es el caso de MoveIt. Además de estas capacidades, el reconocimiento del habla y la síntesis de voz desempeñan un papel esencial en la interacción del robot con los seres humanos. La habilidad de un robot para entender y generar voz es crucial para la comunicación y la colaboración efectiva con las personas, lo que resulta valioso en entornos donde la interacción humano-robot es fundamental, como la asistencia a personas mayores o la educación. Para terminar, el reconocimiento de objetos es otro componente común de las arquitecturas cognitivas. Este componen-

te permite al robot obtener información relevante sobre su entorno al identificar y clasificar objetos presentes en su campo visual. Esta capacidad es importante en diferentes aplicaciones, desde la manipulación de objetos y la navegación hasta la toma de decisiones.

### **2.5.3. ¿Qué métodos se emplean para evaluar las arquitecturas cognitivas?**

Después de extraer y analizar los datos de los 13 artículos seleccionados, se han encontrado diferentes tipos de evaluación de las arquitecturas cognitivas que pueden estar orientados a diferentes tipos de personas. En primer lugar, se tienen las evaluaciones basadas en interacción con humanos, que son las que más se repiten. De esta forma, se tienen evaluaciones como las presentadas en los artículos FS3, en la que el humano intenta enseñar al robot diferentes categorías de objetos; FS8, en la que el humano le pide al robot que coloque un utensilio en la mesa; y FS10, en la que el robot se aproxima a la persona para hablar con ella y la persona le pide que navegue a ciertos puntos. Las evaluaciones basadas en interacciones con humanos pueden ser muy variadas. En el caso de FS5 se tienen tres experimentos: en el primero realiza anuncios aproximándose a personas y las invitará a visitar un panel, en el segundo evalúa el grado de autonomía de personas de edad avanzada y en el tercero atiende a personas con autonomía limitada en un apartamento. La evaluación de FS9 también se realiza en una casa en la que el robot tiene que notificar a una de las personas que recibirá un paquete. Otro ejemplo del uso de un apartamento es el de FS4, en el que se lleva a cabo un juego de preguntas con el humano hasta que suena un timbre y el robot tiene que cambiar la misión. Las interacciones con el robot se pueden realizar de varias formas, como por ejemplo en el caso del artículo FS2 en el que los usuarios teleoperan el robot. Por otro lado, se tiene el caso de FS13 en el que el robot junto a un usuario tienen que montar una caja de madera y una pequeña silla de madera. Por otro lado, se tienen también los casos en los que se utiliza un grupo de participantes para evaluar la arquitectura, como en los artículos FS7 y FS11.

Las evaluaciones que no dependen de humanos también están presentes. De

esta forma, se tienen dos grupos. Por un lado, están las evaluaciones centradas en componentes separados de la arquitectura, como es el caso de FS3, en la que se evalúa el sistema de reconocimiento de objetos; FS6, en la que se evalúa la capacidad de esquivar objetos; FS10, en la que se evalúa la navegación social; y FS13, en la que se evalúa el funcionamiento de los diferentes modelos entrenados por separado. Por otro lado, se tienen evaluaciones de la arquitectura completa sin depender de humanos. Un ejemplo es la evaluación que se presenta en el artículo FS1 en la que se utiliza un brazo robótico para coger y colocar objetos, un rover simulado para adquirir imágenes y utiliza un robot de vigilancia también para adquirir imágenes, pero esta vez teniendo en cuenta la batería. Otros casos son los presentados en los artículos FS6, que se centra en la ejecución de tareas jerárquicas basadas en realizar entregas; y FS12, en el que se realizan múltiples misiones al robot, como cerrar una válvula de gas, traer un libro, abrir las cortinas y monitorizar la casa.

#### **2.5.4. ¿Qué métricas se utilizan para evaluar las arquitecturas cognitivas?**

Existen diferentes métricas utilizadas en las evaluaciones de los artículos. Se tienen algunas que son más recurrentes que otras como es el caso del tiempo de ejecución de las misiones, utilizada en los artículos FS1, FS2, FS6, FS9, FS12; el tiempo empleado en el sistema deliberativo o las llamadas al planificador, utilizada en los artículos FS1, FS8, FS9; y la distancia recorrida por el robot, utilizada en los artículos FS1, FS9, FS10. Otras métricas son el número de ejecuciones exitosas, utilizada en el artículo FS5; y el número de acciones ejecutadas, utilizada en el artículo FS9. También se tienen otras métricas interesantes en el artículo FS8, como la robustez, cuántos intentos necesita el robot para realizar la tarea; y la explicabilidad de la arquitectura, si es posible explicar las decisiones del robot; y en el artículo FS10, en el que se usan métricas de rendimiento el uso de CPU, el uso de red y la memoria utilizada. Por otro lado, en los artículos FS2, FS5, FS7 y FS11 se han utilizado cuestionarios para evaluar la arquitectura centrándose en diferentes cuestiones sobre la interacción y la usabilidad.

Por otro lado, se tienen métricas más específicas de las arquitecturas y de sus componentes. Por ejemplo, en el artículo FS2 se tiene como métrica el tiempo entre colisiones, mientras que en el artículo FS6 se utiliza la precisión al esquivar obstáculos, las entregas realizadas y los planes erróneos. En el artículo FS10 también se utilizan métricas relacionadas con la navegación, en este caso la distancia mínima al humano durante la navegación y las intrusiones en el espacio personal de la persona. Por otra parte, se tienen métricas relacionadas con la obtención de conocimiento, como es el caso del artículo FS3 en el que se utilizan el número de categorías aprendidas, el número de preguntas e interacciones para aprender las categorías, la media de instancias almacenadas por categoría y la precisión global de la clasificación. Además, se tienen otras métricas relacionadas con la interacción con el usuario, como las presentadas por FS11, que son el número de turnos llevados a cabo durante los ejercicios, el número de errores en el diálogo y la duración de las interacciones. Por último, se tiene el caso de FS13 en el que se evalúan los modelos entrenados con métricas como matrices de confusión, error cuadrático medio raíz (RMSE) y error absoluto medio (MAE). En el caso de los experimentos con usuarios, las métricas empleadas en FS13 fueron la media de tareas completadas y la media del tiempo que el usuario esperaba.

## 2.6. Conclusiones

Se ha llevado a cabo una exhaustiva revisión sistemática de la literatura con el objetivo de identificar y analizar las arquitecturas cognitivas empleadas en robots autónomos, así como las herramientas utilizadas para generar sus comportamientos. Este estudio ha implicado un proceso que abarcó las fases de búsqueda, selección, evaluación de calidad y extracción de datos, proporcionando una visión de las tendencias y prácticas en el campo de las arquitecturas cognitivas.

Los resultados obtenidos revelan que las arquitecturas híbridas son las predominantes en el ámbito de los robots autónomos. En estas arquitecturas se utiliza conocimiento simbólico que se almacena en una base de conocimiento. Se pueden utilizar diferentes representaciones del conocimiento como PDDL o grafos de conocimiento. El conocimiento simbólico se emplea de manera significativa en los

sistemas deliberativos para la generación de planes que resolverán las misiones del robot. Paralelamente, las herramientas reactivas, como las máquinas de estados y los árboles de comportamiento, se integran en diferentes niveles de la arquitectura cognitiva, como por ejemplo controlando tanto las acciones del robot como otros aspectos del sistema, lo que resalta la flexibilidad y versatilidad de estos enfoques. En cuanto a los componentes emergentes, se observa su presencia en la implementación de habilidades específicas del robot, como el reconocimiento de objetos. Esta habilidad no solo demuestra la capacidad del robot para interactuar con su entorno, sino también para adquirir información que puede ser almacenada en la base de conocimientos mejorando así su comprensión del mundo circundante.

Por otro lado, el análisis de las evaluaciones realizadas muestra una diversidad significativa en los enfoques utilizados. Algunas evaluaciones implican interacción directa con los humanos, mientras que otras se llevan a cabo sin su participación. En cuanto a las métricas empleadas, las más predominantes son los cuestionarios, el tiempo de ejecución, el tiempo necesario para la generación de planes y la distancia recorrida. Sin embargo, se tienen otras métricas interesantes como el número de misiones resueltas con éxito, el grado de explicabilidad de la arquitectura y métricas de rendimiento, como el uso de CPU, la utilización de red y la memoria utilizada.

Por todo lo anterior, esta revisión sistemática no solo proporciona una visión detallada de las arquitecturas cognitivas en robots autónomos y las herramientas asociadas, sino también destaca la diversidad y complejidad del proceso de evaluación. Este panorama variado subraya la necesidad continua de investigaciones y enfoques innovadores para comprender y mejorar las capacidades de las arquitecturas cognitivas que se utilizan en los robots autónomos en un mundo cada vez más complejo y dinámico.



# Capítulo 3

## Arquitectura Cognitiva

Partiendo de los resultados obtenidos de la revisión sistemática, este capítulo define y formaliza la arquitectura cognitiva propuesta en este trabajo de investigación, .

### 3.1. Visión General

Siguiendo la clasificación de arquitecturas de control y cognitivas revisada en el capítulo anterior, una arquitectura cognitiva híbrida para robots necesita un enfoque que combina diferentes paradigmas o modelos cognitivos para dotar a los robots de capacidades que le permitan generar comportamientos autónomos.

Se puede definir la arquitectura cognitiva híbrida como una tupla de seis elementos. Esta definición se presenta en la ecuación 3.1.

$$CA = \langle \sigma, \alpha, \delta, \Omega, \Phi, A \rangle \quad (3.1)$$

- $\sigma$  define el dominio de habilidades del robot. Estas habilidades representan la unidad mínima de funcionalidad lógica o física, con acceso a uno o múltiples sensores o actuadores y un sistema de conocimiento. De este modo, se tiene la habilidad de navegación, el reconocimiento de objetos, la síntesis del habla o el reconocimiento del habla.
- $\alpha$  define el dominio de acciones del robot. Estas acciones son composiciones del dominio de habilidades del robot para realizar tareas. Las acciones se

pueden implementar como máquinas de estados cuyos estados utilicen las habilidades del robot.

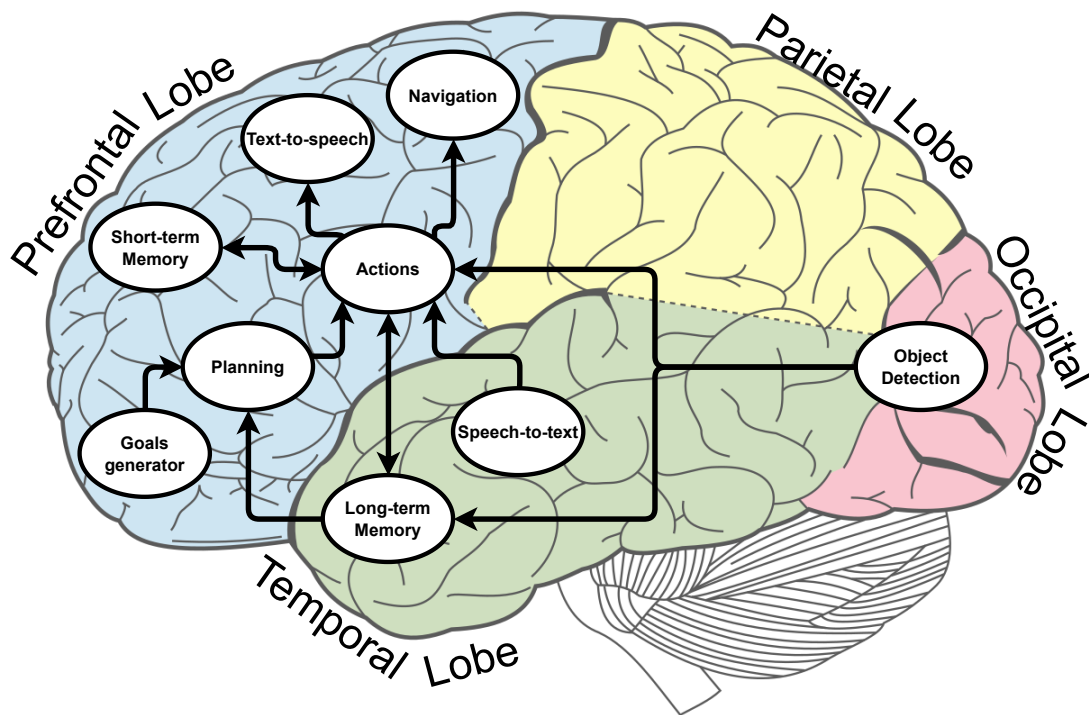
- $\kappa$  es el dominio de símbolos que representan el mundo del robot, es decir, el conocimiento simbólico o subsimbólico que se almacena en una base de conocimiento para ser utilizado por las habilidades del robot.
- $\Omega$  es la función que determina el siguiente objetivo del robot, tal que  $\Omega : \kappa \rightarrow \kappa$ , es decir, la función toma como elemento del dominio la base de conocimiento del robot y devuelve el siguiente objetivo del robot.
- $\Phi$  es la función de planificación, tal que  $\Phi : \kappa \times \alpha \rightarrow \alpha$ , es decir, la función toma como elemento del dominio un par ordenado formado por el conocimiento simbólico y las acciones del robot y devuelve como resultado un plan, es decir, un subconjunto de las acciones del robot.
- $A$  es la función que ejecuta las acciones, tal que  $A : \alpha \times \kappa \rightarrow \kappa$ , es decir, la función toma como elemento del dominio un par ordenado formado por una acción y el conocimiento simbólico del robot y devuelve como resultado el conocimiento del robot actualizado después de ejecutar esa acción.

Los dominios nos permiten definir el punto de partida de las capacidades y conocimiento que puede gestionar el robot. Por otro lado, las funciones nos permiten relacionar el conocimiento, las habilidades, y acciones en el robot que tendrán como resultado la generación de comportamientos que resuelvan tareas. Esta aproximación encaja en muchos aspectos con una aproximación bioinspirada donde tanto dominio como funciones son establecidas de forma natural cuando aplica sobre un humano que explicaremos a continuación.

### 3.1.1. Aproximación Bioinspirada

En la naturaleza, las capacidades cognitivas como la memoria a corto plazo, la percepción o las habilidades motoras se realizan en ciertos lóbulos del cerebro humano [51, 52]. Organizar los componentes de la arquitectura cognitiva como funciones cognitivas del cerebro da como resultado el esquema mostrado en la Figura 3.1.





**Figura 3.1** – Representación bioinspirada de los componentes de la arquitectura cognitiva. Los componentes de la arquitectura cognitiva a los diferentes lóbulos que se encuentran en el cerebro humano. El lóbulo prefrontal se encarga de la planificación la memoria a corto plazo, la síntesis de voz, el control de los motores y las acciones. El lóbulo temporal se encarga de la memoria a largo plazo y el reconocimiento del habla. El lóbulo occipital se encarga de la percepción, es decir, del reconocimiento de objetos. El lóbulo parietal se encargaría de la manipulación de objetos.

El cerebro humano está compuesto por cuatro lóbulos: prefrontal, temporal, occipital y parietal [53]. El lóbulo prefrontal alberga la mayoría de los módulos de la arquitectura cognitiva. Es responsable de la memoria a corto plazo, mientras que la memoria a largo plazo se sitúa en el lóbulo temporal. La memoria a corto plazo corresponde a la memoria de las máquinas de estado. La memoria a largo plazo del lóbulo temporal, que representa el estado del entorno del robot, corresponde a la base de conocimientos simbólica. Además, la memoria a largo plazo se utiliza en el proceso de planificación para elegir las acciones que el robot debe utilizar para

lograr sus objetivos. Estas acciones, que se encuentran en el lóbulo prefrontal, se implementan con máquinas de estado y también pueden modificar el conocimiento de la memoria a largo plazo. Las acciones pueden utilizar diferentes acciones, como la navegación, texto a voz y voz a texto para interactuar con el entorno y los seres humanos. Por otro lado, el lóbulo prefrontal también se encarga del control motor, la navegación y la síntesis de voz; mientras que el lóbulo temporal se encarga del procesamiento de audio y la conversión de voz a texto. El lóbulo occipital es el encargado de los procesamientos visuales, es decir, de la detección de objetos.

Tanto la visión general como la aproximación bioinspirada, permiten definir los componentes centrales para crear una arquitectura híbrida para robots móviles. Posteriormente, se crearan o integraran dichos componentes en la arquitectura desarrollada.

## **3.2. Núcleo de la Arquitectura**

En el contexto de la inteligencia artificial y las arquitecturas cognitivas, el término "núcleo" se refiere a la parte central o esencial de la arquitectura que lleva a cabo las funciones cognitivas fundamentales y, en muchos casos, se relaciona con consciencia. Sin embargo en este trabajo de investigación, la idea es definir los componentes centrales que orquestan y coordinan las diferentes capacidades cognitivas del sistema basadas en gestión y uso del conocimiento.

### **3.2.1. Percepción**

Independientemente de su diseño y propósito, un sistema inteligente no puede existir aislado por lo que necesita la entrada de datos para producir comportamientos. Aunque históricamente todas las arquitecturas cognitivas se centran en el razonamiento, es evidente que la percepción juega un papel importante. La percepción es un proceso que transforma la entrada cruda de los datos de los sensores en una representación interna del sistema para llevar a cabo tareas cognitivas. Dependiendo del origen y las propiedades de los datos entrantes, se distinguen múltiples modalidades sensoriales. Las cinco más comunes son la visión, el oído, el

olfato, el tacto y el gusto. Otros sentidos humanos reconocidos incluyen la propiocepción, la termocepción y el sentido del tiempo. Naturalmente, las arquitecturas cognitivas implementan algunas de estas, así como otras modalidades que no tienen una correlación entre los sentidos humanos, como la entrada a través de teclado o interfaz gráfica de usuario y sensores como LiDAR (Light Detection and Ranging), IMU (Inertial Measurement Unit) o la brújula.

Como se ha visto en la revisión sistemática, la percepción visual es una de las modalidades sensorial más predominantes en las arquitecturas cognitivas. Esta percepción se basa en el procesamiento de las imágenes obtenidas de una cámara del robot con el fin de identificar los objetos que hay en el entorno del robot. Para ello se pueden utilizar diferentes técnicas, como los componentes emergentes. En el caso de este trabajo, se ha elegido YOLOv8 [54], una herramienta que permite detectar objetos. Estos datos se puede utilizar posteriormente para enriquecer la base de conocimiento simbólica o de forma puntual en las acciones del robot.

### 3.2.2. Selección de acciones

La selección de acciones es un aspecto clave en la arquitectura cognitiva. Dependiendo del nivel, se pueden identificar la selección de acciones a alto nivel, a nivel simbólico, y la selección de acciones a bajo nivel, a nivel reactivo. En el nivel simbólico, se tiene una amplia cantidad de planificadores simbólicos que utilizan la base de conocimiento del robot para decidir cuál será la siguiente secuencia de acciones para resolver ciertos objetivos. Por otro lado, en el nivel reactivo se tienen diferentes mecanismos, como las máquinas de estados y los árboles de comportamiento, que deciden cuál será el siguiente paso a bajo nivel, es decir, qué habilidad del robot se utilizará después. En el caso de este trabajo, se han escogido las máquinas de estados, ya que están muy extendidas en entornos industriales y tienen la capacidad para representar de manera clara y organizada los diferentes estados y comportamientos que un robot puede experimentar. Estas máquinas de estados se pueden utilizar en el nivel reactivo y a alto nivel para seleccionar los siguientes objetivos.

### Máquinas de estados

Máquinas de Estados Finitos (MEF o FSM), corresponden a un modelo matemático compuesto de un conjunto finito de estados, un conjunto finito de eventos o entradas, una función de transición que define cómo el sistema se mueve de un estado a otro en respuesta a un evento, y una condición de inicio. Una Máquina de Estados está en un estado particular en un momento dado, y diferentes eventos externos e internos pueden hacer que la máquina transite a un nuevo estado según las reglas definidas por la función de transición

Como se ha descrito en la definición de la arquitectura cognitiva, los dominios de acciones del robot son máquinas de estados que utilizan las habilidades del robot. De forma pormenorizada, una máquina de estados se define como una tupla de siete elementos que consta de una lista de estados, un conjunto de símbolos de entrada, un conjunto de símbolos de salida, un estado inicial desde el cual la máquina comienza y dos funciones: una que determina el comportamiento de transición de los estados y la otra que determina el procesamiento de los símbolos de entrada traducidos en una secuencia de símbolos de salida. Esta definición se presenta en la ecuación 3.2.

$$FSM = \langle \sigma, \tau, \delta, \sigma^*, \tau^*, \Delta, \Omega \rangle \quad (3.2)$$

- $\sigma$  es el conjunto de estados.
- $\tau$  es el conjunto de símbolos de entrada.
- $\delta$  es el conjunto de símbolos de de salida.
- $\sigma^*$  es el estado inicial.
- $\tau^*$  es la entrada inicial.
- $\Delta$  es la función de transición, tal que  $\Delta : \sigma \times \tau \rightarrow \sigma$ , es decir, la función toma como elemento del dominio un par ordenado que consiste en un estado y datos de entrada, devolviendo como imagen otro estado. Esta imagen representa el estado del sistema después del procesamiento indicado por el par ordenado (estado, entrada).

- $\Omega$  es la función de salida, tal que  $\Omega : \sigma \times \tau \rightarrow \delta$ , es decir, la función toma como elemento del dominio un par ordenado formado por un estado y datos de entrada y devuelve como imagen un símbolo de salida. Este símbolo es la salida del sistema en ese instante de tiempo.

Además, las máquinas de estados pueden utilizar *blackboards* para almacenar el conocimiento simbólico o subsimbólico. Los estados y las máquinas de estados anidadas dentro de una máquina de estados utilizan el mismo *blackboard*. Esto significa que los símbolos de entrada y salida se almacenan en este *blackboard*. La definición de un *blackboard* también se presenta en la ecuación 3.3. Un *blackboard* es una tupla de cinco elementos formada por un conjunto de fuentes de conocimiento, un conjunto de símbolos de soluciones intermedias, la entrada inicial, dos funciones: una función que determina qué fuente de conocimiento ejecutar a continuación y una función que ejecuta una fuente de conocimiento.

$$Blackboard = \langle \sigma, \tau, \tau^*, \Delta, \Omega \rangle \quad (3.3)$$

- $\sigma$  es el conjunto de fuentes de conocimiento.
- $\tau$  es el conjunto de símbolos.
- $\tau^*$  es la entrada inicial.
- $\Delta$  es la función planificadora que determina qué fuente conocimiento ejecutar, de manera que:  $\Delta : \tau \rightarrow \sigma$ , es decir, la función toma como elemento del dominio un conjunto de símbolos, devolviendo la siguiente fuente de conocimiento a ejecutar.
- $\Omega$  es la función de ejecución, de manera que:  $\Omega : \sigma \times \tau \rightarrow \tau$ , es decir, la función toma como elemento del dominio un par ordenado que consiste en una fuente de conocimiento y un conjunto de símbolos, devolviendo un símbolo.

Finalmente, ambas definiciones pueden fusionarse para obtener la definición formal de una máquina de estados que utilice *blackboards*, que se corresponden con la memoria a corto plazo de la representación bioinspirada de la arquitectura. Se aplican las siguientes correspondencias:

- Estados → Fuentes de conocimiento
- Conjuntos de símbolos de entrada y salida → Conjunto de símbolos
- Función de transición → Función planificadora
- Función de salida → Función de ejecución

De esta manera, una máquina de estados se define como una tupla de seis elementos formada por una lista de estados, un conjunto de símbolos correspondientes a la solución intermedia, el estado inicial, el conjunto de símbolos inicial y dos funciones: una que determina el comportamiento de transición de los estados y la otra que determina el procesamiento de los símbolos de entrada traducidos en una secuencia de símbolos de salida. La definición se presenta en la ecuación 3.4.

$$FSM_{blackboard} = \langle \sigma, \tau, \sigma^*, \tau^*, \Delta, \Omega \rangle \quad (3.4)$$

- $\sigma$  es el conjunto de estados
- $\tau$  es el conjunto de símbolos de entrada y de salida
- $\sigma^*$  es el estado inicial
- $\tau^*$  es la entrada inicial
- $\Delta$  es la función de transición, tal que,  $\Delta : \sigma \times \tau \rightarrow \sigma$ , es decir, la función toma un elemento del dominio, un par ordenado formado por un estado y un conjunto de símbolos, y devuelve otro estado.
- $\Omega$  la función de ejecución, tal que,  $\Omega : \sigma \times \tau \rightarrow \tau$ , es decir, la función toma un elemento del dominio, un par ordenado formado por un estado y un conjunto de símbolos, y devuelve un símbolo.

Así, para poder seleccionar acciones que den como resultado la generación de comportamientos en el robot mediante máquinas de estados, se tiene YASMIN [55], una librería que permite generar y monitorizar comportamientos basados en máquinas de estados.

### 3.2.3. Memoria

Como se ha mostrado en la representación bioinspirada y en la revisión sistemática, se presentan diferentes aproximaciones para la gestión y mantenimiento de la información en el robot a lo largo del tiempo, siendo las bioinspiradas las aproximaciones más desplegadas. Una de ellas es la memoria a corto plazo, que en este caso se corresponde con los *blackboards* de las máquinas de estados; mientras que la memoria a largo plazo se corresponde con el conocimiento simbólico del robot. Así, es necesario un mecanismo para poder gestionar el conocimiento simbólico del robot. Por esto motivo, se propone KANT [56] para gestionar el conocimiento simbólico del robot, el cual permite consultar, crear, modificar y eliminar el conocimiento del robot.

#### Conocimiento simbólico

Como se ha descrito, el conocimiento simbólico es uno de los elementos que forma la arquitectura cognitiva. Este conocimiento se almacena en una base de conocimientos que se puede definir como una tupla de cinco elementos  $\langle T, O, P, F, A \rangle$ :

- **Tipos:** es el conjunto de los tipos de objetos que se pueden dar en entorno del robot.
- **Objetos:** es el conjunto de objetos que permiten declarar objetos presentes en entorno del robot. Cada objeto pertenece a uno de los tipos declarados.
- **Predicados:** es el conjunto de las características del entorno y de los objetos.
- **Proposiciones:** es el conjunto de características que describen el estado actual del mundo.
- **Acciones:** es el conjunto de las acciones que el robot puede emplear para interactuar con el entorno.

Así pues, estos elementos conforman el conocimiento simbólico que tiene el robot almacenado en su base de conocimientos, que se corresponde con la memoria a largo plazo de la representación bioinspirada de la arquitectura. Por otro lado, los objetos del robot, que es otro de los elementos que forman la arquitectura cognitiva, son proposiciones que el robot quiere que se cumplan.

### 3.2.4. Aprendizaje

El aprendizaje es un proceso que se centra en obtener nuevos conocimientos con el objetivo de mejorar el funcionamiento del sistema. Existen diferentes tipos de aprendizaje, como el perceptual, que tratan de obtener datos de los sensores; el declarativo, que trata de obtener nuevos hechos y relaciones que definan el mundo del robot; el procedural, que trata de aprender a realizar nuevas habilidades; y el asociativo, que se define como el aprendizaje sobre la relación entre dos estímulos separados. En las arquitecturas cognitivas, el aprendizaje puede variar dependiendo del tipo de arquitectura. así, en las arquitecturas simbólicas, el aprendizaje se basa en obtener más conocimiento simbólico, mientras que en las arquitecturas emergentes el aprendizaje se basa en entrenar los modelos. Por otro lado, también se tienen aprendizajes subsimbólicos a niveles reactivos, como la generación de mapas de navegación.

En el caso de las arquitecturas híbridas, como la que se presenta en este trabajo, es necesario obtener conocimiento del mundo en el que se despliega el robot. De esta manera, la percepción es el proceso de convertir datos sensoriales en bruto en una representación interna de las arquitecturas cognitivas, en particular, conocimiento simbólico. Un paso más es el *Symbolic Anchoring* [14] que es el proceso de crear y mantener la correspondencia entre símbolos y datos del sensor que se refieren a los mismos objetos físicos. No solo se necesita la creación de conocimiento, sino también su mantenimiento. Este es un aspecto del *Symbolic Knowledge Grounding* [57], que es el problema de cómo fundamentar los significados de los símbolos utilizados por el robot; mientras que el proceso de fundamentar los símbolos en objetos del mundo real por un agente físico que interactúa en el mundo real se conoce como *Physical Symbol Grounding* [58]. Como resultado, es necesario una herramienta que permita detectar objetos en el entorno del robot y utilizarlos como conocimiento simbólico.

### 3.2.5. Razonamiento

El razonamiento es el proceso mental a través del cual una persona utiliza la información disponible para llegar de manera lógica a una conclusión. Las



arquitecturas cognitivas tratan el razonamiento práctico, cuyo objetivo final es encontrar la próxima mejor acción y llevarla a cabo, a diferencia del razonamiento teórico que tiene como objetivo establecer o evaluar creencias. Esto está fuertemente relacionado con los sistemas de planificación simbólicos, que tratan de generar planes que solucionen los objetivos del robot, en contraposición a las arquitecturas reactivas que carecen de estas habilidades. Por otro lado, en las arquitecturas emergentes el razonamiento tampoco es algo que se tenga en cuenta. Sin embargo, el surgimiento de los *Large Language Models* (LLMs), que son componentes emergentes, puede ayudar a las arquitecturas cognitivas en varias habilidades, siendo una de ellas el razonamiento. Con esto se consigue que el proceso de razonamiento de una arquitectura esté orquestado por un componente emergente que utiliza conocimiento simbólico escrito en lenguaje natural, lo que puede mejorar la interacción con los humanos.

### 3.2.6. Explicabilidad

La explicabilidad de robots autónomos es la capacidad de comprender y comunicar cómo toman decisiones o realizan acciones. Obtener esta información en un entorno bien conocido, como un laboratorio, es una tarea alcanzable donde gran parte de usuarios y desarrolladores tienen conocimiento previo del escenario y la técnica. A medida que los robots autónomos se trasladan a entornos públicos en los que no se utiliza un conjunto de datos de entrada predefinido y los usuarios no tienen experiencia técnica previa, la necesidad de comprender y explicar sus acciones se vuelve crucial, especialmente en aquellos entornos donde interactúan con humanos. Por lo tanto, en la actualidad, es necesario tener un sistema de auditoría que permita conocer el desencadenante de cualquier comportamiento del robot, regular o inesperado, y proporcionar un mecanismo para analizar el estado después de un incidente.

#### **Accountability**

*Accountability* en robótica implica que cualquier robot debe registrar sus actividades con pruebas verificables para que cualquier acción del robot sea rastreable y se identifiquen los eventos que desencadenaron la acción [59]. De esta manera,

la norma de ciberseguridad IEC 62443 define un conjunto de especificaciones y requisitos fundamentales para lograrlo. Un examen más detallado de esta norma permite encontrarse con elementos como el de la IEC 62443-3-3, que especifica sistemas capaces de generar registros de auditoría a través de la generación de logs [60]. Estos registros deben generar *accountability*, el requisito más importante para la explicabilidad.

Como se presenta en [61], para averiguar quién es el responsable del incorrecto funcionamiento de un robot autónomo, es necesario monitorizar y almacenar logs. Por otro lado, a medida que estos robots se integran cada vez más en nuestras vidas, comprender y confiar en su proceso de toma de decisiones se ha vuelto crucial. Esto ha llevado al surgimiento de robots autónomos explicables (XAR) que tienen el objetivo de proporcionar mecanismos para traducir los procesos a los diferentes operadores humanos con explicaciones claras e interpretables de sus acciones y decisiones. La explicabilidad ofrece una forma de establecer la confianza entre humanos y robots al permitir que los humanos comprendan el razonamiento subyacente detrás de las decisiones y acciones de un robot, promoviendo así la responsabilidad y la colaboración.

Un servicio de *accountability* ( $\mathcal{A}$ ) para un robot autónomo  $\mathcal{R}$  puede plantearse como una tupla de tres elementos, que son un conjunto de eventos, un conjunto de componentes y un conjunto de comportamientos. Esto se presenta en la ecuación 3.5.

$$\mathcal{A}_R = \langle \mathcal{C}, \mathcal{E}, \mathcal{B} \rangle \quad (3.5)$$

Un componente ( $\mathcal{C}$ ) identifica tanto un elemento de software con una interfaz y funcionalidad específicas [62]. Un evento ( $\mathcal{E}$ ) describe una ocurrencia que sucede en un instante específico y en un escenario particular como resultado de  $n$  hechos. Un comportamiento ( $\mathcal{B}$ ) describe una acción del robot desencadenada por uno o varios eventos.

