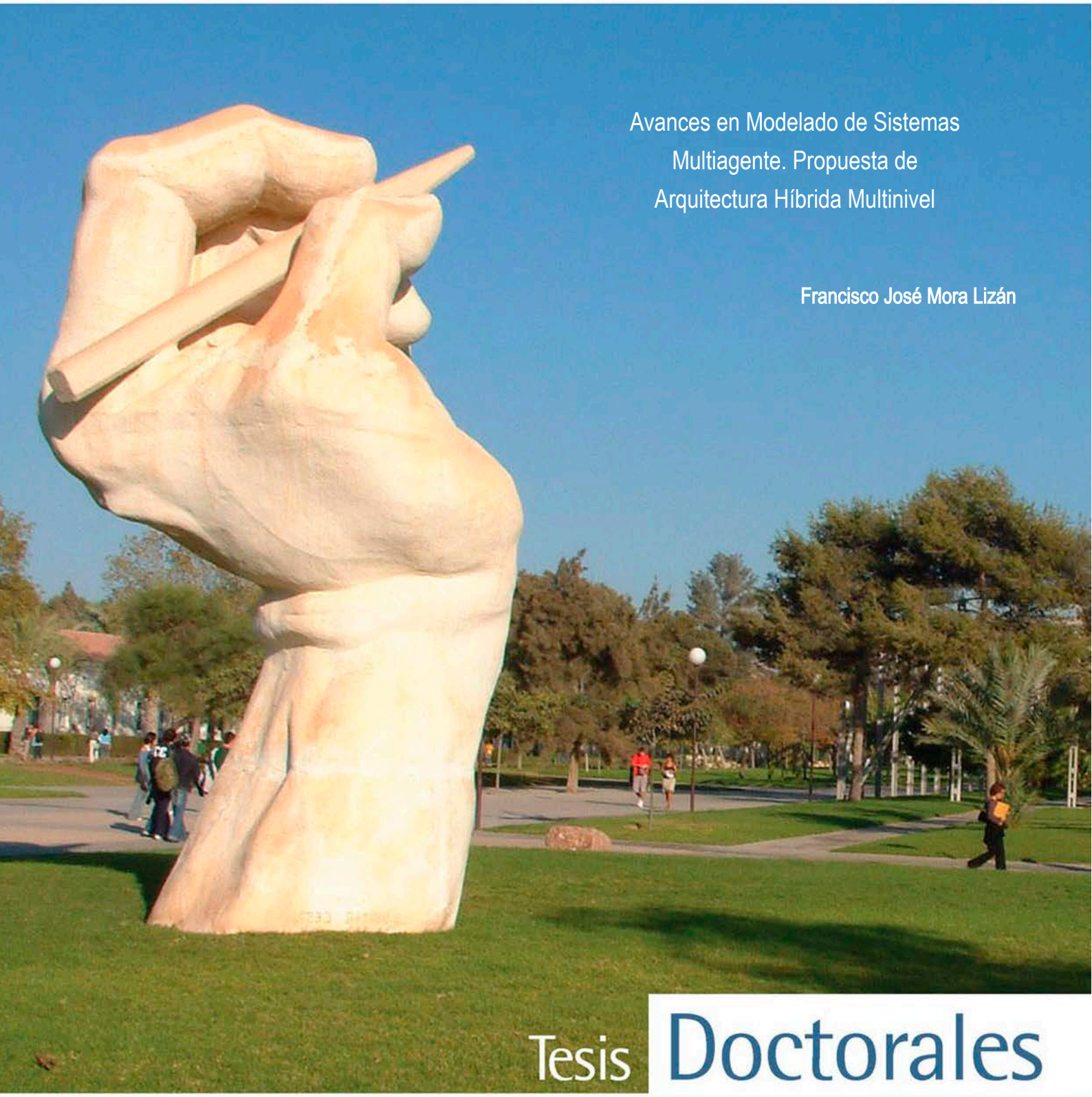
Universitat d'Alacant
Universidad de Alicante



Avances en Modelado de Sistemas Multiagente. Propuesta de Arquitectura Híbrida Multinivel

Francisco José Mora Lizán

Tesis Doctorales

UNIVERSIDAD de ALICANTE

Unitat de Digitalització UA
Unidad de Digitalización UA

Universitat d'Alacant
Universidad de Alicante

Departamento de Ciencia de la Computación
e Inteligencia Artificial
Escuela Politécnica Superior de Alicante

Avances en Modelado de Sistemas Multiagente. Propuesta de Arquitectura Híbrida Multinivel

FRANCISCO JOSÉ MORA LIZÁN

Tesis presentada para aspirar al grado de

DOCTOR POR LA UNIVERSIDAD DE ALICANTE

DOCTORADO EN INFORMÁTICA

Dirigida por:
Dr. Fidel Aznar Gregori
Dr. Carlos Rizo Maestre

Este trabajo cuenta con el apoyo del Ministerio de Ciencia, Innovación y Universidades (España), proyecto RTI2018-096219-B-100. Proyecto cofinanciado con fondos FEDER.

Resumen

Este trabajo de tesis ha tenido un largo desarrollo temporal, que ha permitido abordar campos de aplicación de contextos tradicionalmente muy distantes, permitiendo tener una visión más amplia y fortaleciendo el trabajo realizado.

Un agente inteligente es una entidad que actúa de manera racional intentando lograr un objetivo, para ello percibe su entorno, procesa estas percepciones y responde en su entorno. Pero la resolución de un problema complejo presenta grandes dificultades y restricciones, tanto de tiempo como de recursos, así como necesita gran cantidad de conocimiento. Esto hace que la dificultad de resolución de los problemas aumente o las haga inviables para un agente trabajando individualmente.

En los sistemas multiagente un grupo de agentes colaborarán, combinando sus habilidades y su conocimiento, permitiendo la resolución del problema de manera conjunta.

Los sistemas multiagente son cada vez más demandados y utilizados. Su aplicación la podremos encontrar en un amplio número de campos como robótica, Big Data, aplicaciones de transporte y logística, sistemas de información geográfica, balanceo de redes y móviles, diagnóstico médico, aplicaciones gráficas, juegos, así como en muchos otros campos.

A lo largo de los años se ha trabajado mucho para facilitar el diseño y la construcción de sistemas multiagente, han aparecido diversas arquitecturas, metodologías y herramientas, incluso se ha creado un estándar, el estándar IEEE-FIPA (Foundation for Intelligent Physical Agents). Pero se ha dejado la implementación a bajo nivel en manos de los equipos de desarrollo, los cuales han ido creando soluciones adhoc para cada uno de sus problemas.

En esta tesis se presenta una arquitectura versátil que permita avanzar en el diseño y construcción de sistemas multiagente. Para ello proponemos una arquitectura híbrida multinivel y una metodología que nos facilita la creación de estos sistemas de una manera rápida, con bajo coste y de alta calidad.

Para validar la arquitectura y demostrar su versatilidad la hemos aplicado a tres contextos de diferentes:

- **Edificios inteligentes**: en un sistema multiagente para mejorar el ahorro energético en un hotel, donde podremos observar una de las ventajas de la arquitectura: la integración de elementos de bajo nivel a través de la integración de elementos IoT (Internet of Things).

- **Robótica**: en un sistema de enjambre robótico para la detección de señales de radiofrecuencia, donde se demostrará la flexibilidad, escalabilidad y robustez de la arquitectura al aplicarse en un contexto de múltiples robots y de simulación.
- **Sistemas biomédicos**: en una herramienta tecnológica para medir y registrar movimiento de hiperactividad (ADHD), donde podemos observar la versatilidad de la arquitectura a través de un contexto totalmente diferente, el de diagnósticos en salud.

Abstract

This thesis has been developed over a long period of time, which has made it possible to address fields of application in traditionally very distant contexts, allowing for a broader vision and strengthening the work carried out.

An intelligent agent is an entity that acts rationally in an attempt to achieve an objective. To do so, it perceives its environment, processes these perceptions and responds to them. But the resolution of a complex problem presents great difficulties and restrictions, both in time and resources, and requires a great deal of knowledge. This makes problem solving more difficult or unfeasible for an agent working individually.

In multi-agent systems, a group of agents will collaborate, combining their skills and knowledge, allowing the problem to be solved together.

Multi-agent systems are increasingly in demand and used. Their application can be found in a wide range of fields such as robotics, Big Data, transport and logistics applications, geographic information systems, network and mobile balancing, medical diagnostics, graphic applications, games, as well as in many other fields.

Over the years, a lot of work has been done to facilitate the design and construction of multi-agent systems, various architectures, methodologies and tools have appeared, even a standard has been created, the IEEE-FIPA standard (Foundation for Intelligent Physical Agents). But low-level implementation has been left in the hands of development teams, who have been creating ad-hoc solutions for each of their problems.

This thesis presents a versatile architecture that allows progress in the design and construction of multiagent systems. For this, we propose a multi-level hybrid architecture and a methodology that facilitates the creation of these systems in a fast, low-cost and high-quality way.

To validate the architecture and demonstrate its versatility we have applied it to three different contexts:

- **Smart buildings:** in a multi-agent system to improve energy savings in a hotel, where we can observe one of the advantages of architecture: the integration of low-level elements through the integration of IoT (Internet of Things) elements.

- **Robotics:** a robotic swarm system for the detection of radio frequency signals, which will demonstrate the flexibility, scalability and robustness of the architecture when applied in a multi-robot and simulation context.

• **Biomedical systems:** in a technological tool to measure and record hyperactivity movement (ADHD), where we can observe the versatility of architecture through a totally different context, that of health diagnostics.

Resum

Aquest treball de tesi ha tingut un llarg desenvolupament temporal, que ha permès abordar camps d'aplicació de contextos tradicionalment molt distants, permetent tenir una visió més àmplia i enfortint el treball realitzat.

Un agent intel·ligent és una entitat que actua de manera racional intentant aconseguir un objectiu, per a això percep el seu entorn, processa aquestes percepcions i respon en el seu entorn. Però la resolució d'un problema complex presenta grans dificultats i restriccions, tant de temps com de recursos, així com necessita gran quantitat de coneixement. Això fa que la dificultat de resolució dels problemes augmente o les faça inviables per a un agent treballant individualment.

En els sistemes multiagent un grup d'agents col·laboraran, combinant les seues habilitats i el seu coneixement, permetent la resolució del problema de manera conjunta.

Els sistemes multiagent són cada vegada més demandats i utilitzats. La seua aplicació la podrem trobar en un ampli nombre de camps com a robòtica, Big Data, aplicacions de transport i logística, sistemes d'informació geogràfica, balanceig de xarxes i mòbils, diagnòstic mèdic, aplicacions gràfiques, jocs, així com en molts altres camps.

Al llarg dels anys s'ha treballat molt per a facilitar el disseny i la construcció de sistemes multiagente, han aparegut diverses arquitectures, metodologies i eines, fins i tot s'ha creat un estàndard, l'estàndard IEEE-FIPA (Foundation for Intelligent Physical Agents). Però s'ha deixat la implementació a baix nivell en mans dels equips de desenvolupament, els quals han anat creant solucions adhoc per a cadascun dels seus problemes.

En aquesta tesi es presenta una arquitectura versàtil que permeta avançar en el disseny i construcció de sistemes multiagente. Per a això proposem una arquitectura híbrida multinivell i una metodologia que ens facilita la creació d'aquests sistemes d'una manera ràpida, amb baix cost i d'alta qualitat.

Per a validar l'arquitectura i demostrar la seua versatilitat l'hem aplicada a tres contextos de diferents:

- **Edificis intel·ligents**: en sistema multiagent per a millorar l'estalvi energètic en un hotel, on podrem observar una dels avantatges de l'arquitectura: la integració d'elements de baix nivell a través de la integració d'elements IoT (Internet of Things).

- **Robòtica**: en sistema d'eixam robòtic per a la detecció de senyals de radiofreqüència, que demostrarà la flexibilitat, escalabilitat i robustesa de la arquitectura en aplicar-se en un context de múltiples robots i de simulació.
- **Sistemes biomèdics**: en una eina tecnològica per a mesurar i registrar moviment d'hiperactivitat (ADHD), on podem observar la versatilitat de l'arquitectura a través d'un context totalment diferent, el de diagnòstics en salut.

Agradecimientos

Me gustaría dar mi más sincero agradecimiento a mis directores de tesis, Fidel y Carlos, por su apoyo incondicional, comprensión y dedicación.

De una forma muy especial quiero dar las gracias Ramón, primero como director, luego como tutor, estando a lo largo de estos años siempre ahí, aguantando mis idas y venidas, mis retrasos debido a las tareas docentes y de gestión. Gracias por tu sabiduría, consejos y paciencia.

A los miembros del Grupo de Investigación Informática Industrial e Inteligencia Artificial, sin los cuales no hubiese sido posible sacar hacia delante los artículos de investigación. Y especialmente a Mireia y de nuevo a Fidel, cuyos trabajos han sido claves para el desarrollo de esta tesis.

A todos los miembros del Departamento de Ciencia de la Computación e Inteligencia Artificial de la Universidad de Alicante, en ellos siempre he encontrado amistad y ánimos. Rafa, tranquilo que ya acabo la tesis.

Me gustaría también tener una muestra de agradecimiento con la Comisión Académica del Programa de Doctorado en Informática, especialmente con Jorge y José que también siempre han estado ahí, resolviendo dudas y facilitando todos los trámites de gestión. Acompañar en estos agradecimientos a las personas que conforman la Escuela de Doctorado, de las que sólo nos acordamos cuando las cosas van mal, pero que en los momentos difíciles han echado un capote, para que los tramites los pudiésemos terminar a tiempo.

Y cómo no acordarme de mi familia. A mis padres, que gracias a la educación y sus valores trasmitidos, hoy soy quien soy. A mi mujer Isabel y a mis hijos Paula y Adrián, a los cuales les he tenido que robar muchas horas, conciliando la vida familiar y laboral.

Gracias a todos, que de distintas maneras, me habéis ayudado a terminar esta tesis.

Índice de contenidos

Índice de Ilustraciones

1 Introducción

1.1 Motivación.

Los sistemas MAS (MultiAgent System) están formados por unos miembros interdependientes que actúan individualmente constituyendo una comunidad social. Para vivir dentro de las sociedades son necesarias capacidades de negociación y coordinación.

La resolución de un problema individualmente por un agente presenta grandes restricciones y dificultades. Cuando el problema es complejo se necesita gran cantidad de conocimiento. Muchas veces el problema puede presentar un grado de dificultad que no permita al agente resolverlo de una manera individual. Además, en situaciones reales nos aparecen restricciones de fiabilidad, rapidez y flexibilidad que hacen que la resolución de los problemas aumente o las haga inviables.

Los sistemas multiagente son una solución a la problemática descrita anteriormente, donde un grupo de agentes independientes colaborar para lograr unos objetivos. Cada agente tendrá sus metas y colaborará con otros agentes para obtener información o para participar en la resolución del problema general. Esta colaboración va a permitir la resolución de problemas complejos que de manera individual no se podrían resolver.

La gran ventaja de los sistemas multiagente es que integran en un único sistema al conjunto de agentes, no requiriendo el diseño y la implementación de un nuevo agente, sino que combina las habilidades y el conocimiento de los agentes existentes, permitiendo la resolución del problema de manera conjunta.

A lo largo de los años han aparecido muchas plataformas para la construcción de sistemas multiagente, Java Agent Development Framework (JADE), Zeus Agent Building Toolkit, y Java Intelligent Agent Compontentware (JIAC), entre otros. Cada una de ellas ofrece una serie de agentes y sistemas de comunicación básicos precreados para facilitar el desarrollo de estos sistemas, incluso cuentan con herramientas gráficas para su depuración. Todos estos

frameworks se han diseñado bajo el standard FIPA [1] el cual trata de coordinar y estandarizar los esfuerzos.

Cada día se van necesitando y utilizando más los sistemas multiagente, tanto software como hardware. Su aplicación la podremos encontrar en un amplio número de campos desde sistema de defensa a juegos, pasando por aplicaciones gráficas, aplicaciones de transporte y logística, sistemas de información geográfica, aplicaciones de balanceo de redes y móviles, diagnóstico médico, así como en muchos otros campos. Dentro de su uso en los últimos años cabe destacar su aplicación en sistemas Big Data y Web.

Un tema pendiente es el del diseño y construcción de estos sistemas a bajo nivel. Es importante trabajar en la creación arquitecturas y metodologías que nos faciliten la construcción de los distintos elementos, haciendo que la puesta en marcha de estos sistemas sea más rápida, con unos costes menores costes y aumentando la calidad de los mismos.

1.2 Objetivos.

De una manera genérica, podríamos decir que el objetivo de este trabajo es la creación de una arquitectura versátil y ágil que permita diseñar y construir sistemas multiagente cubriendo las necesidades de planificación, coordinación percepción y actuación típicas de estos sistemas. Una máxima de nuestra propuesta será la adhesión a los estándares, utilizando UML para el modelado y FIPA como plataforma de implementación. El modelo debe de soportar las arquitecturas tradicionales de los sistemas multiagente, así como facilitar el diseño, la implementación y la integración de los componentes tanto de alto como de bajo nivel, permitiendo construir los sistemas multiagente de una manera rápida, con bajo coste y de alta calidad, para lo cual será fundamental el uso de patrones y la reutilización de componentes. Otra meta que pretendemos cumplir es su facilidad de comprensión y aprendizaje del modelo.

Para alcanzar este objetivo se han abordado los siguientes hitos:

- Revisar las características de los sistemas de agentes y multiagente, con la finalidad de conocer los distintos campos donde podemos aplicar los sistemas multiagente, estableciendo sus bondades y limitaciones.
- Revisar de las arquitecturas actuales. Aquí estudiaremos las distintas propuestas que se han ido haciendo a lo largo del tiempo. Para la consecución de este objetivo nos hará falta estudiar las arquitecturas de sistemas de agentes, de los sistemas multiagente y las arquitecturas robóticas. Este estudio nos proporcionará una guía

para describir los elementos, su organización y la interacción entre ellos, así como detectar sus debilidades y fortalezas.

- Revisar los frameworks de desarrollo de sistemas multiagente. La realización de un estudio de las herramientas existentes nos permitirá conocer mejor la construcción de los elementos a bajo nivel, así como las pautas seguidas para cumplir con las normas de estandarización. También nos permitirá detectar las necesidades no cubiertas por parte de estas herramientas.
- Proponer una arquitectura para el desarrollo de sistemas multiagente que sea versátil, flexible, robusta y escalable. Esta arquitectura deberá de recoger las fortalezas y cubrir las debilidades de las arquitecturas actuales. Diseñada para utilizar los frameworks existentes para el desarrollo de sistemas multiagente, pero al mismo tiempo abierta a nuevas posibilidades.
- Proponer una metodología ágil, iterativa y práctica, que permita construir sistemas multiagente, entre ellos, los diseñados con la arquitectura propuesta en esta tesis. Esta metodología vendrá acompañada de una descripción de los documentos y diagramas necesarios para avanzar en el desarrollo de los proyectos, siempre siguiendo los estándares de la industria.
- Verificar la aplicación de la arquitectura y de la metodología propuesta. No nos basta con la definición teórica, sino que validaremos tanto la metodología como la arquitectura, aplicándolas a casos prácticos en diferentes contextos de aplicación.

1.3 Estructura.

El contenido de esta tesis se ha distribuido en seis capítulos.

En este capítulo 1 se exponen las motivaciones, los objetivos de esta tesis y la estructura de del documento.

En el **capítulo 2**, estado del arte, pretende definir los conceptos y características de agentes y sistemas multiagente, así como exponer el estándar FIPA (Fundation for Intelligent Physical Agents). A continuación, pasaremos a abordar las distintas arquitecturas de agentes, de sistemas multiagente y de arquitecturas robóticas. En este capítulo también se realiza un estudio sobre dos de los más importantes frameworks de construcción de agentes (JADE y ZEUS).

El **capítulo 3** se definen la arquitectura y la metodología propuestas en esta tesis. Para ello presentaremos la arquitectura, donde definiremos las

distintas capas del mismo: alto nivel (planificación y coordinación), nivel medio (secuenciación) y bajo nivel (percepción y actuación). A continuación, explicamos la metodología a seguir, así como las características que deben de cumplir las herramientas para construir sistemas utilizando nuestra arquitectura. Para terminar el capítulo describiremos brevemente posibles aplicaciones del modelo con el fin de mostrar su versatilidad.

En el **capítulo 4** se aporta la aplicación de la arquitectura a un problema de un hotel inteligente con el fin de mejorar el ahorro energético. Aquí podremos observar una de las aportaciones de la: la integración de elementos de bajo nivel a través de la integración de elementos IoT (Internet de las Cosas).

En el **capítulo 5** se demuestra la versatilidad de la arquitectura con un problema no típico de sistemas multiagente: La detección de señales RF a través de un enjambre robótico. En este podemos ver el uso de drones y de simulación dentro de nuestro modelo.

Finalmente, en el **capítulo 6** se resumen los resultados y se exponen las conclusiones, se incorpora un detalle de las contribuciones derivadas de la tesis, así como una descripción de las propuestas futuras de trabajo y líneas de investigación.

2 Estado del Arte

2.1 Agentes Inteligentes y Sistemas Multiagente.

Una definición de agente inteligente es la dada por Hípola y Vargas-Quesada [1] donde lo definen "como una entidad software que, basándose en su propio conocimiento, realiza un conjunto de operaciones destinadas a satisfacer las necesidades de un usuario o de otro programa, bien por iniciativa propia o porque alguno de éstos se lo requiere."

Otra definición que se ajusta a este concepto es la dada por Jiménez y Ramos [2] donde definen al agente inteligente como "Pieza de software que ejecuta una tarea dada utilizando información recolectada del ambiente, para actuar de manera apropiada hasta completar la tarea de manera exitosa. El software debe ser capaz de auto ajustarse basándose en los cambios que ocurren en su ambiente de forma tal que un cambio en las circunstancias producirá un resultado esperado.".

Podríamos decir que un agente es módulo de software, el cual tiene un objetivo concreto, se basa en su conocimiento del entorno para cumplirlo, actúa sobre dicho entorno, aprende actos anteriores y es capaz de interactuar con otros agentes.

Tenemos una serie de características fundamentales asociadas al concepto de agente:

- Autonomía: realiza sus propios actos sin ningún tipo de intervención humana directa.
- Reactividad: percibe cambios en el entorno que y reacciona ante ellos.
- Adaptabilidad: los agentes se adaptan continuamente a los cambios del entorno.

- Sociabilidad y comunicación: el agente puede comunicarse con otros agentes e incluso interactuar con humanos.
- Iniciativa o pro-actividad: el agente actúa para conseguir un determinado propósito u objetivo.
- Continuidad temporal: los agentes están realizando acciones u esperando a que eventos del entorno, de otro agente o usuarios lo hagan reaccionar.

Aparte de las características fundamentales anteriores los agentes inteligentes pueden presentar otras peculiaridades. Entre ellas tendríamos la capacidad de interactuar con otros agentes (cooperación), movilidad para moverse entre sistemas y disposición para ayudar a otros agentes siempre y cuando no se comprometan sus propios objetivos (benevolencia).

Al igual que las características, en la literatura de agentes encontramos multitud de clasificaciones de los agentes inteligentes. Una de las más extendidas es la taxonomía de Hyacinth Nwana [3]:

- Agentes de interfaz: ayudan al usuario a utilizar una aplicación, guiándolo en los pasos y/o procesos a realizar. Esta ayuda puede ser dada de forma textual, o gráfica. Internamente el agente puede aprender también a través del comportamiento del usuario.
- Agentes colaborativos o cooperativos: las características principales de estos agentes son la autonomía y la cooperación con otros agentes, entendiendo cooperación como capacidad de negociación para realizar tareas de manera conjunta. La funcionalidad de estos agentes sólo es posible con la participación de otros agentes.
- Agentes móviles: aquí entendemos al agente como un procesos u objeto software que es capaz de viajar por las redes, interactúa con otros equipos, reúne información interactuando con otros equipos, y regresa para informar de los resultados.
- Agentes de información: este tipo de agentes son muy importantes hoy en día para la recuperación de información en la Web. Obtienen información relevante recopilando y manipulando información que se encuentra distribuida en diferentes fuentes.
- Agentes reactivos: son agentes que no poseen un modelo simbólico del entorno, pero que responden a los estímulos del mismo.
- Agentes híbridos: estos agentes serían la combinación de dos o más de los tipos anteriores.

Utilizar un agente de forma individual nos va a causar grandes restricciones. Para resolver problemas complejos un agente individual necesita gran cantidad de conocimiento, incluso el problema puede ser tan complejo que un agente no pueda encontrar una solución útil. Aunque el agente individual fuese capaz de resolver el problema, tenemos unas restricciones de velocidad, fiabilidad, flexibilidad y modularidad, que no harían viable esta solución.

Un sistema multiagente constará de una serie de agentes autónomos independientes, cada uno se dedica a sus propios objetivos y contactará solamente con otros agentes para obtener información, o para contribuir a soluciones coordinadas, permitiendo así la resolución de problemas complejos.

Una de las grandes ventajas de los sistemas multiagente es que permiten la integración en un único sistema de los agentes existentes, no requiriendo del diseño y del desarrollo de un nuevo agente especializado en la resolución del nuevo problema. En su lugar, se combina el conocimiento y las habilidades de los agentes existentes combinándolos permitiendo resolver el problema de una manera conjunta.

También será fundamental trabajar en sistemas multiagente cuando el número de agentes sea muy grande, ya que la gestión individual de ellos sería una tarea difícil. Dentro del sistema multiagente se tratarán de una manera colectiva, como una sociedad de agentes.

Una definición de sistema multiagente la podemos encontrar en [5] como: "un sistema multiagente se define como una red débilmente acoplada de agentes autónomos que trabajan conjuntamente para resolver problemas que están por encima de las capacidades o conocimientos de un único robot".

Los sistemas multiagente tienen una serie de características [5][6]:

- Los agentes sólo perciben y/o tienen información parte de la información del entorno.
- La información está descentralizada.
- No existe un sistema de control global
- La ejecución es asíncrona.

Todas estas características hacen que hablemos de sistemas sociales artificiales [7]. Los estudios en inteligencia artificial, durante muchos años, se han centrado, en los individuos, pero nuestras habilidades sociales es lo que hace únicos, donde no sólo nos podemos comunicarnos, sino también negociar, coordinarnos, y cooperar. De aquí podemos extraer una serie de preguntas que deberíamos de hacerle a un sistema [7]:

- ¿Si los agentes tienen intereses individuales como puede aparecer la cooperación?
- ¿Cómo pueden comunicarse los agentes? ¿Qué lenguaje hablarían?
- ¿Cómo pueden saber los agentes cuando sus acciones u objetivos entran en conflicto con otros agentes?
- ¿Cómo los agentes pueden alcanzar acuerdos?
- ¿Cómo se pueden coordinar las actividades de los agentes?

Para diseñar y construir un sistema multiagente deberíamos de responder a todas esas preguntas, indicando cómo los agentes deben coordinarse, negociar y cooperar. Si a estas preguntas le añadimos que nuestro sistema va a ser un sistema distribuido, su diseño se complica aún más. Como sistemas distribuidos, los sistemas multiagente tendrán unas características a satisfacer [6]:

- Eficiencia y velocidad de ejecución: el paralelismo permitirá un incremento del rendimiento global del sistema.
- Fiabilidad y fiabilidad: el sistema debe ser capaz de funcionar ante el fallo de uno o varios agentes, bien recreándolos o bien sustituyéndolos.
- Escalabilidad y flexibilidad: ante un problema mayor podríamos responder añadiendo nuevos agentes.
- Costes: al estar compuesto de más simples debería de ser más más económico que un sistema centralizado.
- Desarrollo y reusabilidad: al tener elementos más sencillos es más fácil aplicar patrones y herencia. El sistema final puede ser probado y mantenido más fácilmente.

2.1.1 Inteligencia Artificial Distribuida y Resolución de Problemas.

Tradicionalmente se distinguen dos tipos de sistemas: los sistemas multiagente y los sistemas de resolución distribuida de problemas. Mientras que en los sistemas multiagente el énfasis se pone en la coordinación, en los sistemas de resolución distribuida de problemas está en la tarea de descomposición y de síntesis de las soluciones. En la actualidad no existe tal diferenciación y al hablar de sistemas multiagente englobamos ambos tipos.

La solución de problemas distribuidos con sistemas multiagente es apropiada sólo en el caso en que los agentes integrados en el sistema sean capaces de comunicarse y cooperar con otros.

Así, la metodología usada para resolver un problema global de forma distribuida, consta de tres pasos secuenciales: división del problema en subproblemas, solución de los subproblemas y combinación de las subsoluciones. A continuación, pasaremos a explicar cada uno de ellos:

- El primer paso y más importante es dividir el problema global en una serie de problemas parciales, los cuales habrá distribuirlos entre los agentes del sistema atendiendo a las habilidades y especializaciones de los mismos. En el caso de tener varios agentes capaces de resolver el mismo problema tendremos que tomar decisiones en la asignación de subproblemas y establecer estrategias de cooperación entre ellos.
- El segundo paso sería la resolución de los subproblemas. A menudo un agente no es capaz de manera autónoma de resolver un subproblema y tiene que solicitar ayuda. Otros agentes podrían ayudarle a resolver el subproblema o a localizar a agentes que tengan el conocimiento requerido.
- El tercer y último paso es la síntesis de soluciones parciales en una solución global. Aquí aparecen problemas parecidos a los del primer paso; como valorar las subsoluciones y como generar una solución correcta. A veces, la combinación de subsoluciones no produce una solución.

Si cada agente individual es responsable de la solución de un subproblema, el segundo paso del proceso de solución no requiere necesariamente procesos de comunicación y cooperación. Pero en muchas ocasiones la interacción entre diferentes agentes es necesaria porque los subproblemas individuales no pueden resolverse de forma autónoma. Por esta razón, pueden aparecer conflictos directos entre dos o más agentes, lo que requiere que exista comunicación entre los agentes para resolver este conflicto. Así mismo, los agentes deben de comunicarse entre sí para alcanzar la solución general del problema.