Un motor de *accountability* gestiona la asignación de responsabilidades. Puede ser modelado como las funciones de mapeo mostradas en las ecuaciones 3.6 y 3.7.  $\mathcal{C}_e$  denota un subconjunto de eventos causado por uno o varios hechos específicos

disponibles en un componente  $\mathcal{C}$ . Por otro lado,  $\mathcal{E}_b$  denota un subconjunto de eventos  $\mathcal{E}$  que causan un comportamiento específico del robot  $b$ . Por lo tanto, la función  $\alpha$  toma un evento  $e$  como entrada y devuelve los hechos de los componentes que lo desencadenan. La función  $\beta$  toma un comportamiento del robot  $b$  como entrada y devuelve los eventos que lo causan.

$$\alpha : \mathcal{E} \rightarrow \{\mathcal{C}_e | \mathcal{C}_e \subseteq \mathcal{C}\} \quad (3.6)$$

$$\beta : \mathcal{B} \rightarrow \{\mathcal{E}_b | \mathcal{E}_b \subseteq \mathcal{E}\} \quad (3.7)$$

En condiciones ideales, las funciones de mapeo siempre producen resultados correctos. Esta situación se conoce como *Perfect Mapping* (PM) [63]. Cualquier función de mapeo  $\alpha$  se convierte en un PM si y solo si  $\mathcal{C}_e|PM$  incluye todos los hechos que un componente causa el evento  $e$ ; ver la Ecuación 3.8. Además, cualquier función de mapeo  $\beta$  se convierte en un PM si y solo si  $\mathcal{E}_b|PM$  incluye todos los eventos que desencadenan el comportamiento  $b$ ; ver la ecuación 3.9.

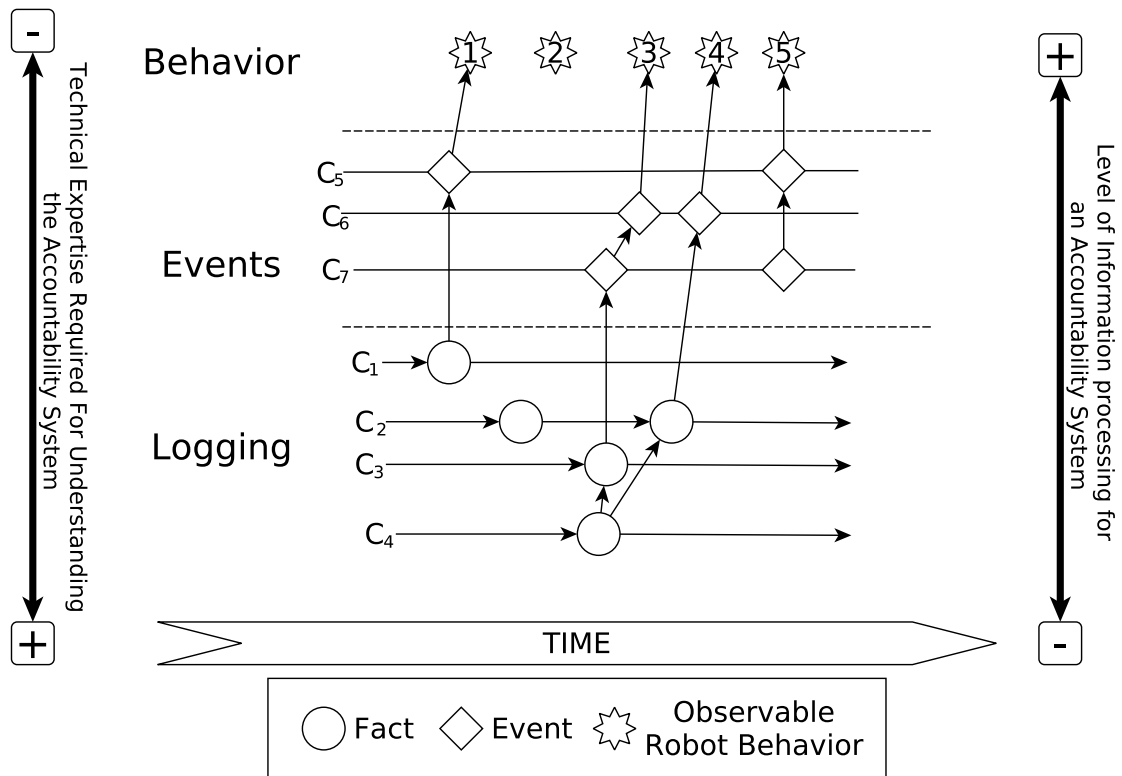
$$\alpha(e) = \mathcal{C}_e|PM \subseteq \mathcal{C} \quad (3.8)$$

$$\beta(c) = \mathcal{E}_b|PM \subseteq \mathcal{E} \quad (3.9)$$

Hoy en día, la diversidad de componentes software ejecutados en local y en la nube dificulta el PM. Cada componente software puede proporcionar suficiente información para un PM siempre y cuando ninguno de los resultados sea incorrecto. Sin embargo, los componentes externos hacen que el sistema de responsabilidad sea menos confiable, ya que no todas las partes responsables pueden identificarse de inmediato. Por ejemplo, un servicio de reconocimiento del habla online ofrecería información de registro mínima basada en una API REST. Típicamente, se tienen conjuntos no completos de componentes  $\mathcal{C}_e|PM$ , ver la ecuación 3.10; o conjuntos no completos de eventos  $\mathcal{E}_b|PM$ , ver la ecuación 3.11.

$$\alpha(e) = \mathcal{C}_e \subset \mathcal{C}_e|PM \subseteq \mathcal{C} \quad (3.10)$$

$$\beta(c) = \mathcal{E}_b \subset \mathcal{E}_b|PM \subseteq \mathcal{E} \quad (3.11)$$



**Figura 3.2** – Dimensiones, a alto nivel, para un robot responsable. En el diagrama se presentan varios comportamientos, componentes, eventos y hechos.

Esta formalización se ilustra en la Figura 3.2. El diagrama presentado simplifica el proceso explicado en [64]. Así pues, se tiene un conjunto de comportamientos, un conjunto de eventos y un conjunto de hechos observables del robot. El PM se ilustra en los eventos 1, 3 y 4, donde es posible rastrear desde los hechos del componente, los eventos desencadenados y los resultados visibles. Sin embargo, no se tienen comportamientos visibles ni rastreables en el evento 2, que surjan sin una traducción directa, por ejemplo, un comportamiento detenido que se desencadena después de un período de tiempo. El componente  $C_2$  muestra un hecho que no está asociado, aparentemente, con ningún comportamiento directo. El componente  $C_7$  muestra un evento que desencadena un nuevo evento y comportamiento sin un hecho claro. Podemos tener esta situación desde una solución en la nube que proporciona eventos fuera del sistema del robot.

### **Interpretabilidad**

XAR ha surgido en paralelo a XAI, o inteligencia artificial explicable, que comparte el objetivo similar de hacer que los sistemas de IA sean más transparentes e interpretables. Tanto XAR como XAI están motivados por la preocupación por la naturaleza de caja negra de los sistemas de IA y robótica, lo que puede dificultar que los humanos comprendan cómo operan estos sistemas y cómo toman decisiones. XAR se basa en los principios y técnicas de XAI y los aplica al contexto específico de la robótica autónoma, donde la necesidad de transparencia e interpretabilidad es particularmente crucial.

La interpretabilidad es fundamental para garantizar la transparencia de las acciones del sistema. Un sistema robótico autónomo que puede proporcionar explicaciones interpretables y comprensibles de sus acciones puede ayudar a mejorar la confianza y seguridad en el rendimiento y la toma de decisiones del sistema. Al incorporar la interpretabilidad en el diseño de robots autónomos, los investigadores buscan crear sistemas que no solo funcionen de manera efectiva, sino que también fomenten la confianza y la colaboración con los humanos.

En las arquitecturas cognitivas, la explicabilidad e interpretabilidad se pueden llevar a cabo de diversas maneras. Como se ha presentado en [65], se pueden aprovechar las capacidades narrativas para explicar e interpretar los logs del robot. Otra posible solución es el uso de visualizadores de máquinas de estados, lo que facilita la monitorización en las arquitecturas reactivas y en las arquitecturas híbridas que las utilicen. En el caso de las arquitecturas simbólicas, el conocimiento simbólico es fácilmente entendible por los operadores humanos. El resultado de los planificadores simbólicos también se puede utilizar para mostrar, a alto nivel, el comportamiento del robot. Por último, en las arquitecturas emergentes, que utilizan diferentes modelos entrenados, la explicabilidad e interpretabilidad son más complicadas, ya que sería necesario explicar los pesos obtenidos después de realizar el entrenamiento.

### 3.3. Sistema Deliberativo

Los sistemas deliberativos son estructuras computacionales diseñadas para permitir la toma de decisiones, la planificación y el razonamiento dentro de sistemas autónomos. Como se muestra en el trabajo de Ingrand y Ghallab [4], estos sistemas están compuestos de un conjunto de funciones deliberativas que puede variar. En primer lugar, la planificación, combina la predicción y la búsqueda para generar una secuencia de acciones abstractas basada en modelos predictivos del entorno y acciones factibles con el fin de lograr algún objetivo. La actuación se basa en ejecutar las acciones planificadas como comandos apropiados para el contexto actual y reacciona a eventos, utilizando las habilidades del robot, es decir, una colección de funciones que procesan secuencias de estímulos de entrada de sensores y genera salida a actuadores, para desencadenar fuerzas motoras y controlar el logro correcto de las acciones elegidas. Por otro lado, la observación detecta y reconoce características y relaciones que caracterizan el estado del mundo, así como eventos y acciones. Combina procesos ascendentes, desde sensores hasta datos significativos, con actividades descendentes como mecanismos de atención, acciones de percepción y planificación para la recopilación de información. La monitorización compara lo que se predice sobre las actividades del robot con lo que se observa en el mundo. Por último, se tiene el aprendizaje, que permite que un robot adquiera nuevo conocimiento, y se adapte y mejore mediante la experiencia.

### 3.4. Sistema de comportamientos

Los sistemas de comportamientos en las arquitecturas cognitivas representan un conjunto de componentes fundamentales para el funcionamiento de robots autónomos. Estos sistemas están compuestos por máquinas de estado, árboles de comportamiento y habilidades reactivas, como la navegación. Las máquinas de estado ofrecen una estructura lógica para definir estados y transiciones en las que el robot puede encontrarse, permitiendo una gestión eficiente de su comportamiento en diferentes situaciones. Por otro lado, los árboles de comportamiento permiten

una representación jerárquica y modular de las acciones y decisiones del robot, facilitando la toma de decisiones complejas a partir de comportamientos simples. Además, las habilidades reactivas, como la navegación o el reconocimiento del habla, constituyen bloques fundamentales que permiten respuestas rápidas y adaptativas del robot ante estímulos del entorno. Estos sistemas son esenciales para dotar a los robots de la capacidad de interactuar de manera efectiva y autónoma con su entorno.

### **3.5. Conclusiones**

En este capítulo se ha presentado formalmente la arquitectura de este trabajo. Desde una aproximación bioinspirada hasta la división la arquitectura, se tienen sus diferentes capacidades cognitivas, como la percepción, la selección de acciones, la memoria, el aprendizaje, el razonamiento y la explicabilidad. Posteriormente, la arquitectura se ha dividido en dos sistemas, el deliberativo y el de comportamientos, que puede hacer uso de las diferentes capacidades cognitivas. De esta manera, el sistema deliberativo tendría una memoria, el razonamiento y la selección de acciones a alto nivel. Por otro lado, el sistema de comportamientos contendría la percepción, la selección de acciones a bajo nivel y el control de los motores del robot. Por último, sería necesario un sistema de aprendizaje, que utilizase elementos de ambos sistemas, y la capacidad de explicar las acciones del robot, en los dos sistemas.





# Capítulo 4

## MERLIN

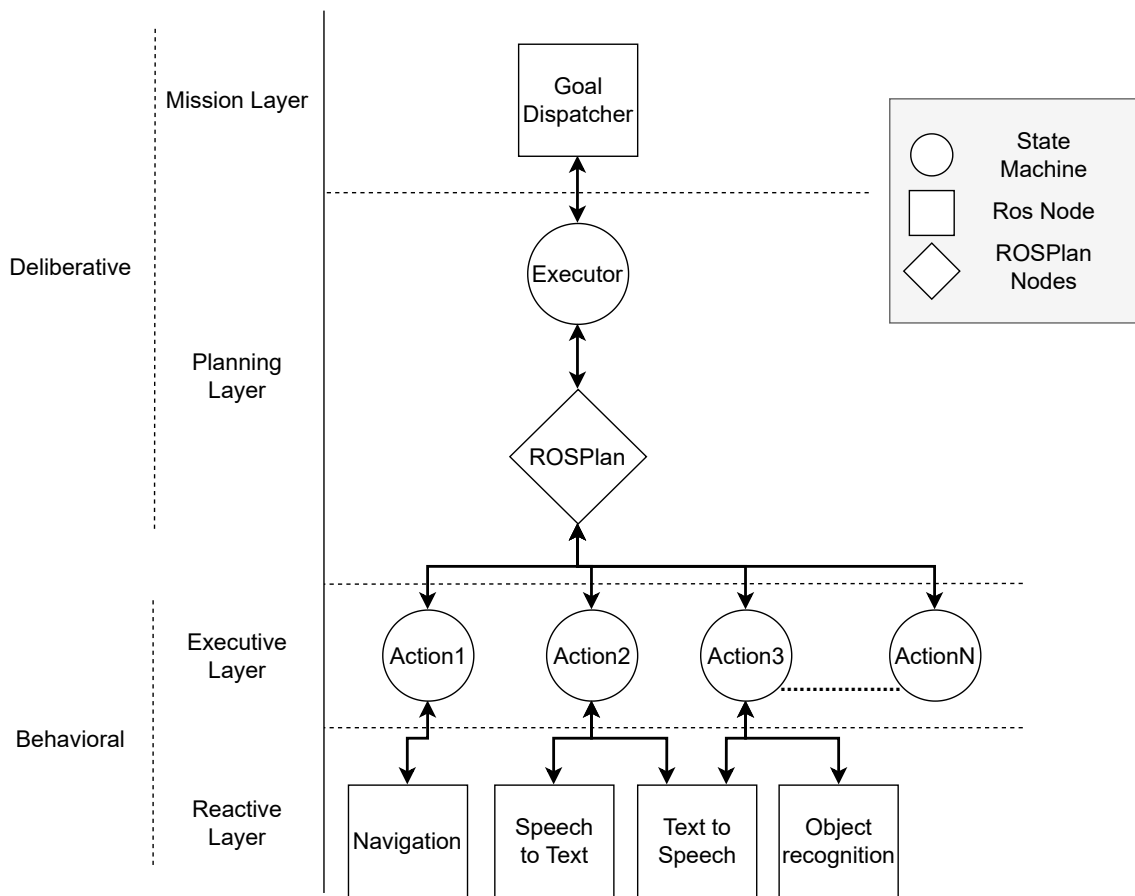
En este capítulo, se presenta MERLIN [66], la primera versión de la arquitectura cognitiva para robots móviles autónomos que se generó en este trabajo de investigación. Estaba implementada sobre el middleware de ROS [19]. MERLIN surgió como una arquitectura híbrida que utiliza como sistema deliberativo un sistema de planificación simbólica, implementado mediante la librería ROSPlan [48], para generar comportamientos a largo plazo; y un conjunto de máquinas de estados para generar comportamientos a corto plazo, implementadas con SMACH [67]. Utiliza una base de conocimientos basada en PDDL [12] y despliega diferentes habilidades como reconocimiento de voz basado en sistemas en la nube o detección de objetos con redes neuronales convolucionales.

### 4.1. Visión General

MERLIN, ilustrado en la Figura 4.1, está compuesto por un sistema deliberativo (*Deliberative System*), que se encarga de tomar decisiones y crear planes, y un sistema de comportamientos (*Behavioral System*), que contiene las acciones y las habilidades que el robot puede realizar. Estos dos sistemas constan de capas que se construyen una encima de otra. El sistema deliberativo se compone de la *Mission Layer* y la *Planning Layer*. El sistema de comportamientos está compuesto por *Executive Layer* y la *Reactive Layer*.

La *Mission Layer* es la capa encargada de crear los objetivos que el robot debe alcanzar. Fue diseñada para que otros sistemas de generación de objetivos puedan

integrarse fácilmente en la arquitectura. El componente principal de esta capa es el *Goal Dispatcher* y se encarga de enviar los objetivos del robot a la siguiente capa.



**Figura 4.1** – Diagrama de la arquitectura MERLIN formada por los sistemas *Deliberative System* y *Behavioral System* que se dividen en cuatro capas, que son *Mission Layer*, *Planning Layer*, *Executive Layer* y *Reactive Layer*.

La *Planning Layer* se encarga de generar y ejecutar los planes. Estos planes son secuencias de acciones para lograr los objetivos del robot. El componente principal de esta capa es ROSPlan, que proporciona las herramientas necesarias para gestionar y mantener el conocimiento PDDL en una base de conocimientos, llevar a cabo la pacificación y ejecutar las acciones de la siguiente capa. Sin embargo, ROSPlan tiene ciertas limitaciones, como la falta de replanificación y el hecho de que las acciones no se cancelen al cancelar el plan, que corrige el otro componente de esta capa, el *Executor*. Este componente se encarga de ejecutar y gestionar ROSPlan.

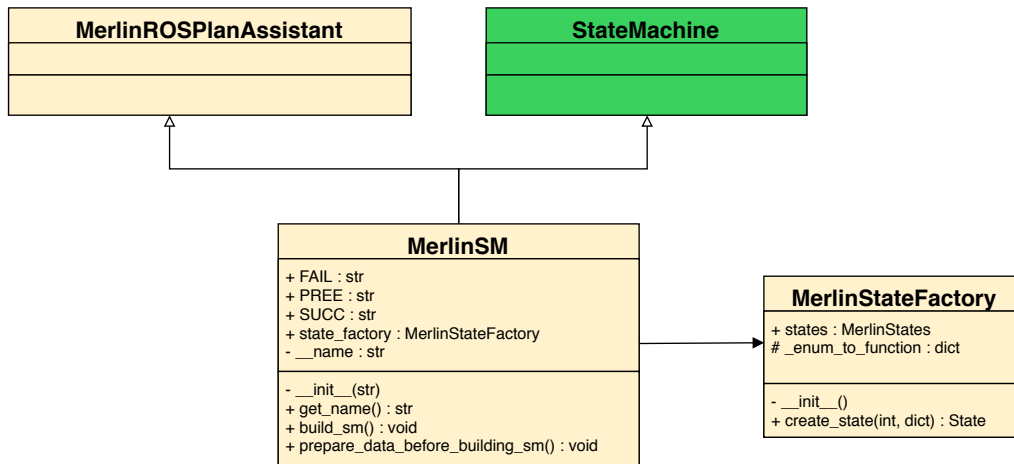
La *Executive Layer* comprende las acciones que se utilizarán para crear los planes. Las acciones se han diseñado como máquinas de estados para controlar mejor su ejecución. Las acciones pueden ejecutar los componentes de la siguiente para generar comportamientos a corto plazo. Además, pueden editar el conocimiento que tiene el robot.

Por último, la *Reactive Layer* está compuesta por los elementos que tienen que gestionar sensores y actuadores. Los sistemas reactivos tratados son navegación, reconocimiento de objetos, voz a texto y texto a voz.

## 4.2. Núcleo de MERLIN

En esencia, MERLIN utiliza máquinas de estados capaces de interactuar con ROSPlan. Como se muestra en la Figura 4.2, la clase *MerlinSM* representa las máquinas de estados de MERLIN. Esta clase se basa en *StateMachine*, que es la clase en verde, y la clase *MerlinROSPlanAssistant*. *StateMachine* pertenece a SMACH y proporciona el motor de la máquina de estados. Por otro lado, *MerlinROSPlanAssistant* tiene un conjunto de funciones para interactuar con la base de conocimiento de ROSPlan. Mediante estas funciones se puede crear, actualizar o eliminar el conocimiento que tiene el robot del escenario. Finalmente, se ha creado una fábrica para gestionar la creación de estados. Esta fábrica está representada en este diagrama de clases por la clase *MerlinStateFactory*, que se ha desarrollado utilizando el patrón de diseño de software de fábrica.

Las máquinas de estados se utilizan especialmente para crear las acciones de la *Executive Layer*. Además, se han utilizado en la *Mission Layer* y en la *Planning Layer*. Las máquinas de estados de la *Executive Layer* y de la *Planning Layer* se han creado utilizando el patrón de diseño fábrica, que se basa en utilizar una clase de fábrica para crear los estados de las máquinas de estado. Este patrón ayudará a organizar y crear nuevos estados asociados con las capacidades básicas del robot, como la navegación y el reconocimiento de objetos. Además, proporciona los estados para la capa de planificación.



**Figura 4.2** – Diagrama de clases de las máquinas de estados de MERLIN. La clase *MerlinSM* representa las máquinas de estados de MERLIN, que es una máquina de estados de SMACH al heredar de *StateMachine* y tienen varias funciones para manipular el conocimiento de ROSPlan al heredar de la clase *MerlinROSPlanAssistant*. La clase *MerlinStateFactory* proporciona una factoría para crear unos estados predefinidos.

### 4.2.1. Percepción

La percepción realizada en la arquitectura MERLIN se basa en el reconocimiento de objetos. Específicamente, se utiliza YOLOv3 [68]. Los objetos detectados de esta forma se utilizaban en las acciones de la arquitecturas.

### 4.2.2. Selección de acciones

#### SMACH

SMACH (*State MACHine*) [67] es una librería de escrita en Python para desarrollar comportamientos mediante máquinas de estados. Permite la construcción de máquinas de estados jerárquicas que se comunican con sistemas basados en ROS mediante sus interfaces. Como resultado, tiene el potencial de generar comportamientos robóticos para realizar diferentes tareas.

SMACH proporciona una interfaz para simplificar el proceso de agregar nuevos comportamientos utilizando código Python personalizado. Además, proporciona una interfaz con un conjunto de clases de estados que facilita la integración de

máquinas de estados de SMACH en ROS. De esta forma, se tienen los estados de *ServiceState*, para ejecutar llamadas a servicio de ROS; *MonitorState*, para monitorizar los mensajes de los topics de ROS; y *SimpleActionState*: para ejecutar llamadas a servidores de acciones de ROS.

### ROSPlan

ROSPlan [48] es un framework basado en ROS que proporciona un conjunto de herramientas para trabajar con planificación basada en inteligencia artificial. También se encarga de gestionar el conocimiento, basado en PDDL, que tiene el robot. Hay dos elementos escritos en PDDL:

- **El dominio**, que se compone de los tipos de objetos que pueden existir en el problema; los predicados, que son las propiedades que pueden tener los objetos; y las acciones que el robot puede realizar para modificar el problema.
- **El problema**, que se compone de los objetos que existen en el mundo del robot; las proposiciones o hechos, que son las instancias de los predicados; y los objetivos, que son las proposiciones que se quieren cumplir en el problema, es decir, que sean verdaderas.

ROSPlan se compone de los siguientes componentes:

- **Knowledge Base**: este componente, es decir, la base de conocimiento; se encarga de almacenar el dominio y el problema PDDL del robot.
- **Problem Interface**: este componente crea el problema PDDL con los datos de la base de conocimiento.
- **Planner Interface**: este componente crea el plan utilizando el dominio y el problema. Un plan es una secuencia de acciones necesarias para lograr objetivos específicos. El planificador utilizado en MERLIN es POPF [13].
- **Parsing Interface**: este componente se encarga de analizar el plan para que pueda ser ejecutado.
- **Plan Dispatch**: este componente ejecuta el plan llamando a las acciones que lo componen. Cada vez que se completa una acción, la base de conocimientos se actualiza agregando o eliminando proposiciones.

ROSPlan dispone de varias interfaces basadas en ROS para utilizar estos componentes. Hay un servicio y una acción (*actionlib*) de ROS para ejecutar cada componente, excepto la base de conocimientos que tiene varios servicios ROS para consultar, editar y eliminar el conocimiento.

### 4.2.3. Memoria

La memoria de esta arquitectura se corresponde con la base de conocimientos de ROSPlan. Esta base de conocimiento almacena el conocimiento simbólico del robot utilizando PDDL. Así, se almacenan tanto los elementos del dominio como los elementos del problema de PDDL.

### 4.2.4. Aprendizaje

El aprendizaje en esta arquitectura se basa en el uso de la percepción para crear nuevo conocimiento simbólico. Esto se puede hacer a través de las acciones que utilizan la percepción. De esta manera, se pueden crear nuevos objetos PDDL a partir los objetos detectados.

### 4.2.5. Razonamiento

En esta arquitectura, no se realizado ningún proceso de razonamiento más allá de la planificación de tareas a alto nivel mediante planificadores de PDDL.

### 4.2.6. Explicabilidad

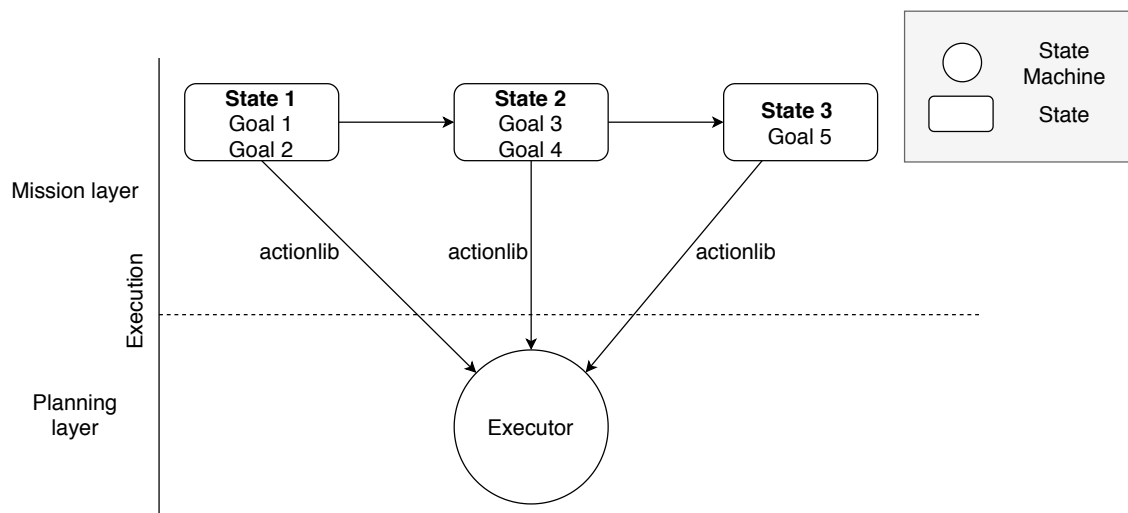
La explicabilidad de esta arquitectura se basa en el uso de logs de ROS y el visualizador de SMACH, mediante el cual se puede conocer el estado de todas las máquinas de estados de SMACH.

## 4.3. Sistema deliberativo

El sistema deliberativo debe controlar el comportamiento que un robot necesita para alcanzar ciertos objetivos. Está compuesto por dos capas:

### 4.3.1. Mission Layer

Esta capa debe gestionar los objetivos, escritos en PDDL, que un robot necesita alcanzar. Puede implementarse como un nodo ROS normal o como una máquina de estados, como se muestra en la Figura 4.3. Cada estado tiene varios objetivos que se envían al *Executor* situado en la siguiente capa, la *Planning Layer*. Como se muestra en la Figura 4.1, este componente se llama *Goal Dispatcher*. Su componente software principal es *MerlinExecutorClient*. Es una clase de Python que utiliza un cliente de acción (*actionlib*) de ROS para comunicar esta capa y con la *Planning Layer*. Esta clase proporciona una funcionalidad que permite cancelar el comportamiento actual del robot. Esto es útil en la robótica asistencial, donde existen objetivos con diferentes prioridades, principalmente orientados a los servicios al usuario.

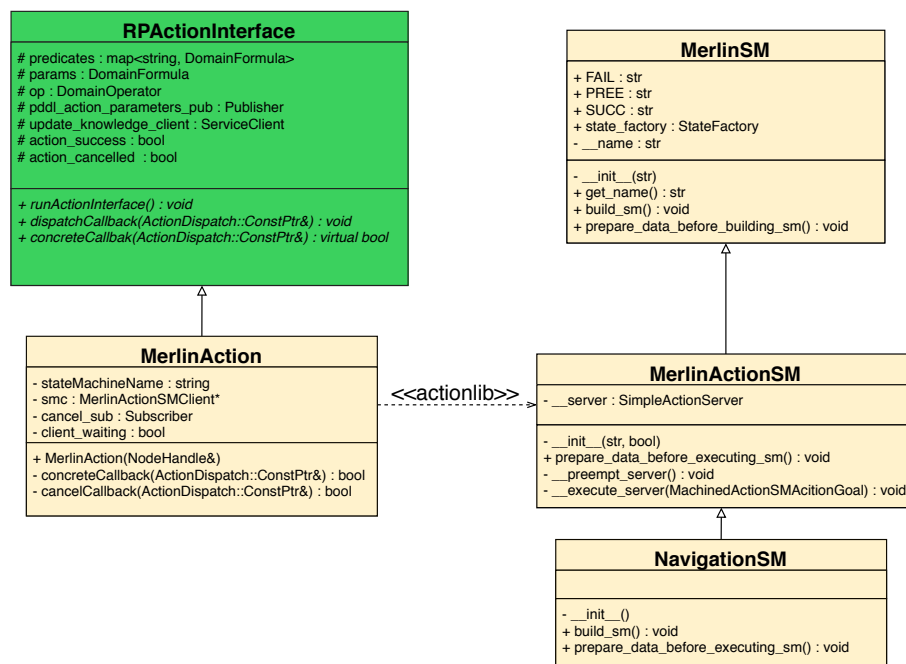


**Figura 4.3** – Diagrama de comunicación entre la *Mission Layer*, implementada como una máquina de estados, y el *Executor* de la *Planning Layer*. La máquina de estados de la *Mission Layer* envía los objetos a la *Planning Layer* utilizando una acción (*actionlib*) de ROS.

### 4.3.2. Planning Layer

Esta capa es responsable de crear los planes, que son las secuencias de acciones necesarias para alcanzar ciertos objetivos. Además, gestiona la ejecución

de estas acciones. Existen dos componentes principales. El primero es ROSPlan, que gestiona la base de conocimientos del robot y proporciona el motor para la gestión de la planificación. El segundo es el *Executor*, que es un nodo de ROS implementado como una máquina de estados. Tiene un estado para ejecutar cada componente de ROSPlan utilizando servicios de ROS. Además, el *Executor* elimina los objetivos utilizados en cada misión y replanifica si ROSPlan falla. Utiliza un servidor de acción (*actionlib*) de ROS que recibe los objetivos, los agrega a la base de conocimientos y comienza ROSPlan.



**Figura 4.4** – Diagrama de clases de la *Executive Layer*. La clase *MerlinActionSM* representa la clase base para crear nuevas acciones como máquinas de estados. La clase *MerlinAction* se encarga de comunicar ROSPlan con las acciones de MERLIN a través de una acción (*actionlib*) de ROS.

## 4.4. Sistema de comportamientos

El sistema de comportamientos proporciona el motor de control para gestionar los comportamientos del robot en las acciones. Está compuesto por una Capa Ejecutiva y una Capa Reactiva.