2.1.2 Comunicación entre Agentes.

Para que dos seres inteligentes puedan interactuar necesitan un canal de comunicación y unas normas: utilizar el mismo lenguaje, tener mecanismos de comunicación para el intercambio de mensajes, etc. Para interactuar debe de existir entre los interlocutores coordinación y comunicación.

2.1.2.1 Actos de Habla.

Para que un agente provoque cambios en los objetivos o creencias de otro tendrá que comunicarse con él de alguna manera. Diremos que dos agentes han establecido un acto comunicativo cuando un agente realiza una serie de acciones que persiguen la modificación de la estructura cognitiva del otro agente.

Como la mayor parte de los actos comunicativos entre personas se realizan mediante el lenguaje hablado, los expertos lingüísticos utilizan el término "acto de habla" para referirse a todos los tipos de actos comunicativos [8][9]. Por ello también llamaremos "hablante" al agente que toma la iniciativa de la comunicación y "oyente" al agente al que va. Podemos clasificar los actos de habla en varias categorías:

- Representativos: exponen una determinada proposición como representación de un estado de cosas del mundo.
- Directivos: mediante los que se realiza una petición, se da una orden o se hace un ruego. En ellos se intenta que el oyente actúe de tal modo que su conducta concuerde con el contenido proposicional.
- Exhortativos: para prometer, comprometer, jurar, amenazar o realizar contratos o garantías. Constituye un compromiso por parte del hablante de adoptar el tipo de acción representado en el contenido proposicional.
- Expresivos: para expresar un agradecimiento, felicitaciones, bienvenidas o pedir disculpas.
- Declaraciones: actos que producen un cambio en el estado del mundo.

La teoría de actos de habla nos ayuda a entender la relación entre las expresiones que intercambia con otros agentes y el estado interno del agente. Esta teoría se basa en la observación de las oraciones expresadas por humanos. Durante la comunicación no únicamente se confirma un hecho, sino que se puede transmitir una un conocimiento, una creencia, una intención o un incluso un deseo. En líneas generales, los actos de habla producen efectos sobre el conocimiento del oyente. Un acto verbal se manifiesta con una sentencia y debe reflejar el contenido proposicional y el tipo de acto de habla. El efecto que el hablante intenta provocar en el oyente se denomina efecto perlocutivo y el efecto que realmente se produce, efecto ilocutivo. Las sentencias transmitidas deben de utilizar un lenguaje común de comunicación, pero lo realmente importa es integrar una semántica en el acto comunicativo para el intercambio de conocimiento exista. Por este motivo, creamos los lenguajes basados en la teoría de actos de habla, y con ella construimos una capa lingüística y formalizamos las acciones lingüísticas de los agentes.

2.1.2.2 Coordinación entre Agentes.

Cuando crece el número de agentes del sistema crece, es necesaria la ayuda para localizar a los agentes que ofrezcan los servicios o la información que se necesite. [10]. Muchos sistemas multiagente utilizan agentes mediadores o facilitadores que proporcionan determinados servicios y asistencia a los agentes del sistema (matchmaker, broker, blackboard, etc.). En general podemos distinguir tres categorías de agentes: P-agentes o agentes proveedores, R-agentes o agentes requeridores y agentes mediadores o intermediarios.

La comunicación es la base de la coordinación y está compuesta por los protocolos de comunicación y es el resultado de los métodos de comunicación.

2.1.2.3 Métodos de Comunicación.

Antes de empezar a hablar de los métodos de comunicación conviene aclarar que la llamada a un procedimiento no se considera un método de comunicación. Dentro de los métodos de comunicación hablaremos de sistemas de pizarra donde utilizamos una zona común para intercambiar información y los sistemas basados en diálogos de mensajes.

2.1.2.3.1 Sistemas de Pizarra

El sistema de pizarra es un mecanismo de comunicación muy utilizado en Inteligencia Artificial y en otros ámbitos. La pizarra ("blackboard") es una estructura de datos que es usada por múltiples participantes (lectores y escritores), y es gestionada y arbitrada por un controlador [11][12] y [13]. Es un área común donde los agentes pueden intercambiar datos, información y conocimiento (Ilustración 1).

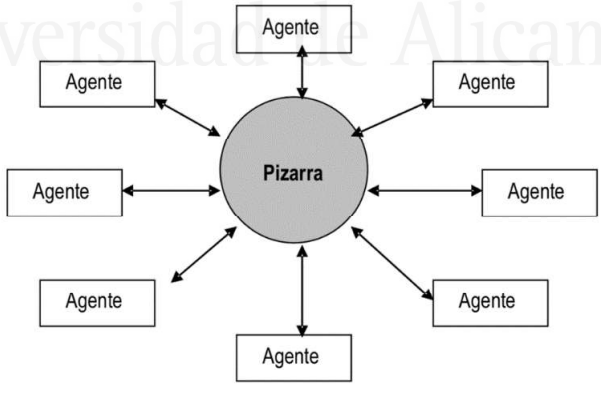


Ilustración 1: Sistema de pizarra.

Cada agente trabajará en la resolución de su subproblema y acudirá a la pizarra para ver la nueva información puesta por otros agentes. Este mismo

agente también utilizará la pizarra para poner sus resultados o la información que le demanden. El funcionamiento de la pizarra es el siguiente: La comunicación se inicia cuando un agente escribe cualquier tipo de información en la pizarra; cada agente comprobará cada cierto tiempo si hay información nueva; generalmente cada agente no recogerá toda la información que se vaya escribiendo en la pizarra, sino que sólo cogerá aquella que le interese para cubrir su misión.

No existe comunicación directa entre los agentes; por esta razón podemos decir que cada agente resuelve de forma autónoma su subproblema. En la pizarra básica no existen áreas privadas, pudiendo cualquier agente leer y escribir cualquier tipo de información. Cuando el número de agentes es grande la información contenida en la pizarra crece de forma exponencial, y esto provocará que se complique la búsqueda de información en la pizarra. Existen variantes más complejas del método de pizarra, por ejemplo, hay una variante para definir regiones en la pizarra donde un agente sólo ve la región de la pizarra que tenga asignada. Un sistema multiagente puede disponer de varias pizarras. Para los datos de pizarra no existe una estructura de datos establecida, por lo que nos tocará definirla de forma que aseguremos que el resto de agentes puede acceder y entender los datos publicados en ella.

La figura de moderador es necesaria para controlar los datos de la pizarra, ya que cualquier agente puede poner datos en ella y esto puede generar problemas. Muchas veces se utiliza el sistema de pizarra no meramente como sistema de intercambio de información, sino que el moderador de la misma se encarga de dirigir el sistema hacia la solución, supervisando el estado del problema, de los subproblemas a ser resueltos y además trata de organizar el trabajo entre y hacia los agentes. De esta forma cualquier agente podría usar la pizarra para acceder a los subproblemas generados y hacer una petición de resolución del subproblema indicándolo usando un registro de activación de fuente de conocimiento (KSAR, "Knowledge Source Activation Record"). El moderador controla y evalúa el conocimiento de la base de conocimiento y selecciona a los agentes más capacitados para resolver el subproblema.

En la Ilustración 2, podemos ver otra mejora del sistema de pizarra, donde un "dispatcher" que se encarga de informar a los agentes interesados en un tipo de información, en lugar de que estos accedan regularmente a la misma.

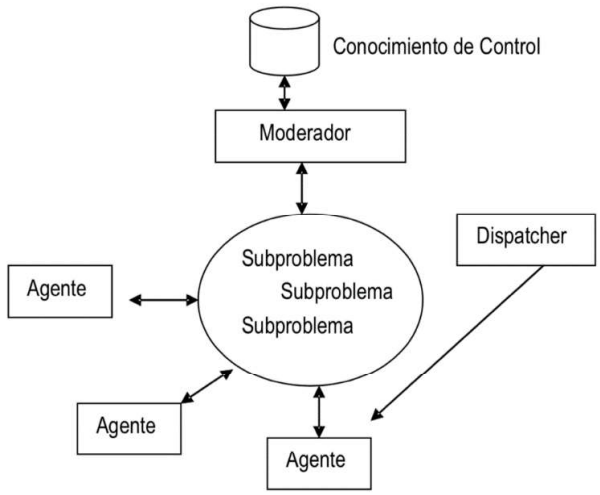


Ilustración 2: Estructura extendida de pizarra utilizando un "dispatcher".

El sistema de pizarra es un mecanismo sencillo y flexible que permite la cooperación y comunicación entre agentes. Sin embargo, tiene el inconveniente de ser un cuello de botella en el rendimiento de la red del sistema multiagente

2.1.2.3.2 Sistemas de Mensajes.

Para la implementación de estrategias de comunicación complejas no es suficiente con el método de pizarra y por ello aparecieron los sistemas basados en mensajes. Los agentes intercambian mensajes a través de protocolos definidos con el fin de cooperar. Las comunicaciones son muy versátiles y no exclusivas de comandos simples.

En la terminología de mensajes en lugar de hablante y oyente, hablaremos de emisores y receptores. En este caso los mensajes son intercambiados directamente entre dos agentes, al contrario que ocurría en los sistemas de pizarra. En este caso no se utiliza zona de memoria intermedia, ni otros agentes son capaces de leer mensajes que no va dirigidos a ellos. El receptor suele ser único, pero existe el llamado "broadcasting" o "multicasting" donde el mensaje es enviado a un grupo específico o incluso a todos los agentes del sistema.

Para que el sistema de mensajes pueda ser utilizado para implementar estrategias de cooperación tenemos que definir un protocolo de comunicaciones (proceso de comunicación, el formato de los mensajes y la elección del lenguaje de comunicaciones) y una semántica del lenguaje de comunicación.

Con la transferencia de mensajes no sólo transmitirán afirmaciones simples, sino que pueden dar lugar a acciones más o menos complejas. Por ejemplo, cuando se pregunta "¿Me puede dar el libro?", normalmente no se espera un simple "sí" o "no", sino que se realiza una solicitud de entrega del libro, iniciándose en el receptor una acción. Las intenciones del emisor, en el mensaje, no son siempre obvias, sino que el contenido del mensaje depende del estado mental del emisor. Y lo mismo puede ocurrir con la interpretación y respuesta por parte del receptor. La teoría de los actos de habla es la disciplina que encarga de atender estas cuestiones.

2.1.2.4 Niveles y Protocolos de Comunicación.

Los protocolos de comunicación están generalmente especificados en diferentes niveles:

- En el nivel inferior del protocolo se especifica el método de interconexión.
- En el nivel medio se especifica la sintaxis o formato de la información que se va a transferir.
- Y en el nivel superior se especifica el significado o semántica de la información.

Dependiendo del número de receptores encontramos protocolos de comunicación binarios, cuando la comunicación es entre un emisor y un receptor, y n-arios, cuando existen múltiples receptores. Un protocolo viene definido por una estructura de datos con los siguientes cinco campos: emisor, receptor (o receptores), lenguaje en el protocolo, funciones de codificación y decodificación, y acciones realizadas por el receptor (o receptores).

Los agentes deberán de conocer el protocolo de comunicaciones, y este debe ser conciso y tener un número limitado de actos de comunicación primitivos.

Otra característica clave será separar la semántica del protocolo de comunicación de la semántica del mensaje encapsulado, ya que la primera es independiente del dominio mientras que la segunda no.

Todo ello hizo que la organización FIPA creara ACL (Agent Communication Language) como lenguaje que permita la intercomunicación entre agentes. En el capítulo 2.2 podemos ver una descripción del estándar ACL de FIPA.

2.1.2.5 Semántica y Ontologías.

El término "ontología" procede de la filosofía (parte de la filosofía que estudia el ente en cuanto tal). El término se ha reutilizado dentro de la inteligencia artificial y en concreto en los sistemas multiagente. De una manera formal, se puede definir la ontología como [14] "Una ontología define los términos y relaciones básicas que forman parte del vocabulario de una determinada área, así como las reglas para combinar términos y relaciones para definir extensiones del vocabulario".

De una manera más informal podemos decir que la ontología consiste en clasificar nuestro mundo en tipos de entidades. Una ontología será un catálogo de las clases de cosas que existen en un domino, desde la perspectiva de la persona que usa un lenguaje con el propósito de hablar acerca de ese dominio.

Dentro de una ontología podemos encontrar los siguientes elementos [15]:

- Conceptos: son las cosas sobre las que podamos emitir un comentario o juicio.
- Relaciones: representan las interacciones entre los conceptos del dominio.
- Funciones: son un caso especial de relaciones.
- Instancias: representan cada uno de los elementos concretos del dominio.
- Axiomas: sentencias que son siempre verdaderas en ese dominio.

Las dos principales fuentes de categorías ontológicas son la observación y el razonamiento. La observación proporciona conocimiento del mundo físico. El razonamiento genera un marco de abstracción llamado metafísica y da sentido a las observaciones. La ontología evita que los símbolos y términos estén mal definidos y/o sean confusos.

Para diseñar la base de datos de conocimiento lo primero que tendremos que hacer es elegir las categorías ontológicas. Esta elección de categorías determinará todas las cosas que pueden ser representadas en nuestro sistema multiagente. Un agente representará sus conocimientos con el vocabulario de una ontología específica. Para crear un agente deberemos de utilizar una ontología específica para representar su conocimiento.

Los agentes emplearán para comunicarse un vocabulario del lenguaje consistente en un diccionario de palabras apropiado para las áreas de aplicación comunes. Cada palabra en el diccionario tiene una descripción, escrita en lenguaje natural, que utilizaremos las personas para entender su significado y una notación formal que será la usada por el software. En los diccionarios es

posible añadir nuevas palabras dentro de áreas existentes y en nuevas áreas de aplicación. Las definiciones formales asociadas con cualquiera de estas ontologías pueden ser utilizadas por los agentes para traducir mensajes de una a otra ontología.

2.1.2.6 Comunicación Eficiente.

Hay un refrán popular que dice "a buen entendedor, pocas palabras bastan". Para conseguir que la comunicación sea eficiente, con menos mensajes y más cortos, podemos asumir que el agente utilizará su propio conocimiento para determinar el significado de un mensaje. Debemos de tener en cuenta los siguientes aspectos para mantener una comunicación eficiente:

- La utilización del contexto: el oyente y el hablante deben compartir un mismo contexto, pudiéndolo utilizar como fuente de conocimiento. De esta manera podemos determinar el significado de las expresiones e incluir en el lenguaje referencias y pronombres.
- La resolución de ambigüedades: al igual que en las expresiones del lenguaje natural, en la lingüística computacional se han identificado diversos tipos de ambigüedades: ambigüedad léxica (una misma palabra puede tener varios significados), ambigüedad sintáctica (mismas oraciones analizadas de distintas formas), ambigüedad referencial (debida al uso de pronombres y anáforas), y ambigüedad pragmática: (si el conocimiento de sentido común y el conocimiento sobre el contexto que posee el oyente no es preciso).

2.1.2.7 Negociación.

En la modelización de sistemas multiagente la negociación es un término importante. La negociación es el proceso de comunicación entre un grupo de agentes para consensuar los términos de un contrato. Dentro de la negociación realizada por los agentes de un sistema multiagente se definirán las asignaciones de subproblemas y recursos así los compromisos de participación y resultados por parte de cada uno de los agentes individuales. Según el comportamiento de los agentes podemos clasificar la negociación en dos tipos: negociación competitiva o cooperativa. A la hora de negociar nos podemos encontrar con varias situaciones:

- Ninguno de los agentes obtiene beneficio. Los intereses particulares no tienen relación con la negociación.
- Uno de los agentes alcanza su objetivo con menos esfuerzo y/o de forma más rápida.

- Existen situaciones reales de conflicto debido a recursos u objetivos compartidos.
- Cuando un agente participa en una negociación el agente puede desempeñar diferentes roles:
- Cooperación simétrica. La negociación aporta mejores resultados que los que cada agente obtendría de manera individual.
- Compromiso simétrico. El resultado individual para cada uno de los agentes es peor que las soluciones independientes, pero de esta forma se logra el objetivo global. Este tipo de negociación va a necesitar de un compromiso.
- Compromiso no simétrico. Uno de los agentes obtiene mejoras en su resultado individual a costa de otro que empeora.
- Conflicto. Debido a que los objetivos entran en conflicto no se llega a un acuerdo.

Dentro de una negociación las características más importantes son: el lenguaje utilizado por los participantes, el protocolo seguido para la negociación y el proceso de decisión (estrategias) que cada agente utiliza. Para tomar las decisiones el agente tendrá que determinar su posición, evaluar las concesiones y establecer los criterios para el acuerdo. Un mecanismo de negociación debe tener las siguientes características:

- Eficiencia: el gasto de recursos por parte de los agentes para llegar a un acurdo debe ser muy contenido.
- Estabilidad: ningún agente debe arbitrariamente desviarse de las estrategias de acuerdo.
- Simplicidad: los mecanismos de negociación deben intentar de ser sencillos desde el punto de vista computacional.
- Distribución: se debe de evitar un sistema de decisión central.
- Simetría: debemos de tratar que los agentes no presenten diferentes pesos en la negociación por razones arbitrarias o inapropiadas.

La negociación automática podría mejorar los contratos [16] y ahorrar tiempo a las personas. Existen distintas técnicas de negociación: votación, subasta, pacto, mecanismos de mercado, contratación y coaliciones.

En la última década han tomado relevancia los algoritmos genéticos dentro de las estrategias de negociación. En estos algoritmos cada agente genera un número aleatorio de estrategias de negociación y las aplica, guarda los

resultados y utiliza a posteriori las que han ofrecido mejores resultados. El problema aparece cuando tenemos un número grande de estrategias, ya que es inviable aplicar estos algoritmos. Otra novedad importante en la negociación es la teoría de juegos, donde las negociaciones son muy agresivas.

2.1.3 Arquitecturas de Agente.

Se han desarrollado muchos tipos de agentes inteligentes atendiendo a las tareas que tenían que ejercer. En cada uno de ellos se han primado unas características en base a los requisitos y restricciones a cubrir. Mientras para unos el recurso fundamental era el tiempo para otros primaba la información sobre la cual tomar decisiones. Esto ha dado lugar a que unos agentes serán más complejos e inteligentes que otros, y necesitaban arquitecturas distintas. No existe una única arquitectura ideal para agentes inteligentes, y esta dependerá de las tareas y el entorno. A continuación, vamos a analizar algunas arquitecturas de referencia que nos van a servir como representación del amplio abanico existente.

2.1.3.1 Arquitecturas Reactivas.

Las arquitecturas reactivas son muy utilizadas y se basan en la correspondencia estrecha que existe entre percepción y acción. Requiere pocos recursos computacionales y funciona bien en entornos de tiempo real.

Este tipo se agentes apareció para cubrir unos requerimientos de agentes compactos, rápidos, flexibles y tolerantes a fallos. Los agentes reactivos no requieren un modelo simbólico del entorno. Tampoco poseen capacidades para realizar procesos de razonamiento complejos. Los agentes reactivos, al contrario que otros tipos de agentes, no obtienen su inteligencia de modelos internos, sino que lo hacen de la interacción con su entorno, siendo de vital importancia esta interacción. Muchos de los investigadores que defienden la estructura reactiva defienden que la inteligencia es una propiedad emergente implícita de un entorno completo frente a la inteligencia en sistemas individuales.

Un agente reactivo no necesita obligatoriamente una estructura compleja para poder actuar en un entorno complejo. Debe de percibir el entorno cada cierto tiempo, reconocer unos estímulos concretos e iniciar unas acciones concretas (reacción).

En la Ilustración 3 podemos ver la arquitectura básica de los agentes reactivos que se corresponde con un sistema simple de estímulo/respuesta. Los sensores recogen la información (perciben), la envían a los módulos de competencia y estos producen una reacción a través de los actuadores que son los encargados de actuar sobre el entorno.

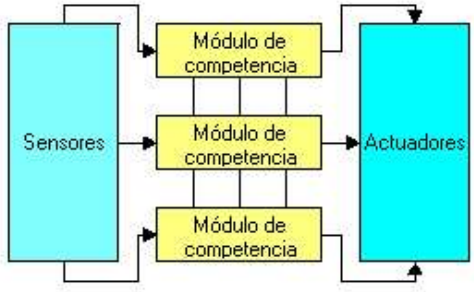


Ilustración 3: Arquitectura reactiva de agentes.

Los robots clásicos son un ejemplo simple de agentes reactivos. Un robot posee un determinado número de sensores que le permiten observar su entorno. Por ejemplo, puede determinar cuándo ha chocado con un obstáculo, que objetos han cambiado de posición o si otro robot está activo en su vecindad. Esta información es pasada a los módulos de competencia del robot. Por ejemplo, un módulo con la tarea de cambiar la dirección de movimiento está activo siempre que se detecte un obstáculo en el camino del robot. Los actuadores en este caso son las ruedas delanteras del robot, cuyo alineamiento cambia según las órdenes del módulo correspondiente.

En general, los sensores de un agente reactivo le permiten obtener información de su entorno y reconocer ciertos estímulos. La percepción del sistema dependerá del tipo de sensor, su número y distribución.

La información recogida por el sensor/es es transmitida al módulo de competencia correspondiente sin utilizan representaciones simbólicas o lenguajes de comunicación de alto nivel. Cada módulo de competencia tiene unas capacidades específicas y una tarea simple asignada. Los módulos tendrán las propiedades necesarias para llevar a cabo las tareas, no existiendo componentes centrales, como planificadores o módulos de razonamiento, por lo que un agente reactivo no podrá resolver una tarea para la que no tenga un módulo de competencia.

Los módulos de competencia trabajan en paralelo y pueden comunicarse directamente entre ellos o a través del entorno. La comunicación directa entre los módulos se basa en mecanismos de comunicación simples y no en lenguajes, lo que permite una rápida reacción. Sin embargo, cuando un módulo produce un cambio en su entorno y este es observado por otro módulo, que inicia su proceso de respuesta, es un método de comunicación es más lento, pero permite reaccionar a situaciones más complejas. Existe la posibilidad, al trabajar en paralelo, que varios módulos se disparen simultáneamente y esto requerirá de

un mecanismo para elegir entre las distintas acciones a realizar. Existen propuestas [17] de organización jerárquica en capas de estos módulos, donde las capas inferiores de la jerarquía estarían las competencias más prioritarias y podrían inhibir a las de capas superiores.

La estructura descentralizada de los módulos de competencia favorece la robustez y la tolerancia a fallos, ya que si un módulo funciona incorrectamente o cae el resto podrían seguir realizando sus competencias.

Los agentes reactivos puros no tienen capacidades de planificación. A pesar de que los planes podrían optimizar el comportamiento de los agentes, en la práctica hay factores que no los hacen viables como por ejemplo, la información incompleta del entorno, el carácter dinámico del mismo y de los objetivos, y la no disponibilidad de recursos.

También nos toca hablar sobre el comportamiento orientado a objetivos. La funcionalidad de los módulos competencia tienen objetivos específicos definidos que no se pueden cambiar; no pueden usar conocimiento interno para generar de forma dinámica nuevos objetivos. A pesar de estas restricciones, algunos investigadores piensan que los agentes reactivos son capaces de realizar acciones orientadas un objetivo, ya que defienden que la orientación hacia los objetivos resulta implícitamente de la interacción con el entorno.

2.1.3.2 Arquitecturas Deliberativas.

Las arquitecturas deliberativas están basadas en la inteligencia artificial simbólica. Newell y Simons [18] enunciaron una hipótesis en la cual un sistema de símbolos físicos capaz de manipular estructuras simbólicas puede exhibir una conducta inteligente. Para poder trabajar en el nivel de Conocimiento de Newell [18], nuestro problema pasa a cómo describir a nivel simbólico los objetivos y medios de satisfacerlos.

Las decisiones se toman empleando mecanismos deductivos: emparejamiento de patrones ("Pattern matching") y diversos formalismos lógicos.

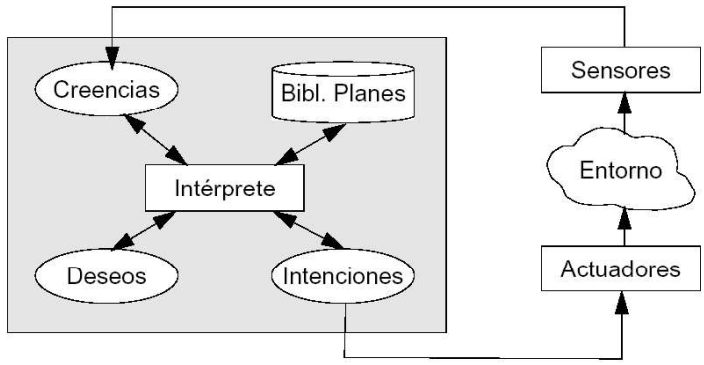


Ilustración 4: Arquitectura de agentes deliberativos.

Las arquitecturas de agentes deliberativos (Ilustración 4) suelen basarse en la teoría clásica de planificación de inteligencia artificial. La deliberación de un agente consiste en determinar qué pasos debe encadenar para lograr su objetivo, siguiendo un enfoque descendente ("top-down") a partir de un estado inicial, un conjunto de operadores/planes y un estado objetivo. Podemos citar los Softbots utilizados en la asistencia a los usuarios en realizar tareas típicas de Unix, como ejemplo de arquitectura cuyo componente principal es un planificador.

Podemos clasificar las arquitecturas deliberativas en 2 tipos: arquitecturas intencionales y arquitecturas sociales. Los agentes intencionales son capaces de razonar sobre sus creencias e intenciones, pudiéndose considerar como sistemas de planificación que incluyen creencias e intenciones en sus planes. Los agentes sociales se pueden definir como agentes intencionales que además mantienen un modelo explícito de otros agentes y son capaces de razonar sobre estos modelos.

2.1.3.3 Arquitecturas basadas en la Lógica.

En las arquitecturas basadas en la lógica, la toma de decisiones es vista como una deducción lógica. Tienen las ventajas de la claridad semántica y el poder aprovechar todos los avances y desarrollos de la lógica y la demostración automática. Tienen la desventaja de que las arquitecturas puramente lógicas no son adecuadas cuando tenemos restricciones de tiempo (respuesta en tiempo real).

Los agentes suelen ser utilizados en entornos de producción complejos, donde el fallo o un comportamiento incorrecto del agente puede ser negativo. Por lo tanto, se busca desarrollar técnicas que aseguren que los agentes se comportarán de una forma determinada. Trabajos previos en Ciencia de la

Computación han estudiado los métodos formales como una base para crear sistemas con un número mínimo de errores. La inteligencia artificial simbólica sugiere que un comportamiento inteligente puede generarse manipulando sintácticamente la representación simbólica del entorno y del comportamiento deseado. En los sistemas basados en la lógica, mediante fórmulas lógicas realizamos la representación simbólica del entorno, y con la deducción lógica y demostración de teoremas realizamos la manipulación sintáctica.

La información del entorno en tipo de agentes se guarda en una base de datos de fórmulas de predicados de lógica de primer orden. El proceso de toma de decisiones se modela a través de un conjunto de reglas de deducción (reglas de inferencia). Y de esta forma el comportamiento del agente quedará determinado ambos (la base de datos y las reglas de deducción).

Las aproximaciones basadas en la lógica presentan las siguientes desventajas:

- Son difíciles de utilizar en la práctica debido a las restricciones de tiempo. La demostración de teoremas es una tarea compleja hablando en términos computacionales, lo que no hace viable en problemas con restricciones temporales.
- Representar entornos complejos y dinámicos es difícil hacerlo con un conjunto de fórmulas.
- Si el entorno cambia significativamente tampoco será posible aplicarlos, ya que cuando vamos a aplicar las reglas de razonamiento estas ya no son válidas.

Debido a estas restricciones la formalización de los sistemas de agentes ha sido utilizada como un lenguaje de especificación interno usado por el agente para su razonamiento, o como metalenguajes externos usados por el diseñador para especificar, diseñar y verificar ciertas propiedades del comportamiento de los agentes situados en un entorno dinámico. El primer uso es más tradicional y presupone que los agentes poseen de forma explícita la capacidad de razonar. Estos agentes son normalmente conocidos como racionales, cognitivos o deliberativos. La segunda aproximación es más reciente en el estudio de agentes y permite usar el formalismo para permitir al diseñador razonar sobre el agente.

Se suelen utilizar los siguientes tipos de lógicas:

- Lógica proposicional y de predicados: es la más simple y la más usada. Las fórmulas se construyen a partir de proposiciones atómicas, que expresan hechos atómicos, y conectivas lógicas, que denotan “y”, “o”, “no” e “implica”. Los estados del sistema son descritos mediante las proposiciones atómicas y sus combinaciones mediante conectivas. No consideran la evolución del sistema.

- Lógica modal: es usado para dar significado a conceptos como creencia y conocimiento utilizando diferentes modos de verdad, como posiblemente cierto y necesariamente cierto. La lógica proposicional es extendida con dos operadores, el de posibilidad y el de necesidad.
- Lógica deóntica: permite indicar lo que un agente está obligado a hacer, introduciendo el operador de obligatoriedad.
- Lógica dinámica: puede ser vista como la lógica modal de la acción. Los operadores de necesidad y posibilidad están basados en el tipo de acciones disponibles. Las acciones son expresadas mediante expresiones regulares, siendo las acciones atómicas aquellas que el agente puede llevar a cabo directamente. La semántica es la de la lógica modal, pero incluyendo un conjunto de estados o mundos definidos por posibles transiciones basados en las acciones.
- Lógica temporal: hay diversas variantes según el tiempo sea visto como curso o cursos de la historia, como espacios temporales discretos o intermedios, o como puntos o intervalos de tiempo.

2.1.3.4 Arquitecturas BDI.

En las arquitecturas BDI (Belief-Desire-Intention), las tomas de decisiones se realizan sobre un proceso de razonamiento que parte de las creencias que el agente tienen del mundo y teniendo en cuenta las intenciones y las acciones.

La arquitectura consta de los siguientes componentes:

- Las creencias (belief), los deseos (desire) y las intenciones (intention) del agente.
- Las funciones que representan la deliberación: proceso donde el agente decide qué metas se desean conseguir la deliberación.
- El razonamiento de fines y medios (means-ends reasoning): el cual indica cómo se lograrán estas metas.

En la Ilustración 5 podemos ver el algoritmo seguido en el proceso BDI. Se empieza intentando comprender las opciones (objetivos) que están disponibles. Una vez identificadas debemos elegir una de las opciones y comprometernos (commit) con ella. Esta opción se convertirá en una intención que determinará las acciones del agente condicionando el razonamiento futuro al descartarse las opciones inconsistentes con la intención.

Una vez adoptada una intención, el agente debe perseverar ("persist") en ella, rectificándola sólo cuando se tenga la certeza que no se podrá alcanzar. Las

intenciones se relacionan con las creencias del futuro, el agente debe creer en las posibilidades de cumplir la intención.

```
InicializarEstado();
Repetir
       Opciones:=GeneradorDeObjetivos(colaDeEventos);
       OpcionesSeleccionadas:=Deliberar(Opciones);
       ActualizarIntenciones(OpcionesSeleccionadas);
       Ejecutar();
       ObtenerNuevosEventosExternos();
       BorrarObjetivosConseguidos();
       BorrarObjetivosImposibles();
FRepetir
```

Ilustración 5: Algoritmo seguido en el proceso BDI.

A nivel práctico un problema clave que nos encontramos en el diseño del razonamiento es el de mantener un buen equilibrio entre los diferentes intereses. A veces, los agentes deben abandonar algunas de sus intenciones, por ser inalcanzables, porque ya las ha conseguido o porque ya no se necesita. El agente tiene que replantearse las intenciones cada cierto tiempo, teniendo que equilibrar entre un comportamiento pro-activo (de cumplimiento de intenciones) y un comportamiento reactivo condicionado por los cambios. Si un agente que está reconsiderando constantemente sus intenciones puede no tener tiempo para realizar el trabajo para. Por otro lado, si no se las replantea podría quedarse anclado en objetivos imposibles de alcanzar. El entorno condicionará este equilibrio, a mayor dinamismo en él mayor exigencia de reacción requerirá el agente.

Este modelo presenta las ventajas de ser intuitivo y tiene un conocimiento informal de los términos creencias, deseos e intenciones similar al razonamiento práctico utilizado día a día. Además, da una descomposición funcional clara indicando que tipo los subsistemas va a requerir.

2.1.3.5 Arquitecturas Híbridas.

Las arquitecturas híbridas combinan módulos deliberativos y reactivos. Los reactivos serán los encargados de procesar los estímulos que no requieran deliberación, mientras que los módulos deliberativos determinarán las acciones para satisfacer los objetivos locales y cooperativos de los agentes.

2.1.3.5.1 Arquitecturas por Capas.

En las arquitecturas por capas, la toma de decisiones está dividida en capas o niveles, siendo el entorno visto con distinto nivel de abstracción en cada una de ellas.

Como vamos a tratar con comportamientos diferentes (reactivos y proactivos) debemos de realizar una descomposición clara en subsistemas. De

una forma natural aparece una arquitectura en la cual los subsistemas son jerarquizados en capas que interactúan entre sí.

Típicamente encontramos dos capas, una para tratar con el comportamiento reactivo y otra con el pro-activo. La comunicación entre las capas se puede establecer de forma horizontal y vertical. Cuando se hace de la horizontalmente cada capa está directamente conectada a las entradas sensoriales y a las salidas actuadoras; actuando cada capa por misma y produciendo sugerencias de qué acción se debe de realizar. Por el contrario, si se hace verticalmente solamente una capa se encarga de manipular las entradas sensoriales y las salidas actuadoras.

La ventaja de las arquitecturas en capas horizontales es su simplicidad conceptual, cada capa implementa un comportamiento diferente, pero existe el problema de conflictos entre módulos pudiéndose generar un comportamiento global del agente no sea coherente. Para asegurar el funcionamiento de las arquitecturas en capas horizontales generalmente se implementa una función de mediación que marca que capa controla al agente en un instante dado. El diseñador se ve obligado a considerar todas las diseñar esta función, la cual se puede convertir en un cuello de botella en el algoritmo de decisión del agente.

Las arquitecturas deliberativas se clasifican en horizontales, ya que los estímulos son recibidos del exterior, se procesan en varias capas con diferente nivel de abstracción y la última capa, la de nivel superior, decide que acciones se llevan a cabo (realizándolas directamente o encomendándolas a capas inferiores).

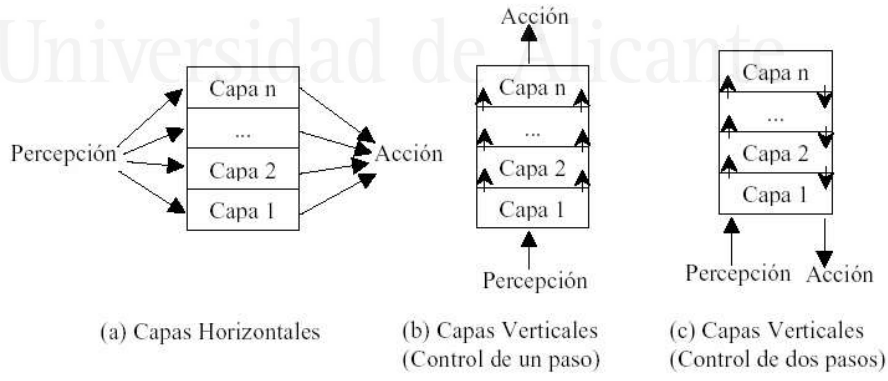


Ilustración 6: Arquitecturas por capas.

La arquitectura de capas verticales puede solucionar algunos de estos problemas. Atendiendo al flujo de control, las arquitecturas de capas verticales

pueden dividirse en 2 subtipos: arquitecturas de capas verticales de uno o dos pasos de control. En las primeras, el control fluye a través de cada capa hasta la capa final que es la que actúa. En las segundas, el flujo de información circula desde los niveles iniciales hacia los niveles superiores, mientras que el control fluye hacia las capas inferiores. En ambos subtipos, la complejidad de las interacciones entre las capas se reduce considerablemente, pero perdemos flexibilidad y robustez, ya que para tomar una decisión, el control debe de pasar todas las capas y se podría producir un fallo en cualquiera de las capas.

2.1.3.5.2 "Touring Machines".

Este tipo de arquitectura está basada en tres capas productoras de actividades organizadas horizontalmente, donde cada capa produce constantemente sugerencias sobre las acciones que el agente debería de realizar. Una de ellas es reactiva y provee de una respuesta más o menos inmediata a los cambios acontecidos en el entorno. Se implementa como un conjunto de reglas situación-acción.

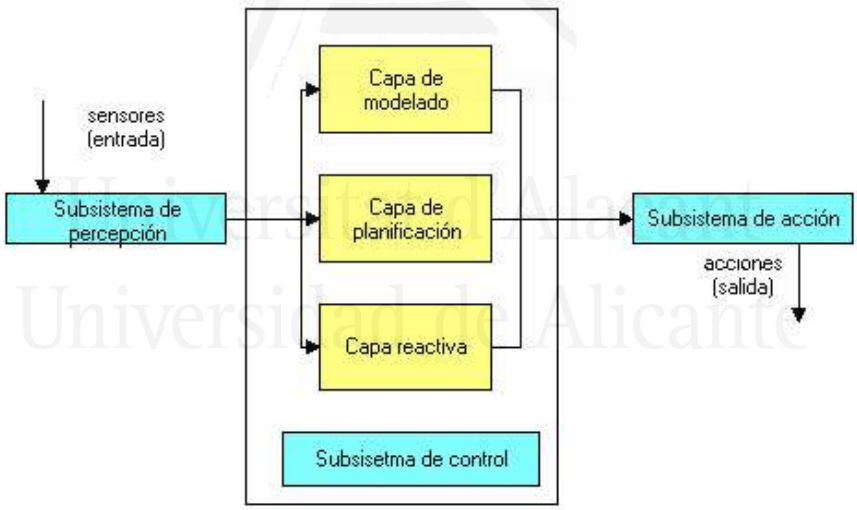


Ilustración 7: Arquitectura "Touring Machine".

La capa de planificación sirve para dotar de un comportamiento proactivo al agente. Bajo circunstancias normales, la capa de planificación es responsable de decidir qué hace el agente. Utiliza un conjunto de esqueletos de planes llamados esquemas. Estos esquemas son en esencia planes jerárquicamente estructurados que el agente elabora en tiempo de ejecución para decidir qué

hacer. Para llevar a cabo un objetivo, la capa de planificación trata de encontrar un esquema que se corresponda a dicho objetivo. Este esquema contendrá subobjetivos, que la capa de planificación utiliza para encontrar otros esquemas que se correspondan con los mismos.

La capa de modelado representa las diversas entidades del mundo (incluyendo el propio agente, así como otros agentes). Predice conflictos entre agentes y genera nuevos objetivos para resolver estos conflictos. Estos nuevos objetivos son pasados a la capa de planificación que busca los esquemas que los satisfagan.

Las tres capas de control están empotradas en un subsistema de control, que decide cuál de las capas tendrá control sobre el agente. Este subsistema de control se implementa como un conjunto de reglas de control. Estas reglas de control pueden suprimir la información de los sensores para alguna capa o censurar las acciones de alguna capa.

2.1.3.5.3 "InteRRaP".

"InteRRaP" es un ejemplo de arquitectura de agente dividida verticalmente en capas y en dos pasos, tal como se ve en el siguiente diagrama:

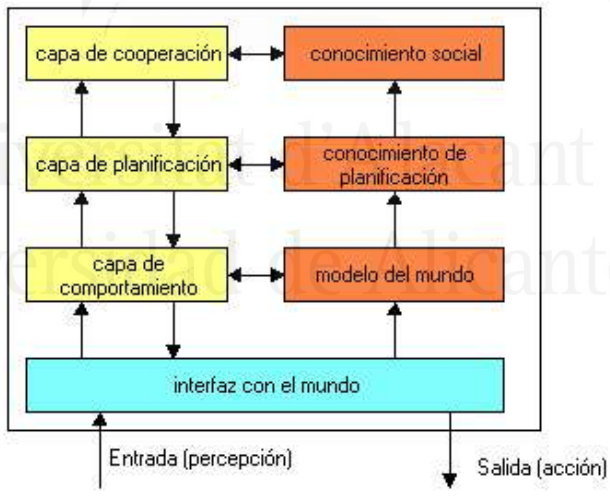


Ilustración 8: Arquitectura InteRRaP.

Como en las "Touring Machines" se tienen tres capas de control. Además, el propósito de cada capa "InteRRaP" parece coincidir con la capa correspondiente de la "Touring Machine". Cada capa está asociada con una base de conocimientos, por ejemplo, una representación del mundo adecuada para cada capa. Estas bases de conocimiento representan al agente y su entorno a

diferentes niveles de abstracción. La base de conocimientos del nivel más alto representa los planes y las acciones de los otros agentes en el entorno; la base de conocimiento del nivel medio representa los planes y acciones del propio agente; y la base de conocimiento del nivel más bajo representa información sobre el entorno. La explícita introducción de estas bases de conocimiento distingue las "InteRRaP" de las "Touring Machines".

La forma en que las diferentes capas en "InteRRaP" funcionan para producir el comportamiento también difiere de las "Touring Machines". La principal diferencia es la forma en que las capas interactúan con el entorno. En las "Toruing Machines" cada capa tenía como entrada las percepciones del entorno y como salida una acción en el entorno. Esto introducía la necesidad de un módulo de control para tratar los conflictos. En "InteRRaP", las capas interactúan con el resto para conseguir la misma finalidad. Los dos tipos de interacción entre capas son la activación "bottom-up" y la ejecución "top-down". La activación "bottom-up" ocurre cuando una capa inferior pasa el control a una capa superior porque no es competente para tratar con la situación actual. La ejecución "top-down" ocurre cuando una capa superior hace uso de los recursos que provee una capa inferior para conseguir sus objetivos. El flujo básico de control comienza cuando una entrada (percepción) llega a la capa más baja de la arquitectura. Si la capa de comportamiento puede tratar con esta entrada, entonces lo hará. En caso contrario, ocurrirá una activación hacia arriba y el control será pasado a la capa de planificación. Si la capa de planificación puede manejar la situación, entonces lo hará, normalmente haciendo uso de la ejecución hacia abajo. En otro caso, usará una activación hacia arriba para pasar el control a la capa más alta. De esta forma, el control fluirá de la capa más baja a las capas más altas de la arquitectura y posteriormente en sentido descendente otra vez.

2.2 El Estándar FIPA (Foundation for Intelligent Physical Agents).

FIPA es una organización sin ánimo de lucro fundada en suiza en 1996 para el desarrollo y establecimiento de estándares de software para [14] heterogéneos que interactúan y sistemas basados en agentes. Nació con el objetivo de definir un conjunto completo de normas para la implementación de sistemas en los que se puedan ejecutar agentes (plataformas de agentes) y especificación de cómo los propios agentes se deben comunicar e interactuar.

Las especificaciones FIPA han ido progresando por varios estados estandarizados: desde un estado preliminar (borrador en construcción, identificados por la inicial P), experimental (especificación estable donde sólo se admiten pequeños cambios, identificados por la inicial X) hasta el estado estándar (especificación implementada con éxito en varias plataformas; que se identifican por la S). Cualquier especificación puede ser reprobada (en este caso se identifica al empezar en D), previo al estado de obsoleta (su identificador empieza por O), cuando se considera innecesaria por cambios en la tecnología o en otras especificaciones y ha quedado por tanto obsoleta.

Entre sus miembros la organización ha contado con varias instituciones académicas y un gran número de empresas, entre ellas: British Telecom , Fujitsu, Hewlett-Packard, IBM, Sun Microsystems. Algunas normas fueron propuestas, y vieron la luz varias plataformas de agentes que adoptaron el "estándar FIPA" para la comunicación de agentes, pero no se logró obtener el apoyo comercial que se había previsto y la organización se disolvió en 2005. En su lugar se creó un comité de estándares IEEE.

Las especificaciones FIPA se organizan en cinco materias: Arquitecturas Abstractas, Comunicación entre Agentes, Aplicaciones, Gestión de Agentes y Transporte de Mensajes de los Agentes. De ellas las más utilizadas han sido la de gestión (Agent Management) y la de comunicación (Agent Communication Language).

2.2.1 Agent Management.

En la Ilustración 9, podemos ver los componentes del sistema de control del modelo de referencia especificado por FIPA (FIPA Agent Management) [19]:

- **AP (Agent Platform):** dota de la infraestructura física donde el conjunto de agentes va a ser desplegado. Está formado por ordenadores, sistema operativo, agentes y componentes de gestión sugeridos por las normas FIPA.
- **Agente:** es un proceso de computación dentro de la plataforma (AP) ofreciendo los servicios que pueden ser utilizados por otros agentes. Estos agentes publicarán en la plataforma los servicios que ofrecen.
- **DF (Directory Facilitator)**: también es conocido como el servicio de páginas amarillas, permitiendo a los agentes publicar una descripción de los servicios que ofrecen y buscar servicios que demandan. Proporciona información del directorio de agentes actualizada, completa y precisa.

- **AMS (Agent Management System):** es un componente muy importante y obligatorio en la plataforma, que realiza el rol de controlar la plataforma y proporciona el servicio de páginas blancas. En él se centralizan las tareas de gestión de la plataforma permitiendo crear, eliminar o migrar de una plataforma a otra agente. Cuando un agente se registra en el AMS obtiene **AID (Agent Identifier),** que será conservado por el AMS como un directorio de todos los agentes de la AP junto con su estado (activo, suspendido, esperando, etc.). En cada AP sólo puede existir un AMS.
- **MTS (Message Transport Protocol)**: es el servicio que proporciona una AP para poder transportar mensajes (comunicar) FIPA-ACL entre agentes de una misma o distinta plataforma.

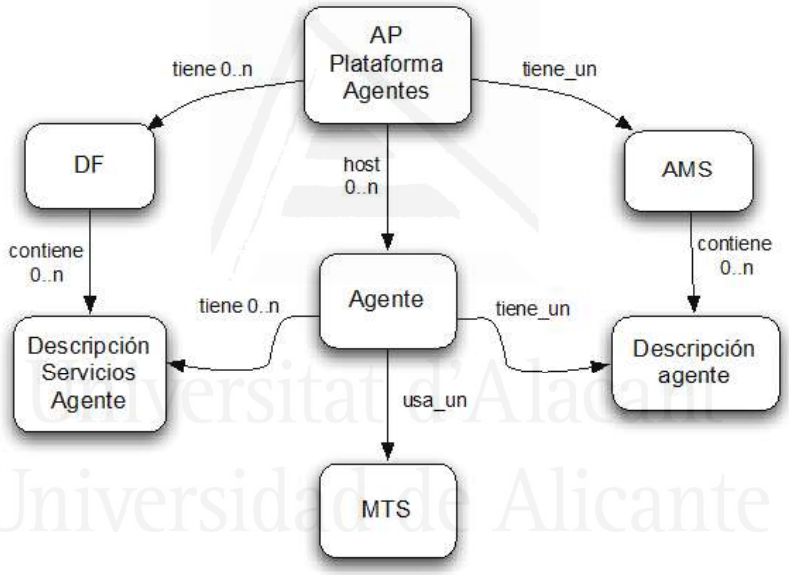


Ilustración 9: Componentes de gestión de agentes especificado por FIPA.

2.2.2 Agent Communication Language.

El lenguaje de comunicación de agentes ACL (Agent Communication Language) está organizado en Protocolos de Interacción para el intercambio de mensajes, Actos Comunicativos basados en la teoría de los actos de habla y Lenguajes de Contenido (Ilustración 10). Cada mensaje ACL está compuesto de una serie de campos como podemos ver en la Ilustración 11, definiendo remitente, destinatario, contenido del mensaje y parámetros de control.

Ilustración 10: Especificaciones de ACL.

FIPA propone distintas especificaciones para representar el contenido de los mensajes en distintos lenguajes: SL ("Semantic Language"), CCL ("Constraint Choice Language"), KIF ("Knowledge Interchange Format") y RDF ("Resource Descripction Framework").

Elemento	Categoría del Elemento
performative	Tipo de acto comunicativo
sender	Participante en la comunicación
receiver	Participante en la comunicación
reply-to	Participante en la comunicación
content	Contenido del mensaje
language	Descripción del contenido
encoding	Descripción del contenido
ontology	Descripción del contenido
protocol	Control de la conversación
conversation-id	Control de la conversación
reply-with	Control de la conversación
in-reply-to	Control de la conversación
reply-by	Control de la conversación

Ilustración 11. Elementos de un mensaje en ACL de FIPA.

FIPA propone distintas especificaciones para representar el contenido de los mensajes en distintos lenguajes: SL ("Semantic Language"), CCL ("Constraint Choice Language"), KIF ("Knowledge Interchange Format") y RDF ("Resource Description Framework").

2.3 Metodología para la Construcción de Sistemas Multiagente.

A finales del siglo XX se acuño el término Agent Oriented Software Engineering (AOSE). Ya en aquella época, se vislumbraba que los agentes serían el futuro de unos sistemas complejos, distribuidos y donde la interacción era un rasgo característico [20] [21]. Las características de estos sistemas hicieron necesarias técnicas específicas para analizar, diseñar y evaluar este tipo de sistemas.

La falta de herramientas adecuadas para la ingeniería del software orientada a agentes llevó a la aparición de gran variedad de metodologías [22]. Una metodología orientada a agentes consiste en un conjunto de métodos orientados a agentes. Un método en el contexto de ingeniería del software son "elementos que estructuran y guían las actividades concernientes a ingeniería del software, con el fin de hacerlas sistemáticas" [22]. En base a esta definición, una metodología de ingeniería del software orientada a agentes supondrá un método ingeniería del software en la cual la abstracción principal será el agente software.

A la hora de definir una metodología habrá que concretar los resultados a producir (Documentación, código, prototipos, pruebas, etc), así como el lenguaje para definir los resultados (Diagramas UML, lenguajes formales, etc. Además, habrá que establecer las guías, las métricas y las pautas de ingeniería del software a seguir (estructuración de actividades, gestión del proyecto, formación del equipo de desarrollo, etc.) y las herramientas a utilizar).

En general, en los sistemas multiagente se aplican una mezcla de técnicas de orientación a objetos (estructura del sistema, encapsulado del comportamiento, etc), de sistemas de gestión de conocimiento (definición y representación del comportamiento) y de inteligencia artificial distribuida (organización, coordinación y comunicación) como hemos visto en apartados anteriores. En la Ilustración 12, podemos ver las distintas áreas de conocimiento (orientación a objetos, sistemas expertos, métodos formales, análisis de roles, integración o BDI) a partir de las cuales han surgido una serie de metodologías para analizar, diseñar e incluso en algunas implementar sistemas multiagente.

En los siguientes apartados de este capítulo haremos una visión general las principales metodologías orientadas al desarrollo de sistemas multiagente.

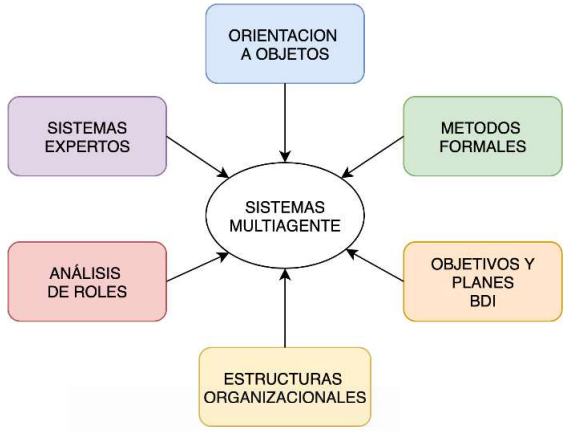


Ilustración 12. Distintas áreas de conocimiento dan lugar a las diferentes metodologías de sistemas multiagente.

2.3.1 GAIA.

Gaia [23][24], es una metodología que a lo largo de los años no ha evolucionado mucho. Pretende ser universal e independiente de las tecnologías destino (framework, plataforma de agentes) con el fin de servir para el número máximo de sistemas. Está basada en la teoría organizacional, modelando los sistemas a través de los siguientes conceptos: entorno, estructuras de la organización, roles, interacciones y reglas. A partir de estos conceptos se construyen los modelos en cada una de las etapas de la metodología.

La metodología Gaia divide el trabajo en dos fases: análisis y diseño, al cual subdivide en diseño arquitectónico y diseño detallado. Esta metodología no cubre la las fases de especificación de requisitos ni de implementación.

El objetivo de la fase de análisis es identificar las posibles suborganizaciones, modelar el entorno, y elaborar los modelos preliminares de roles e interacciones. Dentro de esta fase también se definen las reglas de la organización.

En la fase de diseño, se comienza por definir la estructura organizacional, estructurando los roles elaborados en la fase anterior.

Gaia propone unas guías para establecer la topología jerárquica y control dentro de la organización. Diseña las interacciones en base a estos elementos y las reglas.

Siempre desde el punto de vista de la estructura de la organización, se promueven patrones y representaciones grafica de las relaciones (control, dependencia o igualdad) entre los diferentes roles. No propone una notación específica, dando por válida, por ejemplo, la utilización de un grafo donde las aristas representan las relaciones y los nodos los roles.

Dentro del diseño arquitectónico se elaboran los modelos completos de roles e interacciones. En los modelos de roles se definen todas las actividades y responsabilidades de un rol, y se añaden al conjunto de roles los roles organizacionales, derivados de la elección de la estructura de la organización. Para completar el modelo de interacciones se definen los protocolos que se adoptan como consecuencia de nuevo de la elección de la estructura organizativa.

En la última fase de la metodología, diseño detallado, elaboraremos el modelo de agentes y servicios. Dentro del modelo de agentes se definen las clases de agentes, los roles que podrán desempeñar y el número de instancias de esa clase dentro del sistema. De nuevo la representación está abierta, pudiendo, por ejemplo, utilizar una simple tabla para representar el modelo de agentes donde en una columna tendríamos los agentes y en otro sus respectivos roles.

En Gaia un agente es una entidad software de grano grueso, donde un agente desempeñará sólo un rol o un número muy pequeño. El modelo de servicios determinaría las funcionalidades que se le asignan a cada rol, donde un servicio supone la identificación de sus entradas/salidas, precondiciones y postcondiciones. Gaia no impone restricciones en la implementación de servicios.

Una primera versión de Gaia consideró un modelo adicional, el modelo de "acquintances" en el cual se representaban las comunicaciones entre las clases de agente.

Los modelos de Gaia no se ajustan a ningún estándar ni meta-modelo específico, utilizando siempre una notación bastante informal, incluso dejando libertad al diseñador la elección. En una metodología siempre esperamos un mayor rigor.

Otras características negativas de Gaia son que utiliza un modelo de proceso secuencial y no da ningún soporte a la implementación.

2.3.2 TROPOS.

Tropos [25][26] es una metodología de desarrollo orientada a agentes que pone énfasis en la identificación y el análisis de requisitos.

El desarrollo seguido en Tropos se compone de cinco fases: análisis de requisitos iniciales, análisis de requisitos tardíos, diseño arquitectónico, diseño detallado, e implementación. Inicialmente Tropos no consideraba la fase de implementación, proponiendo simplemente una serie de guías. Debido a esto, es habitual que en otras descripciones de Tropos se diga que tiene sólo cuatro fases.

Durante la fase de análisis requisitos iniciales, se identifica el ámbito de la aplicación, interesados relevantes del dominio y sus relaciones. Para representarlos utiliza los diagramas de actores y de justificación ("rationale").

El diagrama de actores, representa a los interesados identificados (actores) con sus relaciones (dependencias). Se utiliza un grafo dirigido donde los nodos son actores, donde el nodo origen es el actor dependiente y el nodo destino es el actor dependido. La arista representa la dependencia que existe entre ellos, pudiendo ser de tres tipos (objetivos, tareas, recursos).

Los diagramas de actores son completados con los diagramas de justificación, donde se descomponen los objetivos en subobjetivos, y amplían sus dependencias con otros actores. Este proceso se considera incremental, al ser habitual encontrar la identificación de nuevas dependencias. Este diagrama, más tarde, pasó a denominarse diagrama de objetivos, siendo este nombre más representativo con su función.

La siguiente fase, análisis de requisitos tardíos, repite lo visto en pasos anteriores, considerando en este caso un nuevo actor, al del sistema que se pretende construir. Utiliza los mismos diagramas identificando los requisitos funcionales y no funcionales, y describiendo al sistema dentro del entorno donde se desplegará.

En la fase de diseño arquitectónico se configura la estructura de la organización del sistema descomponiéndola en subsistemas, los cuales son relacionados mediante dependencias. Esta fase tiene tres etapas: definición de la estructura general del sistema, especificación de las capacidades disponibles para cada autor y la definición de tipos de agentes, donde se le asignan a los agentes las capacidades identificadas.

Para definir la estructura general del sistema, se utilizan estilos, a modo de patrones, donde se presentan las asignaciones de objetivos y procesos. "Structure-in-5", "Strategic Apex", "Middle Line" son algunos ejemplos de estos estilos.

La fase de diseño detallado, consiste en aumentar el nivel de detalle respecto a los componentes identificados en la fase anterior, definiendo los protocolos que guiarán las comunicaciones. Tropos propone diferentes patrones sociales, que representan situaciones habituales de interacción. Según la naturaleza de la interacción se dividen en dos categorías: pareja y mediación.

Los patrones de pareja representan interacciones entre actores "de par a par". Los patrones de mediación, como su nombre indica, representan situaciones donde es necesario un intermediario.

Durante la fase de implementación, se intenta trasladar el diseño a código de la manera más natural posible, desde la semántica del diseño hacia la plataforma de agentes elegida.

Tropos tiene herramientas de soporte entre las cuales destaca TAOM4E, un software libre que se integra con Eclipse, y que aporta generación de código y test usando un desarrollo dirigido por modelos.

2.3.3 PASSI.

La metodología PASSI [27][28] integra conceptos y modelos orientados a objetos con ingeniería del software orientada a agentes y cubre todas las fases del desarrollo (desde la especificación de requisitos hasta la implementación). PASSI intenta utilizar estándares siempre que sea posible: usa UML como lenguaje de modelado, XML para definir las comunicaciones y FIPA como plataforma destino de la implementación.

PASSI divide el proceso de desarrollo en cinco fases, donde cada una de ellas tiene un modelo asociado. Estos modelos son: System Requirements Model, Agent Society Model, Agent Implementation Model, Code Model y finálmente, Deployment Model. Cada fase está compuesta de una serie de etapas. Utiliza un proceso iterativo, que cuenta explícitamente con una fase de implementación, y donde se realizan tests entre iteraciones.

En [28] se encuentran el esquema de las fases y etapas de PASSI, que se muestra en la Ilustración 13: Fases y etapas de PASSI A continuación pasamos a describir cada una de estas fases.

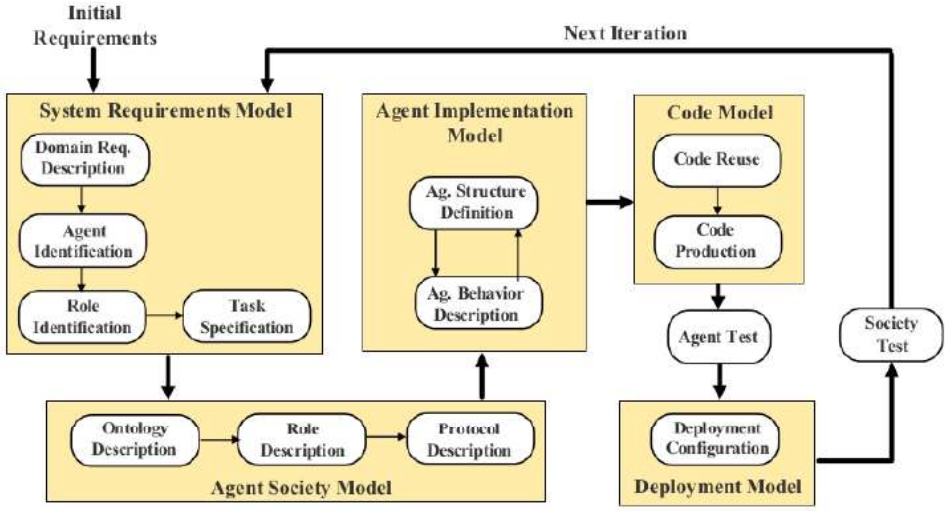


Ilustración 13: Fases y etapas de PASSI [28].