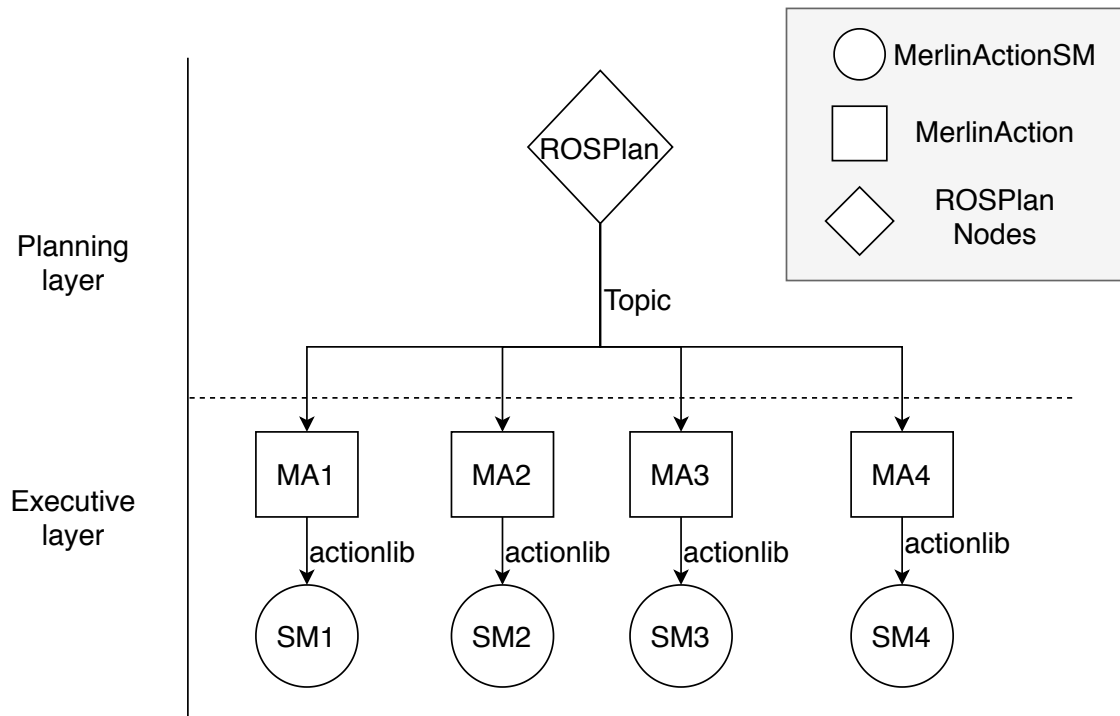
#### 4.4.1. Executive Layer

Esta capa está compuesta por las acciones que se utilizan para crear los planes. Han sido implementadas como una máquina de estados utilizando SMACH. El progreso y el estado de la acción pueden ser monitorizados gracias al SMACH Viewer [69], una herramienta para visualizar las máquinas de estados creadas con SMACH. Esta capa proporciona una funcionalidad crucial para el sistema, ya que ofrece el mecanismo de integración de las máquinas de estados de SMACH con ROSPlan. Se han creado dos clases principales para lograr esto:

- ***MerlinActionSM***: esta es la clase base para crear nuevas acciones de MERLIN. Como se muestra en la Figura 4.4, *MerlinActionSM* hereda de *MerlinSM*, que es la clase de MERLIN para crear máquinas de estados. Estas acciones pueden ejecutarse utilizando una acción (*actionlib*) de ROS.
- ***MerlinAction***: esta clase es el puente entre ROSPlan y las acciones de MERLIN. Heredando de *RPActionInterface*, que es la clase que ROSPlan utiliza para crear nuevas acciones, esta clase puede comunicarse con ROSPlan. *RPActionInterface* es la clase en verde que se muestra en la Figura 4.4. Además, para comunicarse con las acciones de MERLIN, se utilizan acciones de ROS.

La Figura 4.5 muestra el flujo de comunicación entre ROSPlan y las acciones de MERLIN. Existen tantas instancias de *MerlinAction* como acciones de MERLIN. Estas instancias son responsables de ejecutar las acciones de MERLIN. Además, se comunican con ROSPlan utilizando un topic de ROS. Cuando ROSPlan quiere utilizar una acción, escribe el nombre de la acción en este topic, después, las instancias de *MerlinAction* leen el nombre y la instancia a cargo de esa acción debe ejecutarla a través de una acción (*actionlib*) de ROS.

El uso de acciones de ROS para controlar la ejecución de las acciones de MERLIN permite que estas se puede cancelar. Gracias a esto, cuando se cancela ROSPlan, las acciones de MERLIN también se pueden cancelar. En ROSPlan, por defecto, las acciones se ignoran cuando se cancela un plan. De hecho, las acciones de ROSPlan pueden leer el mensaje de cancelación; sin embargo, el manejo de

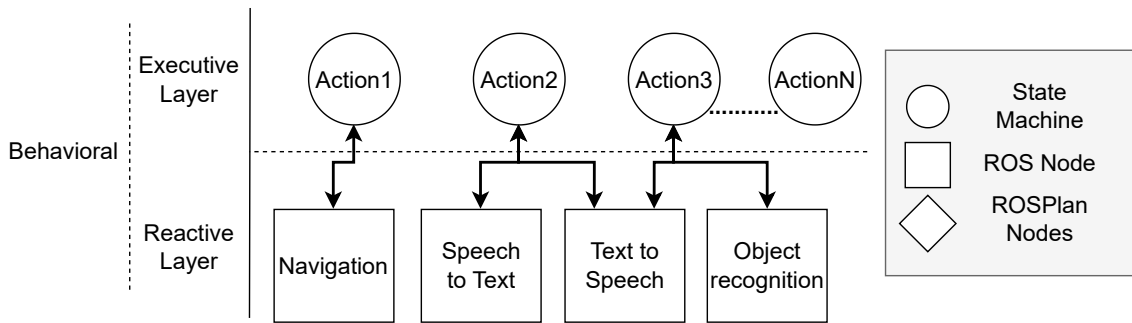


**Figura 4.5** – Diagrama de comunicación entre la *Executive Layer* y *ROSPlan* de la *Planning Layer*. *ROSPlan* envía en un topic el nombre de la siguiente acción que ejecutar. Cada instancia de *MerlinAction* lee ese nombre y ejecuta la máquina de estados asociada si el nombre de la acción coincide.

los mensajes que realizan las acciones de *ROSPlan* es secuencial, por lo que este mensaje se gestiona cuando las acciones se completan. Con esto se corrige una de las limitaciones que tenía *ROSPlan*.

#### 4.4.2. Reactive Layer

Esta capa está compuesta por las habilidades reactivas que el robot necesita para crear las acciones. Estos sistemas incluyen la navegación, el reconocimiento de voz, el reconocimiento de objetos y la síntesis de voz. Para que las máquinas de estados puedan utilizar estos sistemas, deben ser accesibles a través de servicios y acciones de ROS. Además, hay un estado para llamar a cada sistema. En la Figura 4.6 se muestra cómo las acciones pueden utilizar estos sistemas.



**Figura 4.6** – Diagrama de comunicación entre la *Executive Layer* y la *Reactive Layer*. Las acciones de la *Executive Layer*, implementadas como máquinas de estados, utilizan los elementos de la *Reactive Layer* a través de los diferentes tipos de comunicación de ROS.

## 4.5. Limitaciones

La arquitectura presentada consigue integrar los paradigmas deliberativos y reactivos junto a componentes emergentes como la detección de objetos. Sin embargo, tiene ciertas limitaciones, siendo la primera y más importante que MERLIN está basada en ROS y se tiene la nueva versión, ROS 2, que va a sustituirlo, por lo que ROS dejará de tener soporte en un futuro. De esta manera, las herramientas utilizadas, ROSPlan y SMACH, suponen una limitación para mantener y actualizar la arquitectura, ya que inicialmente no estaban disponibles en ROS 2. Además, otra limitación que se tiene es la forma en la que ROSPlan gestiona el conocimiento PDDL, que lo hace mediante una base de conocimiento en memoria en nodo de ROS. La creación, modificación y eliminación del conocimiento se hace a través de servicios de ROS, por lo que este proceso está altamente acoplado a ROS. Por estos motivos, se decidió crear una nueva versión de la arquitectura utilizando ROS 2. Además, se ha desarrollado una nueva librería llamada YASMIN para sustituir a SMACH y una herramienta que permita una mejor gestión del conocimiento PDDL, llamada KANT. La nueva versión de la arquitectura cognitiva, llamada MERLIN2, utiliza estas dos herramientas para generar, planificar, gestionar y monitorizar comportamientos en robots autónomos.



# Capítulo 5

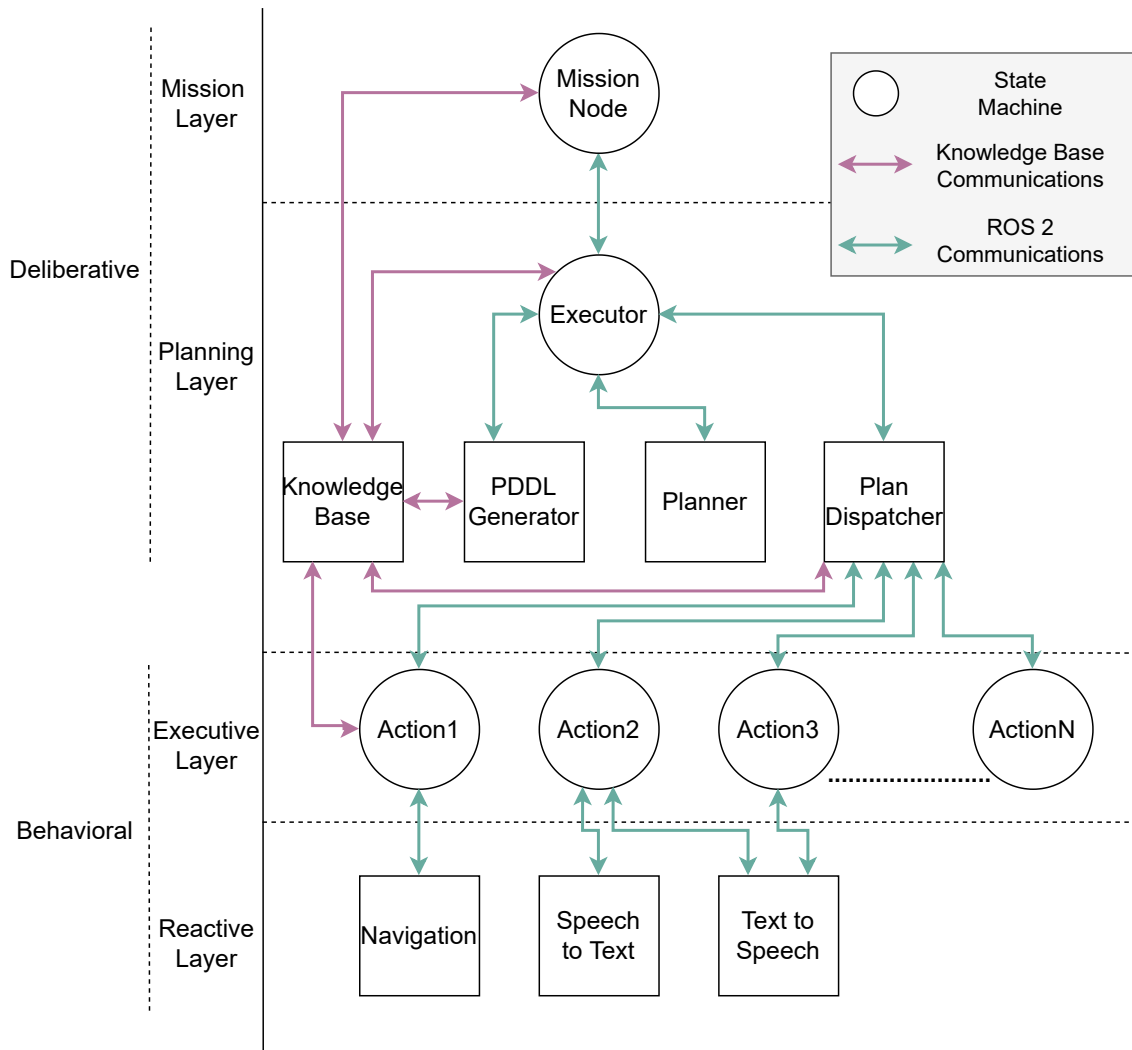
## MERLIN2

Como evolución de MERLIN, este capítulo presenta MERLIN2, una arquitectura cognitiva híbrida totalmente integrada en ROS 2 que utiliza dos nuevas herramientas, YASMIN y KANT. MERLIN2, que viene de *MachinEd Ros 2 pLanning*, es una arquitectura cognitiva híbrida para robots autónomos compuesta por cuatro capas [66]. Se basa en la idea de utilizar la planificación simbólica para establecer el comportamiento de un robot a largo plazo y componentes reactivos para llevar a cabo el comportamiento a corto plazo. Incluye las principales herramientas para generar comportamientos de robots: manipulación del conocimiento para la representación del modelo del mundo, un sistema de toma de decisiones deliberativas para llevar a cabo misiones y la generación de comportamiento para responder a entornos dinámicos.

### 5.1. Visión general

Las arquitecturas cognitivas en robótica se enfrentan al desarrollo y la organización de componentes de software para imitar la mente humana. Es un campo de investigación activo donde se pueden encontrar diferentes tecnologías y mecanismos de software. Algunos de ellos se han presentado anteriormente, como las máquinas de estado y los sistemas de planificación. La arquitectura cognitiva MERLIN2 contiene elementos que pueden ser beneficiosos para la investigación realizada en competiciones de robótica, tanto en entornos reales como en entornos simulados, incluyendo, pero no limitándose a, competiciones como la RoboCup o

la SciRoc. Permite la gestión de información simbólica a través del uso de KANT, la generación de máquinas de estados con YASMIN y un mecanismo para la integración de tareas de robot. De esta manera, MERLIN2 tiene como objetivo acelerar el proceso de producción de comportamientos en Fobots autónomos para hacer frente a las tareas diarias a corto y largo plazo de un robot basado en ROS 2. La figura 5.1 muestra la estructura de la arquitectura.



**Figura 5.1** – Diagrama de la arquitectura MERLIN2 formada por los sistemas *Deliberative System* y *Behavioral System* que se dividen en cuatro capas, que son *Mission Layer*, *Planning Layer*, *Executive Layer* y *Reactive Layer*.

Al igual que MERLIN, MERLIN2 está compuesto por un sistema deliberativo (*Deliberative System*), que se encarga de tomar decisiones y crear planes, y un

sistema de comportamientos (*Behavioral System*), que contiene las acciones y las habilidades que el robot puede realizar. Estos dos sistemas están formados por capas que se construyen una encima de otra. El sistema deliberativo se compone de la *Mission Layer* y la *Planning Layer* mientras que el sistema de comportamientos está compuesto por *Executive Layer* y la *Reactive Layer*.

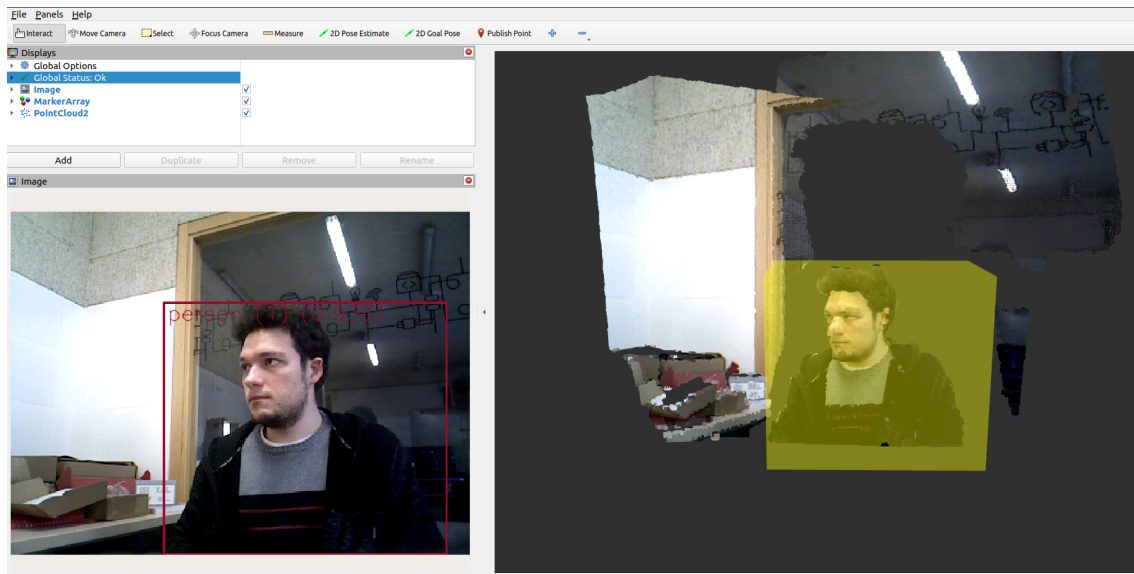
## 5.2. Núcleo de MERLIN2

El núcleo de MERLIN2 está formado por dos partes. La primera se corresponde con un nuevo sistema de planificación basado en la gestión de conocimiento de KANT. Este sistema utiliza KANT para gestionar el conocimiento PDDL para generar planes que cumplan las misiones del robot. Se ha diseñado para poder cambiar el planificador, pudiendo elegir entre varios planificadores PDDL. Por otro lado, la segunda parte del núcleo de MERLIN2 se corresponde con el uso de máquinas de estados, concretamente las de YASMIN. Estas máquinas de estados se utilizan en las diferentes capas de MERLIN2. Gracias a esto, se consigue monitorizar MERLIN2, a alto nivel, utilizando el visualizador de YASMIN.

### 5.2.1. Percepción

La percepción es el proceso de transformar los datos sensoriales, principalmente las imágenes de las cámaras, en una representación interna para realizar diferentes tareas. En el caso de la percepción visual, se tienen diferentes técnicas que utilizan las imágenes, siendo la más utilizada el reconocimiento de objetos, que proporciona información de los objetos que se encuentran en el entorno.

En visión por computador, existen diversas arquitecturas disponibles para la detección de objetos [70]. Se abordan desde tres puntos de vista: las redes neuronales, como AlexNet [71] o ResNet [72]; detectores basados en anclajes de dos etapas, como Faster R-CNN [73], o anclajes de una sola etapa, como YOLO [74]; y detectores basados en transformaciones, como DETR [75]. Sin embargo, YOLOv8 [54] se ha convertido en una referencia líder, ya que el software desarrollado por Ultralytics permite una detección, clasificación y segmentación rápida, precisa y sencilla. Así, se ha integrado en ROS 2 dando lugar a `yolov8_ros` [76].



**Figura 5.2** – Ejemplo del uso yolov8\_ros visualizado en RViz. En la parte izquierda se muestra el reconocimiento de objetos de YOLOv8, la *bounding box*, y el seguimiento, el identificador del objeto. En la parte derecha se muestra la *bounding box* 3D obtenida del reconocimiento de objetos y la imagen de profundidad.

La función de yolov8\_ros dentro de la arquitectura es obtener datos del entorno del robot a partir de los cuales se puede obtener conocimiento simbólico PDDL posteriormente. Esta herramienta utiliza YOLOv8 sobre las imágenes obtenidas del topic de una cámara RGB-D. Después, se aplica el algoritmo de seguimiento ByteTrack [77] para seguir los objetos detectados. Por último, la herramienta también utiliza las imágenes de profundidad de la cámara RGB-D para obtener una aproximación del tamaño y la posición de cada objeto en el entorno del robot.

En la Figura 5.2 se presenta un ejemplo. A la izquierda se muestra el resultado de YOLOv8 después de aplicar el algoritmo de seguimiento, es decir, la *bounding box*, el área rectangular que rodea al objeto detectado, y el ID del seguimiento. A la derecha se muestra la posición y tamaños estimados con la imagen de profundidad. Se visualiza como una *bounding box* sobre la nube de puntos que genera la cámara RGB-D. Estos datos se pueden utilizar en MERLIN2 como se muestra en [78], en el que se utiliza sistema de detección de objetos en la *Reactive Layer*.



### 5.2.2. Selección de acciones

La selección de acciones a alto nivel se realiza mediante planificación, específicamente empleando planificadores PDDL. Estos planificadores utilizan el conocimiento simbólico en PDDL para generar planes, secuencias de acciones a alto nivel, que satisfagan los objetivos del robot. Por otro lado, se utilizan máquinas de estados para llevar a cabo la selección de acciones a nivel reactivo.

#### YASMIN

Un sistema de toma de decisiones debe impulsar eficazmente el proceso de generación de comportamientos al decidir qué acciones o comandos deben activarse o ejecutarse selectivamente en robots autónomos. Un enfoque ampliamente extendido es el uso de máquinas de estados para diseñar estos sistemas. Una máquina de estados es un modelo matemático utilizado para organizar un proceso basado en transiciones entre estados. De esta manera, las máquinas de estados se han utilizado ampliamente en robótica para definir los comportamientos como transiciones desde un estado, que contiene un conjunto de operaciones, hasta otro estado, con otro conjunto de operaciones. Estas operaciones pueden ser de alto nivel, como navegación o síntesis de voz; u operaciones de bajo nivel, como el control de los motores del robot. De la misma manera, los sistemas basados en estados pueden estar compuestos por varias máquinas de estados que cambiarán de un estado a otro ejecutando las tareas del robot. Estos sistemas pueden volverse realmente complejos dependiendo de las transiciones entre los estados y el mecanismo para seleccionar y ejecutarlos.

En ROS existen librerías populares como SMACH [67] o SMACC [79] para diseñar comportamientos robóticos basados en máquinas de estados. Al comienzo de esta investigación, SMACH no se había migrado a ROS 2, por lo que se desarrolló YASMIN (*Yet Another State MachINe*) [55], que se originó durante el trabajo de investigación desarrollado para el Trabajo Final de Máster [80], como alternativa, junto con una herramienta de depuración visual para supervisar su ejecución. Es una librería de código abierto para Python y C++ diseñada para ROS 2. Está destinada a diseñar comportamientos de robots utilizando máquinas de estados.

Todos los estados y cualquier composición de ellos compartirán un *blackboard* para compartir datos. Además, es necesario conocer las transiciones y estados que componen una máquina de estados.

## Diseño

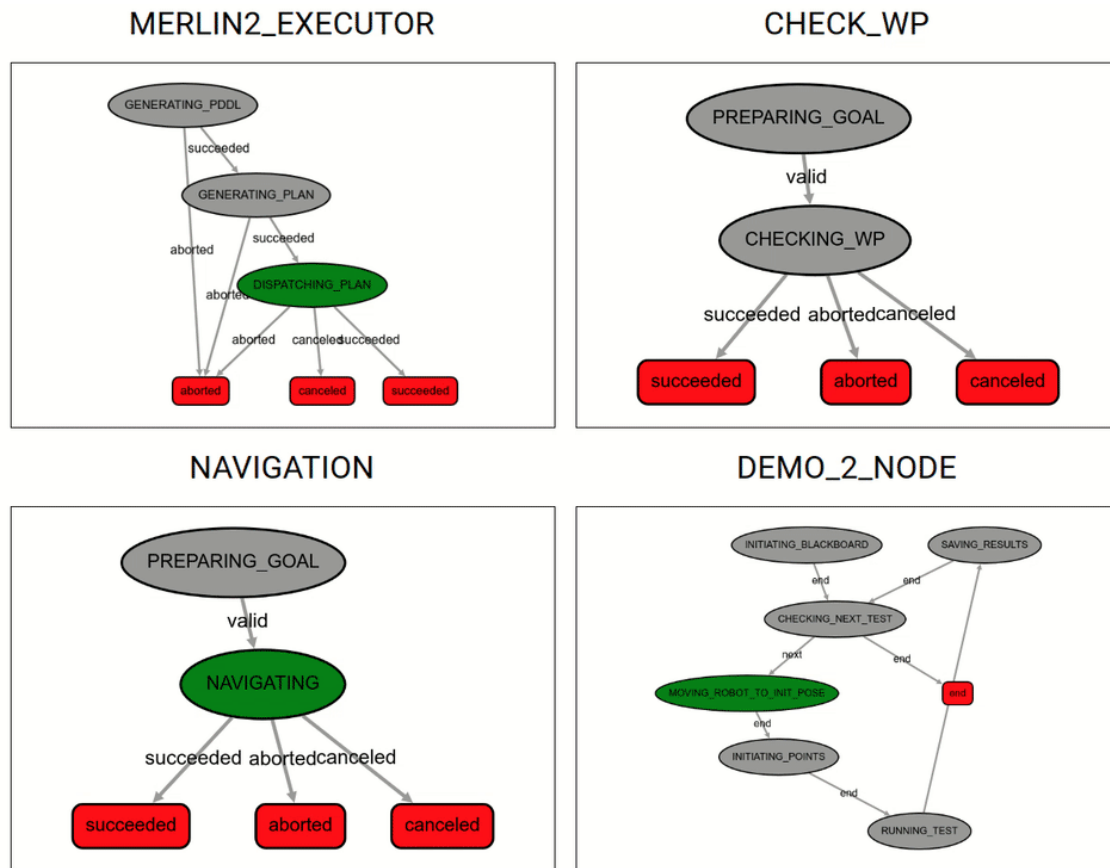
YASMIN es una biblioteca escrita en Python y C++ que permite desarrollar comportamientos mediante la creación de máquinas de estados. Además, YASMIN se puede utilizar para implementar el patrón de estado. Este patrón de diseño de comportamiento permite que un objeto modifique su comportamiento cuando su estado cambia. Así pues, se tienen los siguientes paquetes de ROS 2:

- Paquete **yasmin**: este paquete es la base de de YASMIN y contiene las siguientes clases:
  - **Blackboard**: esta clase se basa en el patrón de *blackboard*. Este patrón de diseño de software incluye el uso de una estructura de memoria global que contiene datos de diferentes fuentes de conocimiento. Por lo tanto, los estados de YASMIN son las fuentes de conocimiento que utilizan el *blackboard* para almacenar y compartir datos. Por otro lado, las máquinas de estados serán los componentes de control que deciden qué fuentes de conocimiento se ejecutan.
  - **State**: esta clase es la base para desarrollar nuevos estados de YASMIN. Para crear un nuevo estado, se hereda de la clase *State* y se implementa el método *execute*. Además, se puede redefinir el método *cancel\_state* que permite definir la forma en que se cancelará el estado. Finalmente, es necesario definir las salidas del estado que serán cadenas de texto utilizadas posteriormente en la definición de las transiciones de estados.
  - **CbState**: esta clase representa un tipo de estado que recibe una función *callback*. Por lo tanto, se puede crear un nuevo estado definiendo una función en lugar de una nueva clase.
  - **StateMachine**: esta clase representa las máquinas de estados de YASMIN. Utilizando la función *add\_state*, se pueden agregar nuevos estados. Esta función recibe una instancia de *State* y un diccionario que repre-

senta las transiciones de ese estado. Una transición consiste en vincular una de las salidas del estado con otro estado o una de las salidas finales de la máquina de estados. Finalmente, dado que *StateMachine* hereda de *State*, se pueden agregar máquinas de estados anidadas.

- Paquete **yasmin\_ros**: este paquete contiene clases de estados para facilitar la integración de YASMIN con ROS 2. Se tiene las siguientes clases:
  - **ActionState**: esta clase representa un estado que ejecuta un cliente de acción de ROS 2. Para usarlo, simplemente se elige el tipo de acción y se define la función *create\_goal*, que instancia el objetivo a enviar al servidor de la acción. Este estado ya está preparado para cancelar el cliente de acción si el estado se interrumpe.
  - **ServiceState**: esta clase representa un estado que ejecuta un cliente de servicio de ROS 2. Para usarlo, simplemente se elige el tipo de servicio y se define la función *create\_request*, donde se crea la instancia de la solicitud que se enviará al servicio.
  - **MonitorState**: esta clase representa un estado que se suscribe a un topic de ROS 2. Para usarlo, simplemente se elige el nombre y tipo de mensaje del topic y se define la función *monitor\_handler*, donde se puede tratar el mensaje.
- Paquete **yasmin\_viewer**: este paquete de ROS 2 contiene el código para visualizar las máquinas de estados de YASMIN. En la Figura 5.3 se muestra un ejemplo de este visualizador en el que se muestran cuatro máquinas de estados ejecutándose. Se tiene las siguientes clases:
  - **YasminViewerPub**: esta clase extrae la información de una máquina de estados de YASMIN y la publica en un topic de ROS 2.
  - **YasminViewerNode**: esta clase lee la información de la máquina de estados del topic y la presenta en el visualizador de máquinas de estados. Este visualizador se basa en una aplicación web formada por un *backend*, creado en Python con Flask; y un *frontend*, creado con React.js. Para mostrar las máquinas de estados, se utilizó Cytoscape.js, que es

una biblioteca de JavaScript para dibujar gráficos. De esta manera, las máquinas de estados se pueden mostrar desde cualquier ordenador utilizando un navegador web.



**Figura 5.3** – Ejemplo del visualizador de YASMIN. La máquina de estados de la parte superior izquierda corresponde con el *Executor* de MERLIN2. La máquina de estados de la parte inferior izquierda se corresponde con la acción de navegación del robot. La máquina de estados de la parte superior derecha se corresponde con la acción de comprobar un punto del mapa. La de la parte inferior derecha se corresponde con la máquina de estados que implementa la *Mission Layer*, encargada de generar los objetivos del robot.

- Paquete `yasmin_msgs`: este paquete contiene los mensajes de ROS 2 interfaces creados para YASMIN. Estos mensajes se utilizan para publicar la información de las máquinas de estados. Se tienen los siguientes mensajes:
  - *Transition.msg*: este mensaje representa las transiciones de los estados.

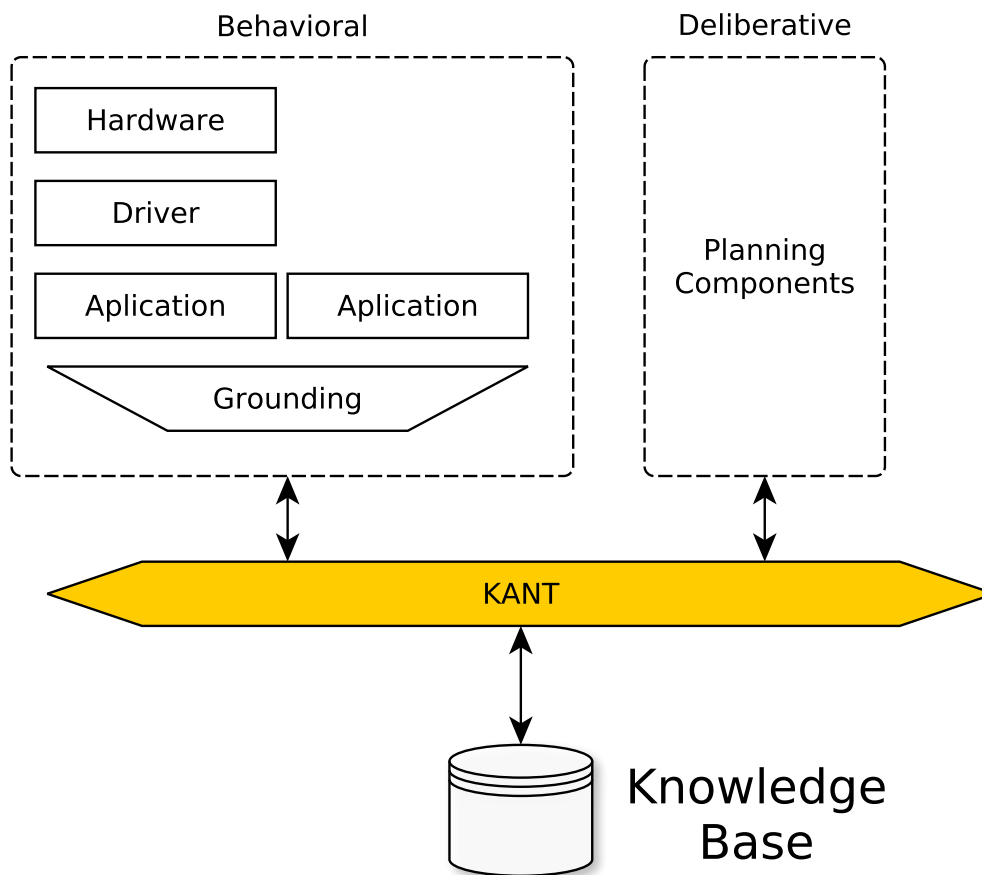
Cada estado tiene un nombre y, cuando se ejecuta, produce una salida. Por lo tanto, este mensaje contiene dos cadenas de texto, una para el nombre de la salida y otra para el nombre del estado.

- ***State.msg***: este mensaje representa un estado. Contiene la información del estado: su nombre, una lista de transiciones (*Transition.msg*) y sus salidas. Además, contiene un booleano para indicar si es una máquina de estados, lo que permite enviar información sobre máquinas de estados anidadas. Si es una máquina de estados, también contiene el nombre de su estado actual. Finalmente, a cada estado se le asigna un ID y el ID del padre.
- ***StateMachine.msg***: este mensaje representa una máquina de estados. Contiene una lista de estados (*State.msg*). El primer elemento de la lista identifica la propia máquina de estados. Los elementos siguientes corresponden a los estados y máquinas de estados anidadas, haciendo referencia al primer elemento como su estado padre.

### 5.2.3. Memoria

Los sistemas de planificación simbólica son un proceso de deliberación abstracto y explícito que elige y organiza acciones [81] para cambiar el estado del robot o del entorno. Para lograrlo, utilizan una representación simbólica de las capacidades del robot proporcionadas por los diseñadores y una representación simbólica de su conocimiento sobre el entorno obtenido mediante los componentes de percepción. La forma estándar de representar todo este conocimiento es mediante el uso de un lenguaje de alto nivel como PDDL [12].

Las definiciones de dominio y problema PDDL del robot deben mantenerse y actualizarse con el tiempo. No existe una forma estándar de gestionar y almacenar el conocimiento PDDL. Como resultado, existen varios métodos diferentes. De esta manera, el objetivo de KANT (*Knowledge mAnagement*) [56], que se originó durante el trabajo de investigación desarrollado para el Trabajo Final de Máster [80], es resolver este problema simplificando el proceso de interacción con la base de conocimiento. En particular, facilita el uso de PDDL desde lenguajes de



**Figura 5.4** – KANT en una arquitectura genérica. Se corresponde con el componente amarillo que hace de interfaz entre los componentes de la arquitectura, tanto del sistema deliberativo como del sistema de comportamientos, y la base de conocimiento.

programación como Python, lo que facilita su uso en robots reales. Para ello, KANT encapsula los elementos del PDDL en clases de Python siguiendo los patrones de diseño software de DTO (*Data Transfer Object*), DAO (*Data Access Object*), fábrica abstracta y método fábrica.