- System Requirements Model: está dividida en cuatro etapas y su objetivo es reunir los requisitos del sistema. Su primera etapa (Domain Requirements Description) recoge los requisitos de los escenarios posibles en diagramas de casos de uso. El resto de etapas de esta fase describen una especificación de alto nivel de los agentes, mediante una identificación de agentes preliminar y repartiendo entre ellos los casos de uso. Se plantean los roles a través de diagramas de secuencia que representan los escenarios principales del sistema. Finalmente, mediante diagramas de actividades, se especifican las tareas asignadas a cada uno de los agentes.
- Agent Society Model: describe el conocimiento y las comunicaciones, es decir, la ontología del sistema. En sus diferentes etapas, se describe la ontología mediante diagramas de clases y se modelan las interacciones. Posteriormente, mediante otro diagrama de clases, donde los agentes se representan con paquetes y los roles con clases, se asignan los roles a las clases. Para concluir esta fase, se especifican los protocolos de comunicaciones entre agentes, siguiendo los contenidos en el estándar FIPA o si hay nuevos definiéndolos mediante diagramas de secuencia AUML.
- Agent Implementation Model: comienza definiendo la estructura general del sistema y la estructura individual de los agentes. Para lo primero, se utilizará un diagrama de clases, el cual representará las interacciones significativas con el entorno mediante actores. Para la estructura individual, utilizaremos un diagrama de clases.

Una vez definida la estructura, definiremos el comportamiento de los agentes a nivel global mediante diagramas de actividades, mostrando las diferentes tareas ejecutadas, así como los mensajes intercambiados. El comportamiento local de los agentes se expresa con una descripción informal. Todo esto se basa en la idea de que la estructura del sistema y de los agentes condiciona el comportamiento de estos, y viceversa.

- Code Model: ocupa el lugar de la fase de implementación tradicional. En esta fase se generará, mediante alguna herramienta de soporte, el código a partir de los modelos. Estas herramientas deben de incluir la generación de plantillas y la reutilización de patrones.
- Deployment Model: describe el despliegue del sistema. Se utilizan diagramas de despliegue, pero con una sintaxis extendida con el fin de poder representar la movilidad de los agentes. El fin de la iteración se realizará mediante los tests en el sistema desplegado.

Entre las herramientas de soporte cabe destacar PASSI ToolKit (PTK), una herramienta que incluye generación de código a nivel de plantilla (template) y una programación manual a posteriori.

2.3.4 PROMETHEUS.

Prometheus [29] es una metodología orientada a agentes que tiene como objetivo ser de propósito general y práctica. Aunque se defina como general está muy especializado en el desarrollo de sistemas con agentes basados en objetivos y planes, proporcionando herramientas para facilitar su desarrollo. Con el objetivo de práctica quiere decir que puedan utilizarla distintos tipos de usuarios, desde personas que se inician en el desarrollo de sistemas multiagente hasta profesionales de la industria.

Prometheus [29] define agente como "una entidad software autónoma, situada en un entorno, reactiva a los cambios en dicho entorno, proactiva en la persecución de objetivos y con habilidades sociales, siendo además flexible y robusta". Para Prometheus identifica los siguientes conceptos: "un agente se relaciona con su entorno mediante sensores y acciones, será proactivo si persigue objetivos, será reactivo si reacciona a eventos, tendrán planes y beliefs, y se comunicarán mediante mensajes que responden a determinados protocolos de interacción". Gracias a estos términos, podemos definir agentes que se ajustan a cualquier arquitectura de agentes (BDI, reactiva, etc.). Como otras metodologías, Prometheus propone un proceso en fases, en su caso concreto en tres: especificación del sistema, diseño arquitectónico y diseño detallado.

- En la fase de especificación del sistema se definen los requisitos del mismo en términos de objetivos, funcionalidades, escenarios, así como el conjunto de sensores y actuadores. Los objetivos representan los requisitos y pueden formar parte de otros objetivos, en cuyo caso conceptualmente hablaríamos de subobjetivos. Las funcionalidades se definen a partir de los objetivos, como partes del comportamiento o visto de otra forma, como las habilidades que poseerá para poder conseguir sus objetivos. Las funcionalidades tienen asociados descriptores los cuales muestran sus objetivos relacionados, las acciones que pueden realizar y las situaciones que desencadenarían dichas funcionalidades. Los escenarios muestran cómo se reacciona ante un evento o las secuencias de acciones seguidas para alcanzar un objetivo. En esta fase también definiremos el entorno donde se situará el sistema y definiremos las interfaces (sensores y actuadores) de los agentes.

- La fase de diseño arquitectónico se centra en diseñar la estructura general del sistema, en identificar las clases de agentes y en describir cómo interactúan entre sí. La estructura general del sistema consiste en un grafo dirigido donde los nodos donde los estos pueden ser agentes, sensores, actuadores, mensajes y conjuntos de datos, mientras que las aristas indican las interacciones. La identificación de agentes se hará en base a las funcionalidades y los conjuntos de datos a manejar, y se representa mediante un "Data coupling diagram". A partir de este diagrama agrupamos las funcionalidades y los conjuntos de datos claramente relacionados. El diagrama "Agent acquaintance diagram" muestra la comunicación entre clases. Y a partir de estos diagramas, se elaboran los "Interaction diagram" e "Interaction protocols", que son diagramas de secuencia UML.

- En la fase de diseño detallado concretaremos los aspectos internos de los agentes: definiremos los agentes en función de "capabilities", desarrollaremos los "process diagrams" a partir de los "interaction protocols", y desarrollar las "capabilities" en términos de otras "capabilities", planes, conjuntos de datos y eventos. Para facilitar el encaje de todas estas piezas, se utiliza un diagrama tipo "Agent overview diagram", donde se representa el agente en función de "capabilities" y de la comunicación de estas.

Junto con esta metodología se ha desarrollado la herramienta Prometheus Design Tool (PDT), la cual incorpora editores gráficos para los diagramas que utiliza la metodología y elementos supervisores de control de la consistencia del sistema. Esta herramienta también permite exportar los diseños hacia JACK

Development Environment (JDE) y generar código JACK automáticamente a través de templates.

Prometheus no soporta la movilidad de los agentes ni trabaja en profundidad la identificación de requisitos, como si hace Tropos.

2.3.5 MaSE.

MaSE [30] (Multiagent System Engineering), es una metodología que fue desarrollada por el Air Force Institute of Technology. Esta metodología tiene en cuenta todas las fases del proceso de construcción de un sistema multiagente, desde la especificación hasta su implementación. Utiliza un lenguaje de especificación basado en UML+OCL y tiene una herramienta de desarrollo (AgentTool) que de momento no cubre todas las fases de proceso de la metodología.

Esta metodología define al agente como "una abstracción útil para resolver problemas en dominios específicos". Atendiendo a esta definición podemos caracterizar a los agentes de la siguiente manera:

- Son entidades que pueden estar distribuidas.
- Son autónomos, trabajan en sociedad y son dirigidas por objetivos.
- La información es compartida entre agentes de manera interactiva.

Dentro de la metodología MaSE se utilizan distintos puntos de vista:

- Captura de objetivos.
- Casos de uso.
- Establecimiento de roles.
- Creación de clases (agentes).
- Diseño de mensajes (conversaciones).
- Integración de agentes.
- Diseño del sistema

2.3.6 MAS-CommonKADS.

La metodología MAS-CommonKADS [31] nace como una extensión CommonKADS incorporando ideas de metodologías orientadas a objetos para la construcción de sistemas multiagente, aunque su enfoque está dirigido al desarrollo de sistemas expertos que interactúan con el usuario. CommonKADS

consideraba solamente 2 agentes básicos: el agente sistema y el agente usuario, mientras que MAS-CommonKADS extiende los modelos para tener más elementos y posibilitar la interacción entre ellos. MAS-CommonKDAS fue la primera metodología en integrar un ciclo de vida software con el desarrollo de sistemas multiagente, en concreto trabajaba con el modelo en espiral dirigido por riesgos.

Esta metodología trabaja en base a modelos del sistema, en concreto tiene los siguientes 7: agente, tareas, experiencia, comunicación, coordinación, organización y diseño. Cada modelo se crea con una descripción gráfica y luego se va complementando con explicaciones en lenguaje natural. Cada modelo está relacionado con otros a través de descripciones de dependencias y de las actividades involucradas. La descripción de los modelos en lenguaje natural, se complementa con otras notaciones como SDL (Specification and Description Language) o MSC (Message Sequence Chart) con el fin de describir la interacción necesaria en el comportamiento de los agentes.

Aunque a define una metodología para sistemas multiagente de propósito general, es una metodología diseñada para el desarrollo de sistemas basados en conocimiento (SBC). Su desarrollo fue financiado por la Comunidad Europea a través de varios proyectos entre 1983 y 1994. Siguiendo la aproximación de los SBC esta metodología construye unos modelos interrelacionados los cuales capturan las características principales del sistema y de su entorno. El proceso de desarrollo implica rellenar un conjunto de "plantillas" para cada modelo.

MAS-CommonKADS propone los siguientes modelos:

- Modelo de Agente (AM): define las especificaciones de un agente: clase de agente, grupo/s a los que pertenece, razonamiento, habilidades, servicios, sensores y actuadores. Un agente puede ser cualquier entidad capaz de utilizar el lenguaje de comunicación establecido, ya sea un proceso software o incluso una persona.
- Modelo de Organización (OM): Es utilizado para el análisis de la organización humana donde se van a integrar los agentes. En este modelo también se describe la organización entre los agentes software y su relación con el entorno.
- Modelo de Tareas (TM): describe las tareas que pueden realizar los agentes: los objetivos, su descomposición, los ingredientes y los métodos de resolución de problemas para la resolución de cada objetivo.
- Modelo de la Experiencia (EM): se describe el conocimiento que necesitan los agentes para alcanzar sus objetivos. Sigue la

descomposición utilizada en CommonKADS reutilizando bibliotecas de tareas genéricas.

- Modelo de Comunicación (CM): describe las interacciones entre los agentes humanos y los agentes software.
- Modelo de Coordinación (CoM): este modelo describe las interacciones entre agentes software.
- Modelo de Diseño (DM): los modelos anteriores trabajan el análisis del sistema multiagente, mientras que este modelo es utilizado para describir la arquitectura y el diseño como paso previo a su implementación.

El desarrollo de sistemas multiagente con CommonKADS sigue un modelo de ciclo de vida con las siguientes fases:

- Conceptuación: se hace un estudio para determinar de dónde se adquiere y extrae el conocimiento, con el fin de obtener una primera descripción del problema. Se determinan los casos de uso con el fin de ayudarnos a entender los requisitos informales y más tarde a probar el sistema.
- Análisis: a partir del enunciado del problema se establecen los requisitos del sistema. En esta fase se desarrollan los modelos siguientes: organización, tareas, agente, comunicación, coordinación y experiencia.
- Diseño: se desarrolla un modelo de diseño capaz de hacer lograr los requisitos establecidos en la fase de análisis. Se establecen las arquitecturas de cada agente y de sistema multiagente.
- Codificación y prueba: se codifica y prueba cada agente.
- Integración: el sistema completo es probado.
- Operación y mantenimiento.

Como hemos dicho al comienzo de este apartado, esta metodología utiliza un modelo de ciclo de vida en espiral dirigido por riesgos, siguiendo la gestión de proyectos de CommonKADS. Sin embargo, para el aprendizaje de la metodología y para proyectos pequeños, se propone seguir un modelo de ciclo de vida basado en el modelo en cascada combinado con la reutilización de componentes.

Ya que en los sistemas multiagente se necesita la interoperabilidad del software, la metodología hace énfasis en el empleo de reutilización, aconsejándola en proyectos grandes y pequeños. A la hora práctica se apuesta

por los estándares tipo OpenDoc, CORBA u OLE2.0. Desde el punto de vista funcional se identifican varios tipos de componentes: el agente, las ontologías, los métodos de resolución de problemas, los servicios del agente, y sus objetivos y sus planes. La tarea de búsqueda y clasificación de componentes se realiza mediante la definición de plantillas textuales para cada uno de los componentes desarrollado en cada uno de sus modelos.

El ciclo de vida en espiral dirigido por riesgos, utilizado en proyectos grandes tiene las siguientes fases:

- Identificación de los objetivos y establecimiento de productos a desarrollar para satisfacer dichos objetivos.
- Identificación y análisis de los riesgos asociados tanto a objetivos como a al desarrollo de productos.
- Planificación del proyecto definiendo las actividades necesarias para desarrollar los productos y asignando los recursos necesarios a dichas actividades.
- Estableciendo los planes de monitorización y control para cumplir con los criterios de calidad establecidos.

2.3.7 MESSAGE.

MESSAGE (Methodology for Engineering Systems of Software Agents) [32] es una metodología orientada a agentes la cual soporta las fases el análisis y diseño de sistemas multiagente, lanzando ideas sobre el resto las fases de implementación, pruebas e implantación. Esta metodología provee un lenguaje, un método y unas guías, centrándose en las fases de análisis

MESSAGE hace uso de un metamodelo para el modelado del sistema multiagente, donde toda entidad etiquetada como un agente tiene que tener un conjunto de atributos determinado. Una extensión de los metamodelos y su puesta en marcha es la metodología INGENIAS.

2.3.8 INGENIAS.

La metodología INGENIAS [33][34], es una metodología soporta el desarrollo de sistemas multiagente desde el análisis hasta la implementación. Sigue un proceso de desarrollo parecido al proceso unificado, y define un conjunto de actividades para guiar durante todo el proceso.

INGENIAS, utiliza diferentes vistas del sistema que se especifican mediante metamodelos. Estos metamodelos se complementan con diagramas

UML. Los metamodelos con los que cuenta esta metodología son: modelo de organización, modelo de agente, modelo de interacción, modelo de entorno, y modelo de tareas y objetivos.

El modelo de organización, diseña la estructura del sistema, para ello define un espacio común para los agentes, sus recursos y sus tareas/objetivos El modelo de organización, divide estos elementos en base a grupos y flujos de trabajo (workflows), y define sus funcionalidades mediante tareas y objetivos.

En el modelo de agente se diseña el comportamiento de agentes individuales en función de sus roles, tareas, objetivos y su estado mental.

El modelo de interacción define el contexto, el objetivo, los actores que interactúan y la información que intercambia. Las interacciones se representarán mediante diagramas UML o AUML.

El modelo de objetivos y tareas define la especificación de las tareas y la relación de estas con los objetivos. Para especificar las tareas, definiremos qué información produce y/o se consume, así como las repercusiones sobre los objetivos y estado mental del agente.

El modelo de entorno define los recursos del sistema y la integración del sistema con otras aplicaciones.

La metodología de INGENIAS se basa en JADE como plataforma objetivo y al igual que Prometheus, no considera a los agentes móviles.

INGENIAS cuenta con una herramienta muy completa llamada INGENIAS Development Kit (IDK). Esta herramienta permite diseñar las vistas del sistema a través de los modelos definidos por la metodología, cuenta con generación automática de código y soporte para validación y verificación.

2.3.9 Resumen Metodologías.

En este capítulo hemos visto que las metodologías AOSE (Ingeniería del Software Orientada a Agentes) investigadas introducen a nivel conceptual a lo largo de su desarrollo términos muy similares, pero ofrecen enfoques muy diferentes (diferentes procesos de desarrollo, diferentes abstracciones y modelos, diferente cobertura del desarrollo de un proyecto). La mayoría de propuestas intentan cubrir las etapas de análisis y diseño del sistema multiagente, pero la mayoría solamente dan guías y directrices sobre la implementación de los sistemas multiagente. A nivel de herramientas sólo unas pocas ofrecen alguna herramienta que apoya parcialmente la metodología. En la Ilustración 14 presentamos una tabla comparativa de las metodologías estudiadas.

Después de estudiar las metodologías, y atendiendo a la tabla, podemos concluir que las más interesantes son PASSI e INGENIAS. La propuesta de nuestra metodología se basará principalmente en la metodología PASSI, pero dándole un carácter más ágil. Las principales razones para basarnos en esta metodología fueron que es una metodología específica para sistemas multiagente, cubre todas las fases (desde la toma de requisitos hasta el despliegue), soporta agentes móviles, utilizaba un modelo iterativo y está mejor documentada que otras. La principal razón de no escoger INGENIAS fue su falta de soporte a agentes móviles.

METODOLOGÍAS	BASADO EN	Ciclo de desarrollo que incluye implementación	ITERATIVO	NOTACIÓN	Tecnología destino	Herramienta de soportes
Gaia	Organizaciones	✗	✗	Propia	-	✗
TROPOS	Agentes	✗ (sólo directrices)	Se considera incremental	Rationale (Diagramas de actores, justificación y dependencias)	-	✗
PASSI	Agentes	✓	✓	UML	FIPA	✓
PROMETHEUS	Objetivos y planes (BDI)	✗	✗	Propia, pero similar a UML	JACK	✓
MaSE	Agentes	✓	✓	UML+OCL	-	✓
Mas-CommonKADS	Sistemas basados en conocimiento	✓	✓ Espiral, dirigido por riesgos	Propia	OpenDoc, CORBA u OLE2.0	-
Message	Agentes	✗	-	UML	FIPA	-
INGENIAS	Agentes	✓	✓	UML-AUML	JADE	✓

Ilustración 14. Tabla comparativa metodologías estudiadas.

2.4 Plataformas para el Desarrollo de Sistemas Multiagente.

2.4.1 JADE.

Hoy en día una de las plataformas más utilizadas y de referencia es JADE [19] (Java Agent Development Framework). Sigue las especificaciones FIPA y permite desarrollar y ejecutar aplicaciones multiagente. Cuenta con una infraestructura flexible capaz de trabajar con agentes de forma distribuida, proporcionando las funcionalidades básicas e independientes de la aplicación específica que permite desarrollar aplicaciones multiagente de una manera sencilla.

JADE implementa de forma completa la especificación FIPA de gestión de agentes, incluyendo los servicios de AMS, DF y MTS.

Una plataforma de JADE está compuesta de contenedores, en los cuales se encuentran los agentes pudiendo estar distribuidos por la red (Ilustración 16). En la JADE existe un contenedor especial, llamado **Main Container.** Es el primer contenedor que hay que lanzar para crear la plataforma, debiéndose el resto de contenedores registrarse en él. Este contenedor principal se encarga de [19]:

- Gestionar la tabla de contenedores (CT, Container Table), manteniendo un registro de las referencias a objetos y las direcciones de todos los contenedores que conforman la plataforma.
- Gestionar la tabla de descriptores de agentes global (GADT, Global Agent Description Table), manteniendo el registro de todos los agentes de la plataforma, incluyendo su ubicación y estado.
- Mantener el AMS y DF, los cuales proporcionan la gestión de agentes y el servicio de páginas blancas, y el servicio de páginas amarillas, respectivamente.

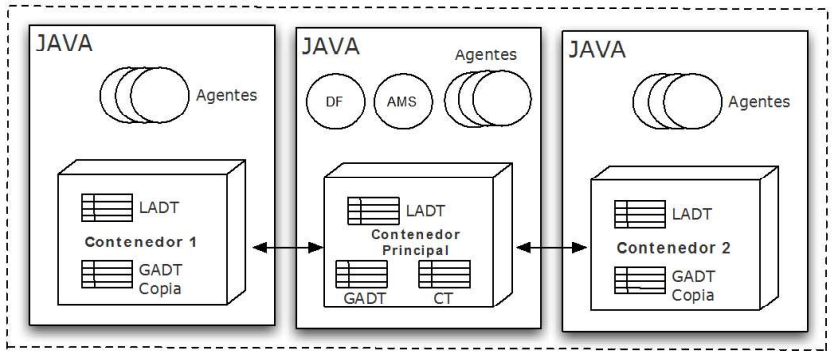


Ilustración 15. Arquitectura de JADE basada en contenedores.

En la práctica la plataforma JADE puede tener una serie de problemas debidos a tener un elemento, el contenedor principal, encargado de gestionar toda la plataforma:

- El rendimiento puede verse afectado al convertirse el contenedor principal en un cuello de botella para la plataforma.
- Puede verse comprometida la robustez del sistema. El contenedor principal contiene el AMS y el DF, dos servicios clave, con lo que un fallo en el podría dejar inoperativa la plataforma. Será necesario gestionar este contenedor de una manera efectiva para asegurar la tolerancia a fallos.

El problema del rendimiento lo solucionaron en versiones nuevas de JADE guardando una copia de la tabla GADT a cada contenedor, donde se hace una gestión local LADT (Local Agent Description Table). De esta forma muchas operaciones se resuelven en local sin hacer uso del contenedor principal. Cuando un contenedor debe localizar a un agente y no tiene la información en local, la solicita al principal y guarda la copia de la información para futuros. Como el comportamiento del sistema es dinámico (los agentes terminan, se mueven, etc.) ocasionalmente el sistema puede fallar, ocasionando una excepción, ante la cual se realiza una nueva copia local desde el contenedor principal.

Para evitar la debilidad del contenedor principal, se necesita utilizar algún mecanismo de tolerancia de fallos el cual asegure la permanencia en el sistema de los agentes AMS y DF. Se han realizado diferentes estudios sobre este problema y la tolerancia a fallos en sistemas multiagente, proponiendo soluciones basadas en manejo de excepciones y redundancia (replicación) [35][36][37].

La replicación consiste es mantener copias de un elemento de forma distribuida y en caso de fallo, una de las copias supliría la función que estaba realizando el elemento que ha fallado. En [37] y [38] se propone un sistema de

replicación dinámico, decidiendo en tiempo de ejecución qué entidades son las más críticas atendiendo a diferentes métricas, planes, dependencias y roles.

Otra de las soluciones propone equipos de agentes intermedios (brokers) En [39] se presenta la arquitectura Adaptive Agent Architecture (AAA) donde se forman equipos jerárquicos de agentes los cuales realizan una redundancia implícita para recuperar el sistema ante fallos en alguno de estos agentes.

El manejo de excepciones consiste en enviar eventos periódicos para comprobar el estado de los agentes. En [40] podemos ver una plataforma para la gestión de fallos en el sistema basada en eventos. FATMAD [41] es una plataforma de desarrollo multiagente, basada en JADE, con tolerancia a fallos a través de puntos de control y recuperación.

La replicación tiene unas ventajas frente al manejo de excepciones: es más rápido, presenta una escalabilidad mayor, y es relativamente más genérico y transparente, al no tener que definir los fallos y sus procesos correctores. Aunque tiene la desventaja de los gastos ocasionados por la réplica, tanto en almacenamiento como en tiempo.

La gran mayoría métodos que acabamos de explicar se centran en conseguir la tolerancia a fallos en los agentes comunes del sistema y no en los agentes de gestión de la plataforma. En [42], se define la arquitectura Virtual Agent Cluster (VAC) que soporta AMS descentralizados en diferentes AP distribuidas. Cada AP tiene una instancia local del AMS y se utiliza un envío periódico de mensajes para comprobar su funcionamiento.

JADE proporciona actualmente herramientas para garantizar que la plataforma permanezca funcional incluso ante errores del contenedor principal [19]. Lo consigue combinando la replicación del contenedor principal y la persistencia del DF. En caso de fallo del AMS, se crea automáticamente un nuevo contenedor principal y se recupera la copia del DF replicado.

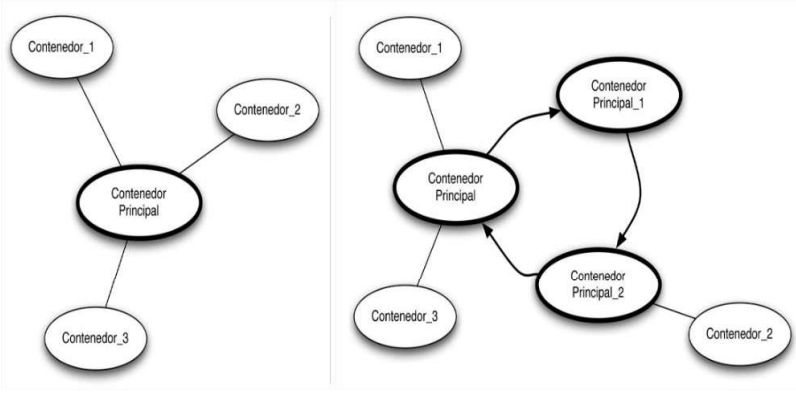


Ilustración 16. Topología de la plataforma JADE sin replicación del contenedor principal y con replicación [20].

En la topología de anillo cada contenedor monitoriza a su vecino, cuando un contenedor principal falla, el contenedor que lo estaba monitorizando informa al resto de contenedores y se reorganizan en un nuevo anillo. El servicio ANS (jade.core.replication.AddressNotificationService) detecta automáticamente detecta cambios en los elementos del anillo y se encarga de actualizar todas las direcciones de los nodos de la plataforma.

Esto asegura que la copia del contenedor principal, pero no mantiene el repositorio del DF. Para esto, y lograr así una máxima fiablidad, JADE ofrece métodos de persistencia del DF.

JADE ofrece dos posibilidades para almacenar el DF: almacenar el catálogo en memoria o en una base de datos relacional. El primero es más rápido, pero presenta limitaciones mayores. Dependiendo de los requerimientos de la aplicación y del número de agentes elegiremos uno u otro.

2.4.1.1 Gestión y monitorización de agentes desde la plataforma JADE.

Se han construido algunas herramientas gráficas de JADE, para facilitar tareas como la depuración de aplicaciones multiagente. Cada herramienta es un agente y obedece las mismas reglas que un agente común.

2.4.1.1.1 Agente RMA.

El agente de monitorización remota (Remote Monitoring Agent) controla el ciclo de vida de una plataforma de agentes y de todos los agentes registrados. Además, como en su nombre lo indica, este habilita el control remoto, donde la interfaz gráfica es utilizada para controlar la ejecución de agentes y su ciclo de vida.

Un agente RMA es un objeto de Java, que se instancia de la clase **jade.tools.rma.rma**, y este puede ser lanzado o ejecutado desde la línea de comandos como cualquier agente ordinario, de la siguiente manera:

java jade.Boot myConsole:jade.tools.rma.rma, o también **java jade.Boot –gui** En la misma plataforma pueden ejecutarse más de un agente RMA, diferenciándolos con el nombre local que tengan, pero en un contenedor solo se puede ejecutar uno.

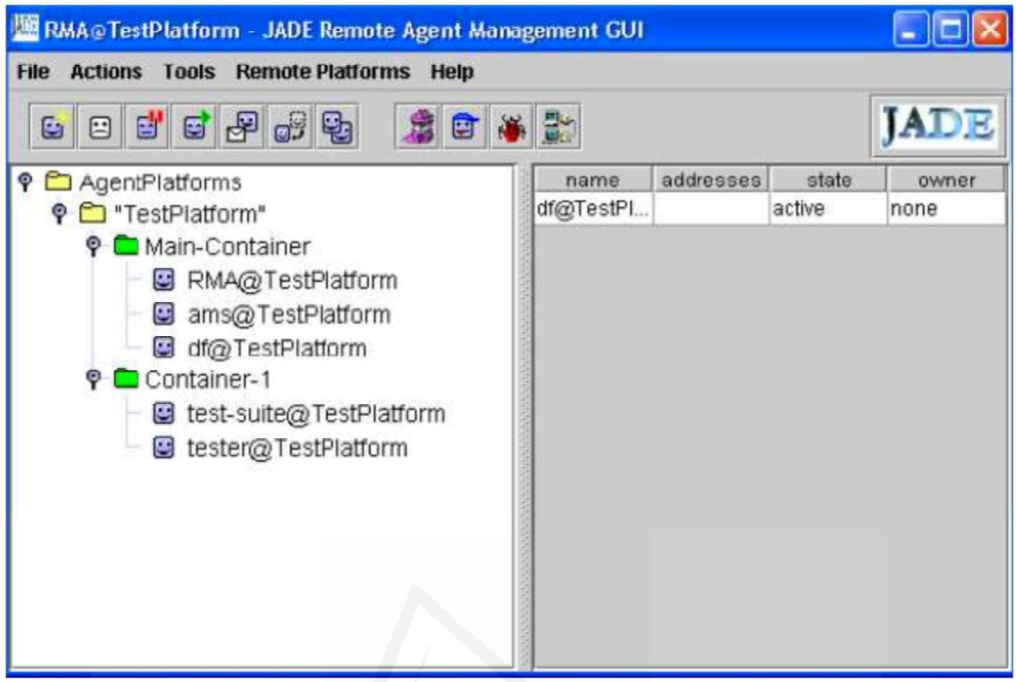


Ilustración 17: Interface gráfica en JADE de un agente RMA (Remote Agent Management).

A continuación se mostraran los comandos que pueden ser ejecutados desde el agente RMA:

- Close RMA Agent: invoca el método doDelete() finalizando el agente RMA. Tiene el mismo efecto que cerrar la ventana.
- Exit this container: finaliza el contenedor donde se está ejecutando el agente RMA y todos los agentes que estén corriendo en él. Si el contenedor es el contenedor principal toda la plataforma será finalizada.
- Shut down Agent Platform: se sale de la plataforma, finalizando todos los contenedores y los agentes.
- Actions menú: son todas las acciones administrativas que se necesitan para usar la plataforma.
- Start New Agent: crea un nuevo agente. Al usuario se le sugiere un nombre y la clase Java de la que será instanciado el nuevo agente. Se puede seleccionar el contenedor donde se quiere que sea lanzado.
- Kill Selected Items: esta acción elimina todos los agentes y los contenedores seleccionados y los borra del registro de la plataforma.

- Suspend Selected Agents: esta acción deja inactivo a cualquier agente que este seleccionado, equivale a llamar el método doSuspend().
- Resume Selected Agents: es equivalente a llamar el método doActivate(). Cuando un agente está suspendido este lo coloca en el estado AP_ACTIVE.
- Send Custom Message to Selected Agents: Al seleccionar esta acción se despliega un cuadro de dialogo donde se puede enviar un mensaje ACL a un agente.
- Migrate Agent: migra agentes de un contenedor a otro, pero no todos los agentes pueden migrar, esto depende de su implementación.
- Clone Agent: clona el agente que este seleccionado, se le debe indicar el nuevo nombre y el contenedor donde el agente se va a ejecutar.

Además desde el RMA se pueden gestionar plataformas remotas. Para ello cuenta con los siguientes comandos:

- RemotePlatforms: Habilita el control de algunas plataformas remotas que sigan las especificaciones FIPA. Estas plataformas no necesariamente tienen que ser de Jade.
- Add Remote Platform vía AMS AID: esta acción obtiene la descripción de una plataforma de agentes remota a través de AMS. Se requiere insertar la información del AID del AMS remoto, para que la plataforma remota se pueda agregar al árbol que se muestra en la interfaz del RMA.
- Add Remote Platform vía URL: agrega la descripción de una plataforma remota vía URL. Se debe insertar la URL para que la plataforma remota se muestre en el árbol de la interfaz gráfica del agente RMA.
- View APDescription: es para ver la descripción de una plataforma seleccionada.
- Refresh APDescription: esta acción llama a través del AMS remoto la descripción de una plataforma y hace una recarga de la misma.
- Remove Remote Platform: elimina la plataforma remota que esta seleccionada en la interfaz gráfica.
- Refresh Agent List: hace una búsqueda en la plataforma remota de todos los agentes que estén contenidos en ella, mostrándolos en forma de árbol en la interfaz del agente RMA.

2.4.1.1.2 Agente Dummy.

El agente Dummy es una herramienta que permite a los usuarios componer y enviar mensajes ACL, manteniendo una lista de todos los mensajes ACL

enviados y recibidos. Cada mensaje de la lista puede ser visto o modificado. Este agente puede ser ejecutado cuantas veces sea necesario. Existen dos maneras de ejecutar este agente, a través de la línea de comandos, o a través del menú de herramientas del agente RMA. El comando es el siguiente:

Java jade.Boot theDummy:jade.tools.DummyAgent.DummyAgent.

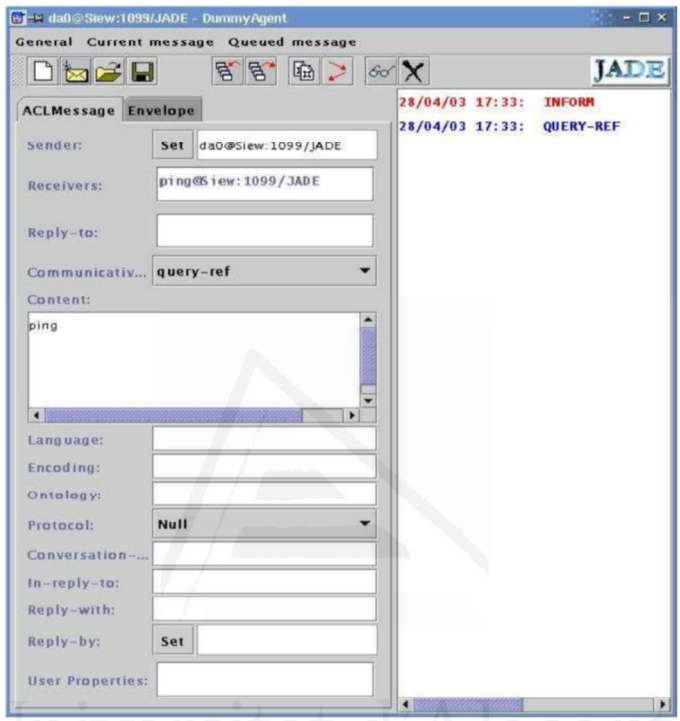


Ilustración 18: Interface gráfica en JADE de un agente Dummy.

2.4.1.1.3 Interface gráfica del agente DF (Directory Facilitator).

Este agente se ejecuta desde el menú de herramientas en la interfaz del RMA, la interfaz gráfica del agente DF solo puede ser mostrada en el host donde está corriendo la plataforma (contenedor principal).

La interfaz gráfica del agente DF se usa para interactuar con el DF, es decir, se puede ver las descripciones de todos los agentes registrados, así mismo, registrar y quitar registros de agentes, modificar las descripciones de los agentes y buscar algún agente a través de su descripción.

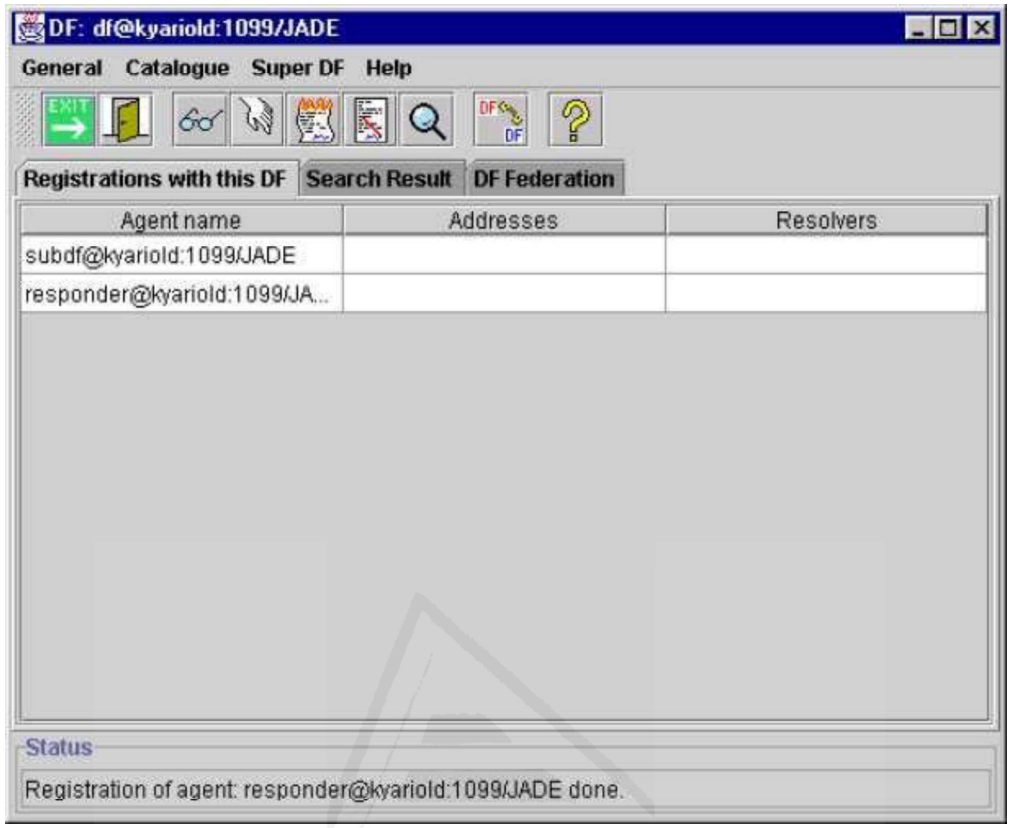


Ilustración 19: Interface gráfica en JADE del agente DF.

2.4.1.1.4 Agente Snnifer.

Existen dos maneras de ejecutar este agente, a través de la línea de comandos, o a través del menú de herramientas del agente RMA. El comando es el siguiente: **java jade.Boot sniffer:jade.tools.sniffer.Sniffer**

Este agente es el que se encarga de interceptar los mensajes ACL lanzados y los visualiza gráficamente en un modo similar a los diagramas de secuencia de UML. Resulta útil para depurar conversaciones entre agentes. El usuario puede ver todos los mensajes de un agente o un grupo de agentes y guardarlos en el disco.

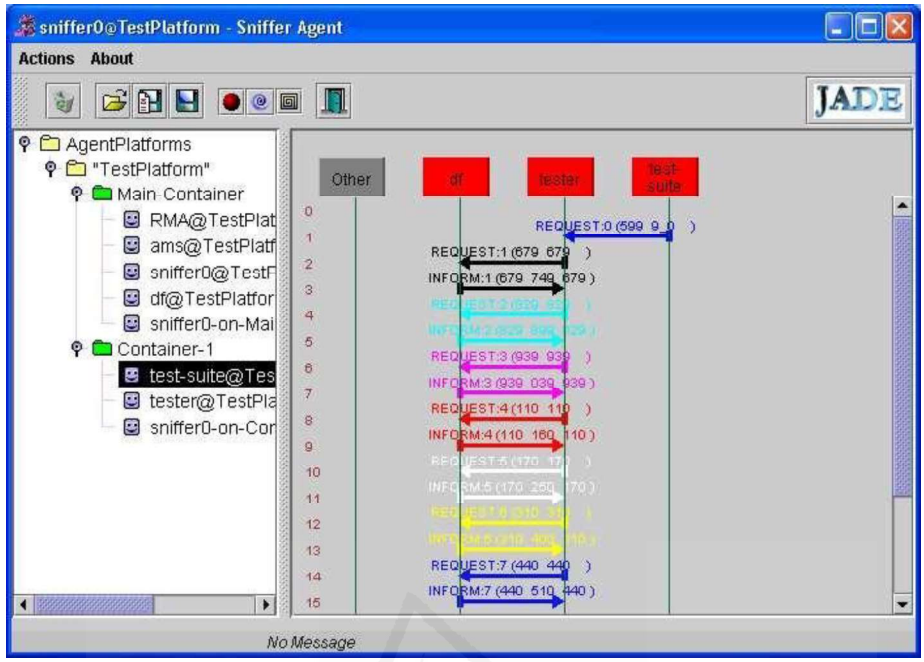


Ilustración 20: Interface gráfica JADE de un agente Sniffer.

El agente Sniffer al iniciar se inscribe con la plataforma, entonces, este informa cada vez que un agente nace o muere, de igual manera cuando un contenedor es creado o es eliminado.

2.4.1.1.5 Agente Instrospector.

Este agente permite monitorizar el ciclo de vida de los agentes y los mensajes ACL que intercambian. Los agentes son pasados como argumentos a través de la línea de comandos, o a través de un archivo de configuración.

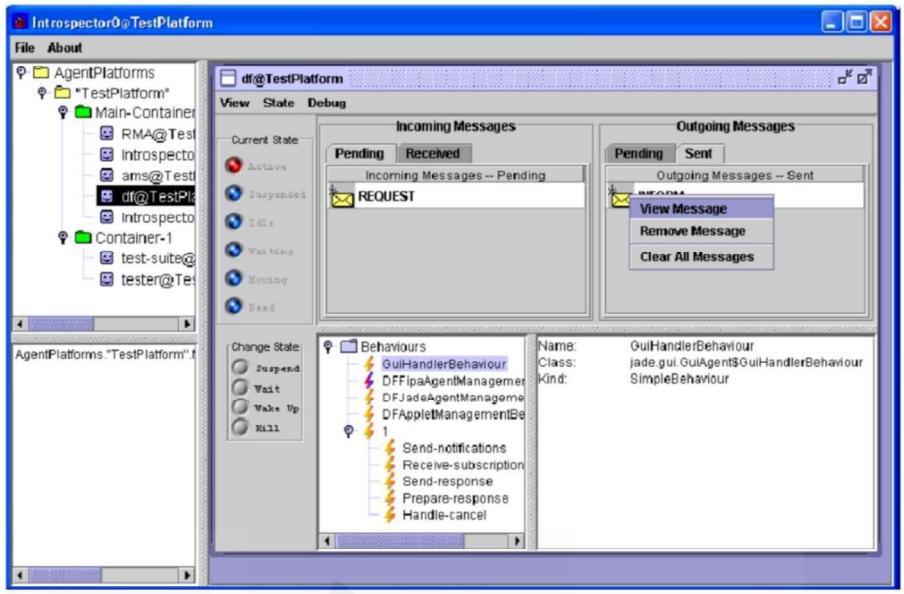


Ilustración 21: Interface gráfica en JADE de agente Introspector.

2.4.2 ZEUS Agent Building Toolkit.

En este capítulo vamos a describir ZEUS [43], una tool-kit avanzado para la construcción de sistemas multiagente. Esta herramienta facilita la construcción de aplicaciones con agentes colaborativos, deliberativos e incluso sistemas que trabajan en entornos de tiempo real [3].

Las razones para utilizar ZEUS son dos: por un lado, existe una demanda creciente de investigadores que necesitan metodologías y herramientas para desarrollar sistemas multiagente y sistemas distribuidos, y las arquitecturas, frameworks y herramientas actuales no se adecuan a estas necesidades. Por otro lado, ZEUS facilita el proceso de ingeniería de estas aplicaciones, reduciendo los tiempos de desarrollo, y aumentando la reutilización y la estandarización de los sistemas multiagente.

2.4.2.1 Filosofía y Especificación en ZEUS.

ZEUS provee un tool-kit genérico, personalizable y escalable para desarrollar sistemas multiagente. Debido a ello cuenta con las siguientes características:

- En primer lugar, debe de definir claramente los niveles de dominio del problema y funcionalidades de los agentes (comunicación, cooperación, coordinación, ejecución de tareas, monitorización, etc.). Por lo tanto, el kit de herramientas debe proporcionar a los desarrolladores de agentes

toda la funcionalidad a nivel de agente; de modo que solo necesitan proporcionar el código que implementa las habilidades de resolución de problemas específicas del dominio de los agentes que definen.

- En segundo lugar, el kit de herramientas debe de facilitar la programación, permitiendo una programación visual y la metáfora de "escoger y elegir". Es decir, los desarrolladores de aplicaciones deberían poder seleccionar de diversos menús, la funcionalidad y las modalidades requeridas por sus agentes.
- En tercer lugar, el conjunto de herramientas debe utilizar tecnología "estandarizada" siempre que sea posible, como lo demuestra el empleo de KQML [44] como nuestro lenguaje de comunicación de agente.

Por último, en cuarto lugar, debe ser compatible con un diseño abierto para garantizar que sea fácilmente extensible.

El entorno de trabajo ZEUS proporciona al desarrollador de sistemas multiagente lo siguiente:

- La definición de diferentes agentes con diferentes funcionalidades y comportamientos.
- La organización de los agentes con relaciones de subordinación, cooperación, supervisión, jerarquía y control.
- Definir en cada agente los mecanismos de comunicación y coordinación a. ZEUS proporciona varios protocolos de coordinación diferentes entre los cuales el diseñador puede elegir, incluyendo maestro-esclavo, contrato neto, una variedad de estrategias de subasta, negociación por tiempo (donde el agente solicita más tiempo para algún objetivo), negociación a precio reducido (donde el agente ofrece reducir el coste que está cobrando a cambio de más tiempo para algún objetivo), etc. También permite la definición de nuevos protocolos proporcionados por el usuario siempre, pero siempre dentro de protocolo de coordinación universal ofrecido por ZEUS.
- La posibilidad de definir el código necesario de cada agente para la resolución de los problemas específicos del dominio necesario; y la generación automática de los ejecutables necesarios para los agentes.

La herramienta ZEUS facilita agentes predefinidos, como los agentes del servidor de nombres (páginas blancas), los agentes facilitadores (páginas amarillas), así como un visualizador. Un objetivo clave del visualizador ese el análisis y la depuración de los sistemas de agentes distribuidos, ya que es bien sabido que este es un problema notoriamente complejo. En [45] se puede ver una descripción de este visualizador.

2.4.2.2 Agentes ZEUS.

Cada agente en ZEUS consiste en una capa de definición, una capa de organización y otra capa de coordinación. La capa de definición comprende las habilidades de aprendizaje y razonamiento, las metas, los recursos, creencias, preferencias, etc. La capa de organización describe las relaciones del agente con otros agentes, por ejemplo, a qué agencias pertenece, qué habilidades sabe que poseen otros agentes, etc. En la capa de coordinación, el agente se modela como una entidad social, es decir, en términos de las técnicas de coordinación y negociación que posee. Construidos sobre la capa de coordinación están los protocolos de comunicación que implementan la comunicación entre agentes; mientras que debajo de la capa de definición está la interfaz del programador de la aplicación (API) que vincula al agente con las realizaciones físicas de sus recursos y habilidades.

El kit de herramientas ZEUS consta de una serie de editores (visuales) que el desarrollador utiliza para especificar la información requerida para definir un agente ZEUS. El conjunto actual de editores incluye lo siguiente:

- • Un editor de ontologías.
- • Un editor de descripción de tareas.
- • Un editor de organización.
- • Un editor de definición de agente.
- • Un editor de coordinación.
- • Un editor de hechos / variables.
- • Un editor de restricciones.
- • Un generador de código.

Todos los editores anteriores facilitan esencialmente la identificación y descripción de un conjunto de agentes, seleccionando la funcionalidad del agente e ingresando tareas y datos relacionados con el dominio. Por lo tanto, la salida del kit de herramientas ZEUS es una descripción lógica de un conjunto de agentes y un conjunto de tareas que se llevarán a cabo en un dominio, junto con el código ejecutable para cada agente y cada tarea.

Para generar ejecutables, ZEUS también viene con las siguientes bibliotecas predefinidas:

- Una biblioteca de gráficos de coordinación que describe algoritmos / protocolos de coordinación.

- Una biblioteca de datos de relaciones que define las relaciones organizacionales definidas por ZEUS.
- Una API para operaciones como la ejecución de tareas y la evaluación de funciones de costes.
- Una biblioteca de shell de agente que proporciona la plantilla de agente, incluidas las comunicaciones KQML, algoritmos de planificación y programación, biblioteca de buzones, etc.
- Un conjunto de agentes de utilidad estándar como servidores de nombres de agentes (páginas blancas), facilitadores (páginas amarillas) y un visualizador.

Para generar el código para un sistema de aplicación específico, el generador de código hereda de las bibliotecas e integra los datos de los distintos editores.

Todo el kit de herramientas ZEUS se ha implementado en el lenguaje de programación Java y se ejecuta en plataformas Linux y Windows.

2.4.2.3 Visualización y Depuración en ZEUS.

La visualización es una característica importante en ZEUS y es altamente demandada por los profesionales en el análisis, la monitorización y la depuración de los sistemas multiagente. En general, visualizar el comportamiento general del sistema en dichos sistemas con control distribuido, datos y procesos es una tarea difícil. Cada agente en el sistema sólo tiene una visión local de la organización, y la carga de integrar en un conjunto coherente las grandes cantidades de información de alcance limitado proporcionadas por agentes individuales recae en el usuario. Además, debido a la complejidad de la interacción y el comportamiento de múltiples agentes, la visualización efectiva adquiere más importancia que en los sistemas de agente único. Es necesario para reducir la sobrecarga de información en los usuarios, y así permitirles confirmar, comprender, controlar y/o analizar el comportamiento del sistema, así como depurar el sistema.

El problema de depuración en los sistemas de multiagente no se puede infraestimar ya que incluso cuando los agentes individuales de una organización son "correctos", el comportamiento general del sistema puede ser diferente al deseado.

El sistema de visualización de ZEUS comprende un conjunto de herramientas, y cada herramienta proporciona una vista diferente de la aplicación multiagente. Cada herramienta interroga a los agentes en la aplicación, recopila la información devuelta y presenta esta información a un

usuario de manera adecuada. En esencia, esto desplaza la carga de inferencia del usuario al visualizador. El conjunto de herramientas actual incluye:

- Una herramienta "social" que muestra el intercambio de mensajes entre agentes en una sociedad.
- Una herramienta de informes que representa gráficamente la descomposición de tareas en toda la sociedad y los estados de ejecución de varias subtareas,
- Una micro herramienta para monitorizar los estados internos de los agentes.
- Una herramienta de control para modificar de forma remota los estados internos de agentes individuales.
- Una herramienta estadística para cotejar agentes individuales y estadísticas de toda la sociedad.
- Una herramienta de video (dentro de la herramienta de sociedad, herramientas de informes y estadísticas) que se puede usar, estilo video, para grabar, reproducir, rebobinar, avanzar, rebobinar y avanzar rápidamente desde una base de datos.

La Ilustración 22 muestra los agentes desde la vista de la herramienta de la sociedad. La información mostrada fue enviada por todos los diferentes agentes y recopilada y presentada por el visualizador para formar esta imagen global de la sociedad.

La Ilustración 23 muestra la herramienta informe, cuya función es proporcionar una visión global de la resolución de problemas en una sociedad de agentes, siendo una herramienta muy útil de depuración y análisis. Permite a un usuario seleccionar un conjunto de agentes y solicita que le informen del estado de todas sus tareas y objetivos. Como cada agente solo tiene una visión local e incompleta, esta herramienta solicita y recopila las vistas locales para proporcionar una imagen más completa generando un gráfico GANTT que muestra la descomposición de la tarea, la asignación de sus subpartes constituyentes a diferentes agentes en la comunidad y cuándo cada agente está programado para realizar su parte de la tarea.

Estas herramientas son invaluables en el análisis y depuración de aplicaciones, ya que juntas fomentan la depuración mediante corroboración. Con esto queremos decir que si dos o más herramientas diferentes que miran a la sociedad desde diferentes puntos de vista apuntan a algún componente (por ejemplo, algún agente) como 'buggy', el usuario está más convencido de que el error emana de ese componente en particular.

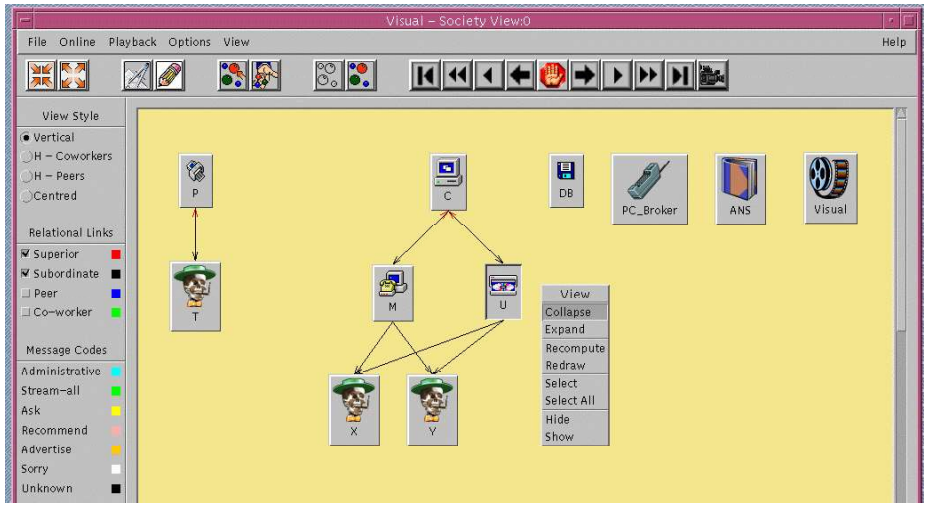


Ilustración 22: Herramienta para la visualización "social" de agentes en ZEUS [43].

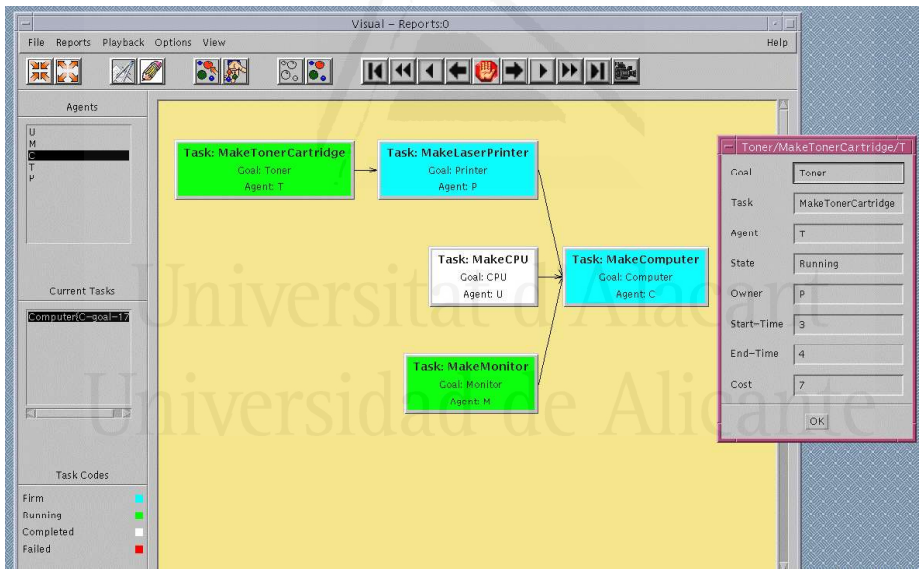


Ilustración 23: Un gráfico GANTT de una tarea que resuelven los agentes colaborando. Observe cómo algunas tareas están en estado de ejecución, algunas se han completado y otras están esperando. El cuadro de diálogo a la derecha muestra detalles de la tarea MakeTonerCartridge [43].

Un objetivo clave de ZEUS era desarrollar una metodología para diseñar aplicaciones de agentes de colaborativos en entornos industriales, y construir un conjunto de herramientas basado en la metodología. Una de las aplicaciones que se desarrollaron con ZEUS fue una aplicación de gestión de viajes que involucra a 17 agentes, en la que un agente de gestión de viajes negocia con los agentes de

reservas de hoteles, aerolíneas, automóviles, trenes y taxis para generar un itinerario para un viaje transatlántico para un usuario. Una vez que se llega a un acuerdo, el administrador de viajes genera el itinerario del viaje, envía correos electrónicos a su usuario y llama por teléfono al usuario para notificarle que el itinerario se ha generado. Este fue un proyecto de 3 semanas de trabajo y fue el primer sistema que se construyó con ZEUS con el fin de demostrar sus bondades. El tiempo del proyecto contrasta con el de un sistema anterior, MII [46] fue un sistema de mucha menos complejidad, que tardó 4 años en completarse. Esta comparación no es del todo justa, ya que los 4 años de trabajo incluyeron el tiempo de diseño, los tiempos necesarios para investigar el prototipo, decidir los entornos de implementación, implementar un prototipo preliminar inicial, etc. Sin embargo, es suficiente para aclarar que estamos haciendo muy ganancias de tiempo significativas en la construcción de aplicaciones usando ZEUS, que desde cero. La Ilustración 24 muestra la aplicación de asistente personal de viaje (PTA) como se muestra en una de las herramientas de visualización de ZEUS.

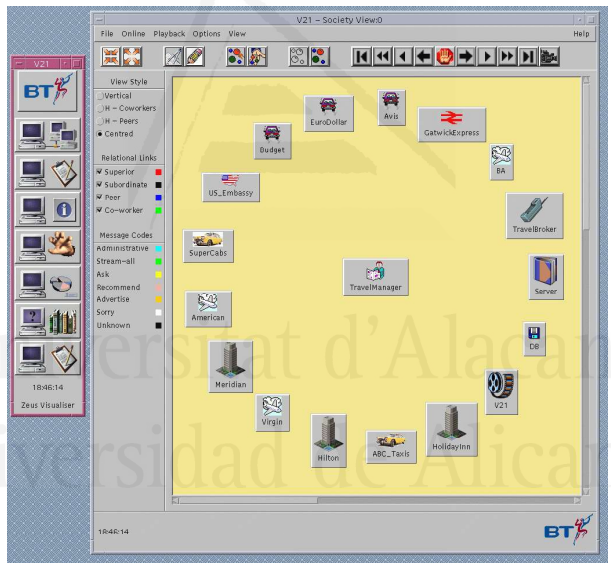


Ilustración 24: Vista social de la aplicación de asistencia personal para viajes [43].

Como resumen podemos decir que ZEUS es un completo conjunto de herramientas de construcción de sistemas multiagente, genérico, personalizable y extensible que facilita y acelera el desarrollo de aplicaciones multiagente complejas generando automáticamente código. Además, permite la visualización y la depuración.

2.5 Arquitecturas Robóticas.

Muchos de los sistemas robóticos han sido desarrollados sobre los paradigmas reactivo y deliberativo, pero la mayoría de sistemas actuales utilizan el paradigma híbrido [47][48][49]. La denominación "híbrido" en este caso, viene dada por incorporar una capa deliberativa sobre una capa reactiva. La capa reactiva es la que proporciona la supervivencia básica del robot en el entorno. La capa deliberativa es la que le permite al robot la realización de tareas más complejas [47]. Gracias a esta organización se puede actuar con rapidez a nivel reactivo y se pueden realizar tareas complejas usando el nivel deliberativo.

Otra cosa muy habitual en las arquitecturas robóticas es el diseño basado en capas. El número de capas puede diferir de una arquitectura a otra, así como los mecanismos de comunicación y coordinación entre ellas. La mayoría utilizan tres capas [48][49][50]: una capa deliberativa, una reactiva y una capa intermedia de control. El nivel superior lo ocupa la capa deliberativa, llevando a cabo tareas de localización, planificación, y razonamiento a nivel global con el fin de conseguir objetivos. La capa reactiva es la inferior, y recibiendo información de los sensores y proporcionando respuestas rápidas, a través de comportamientos. La capa intermedia hace de enlace entre las otras dos, facilitando su comunicación y coordinación.

A lo largo de los años se ha creado una gran variedad de arquitecturas dentro del paradigma híbrido, pero es difícil hacer una clasificación debido a su variedad. En [47] [50] [51] se realiza una clasificación de éstas en: arquitecturas híbridas organizativas, basadas en jerarquías de estados y orientadas a modelos. A continuación, vamos a describir cada una de estas categorías, incluyendo otra centrada en las arquitecturas basadas en sistemas multiagente.

2.5.1 Arquitecturas Híbridas Organizativas.

Las arquitecturas híbridas organizativas están basadas en la gestión empresarial en concreto en su forma de descomponer las responsabilidades. En estas arquitecturas se distingue claramente la capa reactiva de la deliberativa. La capa reactiva es la que contiene los comportamientos básicos mientras que la deliberativa contiene un conocimiento global del mundo. Es en esta capa superior, donde encontramos los módulos que se encargan de la planificación de alto nivel. En la capa inferior está el nivel reactivo, llevando a cabo las tareas básicas.

Dentro de esta categoría de arquitecturas, destacan la arquitectura AuRA [52] y SFX [53], Yavuz [54] y Tripodal [55].

2.5.1.1 Arquitectura AuRA.

La arquitectura AuRA (Autonomous Robot Architecture) nació en 1987 y está basada en la teoría de esquemas. Está compuesta de cinco subsistemas divididos en dos capas claramente diferenciadas (Ilustración 25). En la capa superior, la deliberativa, encontramos los subsistemas de planificación y cartografía. El primero se encarga de la planificación de misiones y las tareas, mientras que el segundo, el cartográfico, encapsula necesarias para la navegación: lectura de sensores y mapeado de entorno. En la capa inferior, la reactiva, encontramos el subsistema de percepción y el subsistema de actuación (motores).

Entre las dos capas principales se encuentra el subsistema homeostático. El papel de este subsistema es muy importante ya que lleva a cabo las tareas de supervivencia del robot modificando su comportamiento.