KANT se representa como elemento amarillo en la Figura 5.4. Cualquier componente del robot puede usarlo para acceder a la Base de Conocimiento. Como resultado, cualquier componente puede consultar y modificar el conocimiento de la misma manera y sin preocuparse por cómo se almacena. Además, actualizar el conocimiento de un robot a partir de datos de sensores es más sencillo gracias a él.

El patrón DTO [82] se utiliza en KANT para crear las clases que encapsulan

la información PDDL que posteriormente se comparte entre los componentes software. El DTO se basa en la creación de atributos privados que representan los datos. Estos atributos deben ser accesibles y editables con las funciones *getter* y *setter*. Por otro lado, el DAO [83] tiene como objetivo utilizar un componente de software para abstraer y encapsular todo el acceso a la fuente de datos. El DAO se encarga de la conexión a la fuente de datos para obtener y almacenar datos, por lo que debe encapsular la lógica para recuperar, guardar y actualizar datos en su almacenamiento de datos. En este caso, la fuente de datos será la base de conocimiento donde se almacena el PDDL.

### Diseño

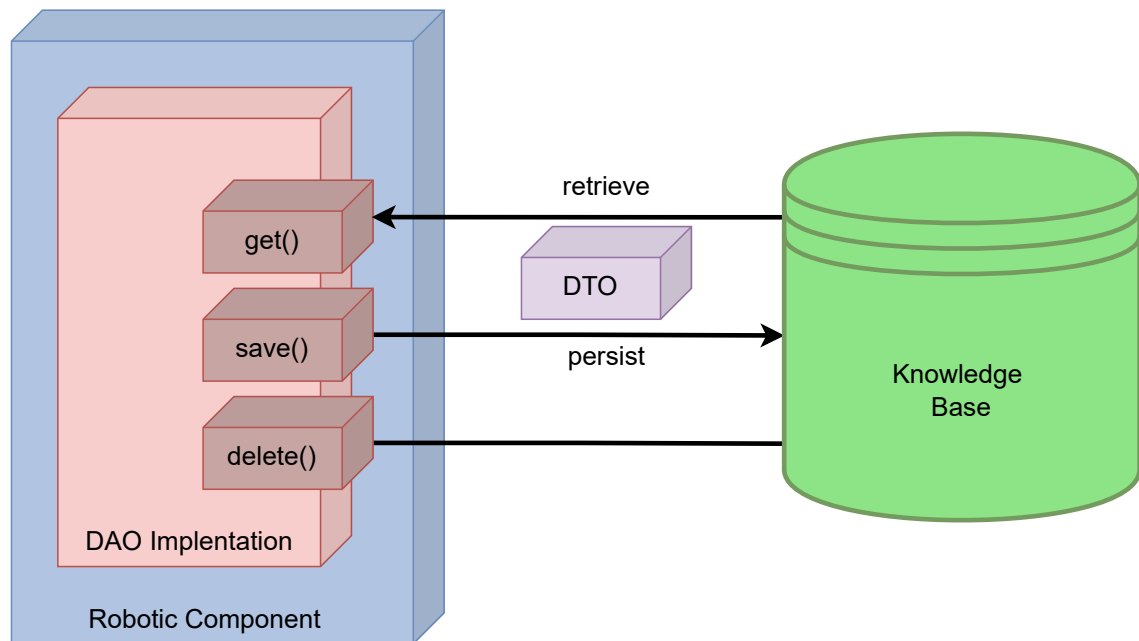
KANT es una herramienta integrada en ROS 2. Por este motivo, su código está organizado en los siguientes paquetes de ROS 2.

- Paquete **kant\_dto**: este paquete contiene las clases DTO de KANT. Los elementos PDDL se han encapsulado en elementos DTO. Siguiendo esto, se ha desarrollado una clase DTO para cada elemento PDDL. La estructura de cada elemento PDDL modelado con el patrón DTO es la siguiente:
  - **Tipos**: un tipo está compuesto por un único atributo *string*, que representa su nombre.
  - **Objetos**: un objeto está compuesto por un atributo de tipo DTO, que representa su tipo PDDL; y un atributo *string*, que representa su nombre.
  - **Predicados**: un predicado está compuesto por un atributo *string*, que representa su nombre; y un atributo de lista de tipo DTO, que representa sus argumentos.
  - **Proposiciones**: una proposición está compuesta por un atributo *string*, que representa su nombre; y un atributo de lista de objetos DTO, que representa sus objetos PDDL. También tiene un atributo booleano, que representa si es un objetivo.
  - **Acciones**: una acción está compuesta por los siguientes atributos:
    - Un atributo *string* que representa su nombre.

- Un atributo booleano que representa si es una acción durativa.
  - Un atributo entero, que representa su duración.
  - Un atributo de lista de objetos DTO que representa sus parámetros.
  - Un atributo de lista de DTO, que representa sus condiciones. El DTO de las condiciones es similar al de las proposiciones, pero también tiene un atributo *string*, que representa el momento en que debe ocurrir la condición; y un atributo booleano, que representa si es negativo.
  - Un atributo de lista DTO similar al de las condiciones.
- Paquete **kant\_dao**: este paquete contiene las instancias de DAO, que se pueden utilizar para interactuar con el conocimiento PDDL en la base de conocimiento. Al combinar los patrones DTO y DAO, cualquier componente software puede consultar, guardar, editar y eliminar datos de la base de conocimiento sin tener que preocuparse por el tipo y la implementación de la fuente de datos. De esta forma, la arquitectura resultante se presenta en la Figura 5.5. Se pueden utilizar tres funciones principales en los elementos DAO:

- **get**: esta función devuelve un DTO de la base de conocimiento. Proporciona acceso al PDDL actual que el robot tiene. En el caso de los tipos, objetos, predicados y acciones de PDDL, esta función toma sus nombres para realizar las búsquedas. Sin embargo, en el caso de las proposiciones PDDL, existen tres funciones *get*: *get\_by\_predicate*, para buscar una lista de proposiciones con un nombre de predicado dado; *get\_goals*, para buscar las proposiciones que son objetivos; y *get\_no\_goals*, para buscar las proposiciones que no son objetivos.
- **save**: esta función se implementa en todos los DAO. Se utiliza para guardar un DTO. Si ese DTO ya existe en la base de conocimiento, se actualiza.
- **delete**: esta función también se implementa en todos los DAO y se





**Figura 5.5** – Arquitectura software de KANT. Los elementos DTO son las encapsulaciones del conocimiento simbólico en PDDL. Se utilizan en los elementos DAO que hacen de interfaz entre la base de conocimiento y cualquier componente software. Los DAO puede consultar, modificar y eliminar el conocimiento PDDL utilizando los DTO.

utiliza para eliminar un DTO de la base de conocimiento.

En KANT existen dos tipos de bases de conocimiento: una base de datos almacenada en MongoDB [84] y una base de datos almacena en memoria en un nodo de ROS 2. De esta manera, se tienen dos familias de DAO:

- **MONGODB:** esta familia DAO usa MongoDB para almacenar el conocimiento, lo que significa que MongoDB es la base de conocimiento. Como resultado, se emplea MongoDB Compass [85] para visualizar el conocimiento actual.
- **ROS2:** esta familia DAO se basa en almacenar el conocimiento en un nodo ROS 2. Este nodo utiliza tres servicios de ROS 2 para cada elemento PDDL, uno para hacer consultas, otro para guardar y editar elementos PDDL, y otro para eliminar elementos PDDL.

Para gestionar la creación de instancias DAO se han usado los patrones

de fábrica abstracta y de método fábrica. Por un lado, la creación de las instancias DAO se gestiona utilizando el patrón de fábrica abstracta. Como resultado, cada familia DAO tiene una clase fábrica para crear sus instancias DAO. Las fábricas de cada familia tienen las mismas funciones para crear las instancias DAO. Por otro lado, para gestionar la creación de las instancias de las fábricas de cada familia DAO se utiliza el patrón de método fábrica.

- Paquete **kant\_knowledge\_base**: este paquete contiene un nodo de ROS 2 que actúa como base de conocimiento. Para comunicarse con el resto de nodos de ROS 2, se han implementado tres servicios para cada elemento PDDL, teniendo en total quince servicios. El primer servicio se utiliza para hacer consultas a la base de conocimientos, el segundo se utiliza para guardar, editar o eliminar el conocimiento y el tercer se utiliza para eliminar todas las instancias de uno de los elementos de PDDL.
- Paquete **kant\_msgs**: en este apartado contiene los mensajes de ROS 2 utilizados en KANT. Se utilizan para comunicarse con el nodo de ROS 2 que almacena en memoria la base de conocimiento de KANT. Se tienen los siguientes mensajes:
  - **PddlType.msg**: este mensaje encapsula la información de un tipo PDDL.
  - **PddlObject.msg**: este mensaje encapsula la información de un objeto de PDDL.
  - **PddlPredicate.msg**: este mensaje encapsula la información, un predicado de PDDL.
  - **PddlProposition.msg**: este mensaje encapsula la información, una proposición PDDL.
  - **PddlAction.msg**: este mensaje encapsula la información, una acción PDDL.
  - **PddlConditionEffect.msg**: este mensaje encapsula la información, una condición o un efecto PDDL.
  - **UpdateKnowledge.msg**: este mensaje encapsula el tipo de actualiza-

ción que se quiere realizar sobre la base de conocimientos. Los tipos son SAVE, para guardar o actualizar el conocimiento, y DELETE, para eliminar el conocimiento.

- *GetPddlType.srv*: este servicio se utiliza para consultar los tipos PDDL.
- *UpdatePddlType.srv*: este servicio se utiliza para actualizar la información de los tipos PDDL.
- *GetPddlObject.srv*: este servicio se utiliza para consultar los objetos PDDL.
- *UpdatePddlObject.srv*: este servicio se utiliza para actualizar la información de los tipos PDDL.
- *GetPddlPredicate.srv*: este servicio se utiliza para consultar los predicados PDDL.
- *UpdatePddlPredicate.srv*: este servicio se utiliza para actualizar la información de los predicados PDDL.
- *GetPddlProposition.srv*: este servicio se utiliza para consultar las proposiciones PDDL.
- *UpdatePddlProposition.srv*: este servicio se utiliza para actualizar la información de las proposiciones PDDL.
- *GetPddlAction.srv*: este servicio se utiliza para consultar las acciones PDDL.
- *UpdatePddlAction.srv*: este servicio se utiliza para actualizar la información de las acciones PDDL.

#### 5.2.4. Aprendizaje

El aprendizaje que se puede llevar a cabo en la arquitectura se corresponde con la generación de nuevo conocimiento simbólico a partir de la percepción. De esta manera, se puede crear nuevos objetos PDDL a partir los objetos detectados.

### 5.2.5. Razonamiento

El uso de *Large Language Models* (LLMs) en robótica puede aportar varios beneficios y capacidades. Por ejemplo, se pueden utilizar para mejorar la interacción en lenguaje natural [86, 87, 88]. Los LLMs permiten que los robots comprendan y generen el lenguaje humano, lo que facilita que los usuarios se comuniquen con los robots y los controlen mediante comandos o consultas en lenguaje natural. También simplifican la recuperación de conocimiento [89, 90, 91]. Además, se proponen como herramientas de resolución de problemas [92, 93, 94, 95], donde los LLMs pueden ayudar a resolver problemas complejos accediendo a recursos textuales para obtener información. Por último, también se pueden utilizar en la interpretabilidad [96], aprovechando las capacidades narrativas de los modelos para explicar e interpretar los logs generados por los robots.

Recientemente, los LLMs han logrado un alto rendimiento en tareas que implican razonamiento aritmético y simbólico utilizando el método del "chain-of-thought" (CoT) [97], que se basa en generar planes y razonamientos divididos en múltiples pasos. Estudios como [98] recientemente lograron un rendimiento significativo en tales tareas utilizando LLMs capaces de razonamiento.

Por este motivo, se propone la integración de LLMs en la arquitectura cognitiva desarrollada en este trabajo de investigación. Para ello, se ha desarrollado la herramienta llama\_ros [99], a una herramienta basada en ROS 2 para acceder a la inferencia de los LLMs desde aplicaciones de ROS 2. Esta nueva herramienta se basa en llama.cpp [100], una librería en C++ que permite ejecutar LLMs cuantificados y con soporte para GPU. Como resultado se tiene que se pueden ejecutar LLMs con los recursos de los que suelen disponer las plataformas robótica con una alta velocidad de inferencia.

### LLMs en la actualidad

Un LLM es un sofisticado sistema de inteligencia artificial entrenado con vastas cantidades de datos de texto para comprender y generar lenguaje humano. Se han experimentado avances recientes en este campo con el lanzamiento de chatGPT [101] (GPT-3.5 y GPT-4). Sin embargo, el lanzamiento de LLaMA [102] y

LLaMA2 [103], en sus diferentes tamaños, 7B, 13B, 33B y 65B; ha marcado una nueva era de LLMs, ya que permite a los investigadores entrenar más modelos en conjuntos de datos personalizados.

No obstante, los LLMs requieren recursos computacionales elevados para la inferencia. Gestionar la carga computacional es un desafío importante en entornos con recursos limitados, como los sistemas robóticos. La cuantificación [104, 105, 106, 107], en este contexto, se refiere al proceso de reducir la precisión de los parámetros y activaciones del modelo, generalmente de números de coma flotante a números enteros. Al hacerlo, se reduce drásticamente los requisitos de memoria y cómputo de que los LLMs necesitan para llevar a cabo el proceso de inferencia. Esto hace factible el uso de los LLMs en robots con limitaciones de potencia de procesamiento y memoria. De esta manera, se pueden aprovechar las ventajas de los LLMs para la comprensión del lenguaje natural, la toma de decisiones y la interacción mientras operan de manera eficiente dentro de sus entornos de hardware limitados.

Los modelos LLaMA y los métodos de cuantificación han hecho posible la proliferación de una gran cantidad de LLMs que se pueden ejecutar en ordenadores personales y sistemas embebidos con herramientas como llama.cpp [100]. El panorama de los LLMs continúa evolucionando rápidamente con la introducción de nuevos modelos como Alpaca [108], Vicuna [109], Airoboros [110], WizardLM [111], Nous-Hermes [112], Marcoroni [113] y Mistral [114].

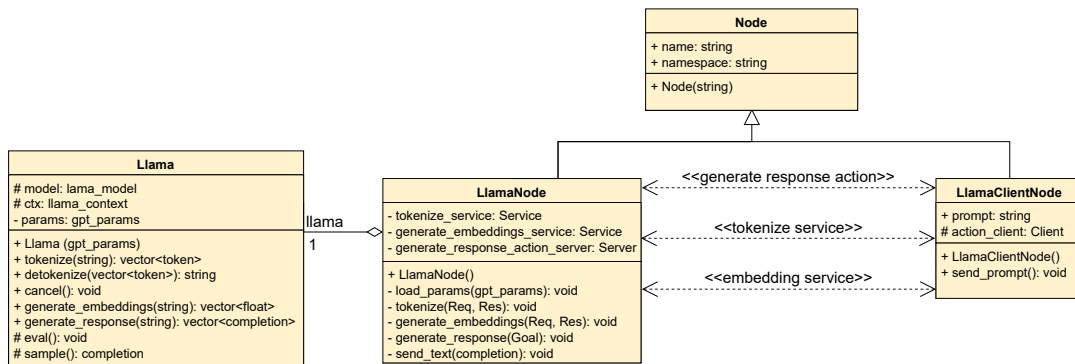
### **Capacidades deliberativas y de razonamiento de los LLMs**

A medida que los LLMs se han vuelto más sofisticados, sus capacidades en una deliberación, de planificación y razonamiento se han convertido en un tema de investigación. Estas capacidades implican considerar cuidadosamente diversas opciones y perspectivas antes de tomar decisiones y organizar estratégicamente las acciones necesarias para lograr un objetivo específico. Algunos trabajos han intentado utilizar LLMs como planificadores. Por ejemplo, [115] utiliza PDDL para la planificación, mientras que [116] explora el uso de la planificación basada en few-shot, es decir, empleando ejemplos en los *prompts* para conseguir que los

modelos se adapten a las nuevas tareas descritas por esos ejemplos.

A pesar de que los modelos preentrenados son ampliamente reconocidos por sus notables habilidades de aprendizaje de *few-shot* en diversas tareas de procesamiento del lenguaje natural, una técnica de *prompt* reciente *chain-of-thought* (CoT) [97] ha logrado mejores rendimientos. En [98], se ha demostrado que los LLMs también pueden destacar como razonadores de *zero-shot*, es decir, sin incluir ejemplos en el *prompt*. Esta técnica se ha ampliado aplicando diferentes algoritmos de búsqueda. De esta forma, el *tree-of-thought* (TOT) [117] permite a los LLMs tomar decisiones considerando diferentes rutas de razonamiento, evaluándolas y decidiendo el próximo curso de acción. Otro caso es el *graph-of-thought* (GOT) [118], que es similar al caso anterior pero distribuye las posibles rutas en un grafo en lugar de un árbol.

En robótica, encontramos más trabajos que intentaron realizar planificación PDDL con LLMs preentrenados, como el caso de [119]. Investigaciones más avanzadas, como ProgPrompt [120], permiten la generación de planes a través de la generación de código que representa el plan para alcanzar los objetivos. Sin embargo, los LLMs rara vez se utilizan dentro de las arquitecturas cognitivas.



**Figura 5.6** – Diagrama de clases de llama\_ros. La clase Llama encapsula el funcionamiento de llama.cpp mientras que la clase LlamaNode es un nodo de ROS 2 que expone ciertas comunicaciones de ROS 2 para poder utilizar las funcionalidades de llama.cpp. Así, se tiene un servicio para tokenizar un texto, un servicio para generar *embeddings* a partir de un text y una acción para generar respuestas a partir de un *prompt*.

## llama\_ros

La herramienta llama\_ros [99] es un conjunto de paquetes ROS 2 que integra llama.cpp [100] en ROS 2. Proporciona mensajes de ROS 2 para generar texto con LLMs, tokenizar el texto y crear *embeddings* o vectores a partir del texto. La Figura 5.6 muestra el diagrama su clases. De esta forma, la clase *LlmaNode* ofrece una acción ROS 2 que permite la generación de texto. Además, se tienen dos servicios, uno para tokenizar el texto y otro para producir vectores a partir del texto. Estos servicios pueden ser útiles cuando se deseen aplicar técnicas de ingeniería de *prompts* más complejos [121, 122], por ejemplo, el servicio de *embeddings* se puede utilizar para convertir el texto en vectores y llevar a cabo *Retrieval Augmented Generation* (RAG) [123].

### 5.2.6. Explicabilidad

La explicabilidad de esta arquitectura se basa en el uso de logs de ROS 2; el visualizador de YASMIN, mediante el cual se puede conocer el estado de todas las máquinas de estados de YASMIN; y en el uso de Compass para visualizar el conocimiento simbólico PDDL almacenado en MongoDB.

## 5.3. Sistema deliberativo

### 5.3.1. Mission Layer

Esta capa se encarga de generar los objetivos de alto nivel de los robots. El componente de software en esta capa establece los objetivos del robot en un momento dado. Este objetivo parte de un estado inicial compuesto por los hechos del entorno actual y establece los hechos que el robot desea que sean ciertos. Se puede implementar como una máquina de estados cuyos estados generan los objetivos.

### 5.3.2. Planning Layer

Esta capa se encarga de generar planes para lograr los objetivos de alto nivel creados previamente. Está asociada con el clásico subsistema deliberativo y requiere una representación formal del entorno, planificadores, y acciones [4].

MERLIN2 propone un conjunto de cinco componentes para esta capa:

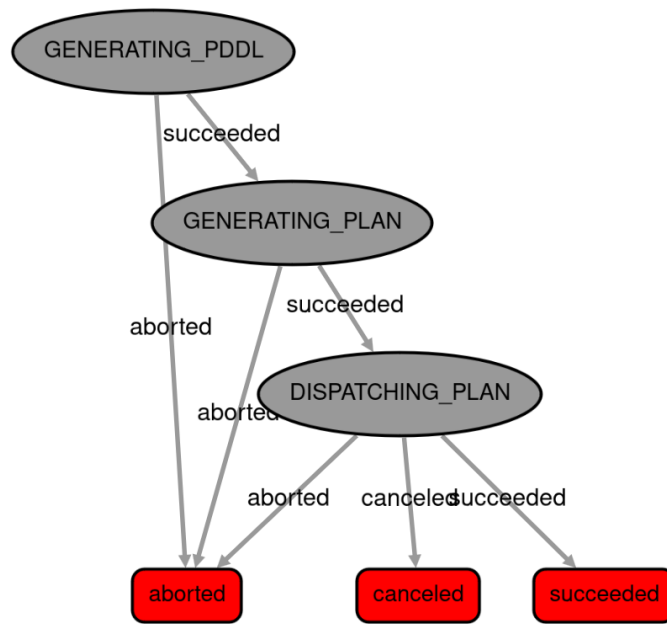
- **Knowledge Base:** este componente es la base de conocimientos de KANT, que corresponde a la memoria a largo plazo. Almacena el conocimiento del entorno utilizando PDDL. Hay dos tipos de bases de conocimientos según el tipo de almacenamiento: un nodo ROS 2 que almacena el conocimiento en la memoria del proceso y una base de datos MongoDB [84], que es más persistente.
- **PDDL Generator:** este componente se encarga de generar los archivos PDDL a partir de la base de conocimientos.
- **Planner:** este componente utiliza los archivos PDDL para crear planes. Se pueden utilizar varios planificadores PDDL. Los planificadores disponibles son POPF [13], SMTP [124], Unified-Planning [125] y VHPOP [126].
- **Plan Dispatcher:** este componente se encarga de ejecutar las acciones del plan generado. También debe cancelar la acción actual si se cancela la misión.
- **Executor:** este componente es una interfaz entre esta capa y la *Mission Layer*. Ejecuta y cancela los componentes de esta capa.

### Integración de LLMs en MERLIN2

La integración de *Large Language Models* (LLMs) en arquitecturas cognitivas se ha presentado en [127]. Para ello, se modifica la *Planning Layer*, reemplazando sus tres componentes por uno solo que utiliza llama\_ros y langchain [128], una librería de Python para simplificar la creación de aplicaciones con LLMs. De esta manera, los LLMs y llama\_ros reemplazan al planificador de la arquitectura. Además, la base de conocimientos PDDL también se reemplaza por un grafo de conocimientos. La capa original estaba controlada por una máquina de estados implementada con YASMIN, presentada en la Figura 5.7. La nueva seguiría estando controlada por una máquina de estados, que se muestra en la Figura 5.8. Esta se compone de dos máquinas de estados anidadas, una para llevar a cabo la planificación y otra para comprobar si los objetivos se han cumplido; y un estado para ejecutar los planes.

La máquina de estados anidada encargada de la planificación se genera planes del robot. El primer estado consiste en generar el estado del mundo del robot, el



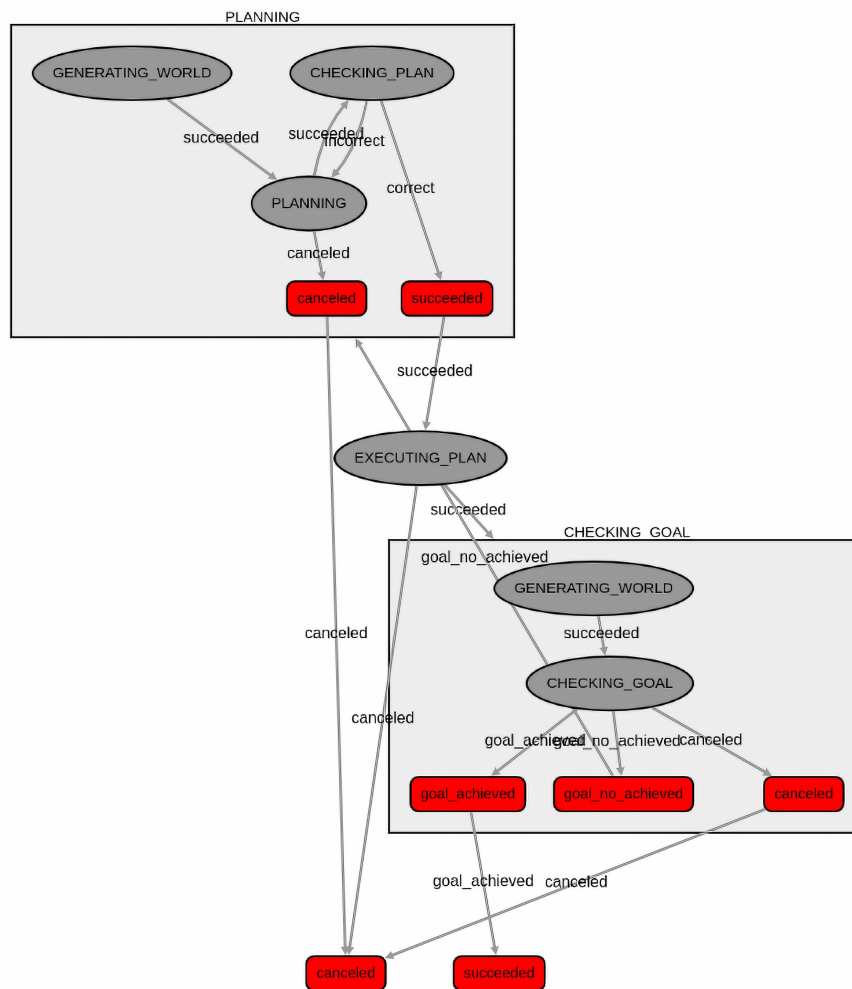


**Figura 5.7** – Máquina de estados de YASMIN encargada de controlar la *Planning Layer* original de MERLIN2. Tiene tres estados, uno para generar el PDDL a partir de la base de conocimientos, otro estado para generar los planes utilizando planificadores de PDDL y otro para estado ejecutar el plan.

cual se utiliza para construir el *prompt* empleado para pedirle al LLM que actúe como planificador. Este estado se corresponde con el estado *Generating PDDL* de la máquina de estados original de MERLIN2, que se muestra en la Figura 5.7. Después de eso, el siguiente estado utiliza *llama\_ros* para solicitar al LLM que genere un plan que resuelva un objetivo dado, lo cual se corresponde con el estado *Generating Plan* de la máquina de estados original. El *prompt* utilizado para generar un plan incluye las acciones del robot, el estado del mundo y el objetivo del robot. Para mejorar la generación de planes y reducir las alucinaciones de los LLMs se utiliza una BNF [129] que obliga al LLM a responder siguiendo el formato JSON, lo que facilita su posterior traducción a la hora ejecutar el plan en el siguiente estado. Por último, se tiene una máquina de estados anidada para comprobar si el plan es correcto, verificando si el objetivo se cumple en el estado del mundo.

El estado del mundo es una representación intermedia del conocimiento obtenida a partir del grafo de conocimiento, como el que se presenta en la Figura 5.9, después de convertir los datos del grafo en elementos de conocimiento. Es-

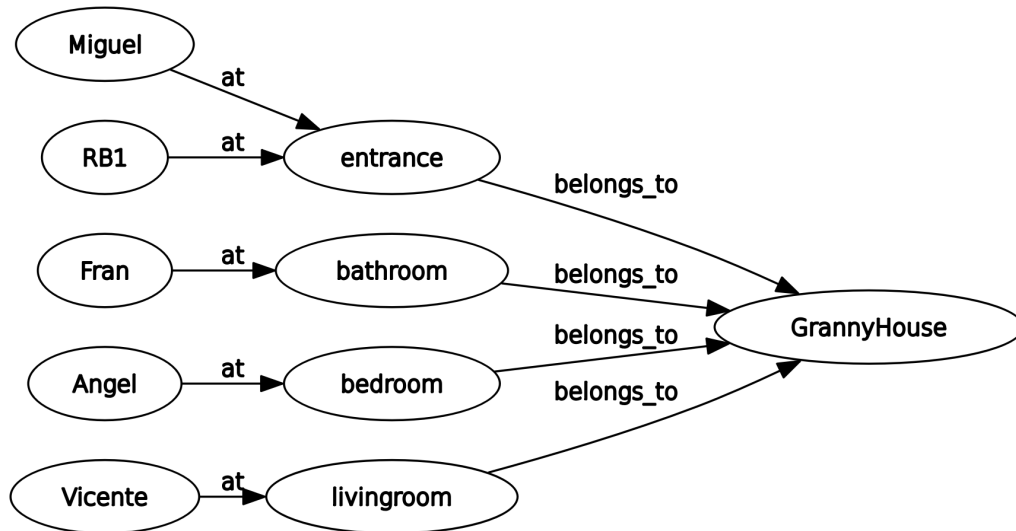
tos elementos de conocimiento están estructurados de la siguiente manera: para los nodos, tienen la forma de "nodo-tipo (propiedades)", mientras que para las aristas, siguen el patrón de "nodo-relación-nodo (propiedades)". El elemento "(propiedades)" incluye pares de clave-valor que abarcan todos los datos asociados a un nodo o una arista. Por ejemplo, puede incluir las coordenadas de objeto.



**Figura 5.8** – Máquina de estados de YASMIN encargada de controlar la nueva *Planning Layer* obtenida después de integrar los LLMs en MERLIN2. Se tiene una máquina de estados anidada para generar planes para los objetivos, un estado para ejecutar el plan y otra máquina de estados anidada para comprobar si el objetivo se cumple en el estado del mundo.

También se puede llevar a cabo la técnica de ingeniería de *prompts* llamada RAG. Para ello, se puede incluir el uso de una base de datos basada en vectores en

la generación del estado del mundo. Los elementos de conocimiento se almacenan como vectores. Luego, se pueden aplicar métodos de recuperación para obtener solo los elementos de conocimiento relacionados con el objetivo del robot. En este trabajo de investigación se utiliza ChromaDB [130], un base de datos basa en vectores que se almacena en memoria.



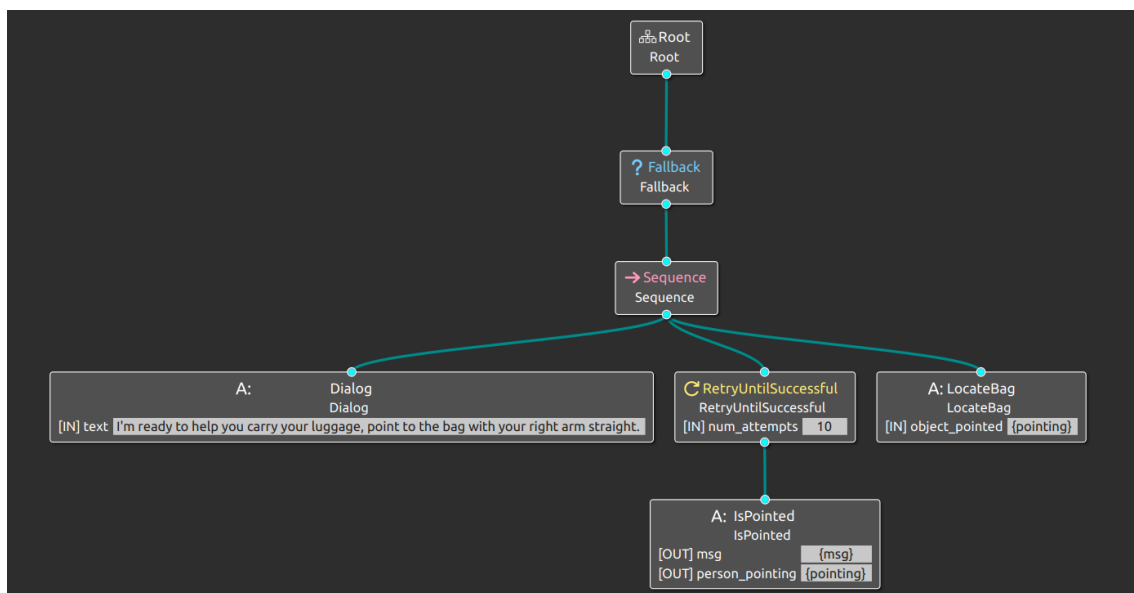
**Figura 5.9** – Ejemplo de grafo de conocimiento que representa un apartamento con cuatro habitaciones, entrada, baño, dormitorio y sala. En cada habitación hay una persona y el robot RB1 está en la entrada.

Una vez que se genera el plan, el estado de ejecución del plan, que corresponde al estado original de *Dispatching Plan*, ejecuta las acciones del plan con los argumentos especificados por el LLM. Después de eso, la máquina de estados anidada de comprobación de los objetivos genera nuevamente el estado del mundo y solicita al LLM, aprovechando sus capacidades de razonamiento, que verifique si se ha alcanzado el objetivo. El *prompt* para comprobar si se ha alcanzado el objetivo incluye el estado del mundo y el objetivo del robot. También se solicita al LLM una explicación de por qué el objetivo no se ha logrado. Si el LLM decide que el objetivo no se ha completado, la máquina de estados intentará nuevamente generar un plan, pero utilizará la explicación de por qué el objetivo no se ha alcanzado, mejorando la generación del plan.

## 5.4. Sistema de comportamientos

### 5.4.1. Executive Layer

Esta capa está compuesta por las acciones que el robot puede llevar a cabo. Cuando se crea la instancia de una acción, su conocimiento PDDL asociado se guarda en la base de conocimientos. Además, estas acciones se implementan como máquinas de estado YASMIN. Como se mencionó anteriormente, el *blackboard* de las máquinas de estado es la memoria a corto plazo, que almacena los objetos iniciales necesarios para ejecutar la acción PDDL y la solución parcial de la máquina de estado. Además, YASMIN tiene un visualizador que se puede utilizar para supervisar las acciones. Por otro lado, las acciones también se puede implementar como árboles de comportamiento [131], los cuales tienen un *blackboard* y un visualizador, Groot [132], parecidos a los de YASMIN.



**Figura 5.10** – Árbol de comportamientos de la acción para detectar bolsas visualizado en Groot. Este acción utiliza un nodo para hacer síntesis de voz, otro nodo para buscar la bolsa a la que está apuntado la persona y otro nodo para localizar la bolsa.

### 5.4.2. Reactive Layer

Esta capa proporciona el conjunto de componentes de software que representan las habilidades del robot. Estas habilidades son la navegación, basada en Navigation2 [133]; y habilidades para interactuar con personas, como la síntesis de voz y el reconocimiento de del habla.

## 5.5. Limitaciones

Aunque MERLIN2 soluciona parte de las limitaciones de MERLIN, se tienen otras limitaciones. En el sistema deliberativo se pueden encontrar ciertas deficiencias. Por ejemplo, la generación de objetivos de la *Mission Layer* se podría mejorar con el uso de LLMs que generan objetivos menos deterministas, ya que se podría utilizar la capacidad de razonamiento de los LLMs para generar los objetivos en vez de estar predefinidos por el desarrollador. En cuanto a la *Planning Layer*, no se tiene un buen visualizador del conocimiento simbólico de la base de conocimientos y del plan generado en esta. Además, es necesario un sistema que genere conocimiento simbólico a partir de los datos sensoriales y mantenga la correspondencia entre ese conocimiento y los objetos reales, es decir, un sistema de *Symbolic Anchoring*. Por otro lado, el sistema de comportamientos también tiene limitaciones, como la falta de habilidades basadas en la manipulación en al *Reactive Layer*, lo que mejoraría la interacción del robot con el entorno.



# Capítulo 6

## Evaluación de la Arquitectura Cognitiva

Este capítulo presenta la evaluación realizada en este trabajo de investigación para realizar la experimentación y evaluación de la arquitectura cognitiva presentada en este trabajo. MARLIN y MERLIN2 se han evaluado desde dos perspectivas: por una parte analizando sus componentes por separado y por otra parte se ha realizado la evaluación de cada arquitectura de forma integral. Para llevar a cabo la evaluación, se ha utilizado el robot RB1, que se presenta en la Figura 6.1, en diferentes entornos simulados en Gazebo y reales.



**Figura 6.1** – Robot RB1 utilizado en la evaluación y experimentación. Este robot de Robotnik consiste en un robot de servicio con una base móvil diferencial, con un LiDAR, y un torso, con una cámara RGB-D y altavoces.

## 6.1. Aplicación y despliegue

Antes de comenzar con la evaluación, es necesario conocer cómo se generan nuevos comportamientos con la arquitectura cognitiva. Así pues, a continuación se presenta cómo se puede llevar a cabo este proceso y cómo ha evolucionado de MERLIN a MERLIN2.

### 6.1.1. MERLIN

Para comprender el proceso necesario para generar nuevos comportamientos en un robot utilizando la arquitectura, se ilustra el procedimiento utilizando una tarea formalizada [134]. Para ello, se ha seleccionado el desafío del restaurante de la SCIROC. Este desafío está definido en el libro de reglas del *European Robotic Web Service Rulebook* [135]. La tarea consiste en crear un robot que pueda atender las mesas de un restaurante. Hay tres etapas en este desafío:

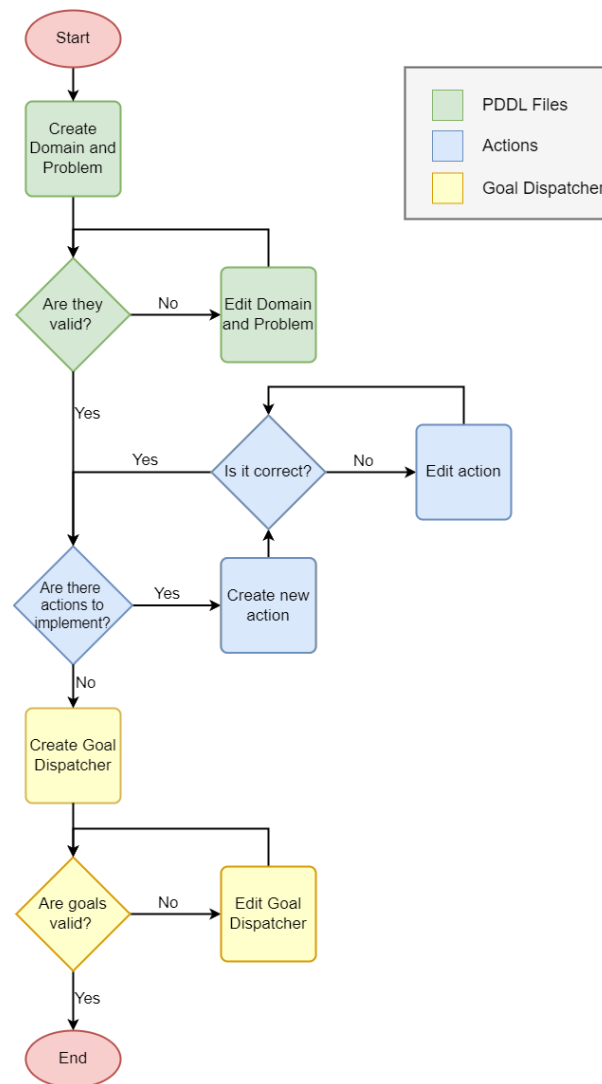
1. Verificar el estado de las mesas: el robot debe verificar el estado de cada mesa. Los estados son: necesita servicio, ya se ha servido, necesita limpieza y está lista.
2. Servir un pedido: el robot debe servir un pedido a una mesa que necesita servicio. También debe verificar si el barman ha preparado el pedido correctamente.
3. Guiar a un nuevo cliente: el robot debe verificar si hay un nuevo cliente en la entrada. Luego, debe guiar al nuevo cliente a una mesa libre.

Para crear nuevos comportamientos en robots utilizando MERLIN, se sigue el flujo de la Figura 6.2. De esta manera, para incluir los comportamientos y acciones necesarios para resolver el desafío del restaurante, se descompone el proceso completo en varias fases, en las que se trata la creación del PDDL, la implementación de las acciones y la generación de las misiones.

La primera fase aborda el problema desde la perspectiva deliberativa. Esta fase está representada por los elementos verdes de la Figura 6.2. Es necesario definir los dos elementos principales de nuestro sistema basado en PDDL, el dominio y el problema. Por ejemplo, el código 6.1 presenta el problema PDDL. Luego, deben



validarse utilizando el planificador POPF. Si no son válidos, deben corregirse y repetir el proceso.



**Figura 6.2** – Diagrama de flujo para crear nuevos comportamientos con MERLIN. Se distinguen tres partes, que son la creación de los archivos PDDL (verde), la implementación de las acciones (azul) y la implementación de la *Mission Layer* (amarillo).

La segunda fase se ocupa de la *Executive Layer* de la arquitectura. Esta fase está representada por los elementos de color azul de la Figura 6.2. Para utilizar la biblioteca SMACH, es necesario dividir el desafío del restaurante en subtarear con diferentes complejidades y funcionalidades. Estas subtarear son equivalentes a las acciones que utiliza ROSPlan y que están definidas en el dominio. Como

resultado, se tienen soluciones jerárquicas, como se presenta en la Figura 6.3. En este ejemplo se presenta una máquina de estados de SMACH que implementa la acción de llevar el pedido de una mesa, la barman. La acción utiliza diferentes habilidades del robot situado en la *Reactive Layer*, como el reconocimiento del habla, la síntesis de voz y el reconocimiento de objetos.

**Listing 6.1** – Problema PDDL para un robot que atiende las mesas de un restaurante de la SCIROC. Como objetos de PDDL iniciales se tienen los puntos del mapa, las mesas y el barman. Como proposiciones iniciales se tiene el punto inicial del robot, el punto de cada mesa, el punto del barman, el punto para que el robot espere y el punto en el los nuevos clientes esperan.

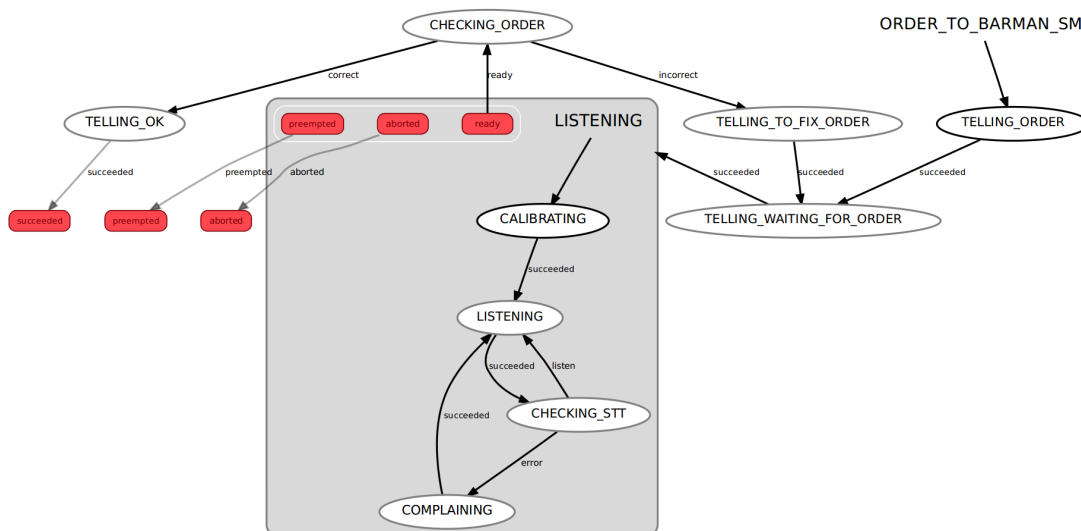
```
(define (problem restaurant_prb)
  (:domain restaurant)

  (:objects
    wp0 wp1 wp2 wp3 barman_wp – waypoint
    robot_waiting_wp – waypoint
    person_waiting_wp – waypoint
    t1 t2 t3 – table
    barman – person
  )

  (:init
    (robot_at wp0)
    (is_wp_near_table wp1 t1)
    (is_wp_near_table wp2 t2)
    (is_wp_near_table wp3 t3)
    (person_at barman barman_wp)
    (is_robot_waiting_wp robot_waiting_wp)
    (is_person_waiting_wp person_waiting_wp)
  )

  (:goal ()))
)
```

Finalmente, es necesario crear el componente que gestione los objetivos que el robot debe alcanzar, es decir, el *Goal Dispatcher* que pertenece a la *Mission Layer* y se puede implementar como una máquina de estados. Esta fase está representada por los elementos de color amarillo de la Figura 6.2.



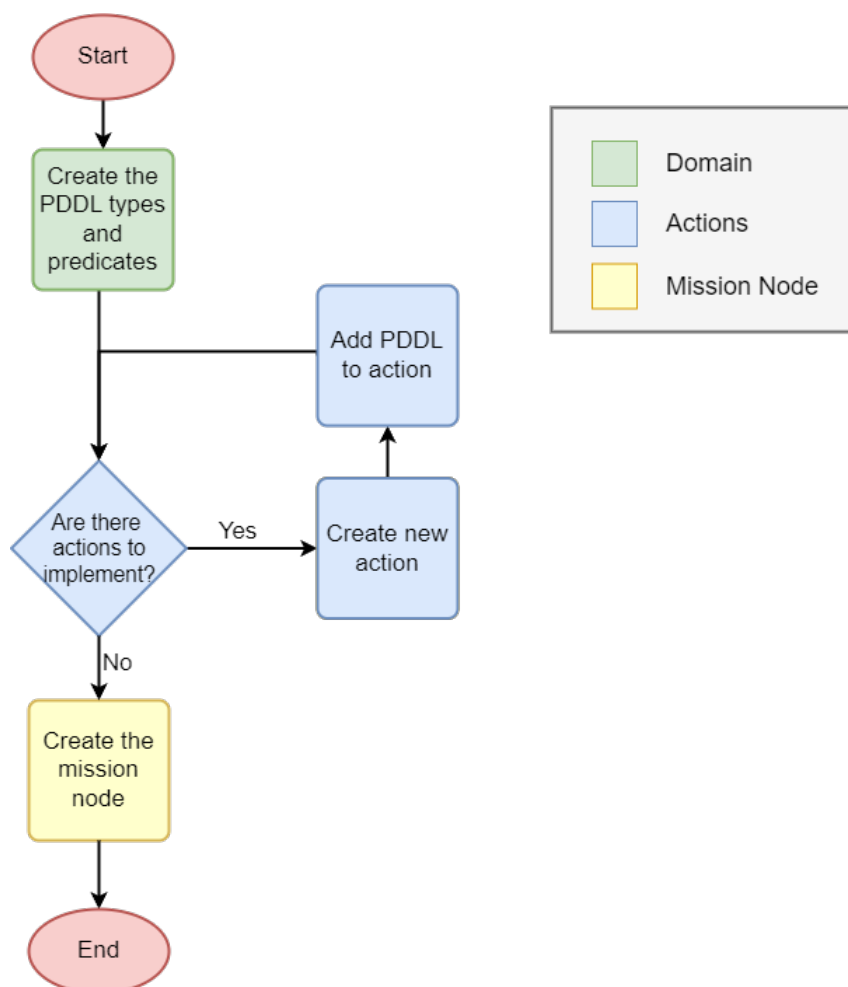
**Figura 6.3** – Ejemplo del SMACH Viewer mostrando la máquina de estados para llevar el pedido de una mesa al barman. Se tienen estados para hacer síntesis de voz y una máquina de estados para hacer reconocimiento del habla, con lo que se puede interactuar con el barman, y un estado que utiliza el reconocimiento de objetos para comprobar si el barman a colocado los objetos correctos del pedido.

### 6.1.2. MERLIN2

Al igual que en MERLIN, ahora se presenta un ejemplo de como utilizar MERLIN2 para generar nuevos comportamientos en robots autónomos. De esta manera, esta vez se ha seleccionado una tarea de la RoboCup 2023 [136], concretamente la tarea *carry my luggage*, que consiste en que el robot ayude a una persona a llevar parte de su equipaje. El diagrama de flujos presentado en la Figura 6.4 resume el procedimiento necesario para crear estos nuevos comportamientos.

La primera fase consiste en crear el dominio PDDL. Este dominio se crea con KANT, en Python o C++. Como resultado, cualquier componente software podría utilizar el nuevo dominio. La segunda fase consiste en crear las acciones

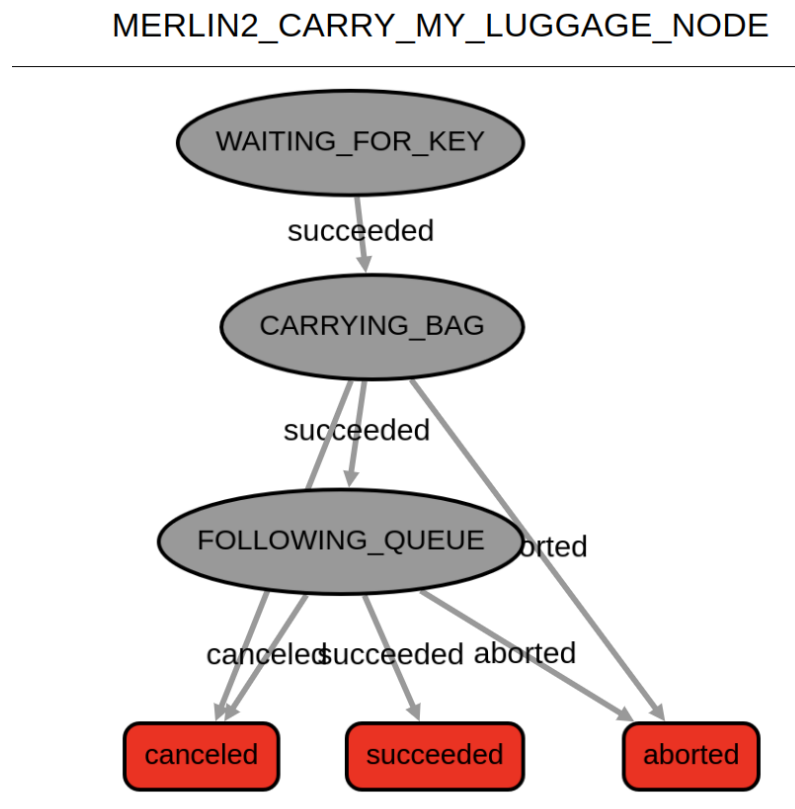
necesarias. Estas acciones pueden ser máquinas de estados de YASMIN o árboles de comportamiento. Las acciones no solo contienen los comportamientos necesarios para llevar a cabo las misiones, sino que también incluyen el conocimiento PDDL que las define y que se utilizará en la planificación. Un ejemplo se presenta en la Figura 5.10. En este caso, la acción de detectar una bolsa se ha implementado como un árbol de comportamiento.



**Figura 6.4** – Diagrama de flujo para crear nuevos comportamientos con MERLIN2. Se distinguen tres partes, que son la creación del conocimiento inicial (verde), la implementación de las acciones (azul) y la implementación de la *Mission Layer* (amarillo).

Por último, es necesario implementar la *Mission Layer* de MERLIN2 ya sea como un nodo normal o como una máquina de estados, que se encargará de crear el estado inicial del conocimiento, es decir, el problema PDDL; y de generar las

misiones del robot. El estado inicial del problema está formado por los objetos y proposiciones que se tendrían al inicio de la misión. En la Figura 6.5 se presenta un ejemplo. En este caso, se tiene un primer estado que espera a que se pulse una tecla. Después, se tienen dos estados para generar los objetivos del robot.

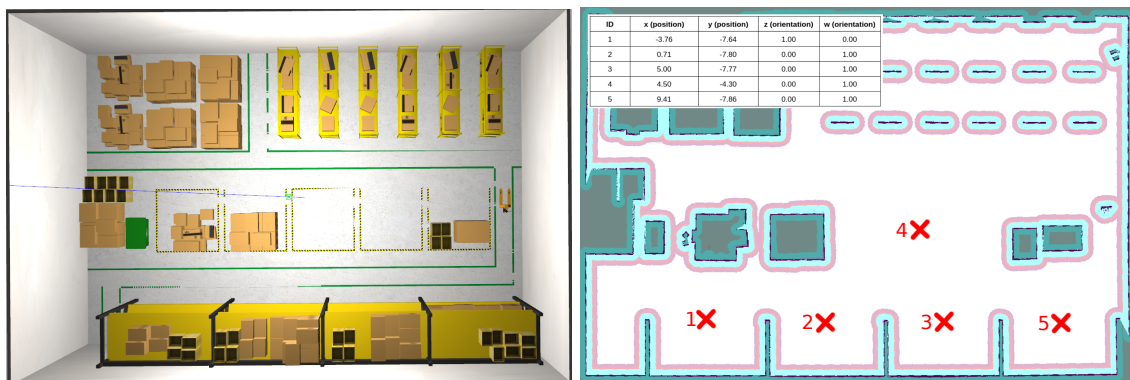


**Figura 6.5** – Máquina de estados que implementa la *Mission Layer* en la prueba *carry my luggage* de la RoboCup 2023. Se tienen tres estados, el primero espera a que el usuario pulse una tecla del teclado, el segundo genera el objetivo de llevar una bolsa y el tercero genera el objetivo de seguir la fila para volver a la arena de la RoboCup.

## 6.2. Evaluación de YASMIN

La evaluación de YASMIN se basa en comparar sus máquinas de estados con las de SMACH. Se plantea una tarea simple en la que un robot patrullar ciertas partes de un almacén. Para ello, el almacén se divide en cinco puntos que el robot debe explorar en un orden específico. Explorar cada punto implica girar el robot 45

grados, avanzar, girar 180 grados y regresar al punto de partida. Esto se repite ocho veces para completar un círculo. Las acciones de avanzar y girar se implementan como una máquina de estados compuesta por dos estados: un estado para calcular la posición a la que navegar y un estado para realizar la navegación. El almacén utilizado es el mundo de almacén de AWS Robomaker [137]. La Figura 6.6 muestra la vista aérea del almacén y los cinco puntos a patrullar con una tabla que indica sus coordenadas exactas y el orden en el que se visitarán.



**Figura 6.6** – Vista aérea del almacén de AWS Robomaker. A la izquierda se muestra el mundo simulado del almacén. A la derecha se presenta el mapa del almacén, marcando los puntos que se van a patrullar.

Las métricas utilizadas para medir cada máquina de estados son las siguientes:

- **Uso de RAM:** esta métrica permite medir el impacto de cada solución en la RAM. Dado que los robots son sistemas embebidos, es importante optimizar el uso de sus recursos (CPU, RAM, GPU, etc.).
- **Duración:** esta métrica permite medir el tiempo requerido para alcanzar los cinco puntos de cada experimento. Sin embargo, es importante destacar que se basa no solo en el tiempo necesario para ejecutar los estados y la transición entre estados, sino que también depende de la navegación utilizada por el robot, es decir, Navigation2 [133].
- **Energía consumida:** esta métrica permite medir el impacto de cada solución en la batería del robot. Nuevamente, la batería es uno de los recursos limitados del robot.

### 6.2.1. Discusión

Se realizaron un total de 20 ejecuciones para cada solución utilizando un equipo de escritorio compuesto por un procesador Intel(R) Core(TM) i7-6700 CPU @ 3.40 GHz, una tarjeta gráfica GeForce GTX 1060 de 6 GB y 16 GB de RAM DDR4. Se llevaron a cabo 10 ejecuciones donde se midió el uso de RAM y 10 ejecuciones donde se midió el consumo de energía. Ambas mediciones se realizaron de forma independiente, ya que la medición del consumo de energía aumenta el uso de RAM.

Se utilizó un conjunto de bibliotecas de Python para medir el rendimiento de las máquinas de estados en este experimento: a) el paquete psutil [138], que es una biblioteca Python para obtener información de los procesos en ejecución y la utilización del sistema, b) la aplicación CodeCarbon [139], que es un paquete software que permite a los desarrolladores de Python estimar la cantidad de dióxido de carbono (CO<sub>2</sub>).

Esta tarea se completó utilizando YASMIN y SMACH en la versión ROS Galactic en Ubuntu 20.04. En ambos casos, el código fuente utilizado se consideraría similar en términos de las siguientes cifras obtenidas:

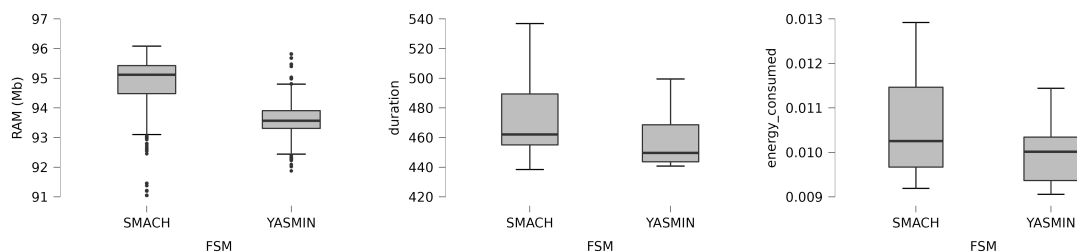
- **SMACH:** 12 archivos Python, 1 archivo CMakeLists.txt, 1 archivo XML, 1 archivo YAML. Esta versión presenta un total de 575 líneas en general: 417 líneas de código, 15 líneas de comentarios y 143 líneas en blanco.
- **YASMIN:** 12 archivos Python, 1 archivo CMakeLists.txt, 1 archivo XML, 1 archivo YAML. Esta versión presenta un total de 565 líneas en general: 407 líneas de código, 14 líneas de comentarios y 144 líneas en blanco.

La Tabla 6.1 muestra las estadísticas de este experimento. Estos resultados revelan que el consumo de RAM tiene un comportamiento similar entre las dos opciones. Sin embargo, la desviación estándar presenta valores más bajos en YASMIN y un rango de 1 MB más pequeño, lo que significa que el sistema de toma de decisiones de YASMIN es más estable en cuanto al consumo de memoria. Además, la duración del mismo experimento presenta una reducción del 3.46 % en el tiempo cuando se utiliza YASMIN, lo que se consideraría menor, pero representa

**Tabla 6.1** – Resultados obtenidos durante la experimentación de YAMSIN. Se muestran diferentes medidas estadísticas del consumo de RAM, el tiempo de ejecución y la energía consumida para comparar YASMIN y SMACH.

	RAM (MB)		Tiempo (segundos)		Energía (Kw)	
	SMACH	YASMIN	SMACH	YASMIN	SMACH	YASMIN
<b>Media</b>	94,776	93,648	473,718	457,301	0,011	0,010
<b>Desv. Típica</b>	0,983	0,548	30,020	19,794	0,001	$8,073e^{-4}$
<b>Rango</b>	5,031	3,941	98,418	58,717	0,004	0,002
<b>Min</b>	91,051	91,875	438,435	440,768	0,009	0,009
<b>Max</b>	96,082	95,816	536,854	499,485	0,013	0,011
<b>25th percentil</b>	94,480	93,309	455,042	443,652	0,010	0,009
<b>50th percentil</b>	95,117	93,566	462,082	449,634	0,010	0,010
<b>75th percentil</b>	95,426	93,906	489,390	468,610	0,011	0,010

una reducción total de 16 segundos. Finalmente, el consumo de energía presenta valores similares. La Figura 6.7 ilustra las estadísticas de SMACH y YASMIN durante el experimento. Estos gráficos respaldan la idea de que el rendimiento de YASMIN en cuanto a la duración es ligeramente mejor. Además, el rango intercuartílico se reduce en comparación con SMACH, lo que significa que parece más estable en YASMIN.



**Figura 6.7** – Diagramas *BoxPlot* que ilustran el consumo de RAM, la duración en segundos y el consumo de energía.



### 6.3. Evaluación de KANT

Para evaluar KANT, se plantea una tarea de alto nivel extraída y adaptada del capítulo de ERL-SciRoc 2019 [135]. Esta tarea consiste en un robot que asistirá a las personas en una cafetería o un restaurante. El robot se encarga de atender a los clientes, tomando pedidos y llevándolos hacia y desde las mesas. Durante la ejecución, el robot necesita una cantidad considerable de objetos, acciones y proposiciones que se consultan, crean, editan y eliminan. De esta forma, se propone como métrica el tiempo para gestionar el conocimiento. Este proceso experimental evalúa el tiempo de finalización de estas cinco tareas consecutivas basadas en la manipulación de PDDL mediante KANT:

1. ***Time to reset PDDL (TtR)***: tiempo empleado para reiniciar la base de conocimiento, es decir, para borrar todo el conocimiento PDDL de la base de conocimientos.
2. ***Time to load initial PDDL (TtL)***: tiempo empleado en crear en conocimiento PDDL inicial para el caso de atender una cafetería o restaurante.
3. ***Time to check tables (TtC)***: tiempo empleado en simular la tarea de comprobar el estado de las mesas añadiendo, modificando y eliminando el conocimiento PDDL necesario.
4. ***Time to serve an Order (TtS)***: tiempo empleado en simular la tarea de servir una orden añadiendo, modificando y eliminando el conocimiento PDDL necesario.
5. ***Time to guide a Person (TtG)***: tiempo empleado en simular la tarea de guiar a una persona a una mesa libre añadiendo, modificando y eliminando el conocimiento PDDL necesario.

Se implementaron dos familias DAO para poder almacenar PDDL: MongoDB y una base de conocimientos en memoria utilizando un nodo de ROS 2. Estas familias se desplegaron junto con la arquitectura cognitiva de este trabajo de investigación, MERLIN2, y todo se simuló utilizando un entorno ROS 2 Foxy.

### 6.3.1. Discusión

Cada prueba se ha ejecutado 3000 veces en un equipo portátil con 32 GB de RAM y un procesador Intel(R) Core(TM) i7-8750H a 2.20 GHz. Posteriormente, se obtuvieron los resultados presentados en la Tabla 6.2, en la que se muestran los resultados utilizando una base de datos almacenada en memoria en un nodo de ROS 2, y la Tabla 6.3, en la que se muestran los resultados utilizando MongoDB como base de conocimiento. La base de conocimiento en memoria en un nodo de ROS 2 presenta un tiempo de ejecución de casi 122 minutos. El tiempo promedio de cada iteración es de 2.432 segundos, con una mediana de 2.385 segundos. El tiempo para crear el PDDL inicial es de 52 minutos, borrar todo el conocimiento lleva 13 minutos y se gastan 23, 18 y 14 minutos en las tareas TtC, TtS y TtG respectivamente.

**Tabla 6.2** – Resultados del experimento de KANT utilizando una base de datos almacenada en memoria en un nodo de ROS 2. Se muestran diferentes medidas estadísticas de los tiempos de ejecución para comprar KANT utilizando un nodo de ROS 2 como bases de conocimiento.

	<b>TtR</b>	<b>TtL</b>	<b>TtC</b>	<b>TtS</b>	<b>TtG</b>	<b>Total</b>
<b>Media</b>	0,263	1,051	0,465	0,364	0,29	2,432
<b>Moda</b>	0,317	0,967	0,341	0,233	0,403	2,261
<b>Mediana</b>	0,324	0,996	0,416	0,321	0,245	2,385
<b>Desv. Típica</b>	0,124	0,285	0,18	0,152	0,137	0,446
<b>Min</b>	0,039	0,527	0,218	0,16	0,114	1,312
<b>Max</b>	0,805	3,041	2,065	1,618	1,489	4,722
<b>Suma</b>	787,571	3152,132	1393,824	1093,036	869,431	7295,995

En el caso de utilizar MongoDB, se tiene un tiempo de ejecución de 76 minutos para las mismas 3000 iteraciones. Del tiempo total, se utilizaron 3.5 minutos para borrar el conocimiento. Aproximadamente 31 minutos se dedicaron a crear el conocimiento inicial. Se invirtieron 16 minutos en TtC, 14 minutos para TtS y finalmente 10 minutos en TtG.

**Tabla 6.3** – Resultados del experimento de KANT utilizando una base de datos basada en MongoDB. Se muestran diferentes medidas estadísticas de los tiempos de ejecución para comprar KANT utilizando MongoDB como bases de conocimiento.

	TtR	TtL	TtC	TtS	TtG	Total
<b>Media</b>	0,069	0,612	0,329	0,288	0,216	1,514
<b>Moda</b>	0,071	0,613	0,327	0,293	0,214	1,525
<b>Mediana</b>	0,069	0,612	0,329	0,288	0,216	1,512
<b>Desv. Típica</b>	0,002	0,014	0,008	0,007	0,006	0,027
<b>Min</b>	0,063	0,569	0,299	0,262	0,194	1,446
<b>Max</b>	0,1	0,756	0,411	0,344	0,264	1,815
<b>Suma</b>	206,676	1835,754	987,505	865,309	646,987	4542,231

Como se muestra en las tablas, la opción de MongoDB obtiene unos resultados mucho mejores que la aproximación que utiliza una base de conocimientos almacenada en memoria en un nodo de ROS 2. Por ejemplo, MongoDB gasta 3,8 veces más rápido en la tarea TtR, 1,72 veces más rápido en la tarea TtL, 1,41 veces más rápido en la tarea TtC, 1,26 veces más rápido en la tarea TtS y 1,34 veces más rápido en la tarea TtG. En total, la aproximación de una base de conocimiento utilizando MongoDB es 1,61 más rápida.