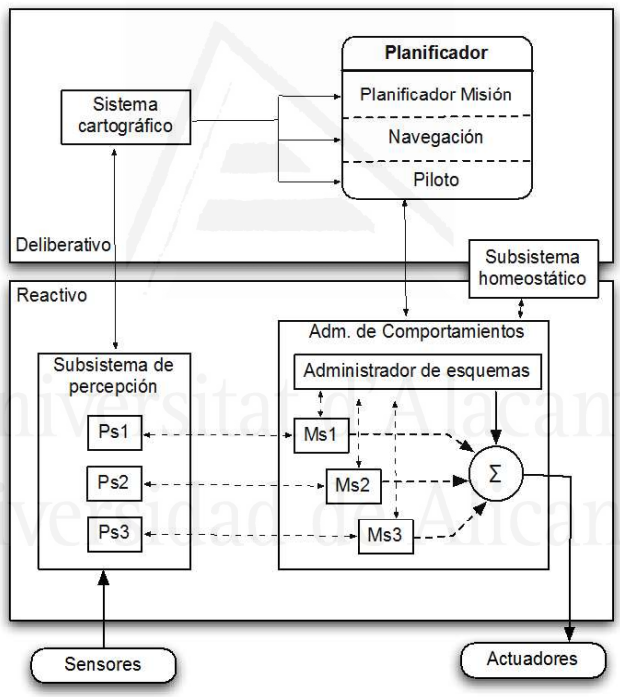


Ilustración 25: Arquitectura híbrida organizativa AuRA.

Esta arquitectura aporta modularidad y flexibilidad gracias a estar compuesta por diferentes módulos se pueden reemplazar. Por el contrario, es una arquitectura poco versátil, diseñada específicamente para navegación de robots móviles y, poco robusta, por la importancia del subsistema de control homeostático.

2.5.1.2 Arquitectura SFX.

La arquitectura SFX (Sensor Fusion Effects) apareció en 1992 como una extensión de la arquitectura AuRA a la cual se le añadió fusión sensorial y gestión de errores [53]. Sigue contando con los dos niveles, el deliberativo y el reactivo, pero subdivide cada uno de estos niveles en módulos. El nivel reactivo consta de dos módulos: comportamientos estratégicos y comportamientos tácticos. SFX utiliza un método de inhibición de los comportamientos tácticos sobre los estratégicos. En ausencia de contingencias se realizarían los comportamientos estratégicos, encargados de desarrollar objetivos de un nivel superior, pero para situaciones especiales priman determinados comportamientos tácticos, como podría ser la evitación de choque con un obstáculo. El componente deliberativo está dividido en módulos que son implementados como agentes, los cuales interactúan entre ellos. Existe un agente supervisor, que hace de interfaz con el usuario y controla la evolución del sistema, llamado "planificador de la misión". A nivel interno, este agente se compone de tres subsistemas administradores: de sensores, de tareas y de actuadores.

Esta arquitectura apareció con la idea de lograr robustez, para ello incluyó mecanismos de tolerancia a fallos, pero al basarse en módulos centrales, como el planificador y estructuras tipo pizarra, la robustez vuelve a estar comprometida.

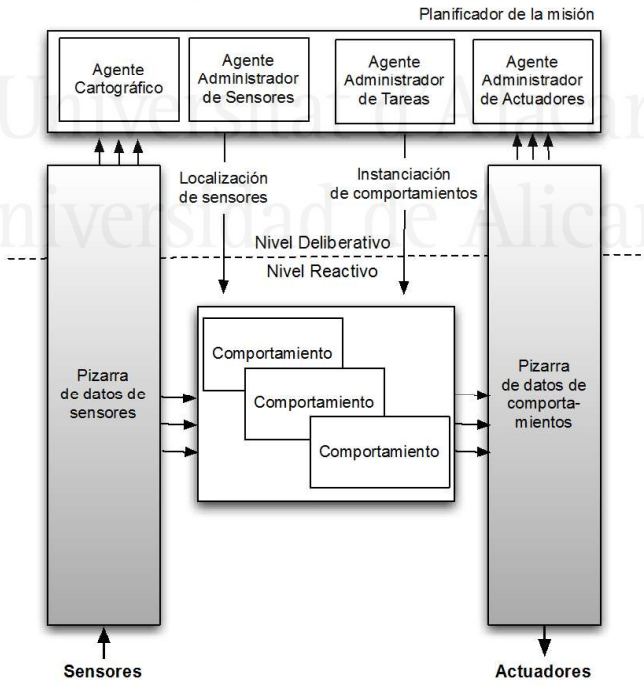


Ilustración 26: Arquitectura híbrida organizativa SFX.

2.5.1.3 Arquitectura Yavuz

En [54] se presenta la arquitectura modular jerárquica Yavuz. Esta arquitectura tiene dos niveles, deliberativo y reactivo, que a su vez se dividen en varias capas. La capa deliberativa tiene varios modos de funcionamiento determinados por los módulos de generación de comando difuso y de arbitraje, permitiendo al robot tener tres modos de funcionamiento: manual, donde el robot es pilotado por un operador; aprendizaje, donde el operador indica al robot como resolver la tarea y el objetivo; y reproducción, donde el robot reproduce la tarea aprendida de manera autónoma. Es una arquitectura adecuada en sistemas donde los robots realizan tareas previamente establecidas.

A pesar de nacer como una arquitectura modular, flexible y escalable, el tener un módulo central imprescindible, dificulta estas pretensiones.

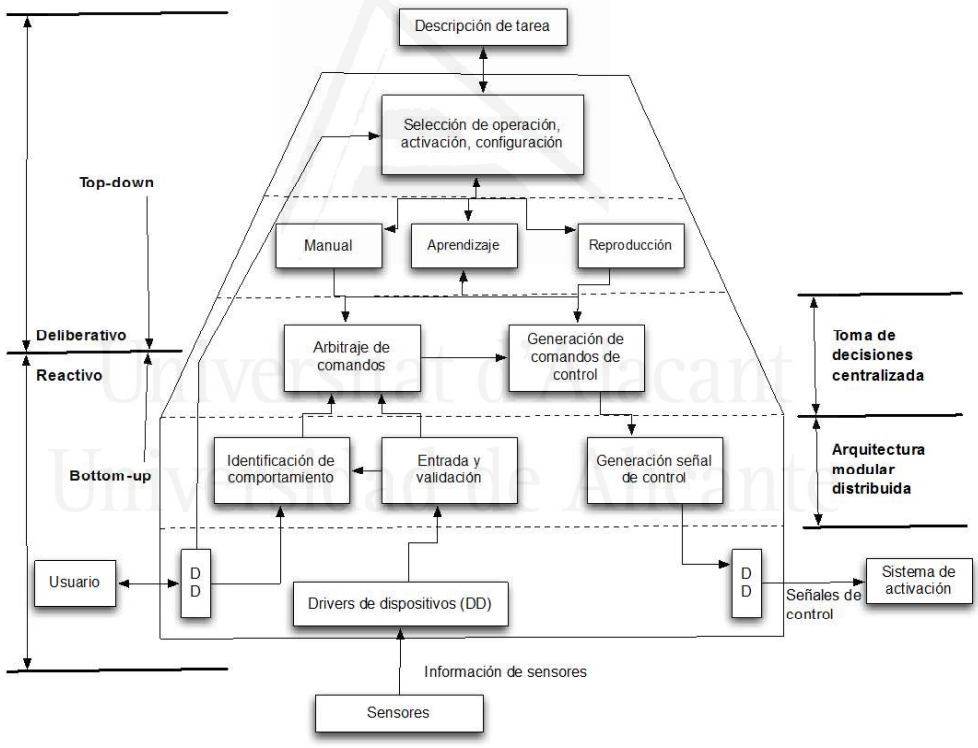


Ilustración 27: Arquitectura híbrida organizativa Yavuz.

2.5.1.4 Arquitectura Tripodal.

La arquitectura Tripodal [55] (Tripodal Schematic Control Architecture) se basa en tres capas: una capa deliberativa, otra reactiva y una última capa intermedia de secuenciación. La capa deliberativa es la encargada de la planificación central y de interactuar con los usuarios. El usuario a través del módulo HRI encarga la tarea a realizar. El planificador descompone esta tarea en una secuencia de procesos y los envía al supervisor de procesos, el cual se encuentra en la capa de secuenciación. La capa de secuenciación está compuesta por un módulo supervisor que ejecuta los procesos a dirigiendo los componentes de la capa reactiva, y los módulos de navegación y manipulación, los cuales extraen información avanzada a partir de los datos obtenidos por los sensores. La capa reactiva es la que contiene los componentes que trabajan en tiempo real interactuando directamente con el hardware. En la Ilustración 28: Arquitectura híbrida organizativa Tripodal. podemos ver la disposición de los diferentes módulos que forman la arquitectura Tripodal, así como el flujo de información entre ellos.

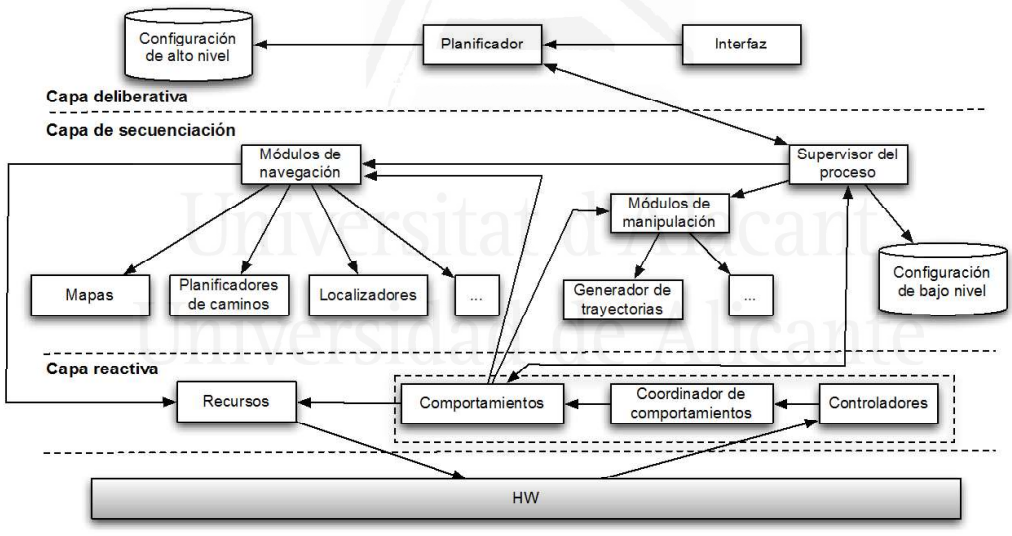


Ilustración 28: Arquitectura híbrida organizativa Tripodal.

2.5.2 Arquitecturas Híbridas basadas en Jerarquía de Estados.

Las arquitecturas basadas en jerarquías de estados organizan las actividades en base al conocimiento temporal. Tienen tres capas, basadas en el estado pasado, presente y futuro de conocimiento. La capa reactiva utiliza únicamente conocimiento del presente, mientras que la capa deliberativa trabaja con el conocimiento pasado y futuro (suposiciones y predicciones). La arquitectura 3T [56] usada por la NASA es el mejor ejemplo de esta arquitectura., aunque existen otras muy utilizadas como BERRA [57] y SSS [58].

2.5.2.1 Arquitectura 3T.

La arquitectura 3T (3 Tiered) nació con el objetivo de resolver tareas de una forma robusta. Es una arquitectura basada en tres niveles, el reactivo, el deliberativo y el intermedio que sirve como interface de los dos anteriores. En la capa superior, capa deliberativa, se encuentra el planificador, el cual se encarga de proporcionar la perspectiva global del sistema, establece los objetivos y planifica las estrategias, manejando información del pasado, presente y futuro. Esta información se pasa a la capa intermedia, que actúa de secuenciador transformando las tareas recibidas del nivel superior en habilidades a desarrollar por el nivel inferior. Esta capa sólo trabaja con información del pasado y del presente. En el nivel inferior, en la capa reactiva, sólo se trabaja con información del presente, encontramos las habilidades en forma de comportamientos. En la Ilustración 29 podemos ver un esquema de esta arquitectura.

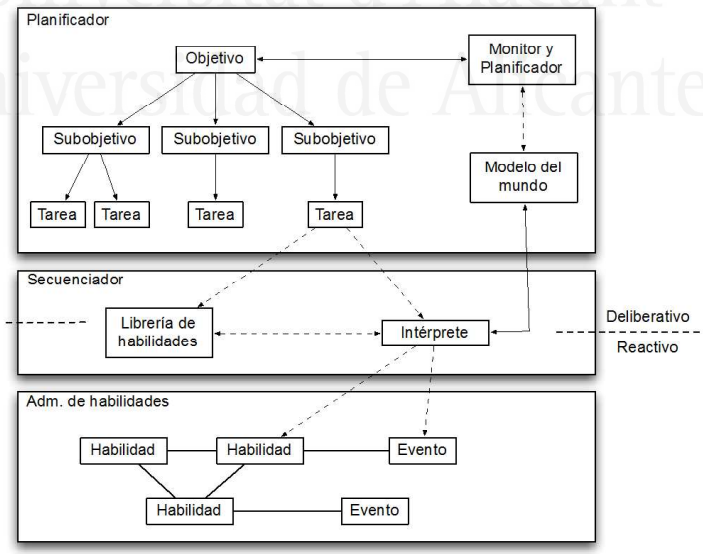


Ilustración 29: Arquitectura híbrida basada en jerarquía de estados 3T.

En la arquitectura 3T los tres niveles definidos operan de manera concurrente y asíncrona, de forma que los algoritmos lentos se alojan en el planificador y los rápidos en el administrador de habilidades. Esta división puede implicar clasificaciones no deseadas. Un ejemplo de esto, lo tendríamos en los sistemas de visión, que por funcionalidad corresponderían al bajo nivel (administrador de habilidades), pero que por lentitud se incorporan a alto nivel, en el planificador.

2.5.2.2 Arquitectura BERRA.

La arquitectura BERRA (BEhavior-based Robot Research Architecture, 2000) [57] fue diseñada para que un robot móvil fuese capaz de llevar a cabo las tareas ordinarias de una oficina. Consta de tres capas: deliberativa, ejecución de tareas y reactiva. El usuario da las órdenes a través de la capa deliberativa. Esta interpreta las órdenes y realiza una planificación tanto del camino como de la tarea encomendada. Por ello, debe de poseer una interfaz con el usuario y un planificador. El planificador convertirá las órdenes en una lista de estados consecutivos, donde cada uno de ellos representa una configuración de la capa reactiva. Estas configuraciones serán enviadas a la capa reactiva, la cual las ejecutará y confirmará su realización de manera secuencial, en orden y de una en una. En el caso de recibir un error de ejecución la capa deliberativa revisará el plan y si no es posible completar la tarea avisará al usuario. La capa intermedia actúa de nexo entre las dos capas anteriores, recibiendo los estados de la capa superior y traduciéndolos a las configuraciones adecuadas para la capa reactiva, y monitorizando la capa reactiva en base a las directrices marcadas por la capa deliberativa. En esta capa también se encuentra el localizador, el cual se encarga del seguimiento de la posición del robot. Por último, la capa reactiva, que se compone de diferentes comportamientos, recibe la información sensorial del robot, envía a cada comportamiento la información que necesita. Cada comportamiento define una tarea sencilla y enviará al módulo controlador las acciones del robot.

Una de las principales ventajas de esta arquitectura es que es totalmente distribuida, ejecutándose cada componente como un proceso individual, permitiendo la distribución de la arquitectura en múltiples máquinas. Como parte negativa, el módulo planificador necesita conocer la topología del entorno y hay una dependencia fuerte del módulo central que gestiona todos los procesos.

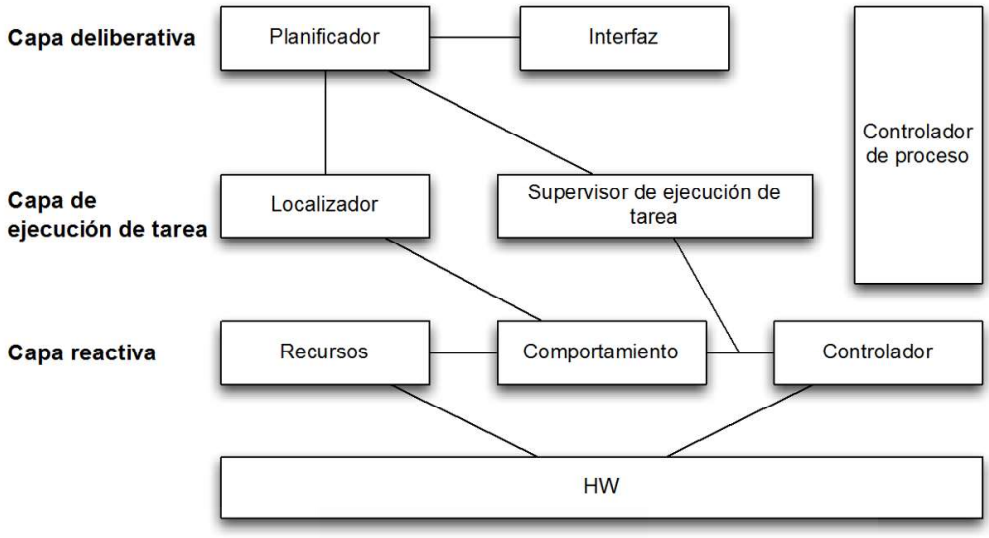


Ilustración 30: Arquitectura híbrida basada en jerarquía de estados BERRA.

2.5.3 Arquitecturas Híbridas Orientadas a Modelos.

Las arquitecturas orientadas a modelos utilizan un modelo global del mundo como percepción, llamado sensor virtual. Plantean la deliberación como todo aquello que esté relacionado con un comportamiento condicionado por una meta o un objetivo, mientras que definen la reactividad como las unidades de control que operan en el presente. Parece que se vuelve atrás, hacía las arquitecturas jerárquicas, pero entre ambas existen las siguientes diferencias [51]:

El modelo se centra en determinadas regiones, regiones de interés, estando mejor organizado.

Las percepciones lentas son recibidas de manera distribuida y asíncrona. Los errores sensoriales y la incertidumbre son filtrados mediante mecanismos de fusión sensorial. Saphira [59] y TCA [60] son dos de las arquitecturas más conocidas de este tipo.

2.5.3.1 Arquitectura Saphira.

La principal finalidad de la arquitectura Saphira es la construcción de agentes móviles autónomos con capacidad para acatar, aprender y ejecutar tareas con robustez [59]. Es una arquitectura de dos capas: una capa inferior reactiva y una capa superior deliberativa. En la Ilustración 31, podemos ver un

esquema que muestra esta arquitectura. La arquitectura está construida en torno al espacio de percepción local LPS (Local Perceptual Space), el cual es el componente central de representación interna. La parte de percepción que se encarga de añadir y extraer la información sensorial al LPS con el fin de reconocer objetos y navegar principalmente. Tiene otra parte, la efectora, que es donde se ejecutan los comportamientos. El control de Saphira se basa en comportamientos, los cuales se definen y coordinan utilizando lógica difusa [61], de forma que la salida de los comportamientos son reglas difusas que se combinarán generar los comandos de movimiento del robot (velocidad y dirección). El controlador PRS-Lite (Procedural Reasoning System-lite), es el sistema de representación y deliberación (BDI), encargado de la selección y coordinación de comportamientos. Este controlador se caracteriza por tener capacidad de integración de actividades dirigidas por objetivos o por eventos y, por una descomposición jerárquica de tareas. Saphira tiene la gran ventaja de estar basada en componentes independientes., pudiéndose ejecutar en diferentes nodos. Y tiene la desventaja de apoyarse en un nodo central, el LPS, elemento crítico para el control del sistema.

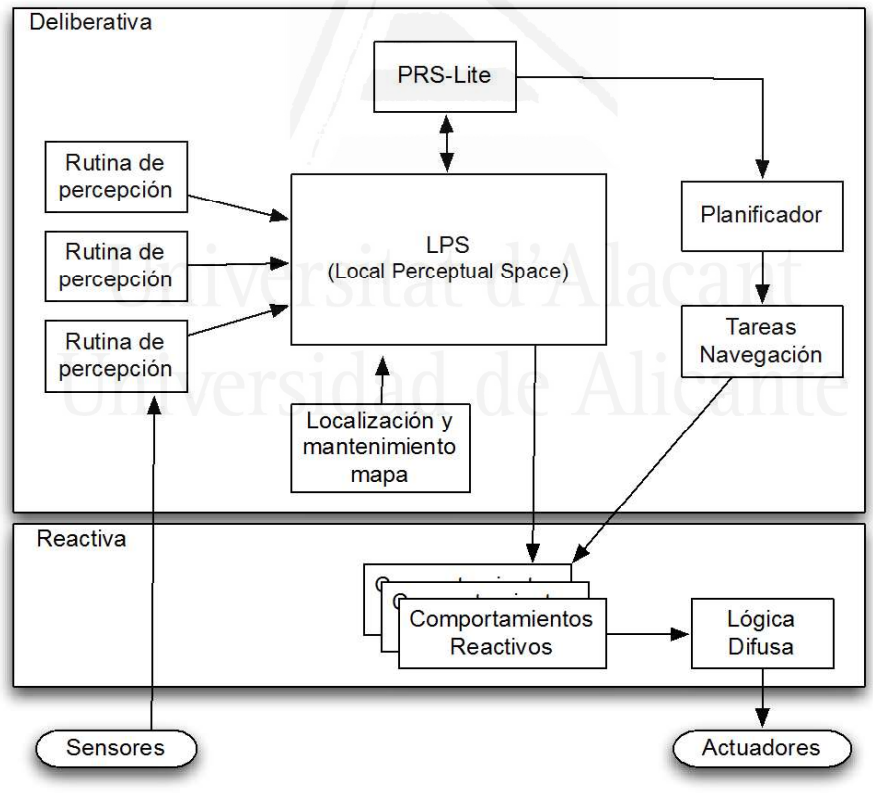


Ilustración 31: Arquitectura híbrida orientada a modelos Saphira.

2.5.3.2 Arquitectura TCA.

Según Murphy en [51], "la arquitectura TCA (Task Control Architecture) se parece más a un sistema operativo que a una arquitectura de propósito general". Esta arquitectura aporta unas estructuras de control que sirven para desarrollar los comportamientos. El sistema se compone de diferentes módulos, con tareas específicas, que se comunican mediante mensajes a través de un servidor central.

Esta arquitectura no es apta para tareas que requieran ejecución en tiempo real según indica su propio manual.

2.5.4 Arquitecturas Basadas en Agentes.

La mayor parte de arquitecturas estudiadas en los puntos anteriores están constituidas por módulos. Actualmente, muchos estudios introducen los sistemas multiagente para implementar sistemas robóticos, en concreto la mayoría de arquitecturas híbridas sustituyen el tradicional modelo de control por sistemas distribuidos basados en agentes [62]. A continuación, describiremos algunas arquitecturas basadas en sistemas multiagente para el control de un robot individual y más adelante describiremos las arquitecturas multirobóticas.

2.5.4.1 Arquitectura Busquets.

La arquitectura Busquets [63] es una arquitectura diseñada para un sistema de navegación basado en visión. En la Ilustración 32 podemos ver la estructura de esta arquitectura de tres capas. En la capa inferior se sitúan los sensores y actuadores. En la capa intermedia encontramos los sistemas ejecutores, los cuales tienen acceso a los sensores y actuadores del robot ofreciendo servicios al resto del sistema. Y en la capa superior es donde se llevan a cabo las tareas de alto nivel, es decir es donde se sitúan los sistemas deliberativos. Esta arquitectura, está compuesta a su vez, de tres sistemas: el sistema piloto, el sistema de visión y el sistema de navegación. Los dos primeros son los sistemas ejecutores y el último es el sistema deliberativo. El sistema piloto es el encargado de todos los movimientos del robot, por un lado, seleccionando los movimientos recibidos desde el sistema de navegación y por otro, de manera independiente, para evitar obstáculos. El sistema de visión es el que identifica y hace el seguimiento de los puntos de interés. Por último, el sistema de navegación es el encargado de guiar al robot hasta el objetivo tomando las decisiones de alto nivel. El sistema está compuesto de varios agentes con distintas responsabilidades donde cada agente propone al sistema

de navegación las acciones a realizar. Existe un agente encargado de coordinar estas acciones propuestas.

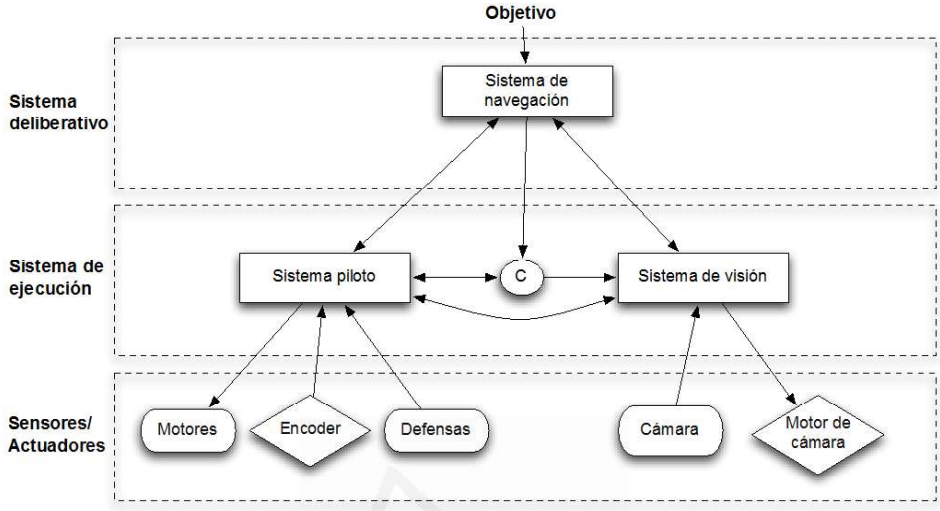


Ilustración 32: Arquitectura basada en sistemas multiagente Busquets.

En esta arquitectura los sistemas necesitan cooperar para alcanzar el objetivo global, pero también, compiten por el control de los actuadores del robot. A pesar de la distinción en capas, la arquitectura no es jerárquica y la coordinación se realiza de forma centralizada por parte de un agente central.

2.5.4.2 Arquitectura Innocenti.

La arquitectura Innocenti [64] es una arquitectura multiagente combinada con control cooperativo. Un sistema multiagente define la arquitectura global, como podemos ver en la Ilustración 33, mientras que para diseñar y desarrollar cada agente individual se utiliza el control cooperativo. Esta arquitectura está formada por cuatro subsistemas: percepción, comportamiento, deliberación y actuación; por un agente que contiene la interfaz con el usuario y la plataforma de agentes. A través del subsistema de percepción se obtiene la información del entorno y de las condiciones internas del robot. El subsistema de comportamiento es el que lleva a cabo las acciones específicas, dirigirse a un punto, evitar obstáculo, etc. utilizando la información enviada por los agentes de percepción. El subsistema deliberativo realiza las tareas de alto nivel (localización, planificación de tareas y planificación de camino) a través de varios agentes. El subsistema de actuación recibe la información del resto de subsistemas y envía al robot las órdenes de movimiento.

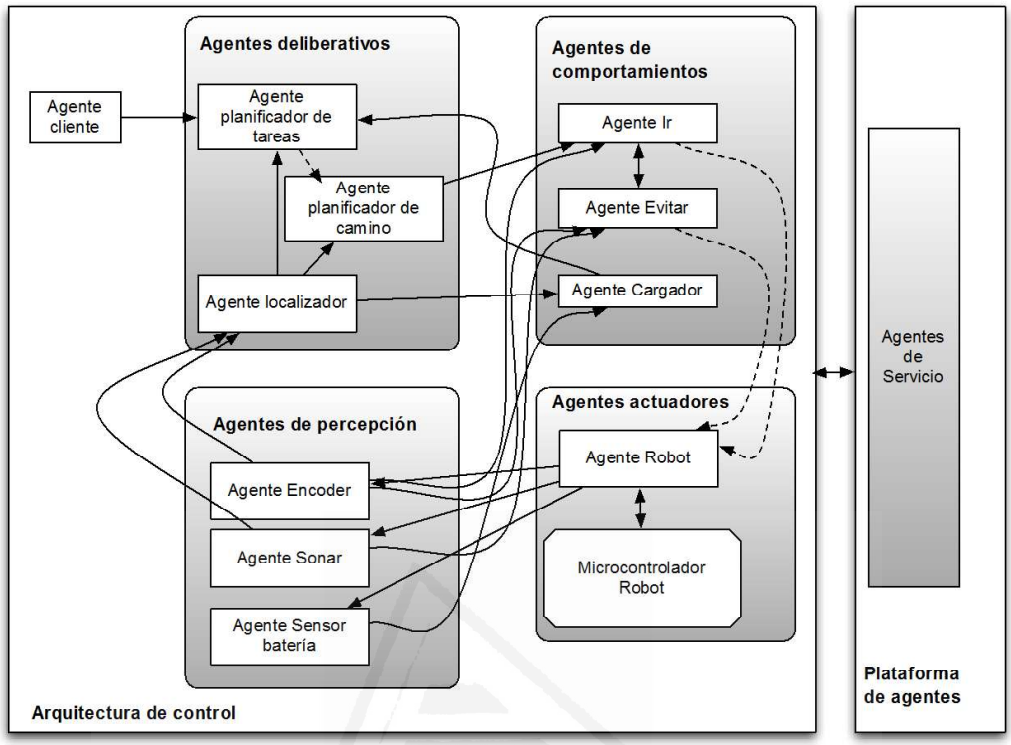


Ilustración 33: Arquitectura robótica basada en agentes Innocenti [64].

Gracias a la utilización de agentes, esta arquitectura es flexible y escalable, pero como muchos sistemas multiagente presenta una sobrecarga de las comunicaciones al hacer uso intensivo del agente de páginas amarillas.