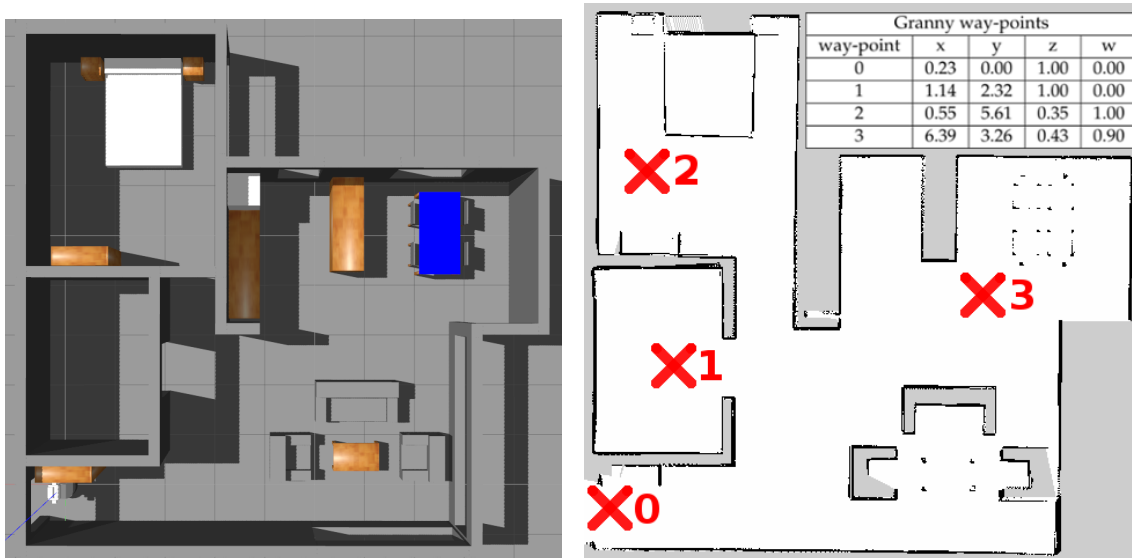
## 6.4. Evaluación de la arquitectura cognitiva

Para evaluar la arquitectura cognitiva presentada en este trabajo de investigación se propone comparar ROSPlan; la arquitectura original, MERLIN; y el segundo enfoque de la arquitectura, MERLIN2. En estos casos se ha utilizado el mismo planificador PDDL, que es POPF. Por otro lado, también se ha comparado el último enfoque, que incluye el uso de LLMs para llevar a cabo la planificación de tareas. En este caso, se han definido las siguientes versiones, dependiendo de los componentes que se han utilizado para integrar los LLMs en la arquitectura:

- **Integración completa (IC):** esta versión utiliza todos los componentes. El estado del mundo se genera mediante Retrieval Augmented Generation

(RAG) con la base de datos de vectores ChromaDB y se tiene un paso para verificar si se cumplen los objetivos.

- **Integración sin RAG (ISR):** esta versión es similar a la IC, pero no utiliza RAG. Como resultado, utiliza todos los elementos de conocimiento como el estado del mundo.
- **Integración sin verificación de objetivos (ISV):** esta versión es similar a la IC pero sin el estado de verificación de objetivos. Como resultado, la arquitectura depende únicamente de la capacidad de planificación del LLM.
- **Integración sin RAG y sin estado de verificación de objetivos (ISRSV):** esta versión no utiliza ni el RAG ni verifica que los objetivos se han cumplido.



**Figura 6.8** – Vista aérea del apartamento simulado utilizado en los experimentos de la arquitectura cognitiva presentada en este trabajo de investigación. A la izquierda se presenta el mundo simulado del apartamento. A la derecha se presenta el mapa del apartamento, marcando los puntos a los que tendrá que navegar el robot para realizar sus misiones.

El experimento consiste en realizar misiones basadas en navegar hasta ciertos puntos de interés y realizar una tarea basada en síntesis de voz. Por ejemplo, en el caso de la versión de la arquitectura que utiliza LLMs, las misiones consisten en saludar a las personas que estarían situadas en esos puntos, mientras que en el

resto de casos consisten en anunciar que el punto ha sido patrullado. Los puntos se corresponden con cuatro zonas de un apartamento: la habitación, el baño, la cocina y la puerta principal. Estos puntos son los puntos marcados con cruces rojas en el mapa de la Figura 6.8 y sus posiciones se definen en la tabla de la misma figura.

Con estos puntos de interés definidos, se han diseñado dos tipos de experimentos. Para cada experimento se tiene una lista aleatoria de puntos de interés que el robot tienen que visitar. Con el fin de evaluar el impacto y la capacidad de recuperación de la arquitectura para gestionar misiones, se cancela la mitad de las tareas asociadas con la misión. Con esto, los dos experimentos consisten en el mismo concepto, visitar los puntos y realizar una tarea, pero se diferencian en las misiones que se realizan. El primer experimento contiene 6 misiones, es decir, visitar 6 puntos de interés; mientras que el segundo tiene 20 misiones, es decir, visitar 20 puntos de interés.

Para evaluar el rendimiento de la arquitectura, se tienen las siguientes métricas:

1. **Tiempo:** se mide el tiempo de ejecución, en segundos, necesario para completar las misiones.
2. **Distancia recorrida:** se mide la distancia, en metros, recorrida durante la ejecución de las misiones.

La elección de estos valores nos permite medir el rendimiento de la arquitectura cognitiva. En primer lugar, la métrica de tiempo mide el tiempo necesario para cada misión, lo que significa que cubrirá el impacto de pasar de una tarea a otra. Esto implicará cancelar ciertas capas de la arquitectura para enviar un nuevo objetivo y activar las nuevas acciones. En segundo lugar, la distancia recorrida permite medir la distancia navegada para cumplir una misión. Un robot que recorre distancias más largas implica un mayor consumo de batería y menos tiempo de servicio efectivo. Al mismo tiempo, significa que el proceso de cambio de una tarea a otra no solo tiene un impacto en el tiempo, sino también implicaciones en el rendimiento.

Para llevar a cabo los experimentos de este experimento, se ha utilizado un equipo con Ubuntu 22.04 y ROS 2 Humble. Además, cuenta con un procesador

Intel(R) i9-13980HX, 32 GB de RAM y una tarjeta gráfica Nvidia RTX 4070 con 8 GB. Se ha configurado el componente llama\_ros para utilizar un LLM Marcoroni de 13B con cuantificación de 4 bits. Además, también se ha configura llama\_ros para acelerar el LLM con GPU mediante cuBLAS, cargando 30 de 43 capas totales.

#### 6.4.1. Discusión

Las Tablas 6.4 y 6.5 presenta los resultados para este experimento después de ejecutar los dos tipos de experimentos cinco veces. Se muestra el rendimiento del robot cuando se establecen 6 o 20 puntos. Estos valores resumen la diferencia significativa entre los enfoques presentados.

De esta manera, los resultados muestran que MERLIN es superior a ROSPlan. Más específicamente, ROSPlan gasta 1,7 veces más de tiempo en el experimento 1 y 1,5 veces más de tiempo en el experimento 2 que MERLIN. Esto se debe a que ROSPlan no está correctamente preparado para cancelar los planes y las acciones que ejecuta, mientras que MERLIN si que es capaz de cancelar sus planes y acciones. Además, como resultado, ROSPlan no cancela sus acciones, por lo que recorre mayores distancias, concretamente recorre 2,1 veces más de distancia en el experimento 1 y 1,7 veces más de distancia en el experimento 2. De esto se obtiene que MERLIN es una arquitectura cognitiva superior a las aproximaciones ya existentes en ROS por separado.

Por otro lado, al comparar MERLIN y MERLIN2, se puede ver que los resultados son bastante similares en el experimento 1, obteniendo MERLIN unos resultados mínimamente mejores. Sin embargo, en el experimento 2, MERLIN gastó 1,4 más de tiempo y recorrió 1,5 más de distancia. De aquí se obtiene que MERLIN2 se comporta mejor en tareas más largas aunque puede ser algo más inestable porque su desviación estándar es mayor. De esta manera, el uso de YASMIN, KANT y el resto de componentes mejoran la arquitectura inicial aunque hay que tener en cuenta que MERLIN2 se basa en ROS 2 y Navigation2, que son versiones más modernas que las que utiliza MERLIN.

**Tabla 6.4** – Resultados obtenidos al evaluar la arquitectura cognitiva de este trabajo de investigación en el experimento 1, compuesto de 6 misiones. El experimento se ha repetido 5 veces. Se muestran los valores medios y las desviaciones típicas.

	<b>Tiempo (segundos)</b>	<b>Distancia (metros)</b>
<b>ROSPlan</b>	178,20 ± 6,03	46,65 ± 2,05
<b>MERLIN</b>	104,46 ± 12,30	21,72 ± 3,94
<b>MERLIN2</b>	107,72 ± 12,02	30,57 ± 1,02
<b>LLM (IC)</b>	251,63 ± 14,40	15,30 ± 5,85
<b>LLM (ISR)</b>	178,48 ± 18,05	17,25 ± 7,68
<b>LLM (ISV)</b>	200,15 ± 23,34	14,54 ± 5,79
<b>LLM (ISRSV)</b>	166,47 ± 11,17	20,80 ± 3,52

**Tabla 6.5** – Resultados obtenidos al evaluar la arquitectura cognitiva de este trabajo de investigación en el experimento 2, compuesto de 20 misiones. El experimento se ha repetido 5 veces. Se muestran los valores medios y las desviaciones típicas.

	<b>Tiempo (segundos)</b>	<b>Distancia (metros)</b>
<b>ROSPlan</b>	556,82 ± 9,89	138,82 ± 4,94
<b>MERLIN</b>	369,14 ± 18,06	80,23 ± 6,26
<b>MERLIN2</b>	257,56 ± 65,45	62,96 ± 27,26
<b>LLM (IC)</b>	848,20 ± 50,05	53,10 ± 13,33
<b>LLM (ISR)</b>	563,29 ± 20,82	51,72 ± 8,65
<b>LLM (ISV)</b>	665,63 ± 26,86	49,37 ± 10,89
<b>LLM (ISRSV)</b>	508,02 ± 35,41	53,70 ± 10,43

Por último, los resultados son sustancialmente mejores utilizando el enfoque clásico de PDDL que al implementar LLM. En el caso del experimento 1, la diferencia entre MERLIN2 y IC es 2.3 veces peor y 1.5 veces peor en comparación con la versión ISRSV. Al mismo tiempo, los resultados en el experimento 2 confirman que MERLIN2 es la mejor opción para la toma de decisiones en comparación con las opciones de LLM (3.2 veces mejor en comparación con IC y 1.9 veces mejor

en comparación con ISRSV). Por otro lado, al comparar ISR e ISV, el tiempo de ejecución de ISV es 1.2 veces peor que el de ISR en ambos experimentos. Este resultado muestra que utilizar RAG tiene un costo más significativo que utilizar el LLM para verificar si se ha logrado el objetivo. También se observan resultados superiores en cuanto a la distancia recorrida. En promedio, los resultados basados en LLM mostraron que el robot recorrió la mitad de la distancia que recorrió el robot con MERLIN2. Esto se debe a que, en el caso de integrar LLMs, cuando una misión se cancela lo suele hacer cuando está planificando mientras que en el caso de MERLIN2 lo suele cancelar cuando está navegando. De esta forma, cuando el robot utiliza MERLIN2 recorre una mayor distancia antes de cancelar sus funciones. Todo esto se debe a que el tiempo de inferencia del LLM elegido es bastante superior al tiempo de planificación. Sin embargo, con el creciente desarrollo de LLMs y de herramientas a fines, es posible que en poco tiempo surjan LLMs más pequeños y nuevas técnicas para optimizar la inferencia. Esto haría posible que los planificadores clásicos se pueden reemplazar por LLMs, con lo que la interacción con los robots autónomos se podrá realizar utilizando un lenguaje más natural.

#### 6.4.2. Conclusiones

En este capítulo de evaluación se ha demostrado la validez de la arquitectura cognitiva desarrollada en este trabajo de investigación. Para ello, se han realizado diferentes experimentos y evaluaciones y se han utilizado métricas encontradas durante la revisión sistemática de la literatura, como el tiempo de ejecución, la distancia recorrida, métricas de rendimiento y consumo de energía.

En primer lugar, se han evaluado los componentes de MERLIN2 por separado, YASMIN y KANT. En el primer caso, se ha evaluado la nueva librería YASMIN con la que se tenía anteriormente en ROS, SMACH. Esta evaluación se basó en un experimento de navegación en un entorno simulado empleando máquinas de estados de ambas librerías. Como se mostró en la discusión, YASMIN obtiene mejores resultados en cuanto a tiempo de ejecución, uso de recursos y consumo de energía, por lo que YASMIN es superior a SMACH. Por otro lado, la evaluación de KANT se basó en medir el tiempo de ejecución en un experimento basado en



manipular el conocimiento, una basada en MongoDB y almacenada en memoria. Los resultados muestran que la versión de MongoDB es superior a la versión que utiliza almacenamiento en memoria, que es el más utilizado en herramientas como ROSPlan.

Por último, se ha presentado la evaluación de la arquitectura cognitiva. Para ello, se ha comparado ROSPlan, MERLIN, MERLIN2 y la integración de LLMs. El experimento realizado se basa en gestionar las misiones de un robot que tiene que moverse por ciertas zonas de un apartamento. Las métricas utilizadas han sido el tiempo de ejecución y la distancia recorrida. Como resultado, se ha obtenido que MERLIN2 es similar a MERLIN, pero obtiene mejores resultados en ejecuciones con un mayor número de misiones. Por otro lado, el uso de LLMs muestra peores resultados debido al uso de modelos que necesitan bastante tiempo de inferencia en comparación con el tiempo que requieren los planificadores clásicos. Por lo que la configuración clásica es la mejor opción para generar comportamientos con MERLIN2. Sin embargo, es posible que en un futuro se creen LLMs con menor tiempo de inferencia y puedan competir con los planificadores clásicos.



# Capítulo 7

## Conclusiones

En este trabajo de investigación se ha presentado una arquitectura cognitiva para generar, controlar, planificar y monitorizar los comportamientos de robots autónomos. Es una arquitectura híbrida, organizada en capas, que se basa en utilizar de forma conjunta un sistema deliberativo, un sistema reactivo y varios componentes emergentes. Para desarrollar la arquitectura, se ha llevado a cabo una revisión sistemática de la literatura, buscando las arquitecturas cognitivas recientes y las herramientas para generar comportamientos. Posteriormente, se ha definido formalmente la arquitectura y se ha diseñado y desarrollado. Por último, se ha realizado la evaluación de la arquitectura y sus herramientas. De esta manera, esta arquitectura responde a la hipótesis de este trabajo de investigación.

### 7.1. Validación de la hipótesis y objetivos

Este trabajo de investigación se ha centrado en el uso de arquitecturas cognitivas híbridas para generar, controlar, planificar y monitorizar comportamientos en robots autónomos. Con esto se trata de responder a la hipótesis principal  $H_0$  del trabajo, definida en el capítulo 1. Esta hipótesis consiste en que la implementación de una arquitectura cognitiva híbrida para robots autónomos, que integre componentes reactivos, deliberativos y emergentes, conducirá a un incremento significativo en su capacidad para adaptarse a entornos dinámicos y tomar decisiones inteligentes en tiempo real, lo que mejorará su autonomía y desempeño en varias tareas. Así, el objetivo principal de este trabajo de investigación es el

desarrollo de una arquitectura cognitiva híbrida que permita generar, controlar, planificar y monitorizar comportamientos en robots autónomos. Además, se tienen los siguientes objetivos secundarios:  $O_1$  conocer las herramientas para generar comportamientos en robots autónomos que se utilizan actualmente en las arquitecturas cognitivas;  $O_2$  diseñar y desarrollar una nueva arquitectura cognitiva que permita generar comportamientos en robots autónomos; y  $O_3$  evaluar la nueva arquitectura cognitiva desarrollada anteriormente con el objetivo de validar su utilidad y aplicabilidad en robots autónomos.

Para comenzar este trabajo de investigación, se ha llevado a cabo una revisión sistemática de la literatura en el capítulo 2, centrándose en los artículos de investigación recientes que tratan este tema. En esta revisión se han presentado dos preguntas de investigación principales, que son ¿Qué arquitecturas cognitivas contempla el estado de la cuestión para robots autónomos?, y ¿Qué herramientas se utilizan en las arquitecturas cognitivas para generar comportamientos en robots autónomos?, con las que se busca conocer las arquitecturas cognitivas recientes y las herramientas empleadas para generar comportamientos, para poder cumplir el objetivo  $O_1$  que ha definido. Por otro lado, también se han definido dos preguntas de investigación secundarias, que son ¿Qué métodos se emplean para evaluar las arquitecturas cognitivas?, y ¿Qué métricas se utilizan para evaluar las arquitecturas cognitivas?, para conocer como se evalúan y se validan las arquitecturas cognitivas. Como resultado, se han encontrado las arquitecturas cognitivas más recientes y las herramientas para generar comportamientos, como las máquinas de estados, los árboles de comportamiento, el PDDL, los planificadores simbólicos, los grafos de conocimiento y diferentes tipos de componentes emergentes. También se han identificado diferentes formas para evaluar las arquitecturas cognitivas, entre las que destaca el uso de experimentos con interacción con humanos; y las métricas empleadas, como el tiempo de ejecución, la distancia recorrida y el rendimiento.

Así pues, para llevar a cabo el objetivo  $O_2$ , en los capítulos 5 y 5, se ha presentado el diseño, y la implementación de la arquitectura cognitiva híbrida para robots autónomos con la que se pretende responder a la hipótesis inicial  $H_0$ . La primera versión de la arquitectura cognitiva, MERLIN [66], se desarrolló en ROS utilizando

herramientas clásicas, como ROSPlan y SMACH. Esta primera versión consistía en una arquitectura híbrida organizada en cuatro capas. Sin embargo, dado que ROS dejará de tener soporte, se desarrolló una segunda versión basada en ROS 2, llamada MERLIN2 [140]. Esta nueva versión se basa en dos nuevas herramientas: KANT [56], para gestionar el conocimiento simbólico del robot; y YASMIN [55], para generar comportamientos mediante el uso de máquinas de estados, las cuales se pueden monitorizar desde un visualizador web. MERLIN2 se ha desarrollado con un sistema deliberativo basado en KANT que contiene un planificador de PDDL. Además, utiliza las máquinas de estados de YASMIN en sus diferentes capas y, gracias al visualizador de YASMIN, se puede monitorizar la arquitectura a alto nivel. Por otro lado, se han presentado los componentes emergentes que puede utilizar MERLIN2, es decir, YOLOv8, para obtener información del entorno, y LLMs, para procesar textos, mejorar la interacción natural con los humanos, llevar a cabo tareas de razonamiento y planificación y mejorar la explicabilidad del sistema y de la arquitectura aprovechando las capacidades narrativas de los LLMs para interpretar los logs que se generan.

Por último, en el capítulo 6, se ha tratado el objetivo  $O_3$ , es decir, se ha validado la arquitectura y sus diferentes herramientas para generar comportamientos. De esta manera, en la evaluación de KANT se ha mostrado que el uso de una base de conocimientos en MongoDB tiene un mejor rendimiento que una base de datos almacenada en memoria en un nodo de ROS 2, que es la más extendida. En segundo lugar, la evaluación de YASMIN es similar o incluso mejor que la alternativa que se tenía en ROS, SMACH. Para terminar, se ha realizado la evaluación de la arquitectura cognitiva completa. Para ello, se han comparado ROSPlan, MERLIN, MERLIN2 y MERLIN2 con diferentes integraciones de LLMs para realizar planificación. Como resultado, se ha obtenido que MERLIN2 es la mejor opción porque se corrigen las limitaciones de ROSPlan. Además, obtienen mejores resultados con los planificadores clásicos de MERLIN2 que con los LLMs. Sin embargo, es importante destacar que es posible que surja nuevos LLMs y técnicas que optimicen los procesos de inferencia, lo que haría posible reemplazar dichos planificadores por LLMs, consiguiendo así una interacción más natural con el robot.

## 7.2. Contribuciones

Como resultado de este trabajo de investigación, se han obtenido diferentes contribuciones tanto en conferencias como en artículos. De esta forma, se tienen las siguientes contribuciones:

- **Rodríguez-Lera, F. J., González-Santamarta, M. Á., Guerrero-Higueras, Á. M., Martín-Rico, F., & Matellán-Olivera, V. (2022).** *Towards explainability in robotics: A performance analysis of a cloud accountability system.* *Expert Systems*, 39(9), e13004
- **González-Santamarta, M. Á., Rodríguez-Lera, F. J., Martín, F., Fernández, C., & Matellán, V. (2022, May).** *KANT: A tool for grounding and knowledge management.* In *International Work-Conference on the Interplay Between Natural and Artificial Computation* (pp. 452-461). Cham: Springer International Publishing
- **González-Santamarta, M. Á., Rodríguez-Lera, F. J., Matellán-Olivera, V., & Fernández-Llamas, C. (2022, November).** *YASMIN: Yet another state machine.* In *Iberian Robotics conference* (pp. 528-539). Cham: Springer International Publishing
- **González-Santamarta, M. Á., Rodríguez-Lera, F. J., Fernández-Llamas, C., & Matellán-Olivera, V. (2023).** *MERLIN2: MachinEd Ros 2 pLanINg.* *Software Impacts*, 15, 100477
- **González-Santamarta, M. Á., Fernández-Becerra, L., Sobrín-Hidalgo, D., Guerrero-Higueras, Á. M., González, I., & Lera, F. J. R. (2023).** *Using Large Language Models for Interpreting Autonomous Robots Behaviors.* In P. García Bringas, H. Pérez García, F. J. Martínez de Pisón, F. Martínez Álvarez, A. Troncoso Lora, Á. Herrero, ... E. Corchado (Eds.), *Hybrid Artificial Intelligent Systems* (pp. 533–544). Cham: Springer Nature Switzerland.
- **González-Santamarta, M. Á., Rodríguez-Lera, F. J., Guerrero-Higueras, Á. M., & Matellán-Olivera, V. (2023).** *Integration of Large Language Models within Cognitive Architectures for Autonomous Robots.* *arXiv preprint arXiv:2309.14945.*

Por otro lado, se ha participado en otros trabajos con cierta relación con el trabajo que se presenta. De esta manera, tienen las siguientes contribuciones secundarias:

- *Álvarez-Aparicio, C., Guerrero-Higueras, Á. M., González-Santamarta, M. Á., Campazas-Vega, A., Matellán, V., & Fernández-Llamas, C. (2022). Biometric recognition through gait analysis. Scientific Reports, 12(1), 14530.*
- *Rodríguez-Lera, F. J., González-Santamarta, M. Á., Esteban-Costales, G., Ayucar, U., Gil-Uriarte, E., Glera, A., & Mayoral-Vilches, V. (2022, November). Threat modeling for robotic-based production plants. In 2022 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR) (pp. 110-115). IEEE.*
- *Hidalgo, D. S., Becerra, L. F., González-Santamarta, M. Á., Álvarez-Aparicio, C. Á., Guerrero-Higueras, Á. M., Conde-González, M. Á. C., ... & Matellán-Olivera, V. (2023). Ciberseguridad en sistemas ciberfísicos: entorno simulado para la evaluación de competencias en ciberseguridad en sistemas con capacidades autónomas. In Actas de las VIII Jornadas Nacionales de Investigación en Ciberseguridad: Vigo, 21 a 23 de junio de 2023 (pp. 461-467). Universidade de Vigo.*
- *Fernández-Becerra, L., González-Santamarta, M. A., Sobrín-Hidalgo, D., Guerrero-Higueras, Á. M., Lera, F. J. R., & Olivera, V. M. (2023, August). Accountability and Explainability in Robotics: A Proof of Concept for ROS 2-And Nav2-Based Mobile Robots. In Computational Intelligence in Security for Information Systems Conference (pp. 3-13). Cham: Springer Nature Switzerland.*

Por último, también se han obtenido varios repositorios públicos durante el desarrollo de este trabajo. De esta forma, se tienen los siguientes repositorios:

- MERLIN2: <https://github.com/MERLIN2-ARCH/merlin2>
- simple\_node: [https://github.com/uleroboticsgroup/simple\\_node](https://github.com/uleroboticsgroup/simple_node)
- waypoint\_navigation: [https://github.com/MERLIN2-ARCH/waypoint\\_navigation](https://github.com/MERLIN2-ARCH/waypoint_navigation)
- speech\_to\_text: [https://github.com/MERLIN2-ARCH/speech\\_to\\_text](https://github.com/MERLIN2-ARCH/speech_to_text)
- text\_to\_speech: [https://github.com/MERLIN2-ARCH/text\\_to\\_speech](https://github.com/MERLIN2-ARCH/text_to_speech)

- YASMIN: <https://github.com/uleroboticsgroup/yasmin>
- KANT: <https://github.com/uleroboticsgroup/kant>
- knowledge\_graph: [https://github.com/mgonzs13/knowledge\\_graph](https://github.com/mgonzs13/knowledge_graph)
- ros2\_rb1: [https://github.com/mgonzs13/ros2\\_rb1](https://github.com/mgonzs13/ros2_rb1)
- ros2\_rover: [https://github.com/mgonzs13/ros2\\_rover](https://github.com/mgonzs13/ros2_rover)
- yolov8\_ros: [https://github.com/mgonzs13/yolov8\\_ros](https://github.com/mgonzs13/yolov8_ros)
- yolonas\_ros: [https://github.com/mgonzs13/yolonas\\_ros](https://github.com/mgonzs13/yolonas_ros)
- llama\_ros: [https://github.com/mgonzs13/llama\\_ros](https://github.com/mgonzs13/llama_ros)
- audio\_common: [https://github.com/mgonzs13/audio\\_common.git](https://github.com/mgonzs13/audio_common.git)
- whisper\_ros: [https://github.com/mgonzs13/whisper\\_ros](https://github.com/mgonzs13/whisper_ros)

### 7.3. Trabajo futuro

Si bien la arquitectura presentada en este trabajo aborda la hipótesis planteada y cumple con los objetivos establecidos inicialmente, se identifican oportunidades de investigación en algunos de los componentes del núcleo de la arquitectura.

Como dirección futura, se contempla la utilización de *Large Language Models* más refinados y compactos, con desempeños superiores. Esto permitiría que la selección de tareas a alto nivel, es decir, la planificación, la generación de objetivos y el razonamiento de la arquitectura se realicen en lenguaje natural, lo que potencialmente mejorarían la interacción con el robot.

Por otra parte, se propone una optimización en la percepción y el aprendizaje de la arquitectura. Una opción sería la implementación de un sistema de *Symbolic Anchoring* que adquiriera conocimiento simbólico del entorno y mantenga la correspondencia entre los símbolos y los objetos reales.

Finalmente, en cuanto a la memoria de la arquitectura, se sugiere reemplazar completamente el uso de PDDL por un grafo de conocimiento, lo que permitiría gestionar el conocimiento simbólico del robot de manera más efectiva.



# Bibliografía

- [1] Michael Brady. “Artificial intelligence and robotics”. En: *Artificial intelligence* 26.1 (1985), págs. 79-121.
- [2] Stuart J Russell y Peter Norvig. *Artificial intelligence a modern approach*. London, 2010.
- [3] R Peter Bonasso et al. “Experiences with an architecture for intelligent, reactive agents”. En: *Journal of Experimental & Theoretical Artificial Intelligence* 9.2-3 (1997), págs. 237-256.
- [4] Félix Ingrand y Malik Ghallab. “Deliberation for autonomous robots: A survey”. En: *Artificial Intelligence* 247 (2017), págs. 10-44.
- [5] Ronald Arkin y Tucker Balch. “AuRA: Principles and Practice in Review”. En: *Journal of Experimental & Theoretical Artificial Intelligence* 9 (feb. de 1970). DOI: 10.1080/095281397147068.
- [6] Iuliia Kotseruba y John K Tsotsos. “40 years of cognitive architectures: core cognitive abilities and practical applications”. En: *Artificial Intelligence Review* 53.1 (2020), págs. 17-94.
- [7] Peijun Ye, Tao Wang y Fei-Yue Wang. “A survey of cognitive architectures in the past 20 years”. En: *IEEE transactions on cybernetics* 48.12 (2018), págs. 3280-3290.
- [8] Erann Gat, R Peter Bonasso, Robin Murphy et al. “On three-layer architectures”. En: *Artificial intelligence and mobile robots* 195 (1998), pág. 210.
- [9] Barbara Ann Kitchenham, David Budgen y Pearl Brereton. *Evidence-based software engineering and systematic reviews*. Vol. 4. CRC press, 2015.

- [10] Hector Levesque y Gerhard Lakemeyer. “Cognitive robotics”. En: *Foundations of artificial intelligence* 3 (2008), págs. 869-886.
- [11] Danial Nakhaeinia et al. “A review of control architectures for autonomous navigation of mobile robots”. En: *International Journal of the Physical Sciences* 6.2 (2011), págs. 169-174.
- [12] Maria Fox y Derek Long. “PDDL2.1: An extension to PDDL for expressing temporal planning domains”. En: *J. Artif. Intell. Res. (JAIR)* 20 (dic. de 2003), págs. 61-124. doi: 10.1613/jair.1129.
- [13] Amanda Coles et al. “Forward-Chaining Partial-Order Planning”. En: ene. de 2010, págs. 42-49.
- [14] Silvia Coradeschi y Alessandro Saffiotti. “An introduction to the anchoring problem”. En: *Robotics and autonomous systems* 43.2-3 (2003), págs. 85-96.
- [15] Rodney A Brooks. “New approaches to robotics”. En: *Science* 253.5025 (1991), págs. 1227-1232.
- [16] John E Laird, Allen Newell y Paul S Rosenbloom. “Soar: An architecture for general intelligence”. En: *Artificial intelligence* 33.1 (1987), págs. 1-64.
- [17] Mark Petticrew y Helen Roberts. *Systematic reviews in the social sciences: A practical guide*. John Wiley & Sons, 2008.
- [18] Cristina Mamédio da Costa Santos, Cibele Andrucio de Mattos Pimenta y Moacyr Roberto Cuce Nobre. “The PICO strategy for the research question construction and evidence search”. En: *Revista latino-americana de enfermagem* 15.3 (2007), págs. 508-511.
- [19] Morgan Quigley et al. “ROS: an open-source Robot Operating System”. En: *ICRA workshop on open source software*. Vol. 3. 3.2. Kobe, Japan. 2009, pág. 5.
- [20] Giorgio Metta, Paul Fitzpatrick y Lorenzo Natale. “YARP: yet another robot platform”. En: *International Journal of Advanced Robotic Systems* 3.1 (2006), pág. 8.
- [21] SoftBank Robotics. “Naoqi developer guide”. En: *Línea*. Disponible en internet: [http://doc.aldebaran.com/2-8/index\\_dev\\_guide.html](http://doc.aldebaran.com/2-8/index_dev_guide.html) (2020).

- [22] Tore Dybå y Torgeir Dingsøy. “Empirical studies of agile software development: A systematic review”. En: *Information and software technology* 50.9-10 (2008), págs. 833-859.
- [23] Pablo Muñoz et al. “MoBAr: a Hierarchical Action-Oriented Autonomous Control Architecture”. En: *Journal of Intelligent & Robotic Systems* 94.3 (1 de jun. de 2019), págs. 745-760. ISSN: 1573-0409. DOI: 10.1007/s10846-018-0810-z. URL: <https://doi.org/10.1007/s10846-018-0810-z> (visitado 26-10-2023).
- [24] Sumant Pushp et al. “A cognitively enhanced collaborative control architecture for an intelligent wheelchair: Formalization, implementation and evaluation”. En: *Cognitive Systems Research* 49 (1 de jun. de 2018), págs. 114-127. ISSN: 1389-0417. DOI: 10.1016/j.cogsys.2017.12.004. URL: <https://www.sciencedirect.com/science/article/pii/S1389041717300293> (visitado 26-10-2023).
- [25] Antonio Chella et al. “Knowledge acquisition through introspection in Human-Robot Cooperation”. En: *Biologically Inspired Cognitive Architectures* 25 (1 de ago. de 2018), págs. 1-7. ISSN: 2212-683X. DOI: 10.1016/j.bica.2018.07.016. URL: <https://www.sciencedirect.com/science/article/pii/S2212683X18300847> (visitado 26-10-2023).
- [26] S. Hamidreza Kasaei et al. “Towards lifelong assistive robotics: A tight coupling between object perception and manipulation”. En: *Neurocomputing* 291 (24 de mayo de 2018), págs. 151-166. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2018.02.066. URL: <https://www.sciencedirect.com/science/article/pii/S0925231218302327> (visitado 26-10-2023).
- [27] Igor Rodriguez et al. “Robots on stage: A cognitive framework for socially interacting robots”. En: *Biologically Inspired Cognitive Architectures* 25 (1 de ago. de 2018), págs. 17-25. ISSN: 2212-683X. DOI: 10.1016/j.bica.2018.07.014. URL: <https://www.sciencedirect.com/science/article/pii/S2212683X18300707> (visitado 27-10-2023).
- [28] Francisco J. Rodríguez-Lera et al. “HiMoP: A three-component architecture to create more human-acceptable social-assistive robots”. En: *Cognitive*

- Processing* 19.2 (1 de mayo de 2018), págs. 233-244. ISSN: 1612-4790. DOI: 10.1007/s10339-017-0850-5. URL: <https://doi.org/10.1007/s10339-017-0850-5> (visitado 26-10-2023).
- [29] Tobias Fischer et al. “iCub-HRI: A Software Framework for Complex Human–Robot Interaction Scenarios on the iCub Humanoid Robot”. En: *Frontiers in Robotics and AI* 5 (2018). ISSN: 2296-9144. URL: <https://www.frontiersin.org/articles/10.3389/frobt.2018.00022> (visitado 26-10-2023).
- [30] Jauwairia Nasir, Deok-Hwa Kim y Jong-Hwan Kim. “ART neural network-based integration of episodic memory and semantic memory for task planning for robots”. En: *Autonomous Robots* 43.8 (1 de dic. de 2019), págs. 2163-2182. ISSN: 1573-7527. DOI: 10.1007/s10514-019-09868-x. URL: <https://doi.org/10.1007/s10514-019-09868-x> (visitado 26-10-2023).
- [31] P. Bustos et al. “The CORTEX cognitive robotics architecture: Use cases”. En: *Cognitive Systems Research* 55 (1 de jun. de 2019), págs. 107-123. ISSN: 1389-0417. DOI: 10.1016/j.cogsys.2019.01.003. URL: <https://www.sciencedirect.com/science/article/pii/S1389041717300347> (visitado 27-10-2023).
- [32] Riccardo Caccavale y Alberto Finzi. “Learning attentional regulations for structured tasks execution in robotic cognitive control”. En: *Autonomous Robots* 43.8 (1 de dic. de 2019), págs. 2229-2243. ISSN: 1573-7527. DOI: 10.1007/s10514-019-09876-x. URL: <https://doi.org/10.1007/s10514-019-09876-x> (visitado 26-10-2023).
- [33] Dalia Marcela Rojas-Castro, Arnaud Revel y Michel Menard. “RHIZOME ARCHITECTURE: An Adaptive Neurobehavioral Control Architecture for Cognitive Mobile Robots—Application in a Vision-Based Indoor Robot Navigation Context”. En: *International Journal of Social Robotics* 12.3 (jul. de 2020), págs. 659-688. ISSN: 1875-4791, 1875-4805. DOI: 10.1007/s12369-019-00602-2. URL: <http://link.springer.com/10.1007/s12369-019-00602-2> (visitado 27-10-2023).

- [34] Pourya Aliasghari et al. “Implementing a gaze control system on a social robot in multi-person interactions”. En: *SN Applied Sciences* 2.6 (28 de mayo de 2020), pág. 1135. ISSN: 2523-3971. DOI: 10.1007/s42452-020-2911-0. URL: <https://doi.org/10.1007/s42452-020-2911-0> (visitado 26-10-2023).
- [35] Ricardo Gudwin et al. “The TROCA Project: An autonomous transportation robot controlled by a cognitive architecture”. En: *Cognitive Systems Research* 59 (1 de ene. de 2020), págs. 179-197. ISSN: 1389-0417. DOI: 10.1016/j.cogsys.2019.09.011. URL: <https://www.sciencedirect.com/science/article/pii/S1389041719304772> (visitado 26-10-2023).
- [36] Arianna Pipitone y Antonio Chella. “What robots want? Hearing the inner voice of a robot”. En: *iScience* 24.4 (23 de abr. de 2021), pág. 102371. ISSN: 2589-0042. DOI: 10.1016/j.isci.2021.102371. URL: <https://www.sciencedirect.com/science/article/pii/S2589004221003394> (visitado 26-10-2023).
- [37] Francisco Martín et al. “Client-Server Approach for Managing Visual Attention, Integrated in a Cognitive Architecture for a Social Robot”. En: *Frontiers in Neurorobotics* 15 (2021). ISSN: 1662-5218. URL: <https://www.frontiersin.org/articles/10.3389/fnbot.2021.630386> (visitado 26-10-2023).
- [38] John Páez y Enrique González. “Human-Robot Scaffolding: An Architecture to Foster Problem-solving Skills”. En: *ACM Transactions on Human-Robot Interaction* 11.3 (13 de jul. de 2022), 24:1-24:17. DOI: 10.1145/3526109. URL: <https://dl.acm.org/doi/10.1145/3526109> (visitado 26-10-2023).
- [39] Jonatan Ginés et al. “Depicting probabilistic context awareness knowledge in deliberative architectures”. En: *Natural Computing: an international journal* 21.4 (1 de dic. de 2022), págs. 565-576. ISSN: 1567-7818. DOI: 10.1007/s11047-020-09798-z. URL: <https://doi.org/10.1007/s11047-020-09798-z> (visitado 26-10-2023).
- [40] Jonatan Ginés Clavero et al. “Impact of decision-making system in social navigation”. En: *Multimedia Tools and Applications* 81.3 (1 de ene. de 2022),

- págs. 3459-3481. ISSN: 1573-7721. DOI: 10.1007/s11042-021-11454-2. URL: <https://doi.org/10.1007/s11042-021-11454-2> (visitado 26-10-2023).
- [41] Alessandro Umbrico et al. "A Mind-inspired Architecture for Adaptive HRI". En: *International Journal of Social Robotics* 15.3 (1 de mar. de 2023), págs. 371-391. ISSN: 1875-4805. DOI: 10.1007/s12369-022-00897-8. URL: <https://doi.org/10.1007/s12369-022-00897-8> (visitado 26-10-2023).
- [42] Rémi Dromnelle et al. "Reducing Computational Cost During Robot Navigation and Human-Robot Interaction with a Human-Inspired Reinforcement Learning Architecture". En: *International Journal of Social Robotics* 15.8 (ago. de 2023), págs. 1297-1323. ISSN: 1875-4791, 1875-4805. DOI: 10.1007/s12369-022-00942-6. URL: <https://link.springer.com/10.1007/s12369-022-00942-6> (visitado 27-10-2023).
- [43] David Loza-Matovelle et al. "Task Planning System with Priority for AAL Environments". En: *Journal of Intelligent & Robotic Systems* 107.2 (25 de ene. de 2023), pág. 19. ISSN: 1573-0409. DOI: 10.1007/s10846-023-01806-5. URL: <https://doi.org/10.1007/s10846-023-01806-5> (visitado 27-10-2023).
- [44] Agnese Augello et al. "Roboception and adaptation in a cognitive robot". En: *Robotics and Autonomous Systems* 164 (1 de jun. de 2023), pág. 104400. ISSN: 0921-8890. DOI: 10.1016/j.robot.2023.104400. URL: <https://www.sciencedirect.com/science/article/pii/S0921889023000398> (visitado 27-10-2023).
- [45] James Male y Uriel Martinez-Hernandez. "Deep learning based robot cognitive architecture for collaborative assembly tasks". En: *Robotics and Computer-Integrated Manufacturing* 83 (1 de oct. de 2023), pág. 102572. ISSN: 0736-5845. DOI: 10.1016/j.rcim.2023.102572. URL: <https://www.sciencedirect.com/science/article/pii/S0736584523000480> (visitado 26-10-2023).

- [46] John R Anderson, Michael Matessa y Christian Lebiere. “ACT-R: A theory of higher level cognition and its relation to visual attention”. En: *Human-Computer Interaction* 12.4 (1997), págs. 439-462.
- [47] Sachin Chitta, Ioan Sucan y Steve Cousins. “Moveit![ros topics]”. En: *IEEE Robotics & Automation Magazine* 19.1 (2012), págs. 18-19.
- [48] Michael Cashmore et al. “ROSplan: Planning in the robot operating system”. En: *Proceedings International Conference on Automated Planning and Scheduling, ICAPS*. Vol. 2015. Ene. de 2015, págs. 333-341.
- [49] Francisco Martín et al. “PlanSys2: A Planning System Framework for ROS2”. En: *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2021, Prague, Czech Republic, September 27 - October 1, 2021*. IEEE, 2021.
- [50] Richard S Sutton, Doina Precup y Satinder Singh. “Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning”. En: *Artificial intelligence* 112.1-2 (1999), págs. 181-211.
- [51] Richard SJ Frackowiak. *Human brain function*. Elsevier, 2004.
- [52] Bryan Kolb, Ian Q Whishaw y G Campbell Teskey. *An introduction to brain and behavior*. Worth New York, 2001.
- [53] Geraldine Ed Dawson y Kurt W Fischer. *Human behavior and the developing brain*. The Guilford Press, 1994.
- [54] Glenn Jocher, Ayush Chaurasia y Jing Qiu. *YOLO by Ultralytics*. Ver. 8.0.0. Ene. de 2023. URL: <https://github.com/ultralytics/ultralytics>.
- [55] Miguel Á. González-Santamarta et al. “YASMIN: Yet Another State Machine”. En: *ROBOT2022: Fifth Iberian Robotics Conference*. Ed. por Danilo Tardioli et al. Cham: Springer International Publishing, 2023, págs. 528-539. ISBN: 978-3-031-21062-4.
- [56] Miguel Á. González-Santamarta et al. “KANT: A Tool for Grounding and Knowledge Management”. En: *Bio-inspired Systems and Applications: from Robotics to Ambient Intelligence*. Ed. por José Manuel Ferrández Vi-

- cente et al. Cham: Springer International Publishing, 2022, págs. 452-461. ISBN: 978-3-031-06527-9.
- [57] Stevan Harnad. “The symbol grounding problem”. En: *Physica D: Nonlinear Phenomena* 42.1-3 (1990), págs. 335-346.
- [58] Paul Vogt. “The physical symbol grounding problem”. En: *Cognitive Systems Research* 3.3 (2002), págs. 429-457.
- [59] Yang Xiao. “Flow-net methodology for accountability in wireless networks”. En: *IEEE network* 23.5 (2009), págs. 30-37.
- [60] Daniel DesRuisseaux. “Practical Overview of Implementing IEC 62443 Security Levels in Industrial Control Applications”. En: *Schneider Electric White Paper* (2018).
- [61] Francisco Javier Rodríguez-Lera et al. “Towards explainability in robotics: A performance analysis of a cloud accountability system”. En: *Expert Systems* 39.9 (2022), e13004.
- [62] Anders Oreback. “Components in intelligent robotics”. En: *MRTC report ISSN* (1999), págs. 1404-3041.
- [63] Zhifeng Xiao, Nandhakumar Kathiresshan y Yang Xiao. “A survey of accountability in computer networks and distributed systems”. En: *Security and Communication Networks* 9.4 (2016), págs. 290-315.
- [64] Vicente Matellán et al. “The Role of Cybersecurity and HPC in the Explainability of Autonomos Robots Behavior”. En: *2021 IEEE International Conference on Advanced Robotics and Its Social Impacts (ARSO)*. IEEE. 2021.
- [65] Miguel A González-Santamarta et al. “Using Large Language Models for Interpreting Autonomous Robots Behaviors”. En: *arXiv preprint arXiv:2304.14844* (2023).
- [66] Miguel Á. González-Santamarta et al. “MERLIN a Cognitive Architecture for Service Robots”. En: *Applied Sciences* 10.17 (2020), pág. 5989. DOI: 10.3390/app10175989. URL: <https://doi.org/10.3390/app10175989>.



- [67] Jonathan Bohren y Steve Cousins. “The SMACH high-level executive”. En: *IEEE Robotics & Automation Magazine* 17.4 (2011), págs. 18-20. doi: 10.1109/MRA.2010.938836.
- [68] Joseph Redmon y Ali Farhadi. “YOLOv3: An Incremental Improvement”. En: *arXiv* (2018).
- [69] Jonathan Bohren. *SMACH Viewer*. [http://wiki.ros.org/smach\\_viewer](http://wiki.ros.org/smach_viewer).
- [70] Ayoub Benali Amjoud y Mustapha Amrouch. “Object Detection Using Deep Learning, CNNs and Vision Transformers: A Review”. En: *IEEE Access* (2023).
- [71] Alex Krizhevsky, Ilya Sutskever y Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. En: *Communications of the ACM* 60.6 (2017), págs. 84-90.
- [72] Kaiming He et al. “Deep residual learning for image recognition”. En: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, págs. 770-778.
- [73] Shaoqing Ren et al. “Faster r-cnn: Towards real-time object detection with region proposal networks”. En: *Advances in neural information processing systems* 28 (2015).
- [74] Joseph Redmon et al. “You only look once: Unified, real-time object detection”. En: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, págs. 779-788.
- [75] Nicolas Carion et al. “End-to-end object detection with transformers”. En: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I* 16. Springer. 2020, págs. 213-229.
- [76] Miguel Á. González-Santamarta. *yolov8\_ros*. Feb. de 2023. URL: [https://github.com/mgonzs13/yolov8%5C\\_ros](https://github.com/mgonzs13/yolov8%5C_ros).
- [77] Yifu Zhang et al. “Bytetrack: Multi-object tracking by associating every detection box”. En: *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXII*. Springer. 2022, págs. 1-21.

- [78] Virginia Riego del Castillo et al. “Vision-based module for herding with a sheepdog robot”. En: *Sensors* 22.14 (2022), pág. 5321.
- [79] Brett Aldrich. *State machine asynchronous C++*. URL: <https://smacc.dev/>.
- [80] Miguel Ángel González Santamarta et al. “MERLIN2: Sistema cognitivo para ROS 2”. En: (2022).
- [81] Malik Ghallab, Dana Nau y Paolo Traverso. *Automated Planning: theory and practice*. Elsevier, 2004.
- [82] Paul B Monday. “Implementing the Data Transfer Object Pattern”. En: *Web Services Patterns: Java™ Platform Edition*. Springer, 2003, págs. 279-295.
- [83] Clifton Nock. *Data access patterns: database interactions in object-oriented applications*. Addison-Wesley Boston, 2004.
- [84] S. Bradshaw, K. Chodorow y E. Brazil. *MongoDB: the Definitive Guide: Powerful and Scalable Data Storage*. The expert’s voice in open source. O’Reilly Media, Incorporated, 2019. ISBN: 9781491954461. URL: <https://books.google.es/books?id=ohGAvGAAAJ>.
- [85] MongoDB Compass. <https://www.mongodb.com/products/compass>. 2023.
- [86] Anis Koubaa. “ROSGPT: Next-Generation Human-Robot Interaction with ChatGPT and ROS”. En: (2023).
- [87] Aadi Palnitkar et al. “ChatSim: Underwater Simulation with Natural Language Prompting”. En: *arXiv preprint arXiv:2308.04029* (2023).
- [88] Mohamed Lamine TAZIR, Matei MANCAS y Thierry DUTOIT. “From Words to Flight: Integrating OpenAI ChatGPT with PX4/Gazebo for Natural Language-Based Drone Control”. En: ()
- [89] Chenliang Li et al. “ModelScope-Agent: Building Your Customizable Agent System with Open-source Large Language Models”. En: *arXiv preprint arXiv:2309.00986* (2023).
- [90] Zhengbao Jiang et al. “X-FACTR: Multilingual factual knowledge retrieval from pretrained language models”. En: *arXiv preprint arXiv:2010.06189* (2020).

- [91] Stephan Linzbach et al. “Decoding Prompt Syntax: Analysing its Impact on Knowledge Retrieval in Large Language Models”. En: *Companion Proceedings of the ACM Web Conference 2023*. 2023, págs. 1145-1149.
- [92] Georgia Chalvatzaki et al. “Learning to reason over scene graphs: a case study of finetuning GPT-2 into a robot language model for grounded task planning”. En: *Frontiers in Robotics and AI* 10 (2023). Cited by: 0; All Open Access, Gold Open Access, Green Open Access. DOI: 10.3389/frobot.2023.1221739.
- [93] Tom Silver et al. “Generalized Planning in PDDL Domains with Pretrained Large Language Models”. En: *arXiv preprint arXiv:2305.11014* (2023).
- [94] Ishika Singh et al. “ProgPrompt: Generating Situated Robot Task Plans using Large Language Models”. En: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. 2023, págs. 11523-11530. DOI: 10.1109/ICRA48891.2023.10161317.
- [95] Boyuan Chen et al. “Open-vocabulary Queryable Scene Representations for Real World Planning”. En: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. 2023, págs. 11509-11522. DOI: 10.1109/ICRA48891.2023.10161534.
- [96] Miguel Á. González-Santamarta et al. “Using Large Language Models for Interpreting Autonomous Robots Behaviors”. En: *Hybrid Artificial Intelligent Systems*. Ed. por Pablo García Bringas et al. Cham: Springer Nature Switzerland, 2023, págs. 533-544. ISBN: 978-3-031-40725-3.
- [97] Jason Wei et al. “Chain-of-thought prompting elicits reasoning in large language models”. En: *Advances in Neural Information Processing Systems* 35 (2022), págs. 24824-24837.
- [98] Takeshi Kojima et al. *Large Language Models are Zero-Shot Reasoners*. 2023. arXiv: 2205.11916 [cs.CL].
- [99] Miguel Á. González-Santamarta. *llama\_ros*. Abr. de 2023. URL: [https://github.com/mgonzs13/llama%5C\\_ros](https://github.com/mgonzs13/llama%5C_ros).

- [100] *GitHub - ggerganov/llama.cpp: Port of Facebook's LLaMA model in C/C++ — github.com.* <https://github.com/ggerganov/llama.cpp>. [Accessed 12-09-2023].
- [101] OpenAI. *GPT-4 Technical Report.* <https://arxiv.org/abs/2303.08774>. 2023. arXiv: 2303.08774 [cs.CL].
- [102] Hugo Touvron et al. *LLaMA: Open and Efficient Foundation Language Models.* 2023. arXiv: 2302.13971 [cs.CL].
- [103] Hugo Touvron et al. *Llama 2: Open Foundation and Fine-Tuned Chat Models.* 2023. arXiv: 2307.09288 [cs.CL].
- [104] Hao Wu et al. "Integer quantization for deep learning inference: Principles and empirical evaluation". En: *arXiv preprint arXiv:2004.09602* (2020).
- [105] Zhaowei Cai et al. "Deep learning with low precision by half-wave gaussian quantization". En: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2017, págs. 5918-5926.
- [106] Zhouhan Lin et al. "Neural networks with few multiplications". En: *arXiv preprint arXiv:1510.03009* (2015).
- [107] Darryl Lin, Sachin Talathi y Sreekanth Annapureddy. "Fixed point quantization of deep convolutional networks". En: *International conference on machine learning.* PMLR. 2016, págs. 2849-2858.
- [108] Rohan Taori et al. *Stanford Alpaca: An Instruction-following LLaMA model.* [https://github.com/tatsu-lab/stanford\\_alpaca](https://github.com/tatsu-lab/stanford_alpaca). 2023.
- [109] Lianmin Zheng et al. *Judging LLM-as-a-judge with MT-Bench and Chatbot Arena.* 2023. arXiv: 2306.05685 [cs.CL].
- [110] *GitHub - jondurbin/airoboros: Customizable implementation of the self-instruct paper. — github.com.* <https://github.com/jondurbin/airoboros>. [Accessed 11-09-2023].
- [111] Can Xu et al. *WizardLM: Empowering Large Language Models to Follow Complex Instructions.* 2023. arXiv: 2304.12244 [cs.CL].

- [112] *NousResearch/Nous-Hermes-13b · Hugging Face* — *huggingface.co*. <https://huggingface.co/NousResearch/Nous-Hermes-13b>. [Accessed 11-09-2023].
- [113] *AIDC-ai-business/Marcoroni-13B · Hugging Face* — *huggingface.co*. <https://huggingface.co/AIDC-ai-business/Marcoroni-13B>. [Accessed 13-09-2023].
- [114] Albert Q. Jiang et al. *Mistral 7B*. 2023. arXiv: 2310.06825 [cs.CL].
- [115] Lin Guan et al. *Leveraging Pre-trained Large Language Models to Construct and Utilize World Models for Model-based Task Planning*. 2023. arXiv: 2305.14909 [cs.AI].
- [116] Chan Hee Song et al. *LLM-Planner: Few-Shot Grounded Planning for Embodied Agents with Large Language Models*. 2023. arXiv: 2212.04088 [cs.AI].
- [117] Shunyu Yao et al. “Tree of thoughts: Deliberate problem solving with large language models”. En: *arXiv preprint arXiv:2305.10601* (2023).
- [118] Maciej Besta et al. “Graph of Thoughts: Solving Elaborate Problems with Large Language Models”. En: *arXiv preprint arXiv:2308.09687* (2023).
- [119] Tom Silver et al. “PDDL planning with pretrained large language models”. En: *NeurIPS 2022 Foundation Models for Decision Making Workshop*. 2022.
- [120] Ishika Singh et al. “ProgPrompt: program generation for situated robot task planning using large language models”. En: *Autonomous Robots* (2023), págs. 1-14.
- [121] Nan Li, Bo Kang y Tijn De Bie. “SkillGPT: a RESTful API service for skill extraction and standardization using a Large Language Model”. En: *arXiv preprint arXiv:2304.11060* (2023).
- [122] Udit Gupta. “GPT-InvestAR: Enhancing Stock Investment Strategies through Annual Report Analysis with Large Language Models”. En: *arXiv preprint arXiv:2309.03079* (2023).
- [123] Patrick Lewis et al. “Retrieval-augmented generation for knowledge-intensive nlp tasks”. En: *Advances in Neural Information Processing Systems* 33 (2020), págs. 9459-9474.

- [124] Michael Cashmore et al. “A compilation of the full PDDL+ language into SMT”. En: *Workshops at the Thirtieth AAAI Conference on Artificial Intelligence*. 2016.
- [125] aiplan4eu. *AIPLAN4EU/unified-planning: The AIPLAN4EU Unified Planning Library*. URL: <https://github.com/aiplan4eu/unified-planning>.
- [126] Håkan LS Younes y Reid G Simmons. “VHPOP: Versatile heuristic partial order planner”. En: *Journal of Artificial Intelligence Research* 20 (2003), págs. 405-430.
- [127] Miguel Á. González-Santamarta et al. *Integration of Large Language Models within Cognitive Architectures for Autonomous Robots*. 2023. arXiv: 2309.14945 [cs.R0].
- [128] Harrison Chase. *LangChain*. Oct. de 2022. URL: <https://github.com/hwchase17/langchain>.
- [129] Michael O’Neill et al. *Grammatical evolution*. Springer, 2003.
- [130] *The AI-native open-source embedding database — trychroma.com*. <https://www.trychroma.com/>. [Accessed 13-09-2023].
- [131] Michele Colledanchise y Lorenzo Natale. “On the implementation of behavior trees in robotics”. En: *IEEE Robotics and Automation Letters* 6.3 (2021), págs. 5929-5936.
- [132] *GitHub - BehaviorTree/Groot: Graphical Editor to create BehaviorTrees. Compliant with BehaviorTree.CPP — github.com*. <https://github.com/BehaviorTree/Groot>. [Accessed 06-11-2023].
- [133] Steve Macenski et al. “The Marathon 2: A Navigation System”. En: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020. URL: <https://github.com/ros-planning/navigation2>.
- [134] Meysam Basiri et al. “Benchmarking Functionalities of Domestic Service Robots Through Scientific Competitions”. En: *KI-Künstliche Intelligenz* 33.4 (2019), págs. 357-367.

- [135] Meysam Basiri et al. *Rulebook of the European robotic league for consumer service robots*. 2018. URL: [https://eu-robotics.net/robotics\\_league/erl-consumer](https://eu-robotics.net/robotics_league/erl-consumer).
- [136] *Home - ROBOCUP 2023* — [2023.robotcup.org](https://2023.robotcup.org/en/home/). <https://2023.robotcup.org/en/home/>. [Accessed 06-11-2023].
- [137] Aws-Robotics. *AWS-Robotics/AWS-robomaker-small-warehouse-world at foxy-devel*. URL: <https://github.com/aws-robotics/aws-robomaker-small-warehouse-world/tree/foxy-devel>.
- [138] *Psutil*. URL: <https://github.com/giampaolo/psutil>.
- [139] Victor Schmidt et al. *CodeCarbon: Estimate and Track Carbon Emissions from Machine Learning Computing*. <https://github.com/mlco2/codecarbon>. Mayo de 2021. DOI: 10.5281/zenodo.4658424.
- [140] Miguel Á. González-Santamarta et al. “MERLIN2: MachinEd Ros 2 pLanINg”. En: *Software Impacts* 15 (2023), pág. 100477. ISSN: 2665-9638. DOI: <https://doi.org/10.1016/j.simpa.2023.100477>. URL: <https://www.sciencedirect.com/science/article/pii/S2665963823000143>.





# Apéndice A

## Diagramas de clases UML

En este apéndice se presentan los diagramas UML de MERLIN2 y sus dos herramientas base, YASMIN y KANT.

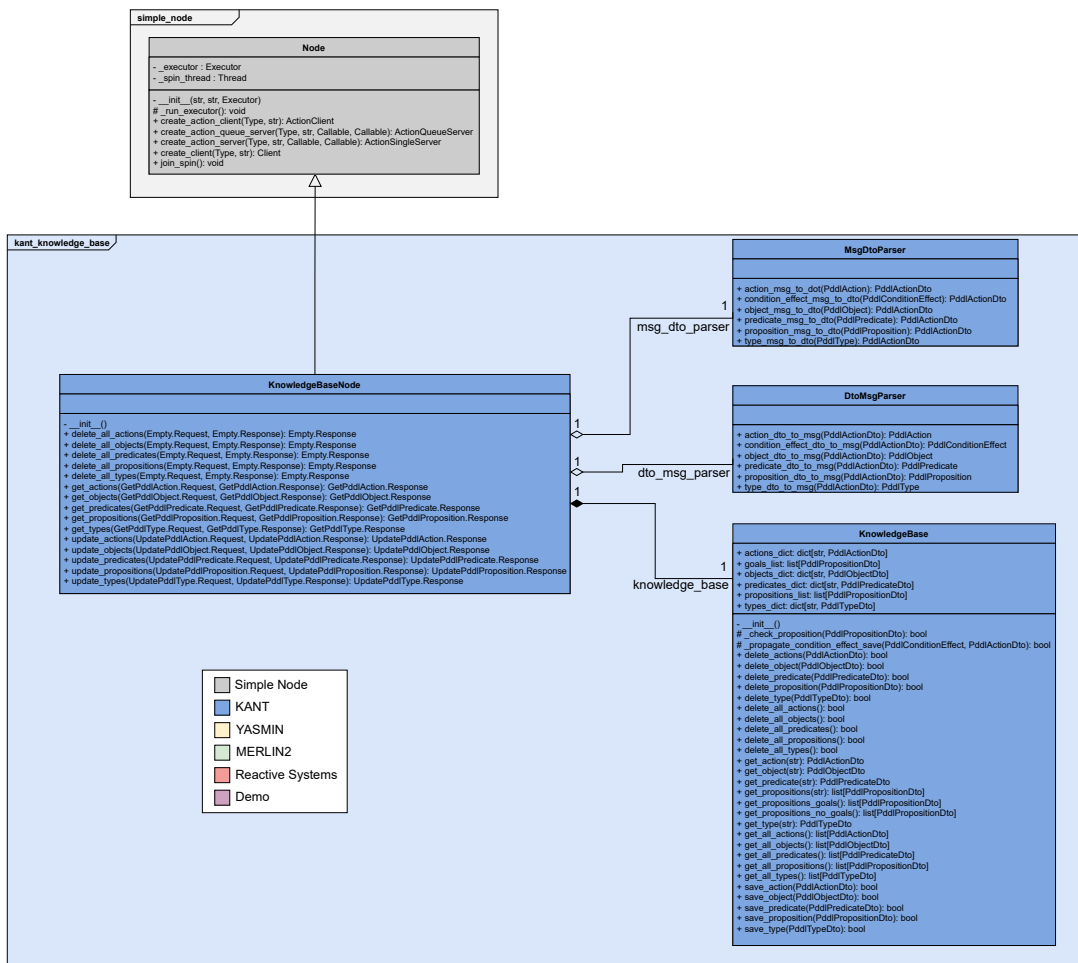


Figura A.1 – Diagrama de clases del paquete de ROS 2 kant\_knowledge\_base.

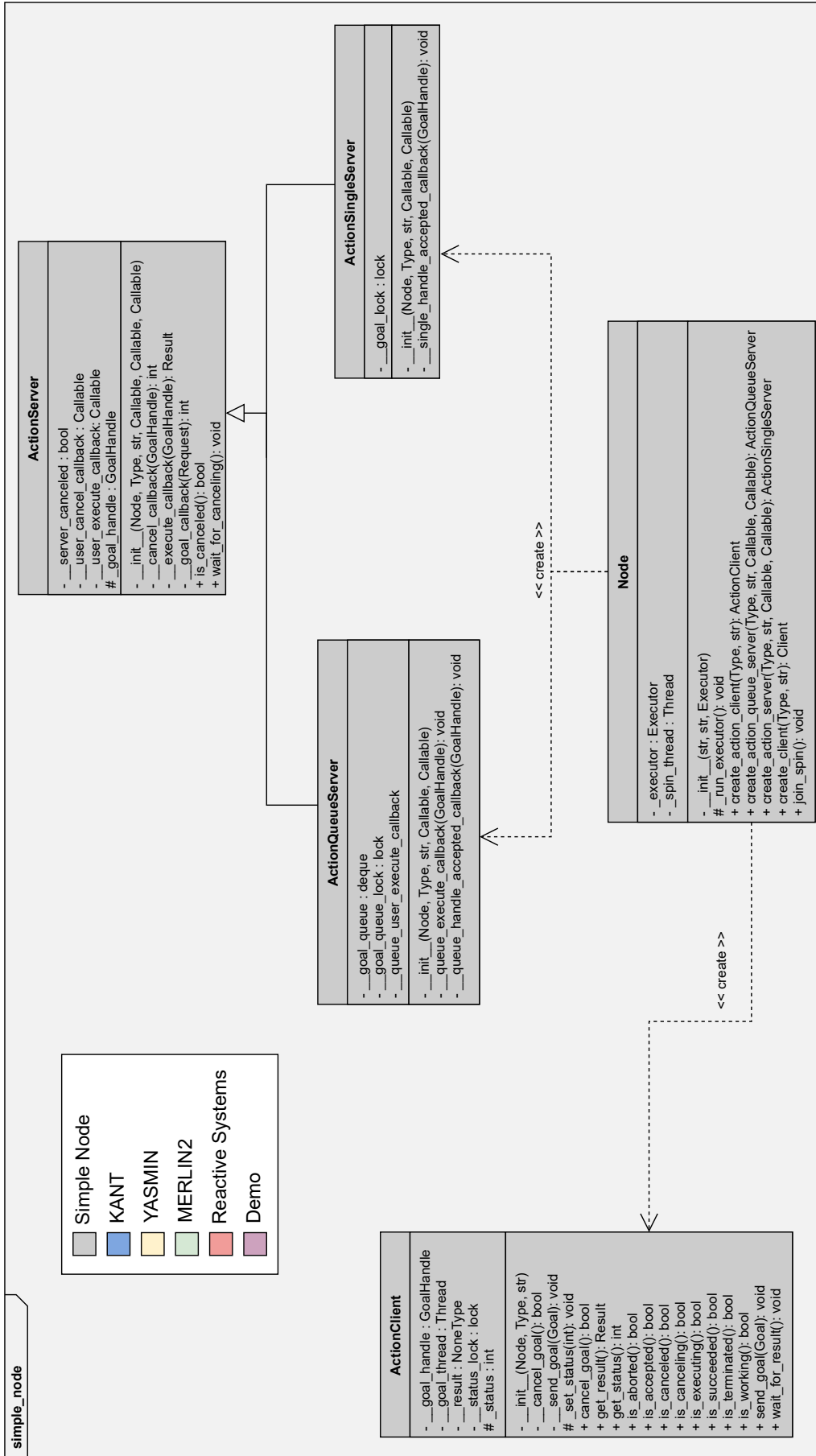


Figura A.2 – Diagrama de clases del paquete de ROS 2 simple\_node.