2.5.4.3 Arquitectura SC-Agent.

La arquitectura SC-Agent [65] es una arquitectura híbrida, distribuida y multinivel, basada en agentes software que interactúan a través de una pizarra, diseñada para el control de robots móviles. La arquitectura está compuesta por tres niveles (como se muestra en la Ilustración 34): un nivel deliberativo, un nivel reactivo, y una plataforma intermedia para las comunicaciones. Todos los componentes de la arquitectura se relacionan a través de esta plataforma intermedia. El nivel deliberativo representa el nivel más alto de conocimiento, ejecutando las tareas de planificación sobre un modelo simbólico interno del estado del entorno. El planificador de la misión divide los objetivos en patrones de comportamiento y los envía al nivel reactivo en forma de esquemas de percepción y de motor. Estos esquemas son agentes que se ejecutan

concurrentemente y utilizan la información de los sensores para decidir las acciones a llevar a cabo por los actuadores. Este nivel reactivo está compuesto de: los sensores y actuadores; los esquemas de percepción (que acceden a los sensores y producen percepciones) y motores (que acceden a las percepciones y producen acciones); y las motivaciones (cada esquema motor posee un proceso de motivación que indica el comportamiento a llevar a cabo). Todos los agentes del nivel reactivo están unidos a una pizarra intermedia. La capa intermedia entre los dos niveles anteriores es la plataforma de esquemas y comunicación, la cual proporciona la infraestructura de acceso a los sensores y actuadores. El nivel deliberativo accede al valor de los sensores, transforma la información obtenida al modelo simbólico interno del entorno, y envía los esquemas necesarios al nivel reactivo a través de esta plataforma intermedia.

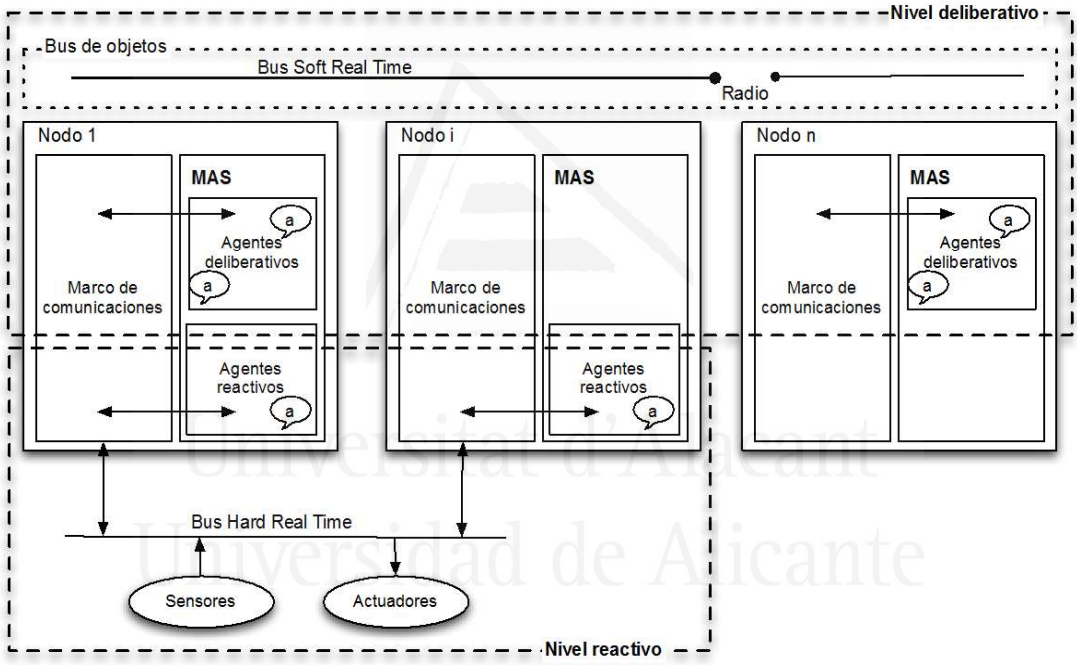


Ilustración 34: Arquitectura robótica basada en agentes SC-Agent.

2.5.5 Arquitecturas Multirobóticas.

En los últimos años se ha multiplicado el interés por el desarrollo de sistemas formados por múltiples robots autónomos, donde a partir de las interacciones entre los individuos del grupo emerge un comportamiento colectivo. Disponer de varios robots facilita que el sistema sea más flexible, robusto y efectivo para el desarrollo de determinadas tareas, donde se tiene una distribución intrínseca y paralela en cuanto a sensores y actuadores [66].

A continuación, vamos a describir algunas arquitecturas diseñadas para sistemas multirobóticos.

2.5.5.1 Arquitectura ALLIANCE.

ALLIANCE [67] es una arquitectura distribuida diseñada para facilitar la cooperación de un grupo de robots heterogéneos, que pone el foco en la robustez de los comportamientos haciéndolos tolerantes a fallos. Los robots están dotados de funciones de alto nivel y son organizados en equipos. Cada uno de los robots seleccionará las acciones a realizar dependiendo de la misión, las características del entorno y del estado de él y de los robots. Esta selección está basada en motivaciones, las cuales se han modelado matemáticamente creando el llamado "comportamiento motivacional" del robot. Las motivaciones internas del robot se dividen en dos grupos: impaciencia (robot impatience) y consentimiento (robot acquiescence). Las primeras hacen al robot realizar tareas cuando otros erran, mientras que las segundas intentan corregir acciones fallidas del propio robot.

Esta arquitectura tiene una variante más moderna, L ALLIANCE, en la cual se le ha dotado a la arquitectura de aprendizaje por refuerzo con el fin de calibrar mejor los parámetros de activación de los comportamientos.

El problema de esta arquitectura es tratar de mantener una comunicación eficiente y escalabilidad con la asunción de que cada robot conozca todo lo que está haciendo el resto del equipo, incluidos los fallos.

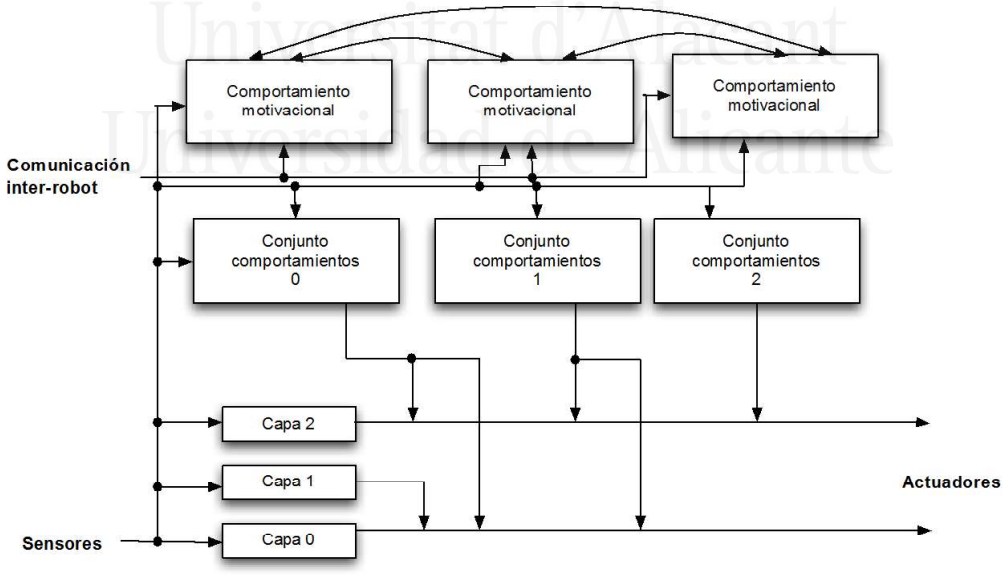


Ilustración 35: Estructura interna de cada robot de la Arquitectura multirobótica Alliance.

2.5.5.2 Arquitectura HEIR.

Dentro de la arquitectura HEIR [68] (Ilustración 36) podemos encontrar 3 componentes diferenciados en base al conocimiento que manejan:

- Simbólico (S): Formalismo declarativo, explícito y proposicional. Es utilizado cuando un robot tiene que realizar tareas complejas y requiere de planificación.
- Diagramático (D): Representaciones analógicas e icónicas (sintetizadas del conocimiento simbólico). Se utiliza en pequeños razonamientos donde se utiliza información sensorial.
- Reactivo (R). Esta relacionado con comportamientos de seguridad, como evitar un obstáculo.

Es una arquitectura donde la actividad se centra en determinados componentes en momentos distintos, no poseyendo una organización jerárquica. Dependiendo del instante de tiempo, como consecuencia de los eventos internos o externos, cualquiera de los tres componentes podría ocupar el nivel superior.

Para trabajar con esta arquitectura utilizaremos una base de conocimiento común llamada KB, sobre la cual el componente simbólico operará; y tendremos que definir las representaciones sintetizadas (DN) para aplicar el componente diagramático.

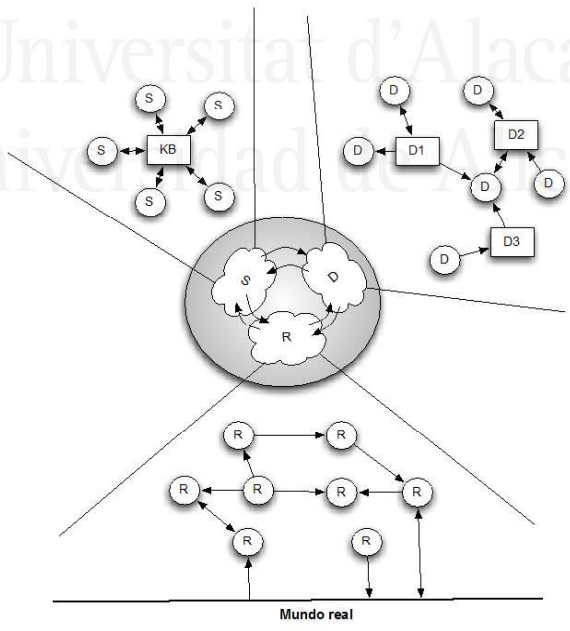


Ilustración 36 : Arquitectura multirobótica HEIR [68].

2.5.5.3 Arquitectura Goldberg.

Goldberg [69] es una arquitectura donde para cada uno de los robots del sistema se definen tres niveles. Cada uno de los niveles, de cada robot, puede comunicarse directamente con el mismo nivel de los otros robots, como se puede observar en la Ilustración 37. Gracias a esto, los robots pueden actuar de forma autónoma, pero al mismo tiempo coordinarse con el resto. Los tres niveles de la arquitectura son: conducta, ejecutor y comportamientos. En el nivel de conducta son coordinados los comportamientos; en el nivel de ejecución son sincronizadas las ejecuciones de las tareas; y en el nivel de planificación se localizan los recursos y son asignadas las tareas.

El nivel de conducta está basado en estudios de mercado y tiene a su vez dos componentes: un agente de mercado encargado de participar en la subasta de tareas y un planificador que estudia el coste y la viabilidad de las tareas, e interactúa con el nivel de ejecución para llevarlas a cabo. El nivel de ejecución es el encargado de descomponer las tareas y de monitorizarlas. Por último, a bajo nivel, se crean bucles de comportamientos, donde los comportamientos sensitivos de un robot se unen con los comportamientos actuadores de otro.

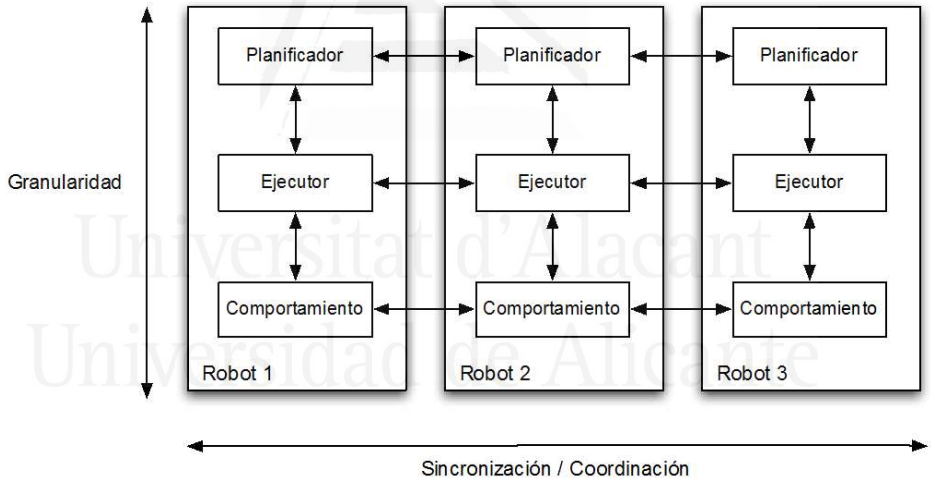


Ilustración 37 : Arquitectura multirobótica Goldberg [69].

Debido a la interconexión entre robots y niveles, esta arquitectura sufre una penalización en la carga de las comunicaciones.

2.5.5.4 Arquitectura Lei.

La arquitectura Lei [70] fue creada en 2010 para modelar la planificación de movimiento de un sistema multirobótico. Es una arquitectura jerárquica que consta de cuatro capas: monitorización, planificación de la misión, coordinación del movimiento y control de comportamiento. En la capa de control encontramos

los controles de teleoperación, ya sea realizada por un humano o una máquina, aunque cuando es operada por una máquina sólo intervendrá en casos de conflicto, bloqueo, o de una excepción. La capa de planificación de la misión, se encarga de controlar al robot planificando el punto de partida y el objetivo de la misión. En la siguiente capa, los robots coordinadores distribuirán la información de las misiones entre los robots autónomos. La capa de coordinación de movimiento es la encargada de planificar el camino. Esta tarea será llevada a cabo por parte de los robots coordinadores los cuales incorporarán una planificación de evitación de obstáculos. Por último, la capa de control de comportamiento, está formada por los robots autónomos ejecutarán la planificación recibida.

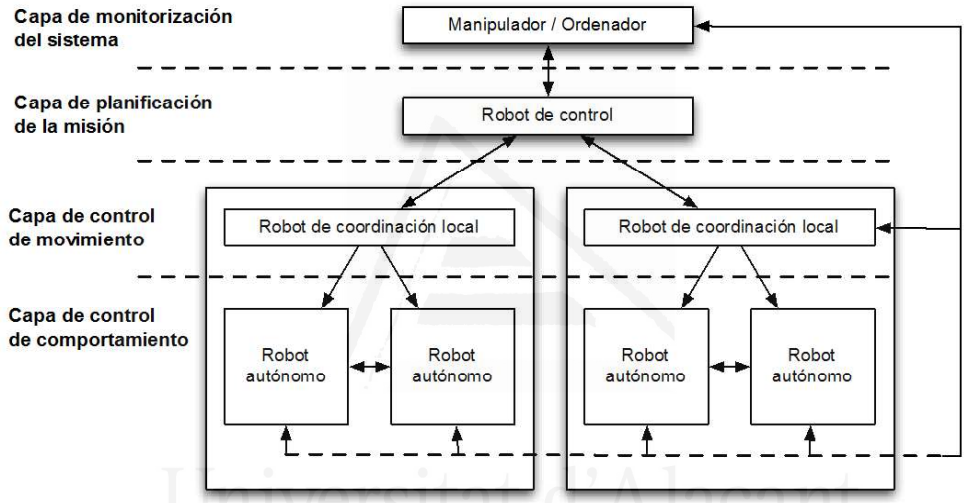


Ilustración 38: Arquitectura multirobótica Lei [70].

Al tener un robot central responsable de la planificación global, es difícil cumplir con la tolerancia a fallos y la escalabilidad.

2.5.5.5 Arquitectura Marino.

En [71] A. Marino presenta una arquitectura de control para un sistema multirobótico especializado en vigilancia, que utiliza el modelo de control reactivo Null-Space-Based Behavioral Control (NSB). En este modelo, las tareas se dividen en tareas más sencillas, a las cuales se les denomina comportamientos, los cuales son ordenados en base a su prioridad. Esta arquitectura aporta una reordenación dinámica de esos comportamientos. En la Ilustración 39, podemos ver que la arquitectura está compuesta de tres niveles. En la capa inferior encontramos los agentes que son las unidades robóticas individuales que llevan a cabo las tareas. En la capa intermedia se definirán los

comportamientos elementales, los cuales con combinados por el NB en forma de tareas más complejas. La capa superior de supervisión es donde se seleccionarán las acciones adecuadas para ser ejecutadas.

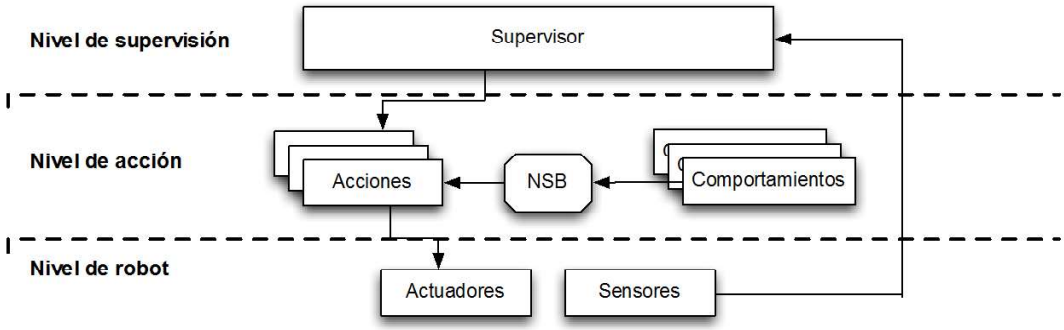


Ilustración 39: Arquitectura definida por A.Marino [71].

Según el autor, la ventaja de esta arquitectura está en que es descentralizada y sin uso de comunicación, es robusta y previene de colisiones. Pero debemos de tener en cuenta que en ella se asume que los robots conocen su localización en todo momento y poseen una descripción del entorno cercano.

2.6 Resumen del Capítulo.

En este capítulo hemos definido el concepto de agente y sus características: autonomía, reactividad, adaptabilidad, sociabilidad, iniciativa y continuidad temporal. También hemos expuesto una clasificación de los distintos tipos de agentes: de interfaz, colaborativos, móviles, de información, reactivos e híbridos.

A continuación, pasamos a definir y a describir los sistemas multiagente, así como las características deseadas en estos sistemas: eficiencia, escalabilidad, fiabilidad, velocidad de desarrollo y reusabilidad. En uno de los apartados nos centramos en una de las cuestiones clave de estos sistemas: la comunicación, detallando los distintos métodos, protocolos y lenguajes de comunicación y coordinación entre los agentes.

No solamente nos centramos en las arquitecturas de sistemas de agentes, sino que decidimos ampliar el estudio a arquitecturas robóticas debido a que en los sistemas multiagente se necesitan cubrir componentes de bajo nivel. Estas arquitecturas robóticas se entremezclan muchas veces con las arquitecturas para sistemas multiagente, incluso algunas de ellas las utilizan (Busquets, Innocenti, SC-Agent).

Las arquitecturas estudiadas describían los diferentes componentes del sistema, su organización y su interacción, con el fin de lograr las capacidades y funcionalidades necesarias para llevar a cabo determinadas tareas. La mayor parte de arquitecturas actuales utilizan el paradigma híbrido, se estructuran en capas o niveles, los cuales están compuestos a su vez por subsistemas o módulos, y buscan una mayor robustez, flexibilidad y escalabilidad de los sistemas.

En el estudio de las arquitecturas robóticas individuales hemos visto que la mayoría se coordinan de una manera centralizada, incluso las basadas en agentes, a pesar de ser arquitecturas distribuidas, se basan en un módulo central coordinador como BERRA [57], Saphira [59], Busquets [63], Innocenti [64] o SC-Agent [65]. De las arquitecturas multirobóticas vistas algunas tienen una coordinación descentralizada como es el caso de ALLIANCE [67], Goldberg [69] o Marino [71]. Lei [70] sin embargo, se basa en un robot central que lleva a cabo toda la planificación y en HEIR [68] la coordinación depende del tipo de conocimiento necesario. Dentro de todas ellas pudimos ver internamente arquitecturas reactivas, deliberativas, BDI e híbridas.

Para estudiar las metodologías lo tuvimos algo más difícil por la falta de consenso y por el hecho de que hayan aparecido distintas metodologías a través de distintas áreas de conocimiento (orientación a objetos, sistemas expertos, métodos formales, análisis de roles, integración o BDI), describiendo al final las siguientes: GAIA,TROPOS, PASSI, Prometheus, MaSE, MAS-CommonKADS, MESSAGE e INGENIAS. Todas ellas se centran principalmente en el análisis y diseño del sistema a nivel de agentes, pero entran nada o poco en el diseño y construcción del agente. Algunas de las metodologías (GAIA, PROMETHEUS, MaSE, utilizan modelos de desarrollo secuenciales, mientras que otras ya utilizan modelos iterativos (TROPOS, PASSI, MAS-CommonKADS, MESSAGE e INGENIAS), más modernos y recomendados.

Para abordar la construcción tanto del sistema como de los agentes hicimos un estudio de dos de los frameworks de agentes más utilizados: JADE y ZEUS. En estos frameworks observamos que facilitan la construcción, ejecución y monitorización de sistemas multiagente que cumplen con el estándar FIPA. Se basan en herencia, y ofrecen unas clases base de agentes (AMS, DF, MTS) para permitir la comunicación y coordinación de agentes del sistema. Sin embargo, estos frameworks dejan al investigador la labor de desarrollo de componentes de bajo nivel.

3 Propuesta. Arquitectura Híbrida Multinivel

3.1 Introducción.

En el capítulo anterior vimos que la mayoría de diseños definidos para un único agente suele seguir el paradigma híbrido, ya que intenta obtener las ventajas de los paradigmas reactivo y deliberativo reduciendo así sus puntos débiles.

La mayoría de estas arquitecturas multiagente se estructuran (o pueden estructurarse) en capas o niveles, cada uno de ellos formado por varios módulos. Todas estas arquitecturas presentar un grado de flexibilidad, gracias a la existencia de un módulo planificador o de supervisión que modifica los comportamientos según las necesidades. La mayoría de arquitecturas incluyen algún mecanismo genérico para dotar a la plataforma de robustez como SFX [53] o ALLIANCE [67].

Un aspecto importante a tener en cuenta en el diseño de una arquitectura es la escalabilidad del sistema. Muchas de estas arquitecturas (Yavuz [54], 3T [56] o Busquets [63]) tienen una escalabilidad baja o media, donde incluir nuevos elementos supondría cambios importantes en los ya existentes. Otras arquitecturas permiten, de manera sencilla, solamente la incorporación de nuevos comportamientos como AuRA [52], SFX [53], Saphira [59] o Innocenti [64]. Muchas veces cuando escalamos un sistema aparecen problemas de sobrecarga en los canales de comunicación. Esto lo podemos observar en ALLIANCE [67] o en Goldberg [69] por el gran número de agentes que utilizan.

Algunos problemas nuevos nos exigen sistemas descentralizados, mientras que la mayoría de las arquitecturas estudiadas tienen una coordinación centralizada, como Innocenti [64] o SC-Agent [65], BERRA [57], Saphira [59] y

Busquets [63]; o se basan en un agente central (ALLIANCE [67], Goldberg [69] o Marino [71], Lei [70]).

Además de las arquitecturas, estudiamos distintas metodologías para la construcción de sistemas multiagente. Algunas de ellas, como GAIA o TROPOS y MESSAGE, no cubrían las fases de codificación y despliegue. Otras como INGENIAS no tenía soporte para agentes móviles, requisito necesario para nuestras pretensiones de desarrollar de un modelo versátil. Y otras estaban muy orientadas a un tipo de problemas, como el caso de MAS-CommonKADS que tiene una orientación muy fuerte hacia sistemas expertos con interacción con el usuario.

A nivel de herramientas y frameworks observamos que la mayoría de esfuerzos han sido dirigidos al desarrollo de la plataforma de agentes, ofreciendo a nivel de implementación sólo una serie de agentes básicos (AMS, DF, MTS) y las herramientas para permitir la comunicación y coordinación de los distintos agentes. En algunos frameworks como JADE se añaden exclusivamente unos agentes de depuración y monitorización. Podemos decir que el mayor vacío se encuentra en la construcción de los elementos de bajo nivel, donde los equipos de desarrollo construyen prácticamente desde 0 estos elementos en base a sus necesidades específicas, haciendo que la implementación real de sistemas multiagente tenga unos costes económicos y temporales altos.

Debido a los problemas anteriores nosotros proponemos una arquitectura híbrida multinivel la cual va a ser versátil, flexible y escalable. Esta arquitectura simplificará el diseño de los sistemas multiagente tanto a alto como a bajo nivel y permitirá construir los sistemas de una manera ágil utilizando los estándares que nos proporciona el mercado.

La parte clave que aporta este modelo es la diferenciación del sistema a bajo nivel, el cual tendrá su propio diseño, sus propios estándares y canales de comunicación, así como una implementación diferenciada del resto del sistema multiagente. Muchas arquitecturas se han estructurado en niveles, pero nosotros, aparte de separar en niveles, hacemos que cada nivel sea completamente independiente en diseño, herramientas e implementación, ya que cada una de las capas tiene unos requerimientos específicos. Pero no nos olvidamos de factores tan importantes como la estandarización o la reutilización, características sin las cuales no lograríamos proyectos viables. Está reutilización se puede apreciar en todos los niveles de la arquitectura, y nos valemos de ella para diseñar e implementar la capa intermedia que hace las funciones de Gateway entre las capas.

Acompañaremos la propuesta de arquitectura con una metodología ágil e iterativa, la cual estructuraremos en una serie de fases y etapas, la cual nos permitirá diseñar y desarrollar un sistema multiagente. Dentro de cada una de

estas etapas indicaremos los artefactos que deberemos de ir generando hasta completar la producción del sistema.

3.2 Arquitectura Propuesta.

La arquitectura que proponemos es una arquitectura híbrida multinivel la cual podemos ver en Ilustración 40. Es híbrida al combinar módulos deliberativos y reactivos. Los reactivos se encargarán de procesar los estímulos que no requieran deliberación, mientras que los módulos deliberativos determinarán las acciones para satisfacer los objetivos locales y cooperativos de los agentes. Es multinivel porque tiene tres niveles claramente diferenciados. El nivel superior será el encargado de la coordinación, encontrándose en ella los agentes deliberativos, de planificación y de control; mientras que el nivel inferior se encargará de la percepción y actuación. El nivel intermedio actúa como Gateway entre la capa de alto y la de bajo nivel.

Una característica clave en nuestra arquitectura es que tendremos 2 canales de comunicación diferentes para los elementos de cada nivel, adecuándose mejor las comunicaciones a las características requeridas en cada uno de ellos. Incluso, atendiendo a la naturaleza del problema, podríamos tener distintos canales de comunicación físicos a bajo nivel con el fin de mejorar el rendimiento, la escalabilidad y la privacidad.

En el nivel superior encontramos los agentes de servicio encargados de controlar la plataforma. Su misión será, como vimos en el capítulo 2, la marcada por la normativa FIPA, proporcionar la infraestructura física donde los agentes serán desplegados, la publicación de los servicios ofrecidos y la localización de agentes dentro de la plataforma. Dentro de este bloque podríamos introducir agentes que nos permitan sincronizar plataformas distribuidas o gestionar la caída y recuperación de agentes o incluso de la plataforma completa.

Otro bloque importante dentro de este nivel son los agentes de deliberación y de planificación. Estos son los que van a contener la parte de comportamiento y es donde se encontrará el corazón de la inteligencia artificial del sistema multiagente. Dependiendo del problema deberemos de identificar que agentes van a pertenecer a este subconjunto así como sus diferentes roles.

En este alto nivel nos encontramos también agentes específicos para comunicarse con los usuarios y con otros sistemas, permitiéndonos encapsular la comunicación con otros subsistemas. Dejamos aquí libertad para determinar el número de agentes necesarios, pero recomendamos centrar la interfaz de usuario en un agente que actuará como cuadro de mandos, y tener tantos agentes de interfaz como subsistemas tengamos, aunque este último número,

dependerá de la complejidad de los sistemas con los que necesitamos interactuar.

La comunicación entre los elementos de este nivel (agentes) la haremos a través del canal proporcionado por los agentes de servicios de la plataforma, utilizando protocolo de transportes de mensajes FIPA y el lenguaje FIPA-ACL.

En el nivel más bajo nivel es donde encontraremos los elementos que van a recibir la información de los sensores y mandarán los comandos a los actuadores. La arquitectura de estos elementos es más reactiva, aunque es capaz de realizar comportamientos sencillos, utilizando principalmente de forma interna arquitecturas puramente reactivas o deliberativas (Ilustración 3) de capas horizontales o verticales de 1 o 2 pasos (Ilustración 6). A estos elementos de bajo nivel les vamos a llamar "microagentes" para diferenciarlos claramente de los agentes del nivel superior.

Para este nivel, atendiendo al problema que tengamos, podemos hacer la comunicación en un único canal o en varios. Tener un único canal nos simplifica la infraestructura necesaria, mientras que el tener varios nos va a permitir un rendimiento mayor al no tener que compartir el canal. Además, la existencia de varios canales, ofrece una privacidad total, aunque esto lo podríamos conseguir en un único canal cifrando los contenidos.

Para entender la arquitectura podemos poner el símil con personas. A alto nivel tendríamos personas, cada una con su rol, que dialogan entre sí. Mientras, a bajo nivel, tendríamos los órganos que componen a cada persona. Cada órgano tendría su función y su comunicación sería a través de la espina dorsal. Los requerimientos y características de ambos sistemas son tan distintos que tienen que implementar de formas totalmente diferentes. A lo largo de los años se ha tratado de diferenciar los elementos conceptualmente, pero no en términos de implementación reales, lo que ha supuesto un lastre importante en la implementación de los sistemas multiagente.

Para terminar de describir la arquitectura hablaremos del nivel intermedio. La función de este nivel va a ser doble: por un lado, como hemos comentado anteriormente, va a servir de nexo entre los niveles superior e inferior haciendo las funciones de gateway. Además, realizará funciones de secuenciación, dividiendo las órdenes recibidas por los agentes del nivel superior en partes y repartiéndolas entre los elementos del nivel inferior. Este nivel puede estar compuesto de varios agentes pudiendo formar agrupaciones lógicas de los elementos de bajo nivel.