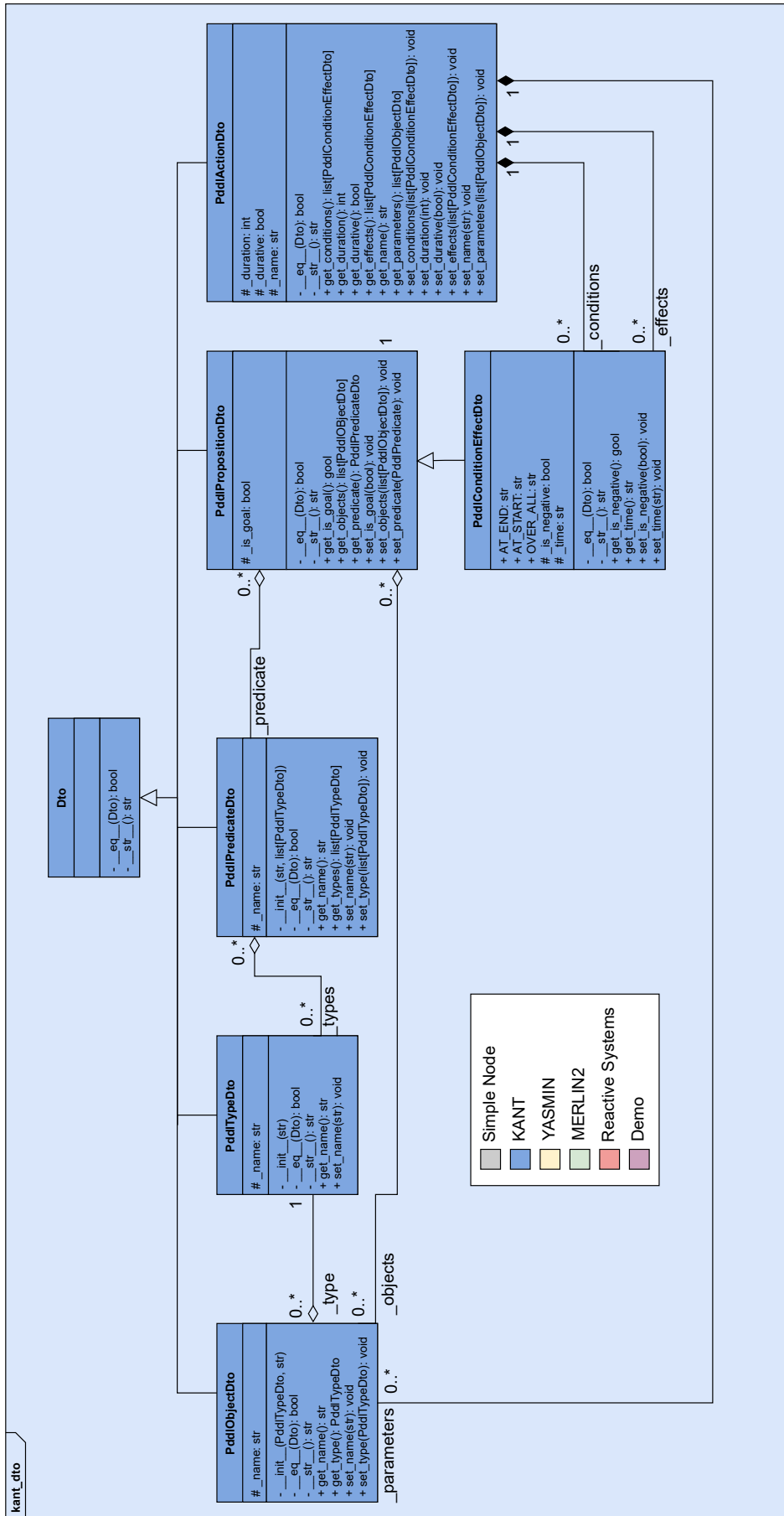


Figura A.3 – Diagrama de clases del paquete de ROS 2 kant\_dto.

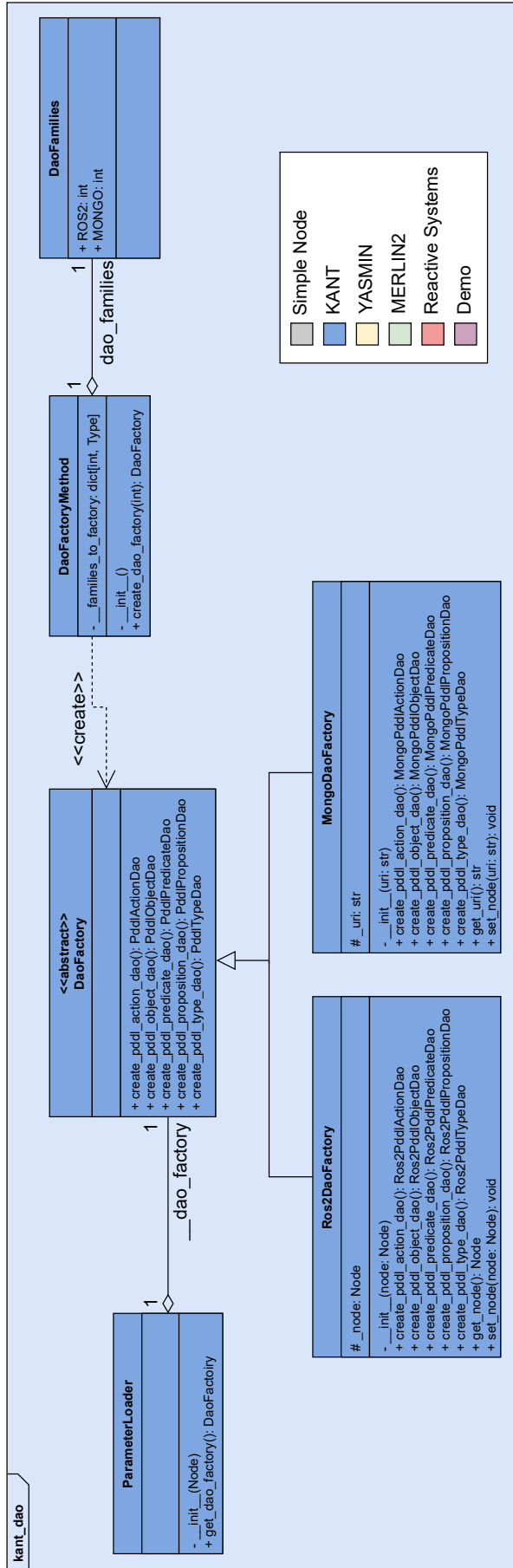


Figura A.4 – Diagrama de clases del paquete de ROS 2 dao\_factory de KANT.

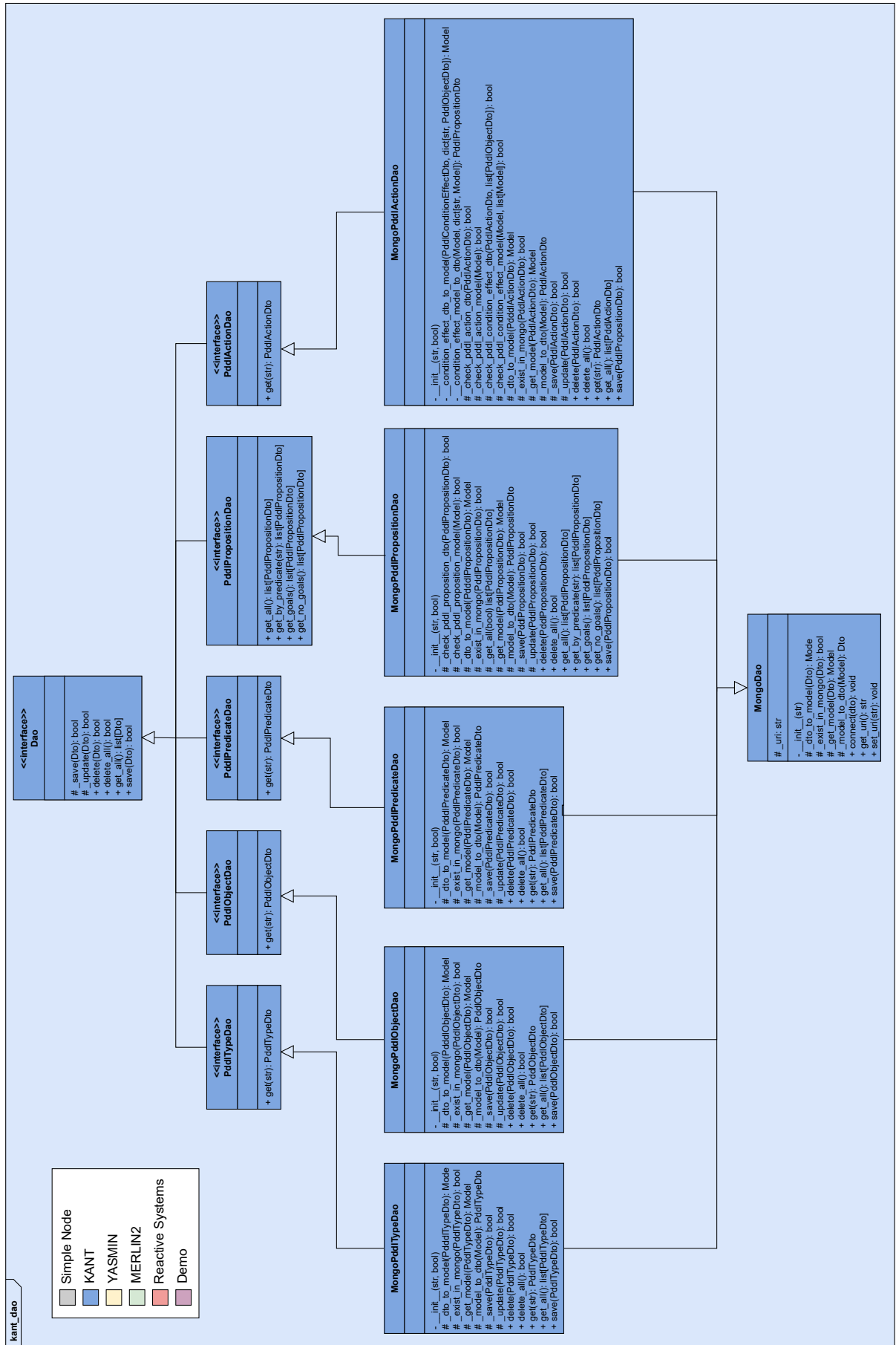


Figura A.5 – Diagrama de clases del paquete de ROS 2 mongo\_dao de KANT.

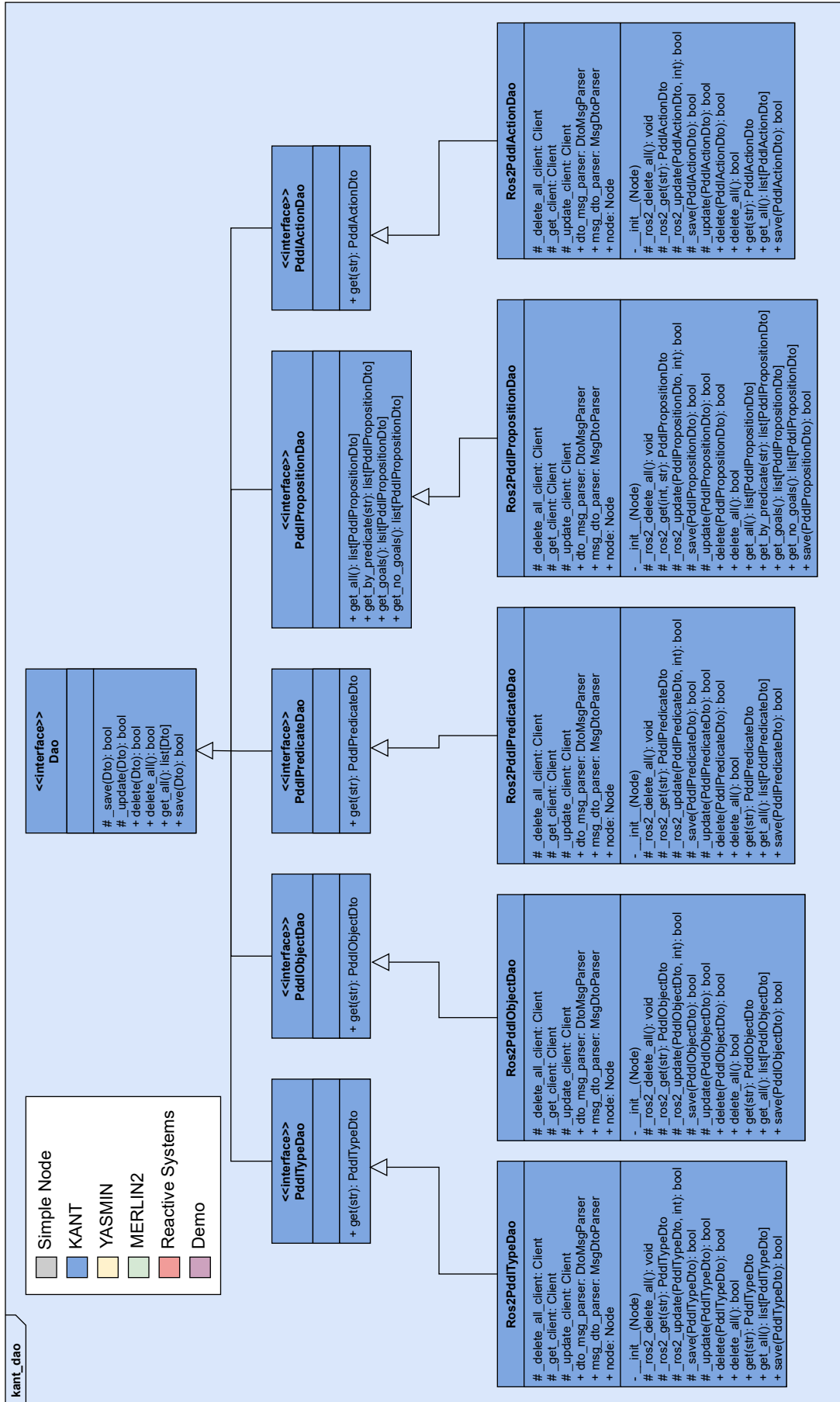


Figura A.6 – Diagrama de clases del paquete de ROS 2 ros2\_dao de KANT.







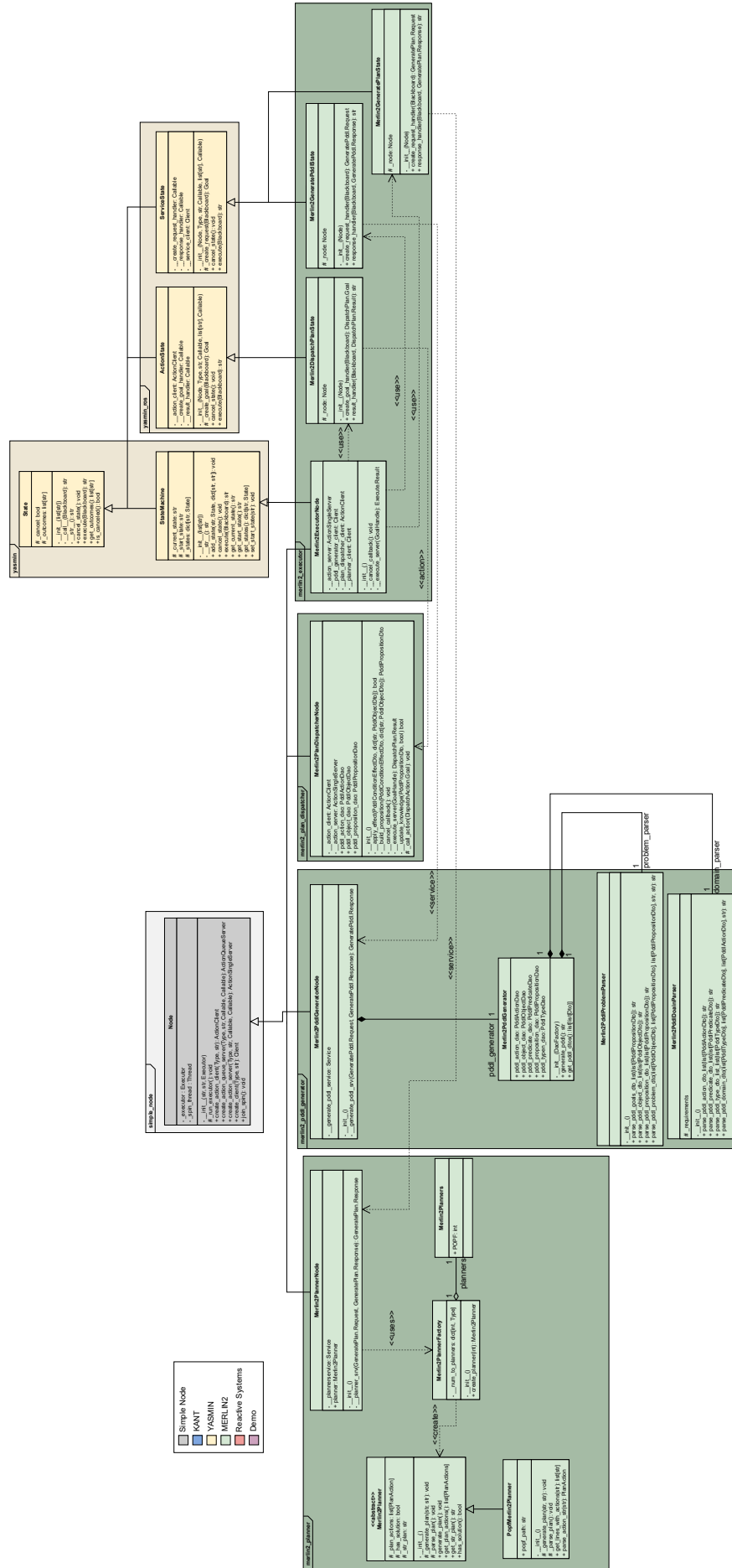


Figura A.9 – Diagrama de classes de la Planning Layer de MERLIN2.





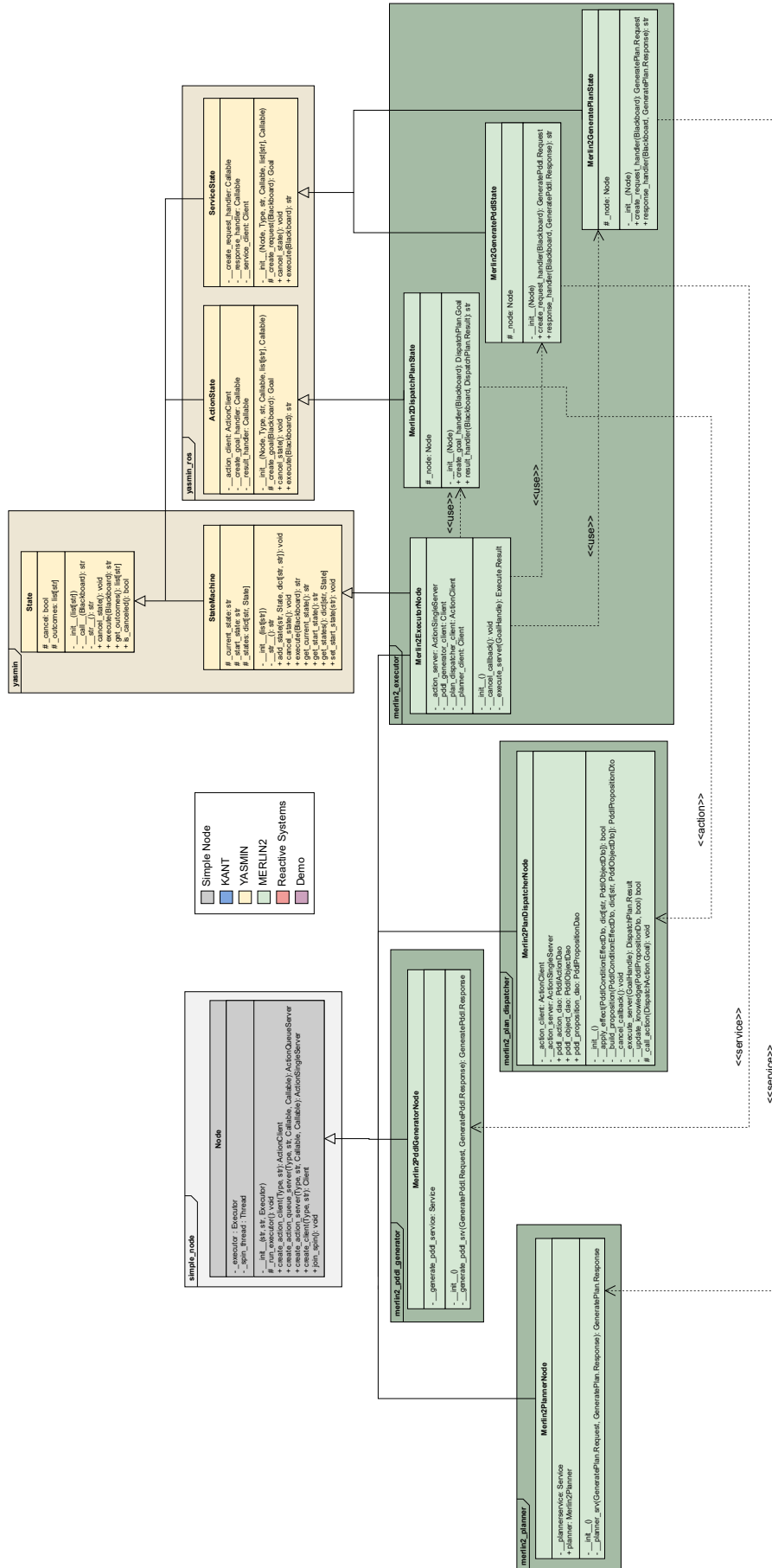


Figura A.12 – Diagrama de clases del *Executor* de MERLIN2.





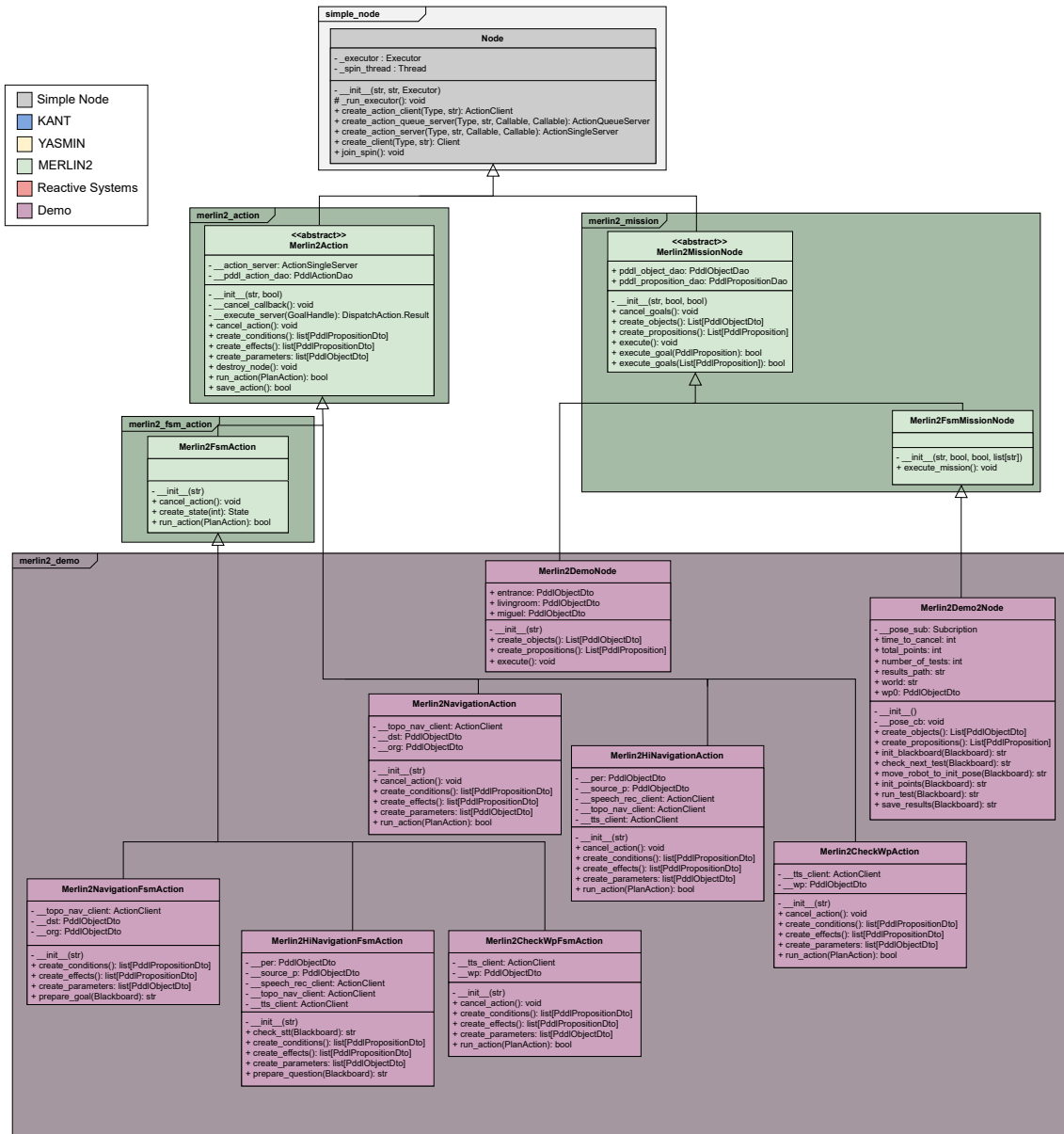


Figura A.15 – Diagrama de clases de la demo de MERLIN2.





# Apéndice B

## Repositorios públicos

En este apéndice se presentan los repositorios públicos de GitHub que se han obtenido como resultado de este trabajo de investigación. Así, la Tabla B.1 resume estos repositorios y sus datos. Se tienen los siguientes repositorios públicos de GitHub:

**Tabla B.1** – Repositorios públicos de GitHub creados durante el desarrollo de este trabajo de investigación.

Nombre	Estrellas	Forks	Licencia	URL
MERLIN2	11	2	GPL-3.0	<a href="https://github.com/MERLIN2-ARCH/merlin2">https://github.com/MERLIN2-ARCH/merlin2</a>
simple_node	4	1	GPL-3.0	<a href="https://github.com/uleroboticsgroup/simple_node">https://github.com/uleroboticsgroup/simple_node</a>
waypoint_navigation	0	0	GPL-3.0	<a href="https://github.com/MERLIN2-ARCH/waypoint_navigation">https://github.com/MERLIN2-ARCH/waypoint_navigation</a>
speech_to_text	1	0	GPL-3.0	<a href="https://github.com/MERLIN2-ARCH/speech_to_text">https://github.com/MERLIN2-ARCH/speech_to_text</a>

text_to_speech	0	1	GPL-3.0	<a href="https://github.com/MERLIN2-ARCH/text_to_speech">https://github.com/MERLIN2-ARCH/text_to_speech</a>
YASMIN	66	12	GPL-3.0	<a href="https://github.com/uleroboticsgroup/yasmin">https://github.com/uleroboticsgroup/yasmin</a>
KANT	5	3	GPL-3.0	<a href="https://github.com/uleroboticsgroup/kant">https://github.com/uleroboticsgroup/kant</a>
knowledge_graph	0	0	Apache-2	<a href="https://github.com/mgonzs13/knowledge_graph">https://github.com/mgonzs13/knowledge_graph</a>
ros2_rb1	4	1	GPL-3.0	<a href="https://github.com/mgonzs13/ros2_rb1">https://github.com/mgonzs13/ros2_rb1</a>
ros2_rover	35	2	MIT	<a href="https://github.com/mgonzs13/ros2_rover">https://github.com/mgonzs13/ros2_rover</a>
yolov8_ros	80	14	GPL-3.0	<a href="https://github.com/mgonzs13/yolov8_ros">https://github.com/mgonzs13/yolov8_ros</a>
yolonas_ros	12	1	GPL-3.0	<a href="https://github.com/mgonzs13/yolonas_ros">https://github.com/mgonzs13/yolonas_ros</a>
llama_ros	58	9	MIT	<a href="https://github.com/mgonzs13/llama_ros">https://github.com/mgonzs13/llama_ros</a>
audio_common	3	0	MIT	<a href="https://github.com/mgonzs13/audio_common.git">https://github.com/mgonzs13/audio_common.git</a>
whisper_ros	10	0	MIT	<a href="https://github.com/mgonzs13/whisper_ros">https://github.com/mgonzs13/whisper_ros</a>

# Apéndice C

## Prompts utilizados

En este apéndice se presentan los *prompts* utilizados en la integración de LLMs en la arquitectura cognitiva MERLIN2. El primer *prompt*, que se presenta en las figuras C.1 y C.2, se utiliza para llevar a cabo la planificación. Está formado por unas reglas básicas que explican que el LLM se tiene que comportar como un planificador, el formato en el que tiene que responder, las acciones del robot, el estado del mundo y el objetivo.

Por otro lado, en la Figura C.3, se presenta la respuesta que el LLM produce al *prompt* de planificación. En este caso, la respuesta está formada por un *reasoning*, en el que el LLM razona sobre el estado del mundo y el objetivo; un plan en lenguaje natural; y una lista de acciones que se corresponden con el plan que el robot tiene que seguir para alcanzar el objetivo.

Por último, se tiene el *prompt* para verificar si el objetivo se cumple en el estado del mundo del robot. Este *prompt* se presenta en la Figura C.4. Está formado por unas reglas básicas que explican que el LLM se tiene que comportar como un verificador de objetivos, el formato en el que tiene que responder, el estado del mundo del robot y el objetivo.

```

You are an AI planner for a robot named RB1 whose role is assistant.
You have to generate a plan (sequence of actions) to achieve a goal.
The plan may contain no actions, one action, or several actions.
Use ONLY the actions listed below with the specified format.
Use the fewest actions you can to generate the plan.
If the goal is already achieved in the world state, return an empty sequence of actions.

You can ONLY respond with the following JSON format:
{
  "reasoning": "reasoning about the world state and the goal",
  "plan": "plan to achieve the goal (in natural language)",
  "actions": [
    {
      "name": "action name",
      "args": "action args"
    }
  ]
}

ACTIONS:
- move_to: Robot action to move from one waypoint to another, args json schema: {
  "robot_name": {
    "title": "Robot Name",
    "type": "string"
  },
  "source_waypoint": {
    "title": "Source Waypoint",
    "type": "string"
  },
  "target_waypoint": {
    "title": "Target Waypoint",
    "type": "string"
  }
}
- greet_person: Robot action to greet a person, the robot and the person must be at the same waypoint, args json schema: {
  "robot_name": {
    "title": "Robot Name",
    "type": "string"
  },
  "person_to_greet": {
    "title": "Person To Greet",
    "type": "string"
  },
  "person_waypoint": {
    "title": "Person Waypoint",
    "type": "string"
  }
}
}

```

Figura C.1 – Prompt para planificar (I).

```

WORLD STATE:
- GrannyHouse is a house
- entrance is a waypoint (x: 0.23, y: 0.0, yaw: -3.142)
- bathroom is a waypoint (x: 1.14, y: 2.32, yaw: -3.142)
- bedroom is a waypoint (x: 0.55, y: 5.61, yaw: 0.673)
- livingroom is a waypoint (x: 6.39, y: 3.26, yaw: 0.891)
- Miguel is a person (is_greeted: No)
- Fran is a person (is_greeted: No)
- Angel is a person (is_greeted: No)
- Vicente is a person (is_greeted: No)
- RB1 is a robot
- entrance belongs to GrannyHouse
- bathroom belongs to GrannyHouse
- bedroom belongs to GrannyHouse
- livingroom belongs to GrannyHouse
- Miguel at entrance
- Fran at bathroom
- Angel at bedroom
- Vicente at livingroom
- RB1 at entrance

GOAL: RB1 greeted Vicente

### Instruction:
Generate a plan to achieve the above goal

### Response:

```

Figura C.2 – Prompt para planificar (II).

```
[INFO] [1695052408.503526771] [planning_layer_node]: {
  "reasoning": "RB1 is currently at the entrance and needs to greet Vicente who is at the livingroom.",
  "plan": "Move RB1 to the livingroom, then greet Vicente.",
  "actions": [
    {
      "name": "move_to",
      "args": {
        "robot_name": "RB1",
        "source_waypoint": "entrance",
        "target_waypoint": "livingroom"
      }
    },
    {
      "name": "greet_person",
      "args": {
        "robot_name": "RB1",
        "person_to_greet": "Vicente",
        "person_waypoint": "livingroom"
      }
    }
  ]
}
```

Figura C.3 – Respuesta al *prompt* para planificar.

```
You are an AI goal validator for a robot named RB1 whose role is assistant.
You have to check if the goal is achieved in the world state.
You can ONLY response with the following JSON format:
{
  "response": "yes/no (Is the goal achieved in the world state?)",
  "explanation": "explain why the goal is achieved or not in the world state"
}

WORLD STATE:
- GrannyHouse is a house
- entrance is a waypoint (x: 0.23, y: 0.0, yaw: -3.142)
- bathroom is a waypoint (x: 1.14, y: 2.32, yaw: -3.142)
- bedroom is a waypoint (x: 0.55, y: 5.61, yaw: 0.673)
- livingroom is a waypoint (x: 6.39, y: 3.26, yaw: 0.891)
- Miguel is a person (is_greeted: No)
- Fran is a person (is_greeted: No)
- Angel is a person (is_greeted: No)
- Vicente is a person (is_greeted: Yes)
- RB1 is a robot
- entrance belongs to GrannyHouse
- bathroom belongs to GrannyHouse
- bedroom belongs to GrannyHouse
- livingroom belongs to GrannyHouse
- Miguel at entrance
- Fran at bathroom
- Angel at bedroom
- Vicente at livingroom
- RB1 at livingroom

GOAL: RB1 greeted Vicente

### Instruction:
Check if the goal is achieved in the world state
```

Figura C.4 – *Prompt* para verificar el objetivo.