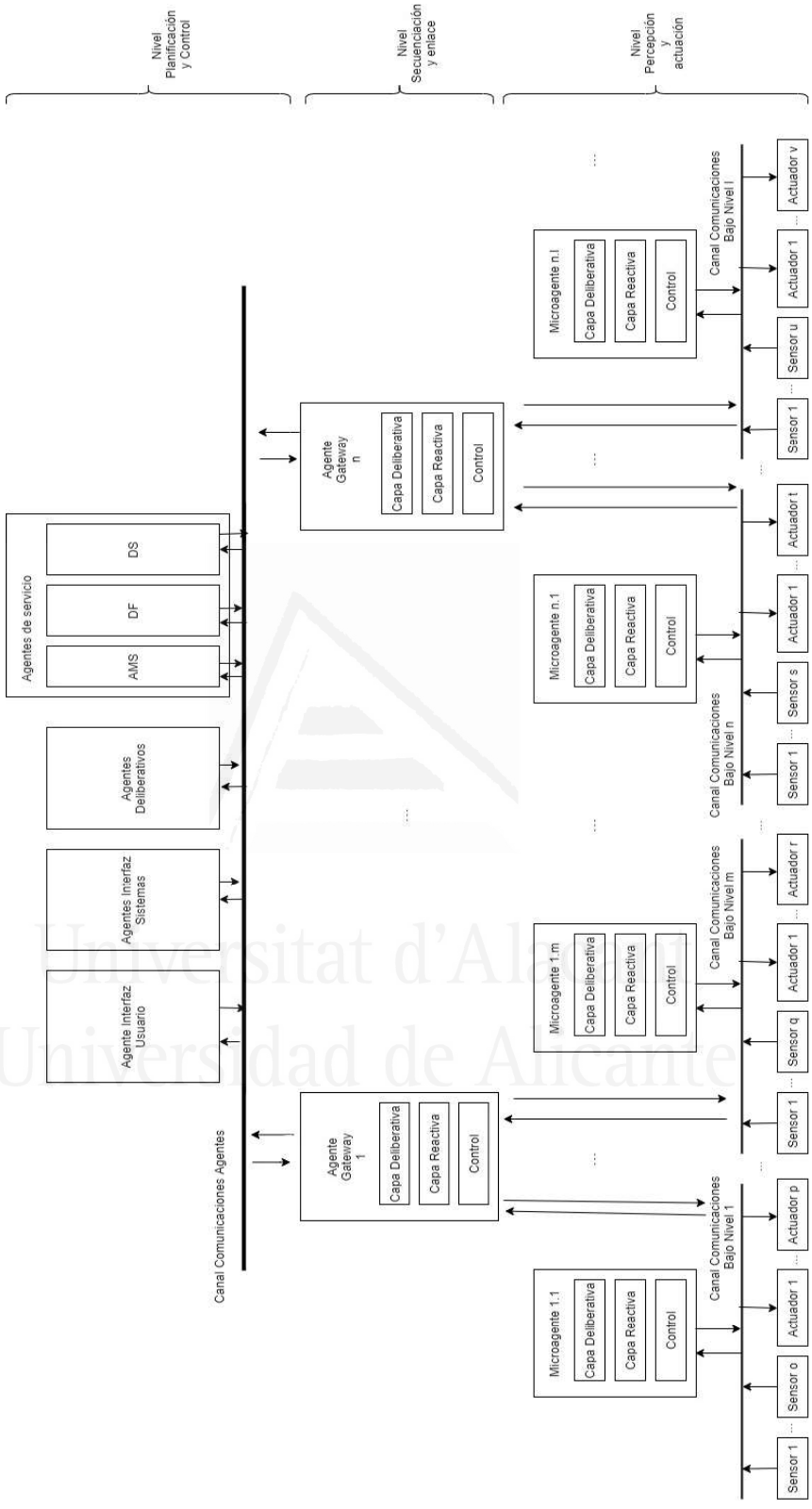


Ilustración 40. Arquitectura híbrida multinivel propuesta.

3.3 Metodología Propuesta.

La propuesta de nuestra metodología está basada principalmente en la metodología PASSI, pero dándole un carácter más ágil. Las principales razones para basarnos en esta metodología fueron que es una metodología específica para sistemas multiagente, cubre todas las fases (desde la toma de requisitos hasta el despliegue), soporta agentes móviles, utilizaba un modelo iterativo y está mejor documentada que otras.

Como en la metodología PASSI hemos dividido la nuestra en 5 fases, pero hemos preferido mantener la nomenclatura de la mayoría de metodologías donde dividimos en modelo de proceso en fases y las fases en etapas (En PASSI a las fases las llama modelos y a las etapas fases). En la Ilustración 41 podemos ver las fases seguidas por nuestra metodología: Análisis de requerimientos, Diseño de Sociedad de Agentes, Diseño de Implementación de Agentes, Codificación y Pruebas y Despliegue.

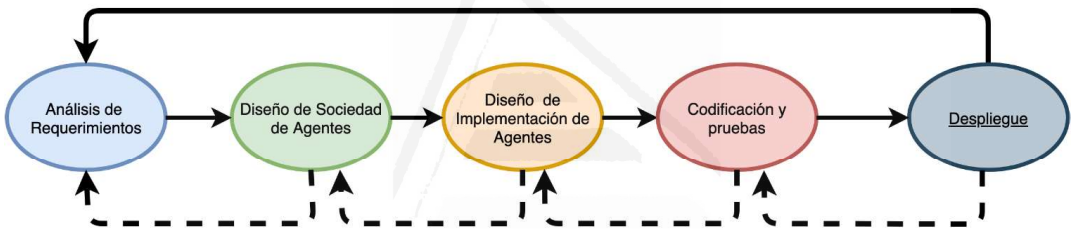


Ilustración 41. Fases de la metodología propuesta

La idea de esta metodología es utilizarla junto a un modelo iterativo e incremental en el cual en cualquier momento podríamos hacer cualquier tipo de actividad, permitiéndonos modificar los modelos creados en pasos anteriores, no siendo necesario pasar por todas las fases para volver a fase la inicial. Lo habitual es volver a la fase anterior para hacer alguna pequeña modificación debido a cambios en el diseño o a errores descubiertos.

La arquitectura propuesta será introducida en la fase de "Diseño de Sociedad de Agentes". Para poder adecuarla al problema serán clave los roles identificados en la fase anterior. De la aplicación de la arquitectura se derivarán dos tipos de diagramas de clases, uno donde nos interesa representar las relaciones de herencia (generalización) entre los agentes del sistema y otro con los distintos tipos de mensajes de los canales de comunicación.

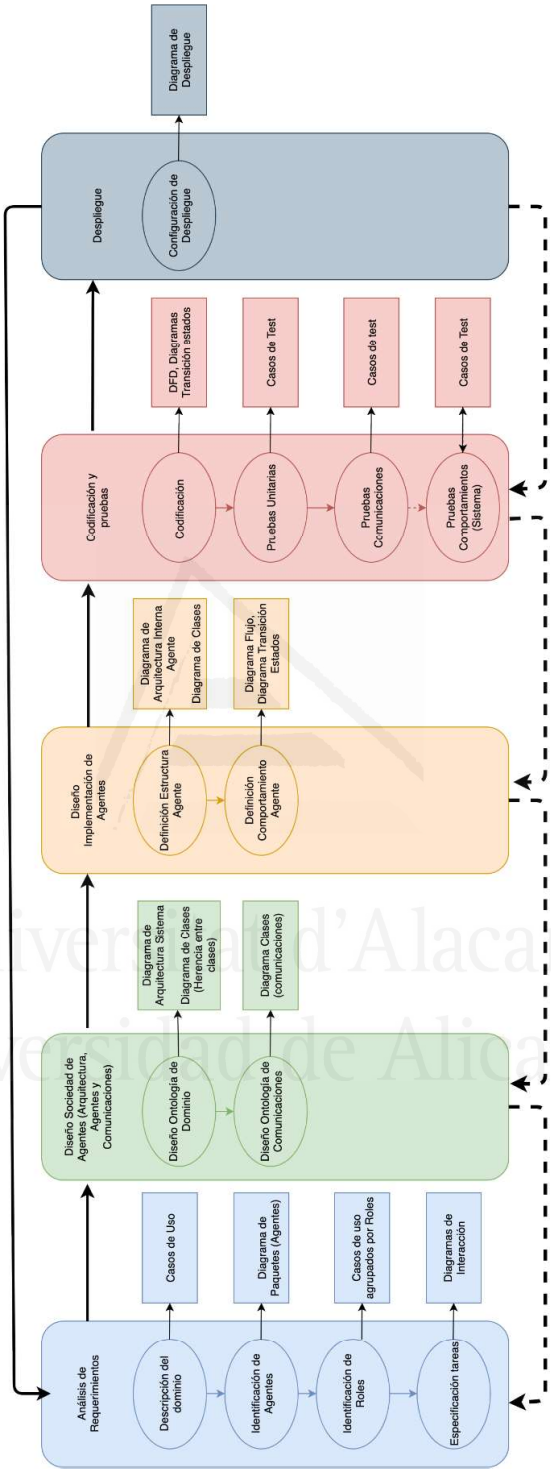


Ilustración 42. Fases y etapas de la metodología propuesta.

En la fase de "Diseño de la Implementación de Agentes" es donde nos centraremos, como su nombre indica, en el diseño interno del agente estableciendo su arquitectura interna. Gracias a la arquitectura general, donde se establecieron tipos de agentes, el diseño interno se simplificará al ir heredando de clases padre muchas de las estructuras y métodos a desarrollar. Los resultados de esta etapa serán los diagramas de clases para cada tipo distinto de agentes. La estructura definida será completada con la especificación del comportamiento de los agentes, para lo cual serán fundamentales los diagramas de actividad creados en fases anteriores. La arquitectura del sistema y de los agentes condiciona el comportamiento de estos. Nuestra arquitectura facilitará el diseño e implementación de la estructura, el comportamiento y las comunicaciones de todos los elementos del sistema.

En la Ilustración 42, podemos ver un diagrama de proceso de nuestra metodología. En los apartados siguientes, vamos a describir cada una de las fases y etapas, indicando en cada etapa los artefactos generados. En los capítulos 4 y 5, de validación de la propuesta, veremos la aplicación de la misma.

3.3.1 Análisis de Requerimientos del Sistema.

Empezamos a describir el dominio del problema y las funcionalidades, así como identificar los agentes y sus roles y las tareas. Esta fase está compuesta por cuatro etapas: descripción del dominio, identificación de agentes, identificación de roles y especificación de tareas. La primera etapa recopila las funcionalidades del sistema en diagramas de casos de uso convencionales. En el resto de etapas se describe una especificación de alto nivel de los agentes, haciendo en primer lugar una identificación preliminar de los agentes, repartiendo los casos de uso, planteando los roles y presentando los escenarios principales del sistema a través de diagramas de secuencia.

3.3.1.1 Descripción de Dominio.

En esta etapa se produce el diagrama de casos de uso donde se identifican todas las funcionalidades a realizar, así como las entidades que van a interactuar con el sistema, siguiendo el estándar UML, se representarán como actores. Como entidades tendremos los usuarios, que interactúan con nuestro sistema, y los sistemas externos, con los que nuestro sistema va a colaborar. Hoy en día es muy habitual, hacer uso de sistemas externos ya existentes con el fin de recabar información de ellos, así como delegar acciones sobre ellos.

3.3.1.2 Identificación de Agentes.

La identificación de agentes corresponde a la agrupación de funcionalidades atendiendo a las responsabilidades de los agentes. Será una

primera aproximación, donde utilizaremos un diagrama de paquetes donde cada paquete representará un agente.

3.3.1.3 Identificación de Roles.

En esta fase vamos a dividir las funcionalidades en grupos, denominados roles. Algunos de estos roles ya vienen predeterminados por la normativa FIPA, lo cual va a aligerar y simplificar la identificación, haciendo que nos podamos centrar en los roles del problema concreto a resolver. Tener en mente la arquitectura propuesta, aunque la vayamos a concretar en la siguiente fase, nos facilitará una creación de roles más fáciles de una asignar a los diferentes agentes de la futura arquitectura. La identificación de roles la haremos con diagramas de caso de uso.

3.3.1.4 Especificación de Tareas.

La especificación de tareas corresponde al último paso de la fase de análisis de requerimientos de sistema. En esta etapa se trabaja realizando un diagrama de secuencia para cada caso de uso identificado, de forma que vemos la interacción entre los distintos participantes (agentes y entidades), con el fin de representar todas las tareas.

3.3.2 Diseño de Sociedad de Agentes.

Esta es una fase clave, donde se diseña la arquitectura del sistema multiagente. En nuestro caso, partiendo de la arquitectura propuesta se concretará una arquitectura final que recoja los agentes involucrados, sus dependencias y las interacciones sociales. Nos tocará ir definiendo los agentes de cada nivel, así como los canales de comunicación en cada uno de ellos. Aquí ya diferenciamos entre agentes (agentes de nivel alto o medio) y microagentes (agentes de bajo nivel). Es importante en este punto tener seleccionadas los frameworks a utilizar, ya que junto con la arquitectura marcará el diseño final del sistema.

3.3.2.1 Diseño de la Ontología del Dominio.

En esta etapa se van a crear 2 diagramas clave: el diagrama de arquitectura y el diagrama de clases (agentes).

En el diagrama de arquitectura separaremos a los agentes en 3 niveles. Los agentes de alto nivel, van a ser los encargados de la coordinación y la planificación, mientras que los de bajo nivel serán los responsables de percepción y actuación. Según el tipo de problema que tengamos los agentes de nivel medio pueden tener un mero papel de comunicación entre los otros dos niveles o puede aparecer la necesidad de secuenciación de tareas. Cuando esto último ocurre nos

encontramos con la necesidad de modificar los diagramas de identificación de agentes y de roles, así como los diagramas de actividad.

Otro factor clave será la definición los canales de comunicación de cada uno de los niveles. En ellos se establecerá el estándar a seguir, lo cual necesitará de un estudio de opciones y viabilidad. Esta elección dependerá de la naturaleza del problema, de las características de las herramientas y a las habilidades del personal del equipo.

Otra labor que tendremos que hacer con los agentes identificados será clasificarlos según su tipología. Por ejemplo, a alto nivel dividiremos los agentes en: agentes de interfaz usuario, agentes de interfaz sistemas, agentes de control de la plataforma (AMS, DS, DF), agentes de planificación, agentes de deliberación etc. Esto lo haremos a través del diagrama de clases y en concreto con mediante las relaciones de herencia.

3.3.2.2 Descripción de la Ontología de Comunicación.

Cada comunicación es representada por la relación entre los dos agentes y se detalla en la clase de atributos de relación. Dicha clase es identificada por un nombre y es descrita por los campos ontología, lenguaje y protocolo. El "lenguaje" describe el lenguaje utilizado en el contenido de la comunicación, mientras que el "protocolo" precisa el protocolo de interacción adoptado (FIPA) o modificado si fuese necesario.

Recordemos que la arquitectura propuesta es una arquitectura a multinivel donde existen distintos canales de comunicación y la naturaleza de estos puede ser totalmente distinta, teniendo que definir para cada uno de los niveles los protocolos y lenguajes a utilizar.

3.3.3 Diseño de Implementación de Agentes.

La fase de diseño de implementación de agentes está formada por dos etapas estrechamente relacionadas entre sí: Definición de Estructura Interna de Agente y Descripción de Comportamiento de Agentes.

3.3.4 Definición de Estructura de Agente.

Para la estructura interna de agentes utilizaremos dos diagramas. Uno para definir la arquitectura interna del agente y otro diagrama, de clases, donde definiremos por cada clase (agente) los atributos, los métodos y sus relaciones. Esta será completada con la especificación del comportamiento de los agentes mediante diagramas de actividades o secuencia. Se define de esta forma el

diseño del sistema que será implementado con ellos, definiendo la estructura interna de cada agente y su comportamiento.

3.3.4.1 Descripción del Comportamiento de Agentes.

Esta fase también tiene dos niveles de abstracción. En los diagramas multiagente se representan los flujos de eventos, que invocan métodos e intercambio de mensajes entre agente, mediante diagramas de actividades. Por su parte los diagramas de agentes individuales muestran la implementación de las formalidades usadas en los métodos, se puede utilizar el método que se considera más apropiado para describirlo (diagramas de flujo, diagramas de transición de estados, et).

3.3.5 Codificación y Pruebas.

La etapa de codificación genera desde los modelos el código gracias a las herramientas de soporte, que suelen incluir patrones para facilitar la reutilización y la generación de plantillas. En nuestro caso la vamos a utilizar sobre todo para diseñar los algoritmos de planificación y comportamientos específicos de nuestros agentes. Nuestro modelo nos permite centrarnos en lo que realmente nos interesa, en la parte diferenciada de nuestro problema, ya que gracias a la arquitectura empleada y a la herencia y agregación, la funcionalidad general la tendremos resuelta.

Corresponde al código de la solución. Aquí trataremos de hacer uso de los patrones, la herencia y la agregación, así como de la reutilización de código de tareas existentes. A nivel de programación nos centraremos en resolver los detalles de la implementación específicos del problema a resolver.

En esta fase será donde se diseñen las pruebas. Estas pruebas se suelen dividir en pruebas unitarias, pruebas de comunicaciones y pruebas de sistema. En las pruebas unitarias vamos a testear el comportamiento de elementos individuales (comportamientos y habilidades de un agente). En las de comunicación las dividiremos en dos partes: Pruebas de comunicación entre elementos de un mismo nivel y pruebas de comunicación entre elementos de distintos niveles. Por último, las pruebas de sistema servirán para comprobar el funcionamiento de las funcionalidades ofrecidas por el sistema.

La fase de codificación y pruebas va muy ligada a la de despliegue y generalmente no obliga a generar conductores y resguardos para simular el entorno de ejecución.

3.3.6 Despliegue.

Se utiliza un diagrama del despliegue para describir la diseminación de los agentes a través de las plataformas disponibles y de sus movimientos. Las plataformas se describen como nodos de proceso, los agentes como componentes y sus comunicaciones (si no son demasiados) conectan al iniciador con el símbolo de interfaz del participante.

En esta última fase también nos vamos a encontrar con la dualidad de los niveles debiendo de hacer un despliegue para cada uno de ellos y diseñando algún método de integración entre ambos.

3.4 Herramientas.

En el capítulo 2 hicimos un estudio de las herramientas y frameworks más utilizados en la construcción de sistemas multiagente. Estas herramientas nos permitirán desarrollar nuestro alto nivel y parte del medio, pero hay un vacío para desarrollar el bajo nivel. Este nivel va a ser muy específico del sistema a desarrollar, pero hemos visto necesario introducir una nueva sección donde al menos se describan las características que debería de tener un framework para facilitarnos la implementación de este nivel.

La capa de bajo nivel va a realizar principalmente funciones de percepción y actuación, por lo cual, vamos a pedirle a la herramienta la facilidad para utilizar estos dispositivos de bajo nivel. Debe de recibir, mandar y multiplexar mensajes de sensores, control, estados, planificaciones y actuadores, entre otros.

Lo ideal es que la herramienta actúe como capa de abstracción del hardware permitiendo que nuestra programación sirva para distintos dispositivos. Esto nos aportará flexibilidad e independencia.

El framework debería de proveer librerías y herramientas a los desarrolladores de software para crear sus propias aplicaciones en distintos lenguajes de programación modernos y distintos sistemas operativos. Es importante su compatibilidad con Java, con el fin de facilitar de su integración con las otras capas, ya que la mayoría de frameworks de utilizados a alto nivel están implementados en Java.

Por otra parte, debe permitir y facilitar el uso de otras librerías especializadas como OpenCV o CUDA, ya que el rendimiento y la reutilización son aspectos claves en este nivel.

También proporciona herramientas y bibliotecas para obtener, construir, escribir y ejecutar código en varios ordenadores. La parte más difícil de cumplir aquí será la de la ejecución distribuida en módulos que se diseñen individualmente y se acoplen libremente en tiempo de ejecución.

Otra parte importante es la del soporte para comunicaciones facilitando un sistema de transporte, lenguajes y protocolos sencillos, versátiles y eficientes. Debería de incorporar una semántica de publicación/suscripción.

Y para finalizar debería de ofrecer los elementos necesarios para gestionar la plataforma, con el fin de llevar un control de los elementos, permitir la búsqueda de elementos y los servicios ofrecidos por estos, así como los mecanismos que den robustez y tolerancia a fallos.

3.5 Contextos de Aplicación.

Uno de nuestros objetivos era proponer una arquitectura que fuese versátil pudiéndose aplicar a distintos contextos. Vamos a exponer 3 contextos, en este capítulo, los cuales hemos modelado con nuestra arquitectura. Haremos una breve descripción de los requerimientos y mostraremos la arquitectura resultante con el fin de facilitar la comprensión y la aplicación práctica de la arquitectura propuesta. En los dos siguientes capítulos aplicaremos de manera detallada la metodología sobre 2 de ellos, con el fin de ver toda la propuesta de la tesis de una manera completa.

El primero sobre el cual vamos a validar la arquitectura es un contexto con gestión centralizada, el de un hotel inteligente. Una de las metas de este proyecto era la integración de elementos IoT (Internet de las Cosas) de una forma estándar, facilitando la integración de nuevos elementos. Otro de los objetivos de este problema era mejorar el comportamiento de ahorro energético de los distintos elementos que componen el hotel.

El segundo contexto escogido ha sido un problema con gestión descentralizada de detección de señales RF a través de un enjambre robótico de drones. En este caso vemos la aplicación de la arquitectura a problemas de robótica y su integración con simuladores.

Por último, vamos a aplicar la arquitectura a un contexto totalmente distinto, el de la creación de una herramienta tecnológica que nos permita medir y registrar movimiento de hiperactividad de forma objetiva sobre estudiantes. Ese último caso lo extenderemos un poco más al no tratarse en los capítulos siguientes.

3.5.1 Aplicación de la Arquitectura sobre un Problema de Ahorro Energético en un Hotel Inteligente.

Los hoteles consumen energía eléctrica y algún combustible, generalmente gas o gasóleo. La energía eléctrica es consumida por el aire acondicionado y bombas de calor, las bombas de presión de agua, el alumbrado, los ascensores y los electrodomésticos de cocinas, restaurante y lavandería. El consumo con gas y/o gasóleo viene dado por el calentamiento de agua para calefacción (cuando es por radiadores) y para la producción de agua caliente sanitaria, para los fogones de la cocina y para la calefacción de la piscina.

Además del consumo, cada día es más importante la generación de energía propia. Esta generación no solamente aporta reducciones en la factura, sino que colabora a mantener el medio ambiente. El elemento principal de generación son las placas solares, las cuales se utilizan en dos vertientes: el calentamiento de agua corriente sanitaria (ACS) y la generación de energía a través de placas fotovoltaicas. Las placas solares suelen instalarse en una posición fija, atendiendo a la altitud y latitud. El rendimiento de estas placas se puede mejorar haciendo el sistema móvil, pero para maximizarlo debemos de contar con información astronómica, ya que la posición del sol depende de la ubicación, el día y la hora.

Los hoteles contar con un sistema de reservas y mantenimiento centralizado. Para el tema del ahorro se utilizan sistemas independientes, en la mayoría de casos muy básicos (controlados manualmente o con simples programaciones de tiempo) y autónomos para cada zona.

Se quiere modelar un sistema multiagente que nos ayude a controlar el gasto energético desde las perspectivas de consumo, confort, seguridad y mantenimiento. La idea es buscar una arquitectura que permita integrar soluciones de bajo nivel con sistemas multiagente de alto nivel, facilitando la integración y desarrollo de sistemas de bajo nivel y aportando mejoras desde el punto de vista de estandarización e implementación.

En la Ilustración 43 podemos ver la aplicación de nuestra arquitectura sobre para el problema del establecimiento hotelero donde queremos implantar el sistema multiagente para mejorar el control energético.

Tal como marca la propuesta tenemos 3 niveles. El nivel más alto lo componen el sistema multiagente propiamente dicho. A bajo nivel tenemos los elementos encargados de la obtención de datos (temperatura, humedad, presencia, etc.) y actuación (reguladores, accionadores, etc.). El nivel central se ha utilizado para comunicar los dos niveles y para dividir los planes en grupos.

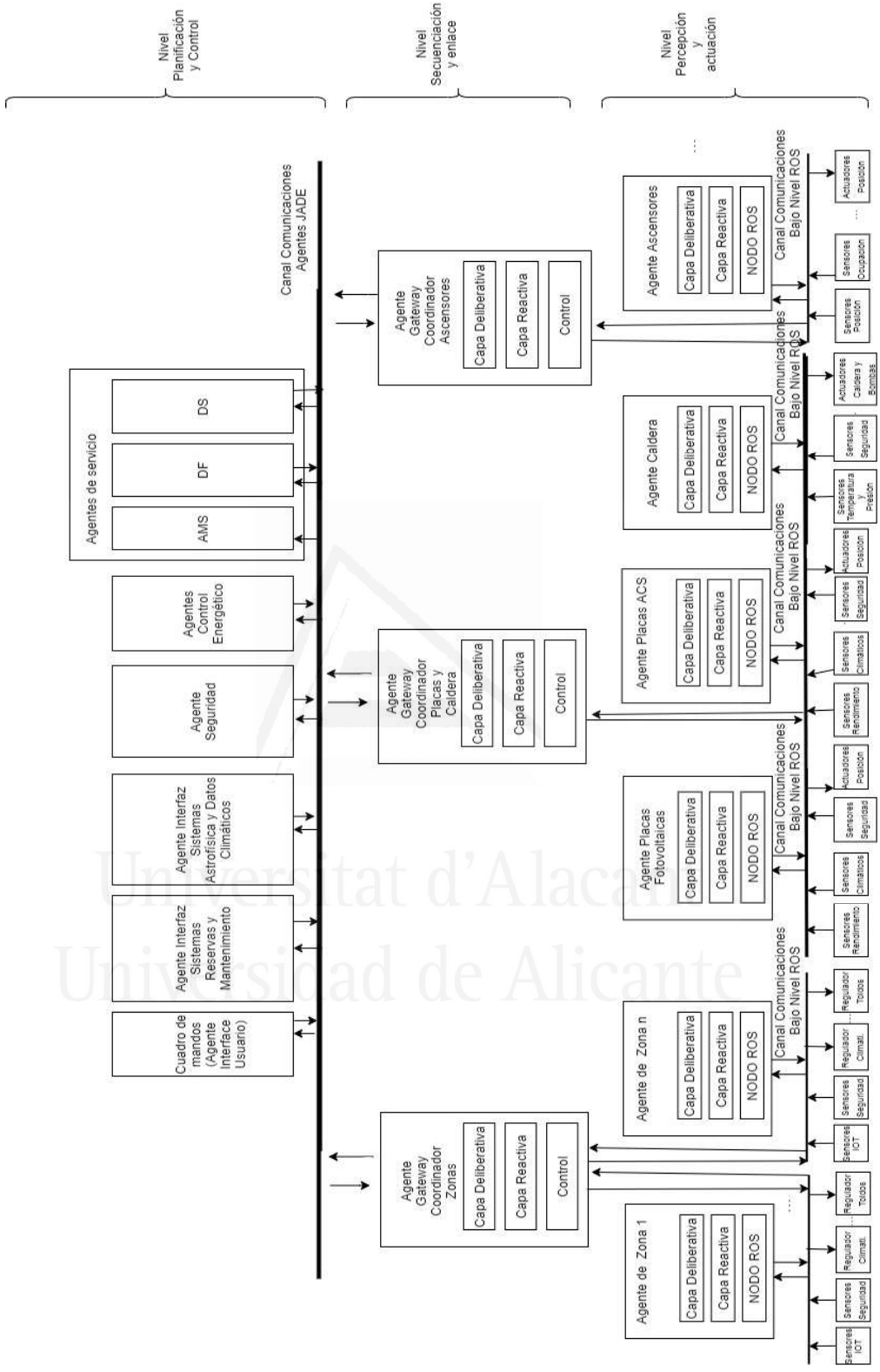


Ilustración 43. Arquitectura Híbrida Multinivel aplicada a Establecimiento Hotelero.

A alto nivel nos encontramos los agentes encargados de mantener la plataforma multiagente; Un agente que va desempeñar el rol de cuadro de mandos, desde el cual los controladores podrán supervisar la gestión de los distintos elementos del hotel; los agentes de interfaz usados para comunicar con los sistemas externos de reservas, mantenimiento, datos astrofísicos y climáticos; un agente cuya misión es velar por la seguridad en el hotel, seguridad desde el punto de vista de presencia de personas en zonas donde tenga permisos y seguridad entendida como el buen funcionamiento de los elementos; y para terminar de describir este nivel tendríamos el agente de control energético que es el encargado de coordinar al sistema multiagente.

En el nivel intermedio encontramos agentes de secuenciación y comunicación, los cuales agrupan a los elementos de bajo nivel para facilitar la planificación de tareas y comunican a los elementos de alto y bajo nivel. El agente coordinador de zonas coordinará el estado y las acciones sobre las estancias (salas comunes, habitaciones, etc.). El coordinador de placas solares y calderas gestionará tanto las placas solares fotovoltaicas como las placas ACS y la caldera. La gestión de las placas solares se ha agrupado por su similitud en datos necesarios y comportamiento, mientras que la caldera tiene un funcionamiento muy estrecho con las placas solares ACS. Para completar este nivel intermedio contamos con un agente encargado del sistema de ascensores, coordinando el funcionamiento de los distintos ascensores.

En el nivel más bajo tenemos los microagentes. Entre ellos tenemos los de zona, los cuales recogerán los datos climáticos, de iluminación y de presencia de las estancias y serán los encargados de regular el aire acondicionado, la iluminación, los toldos, las persianas, las ventanas, etc. Dentro de este nivel tenemos los de gestión de placas solares los cuales permitirán regular la orientación e inclinación de las placas y facilitarán información de rendimiento, temperatura del agua precalentada, etc. Lo mismo ocurre con el agente de caldera el cual facilitará la información de temperatura y presión del agua y permitirá aumentar la temperatura de la misma. En este último grupo tenemos microagentes de ascensor que facilitarán la información de posición, dirección, número de pasajeros y será el que mande las peticiones a los motores físicos.

3.5.2 Aplicación de la Arquitectura sobre un Problema de Detección de Señales de Radiofrecuencia a través de un Enjambre Robótico.

Un campo de investigación dentro del área de la robótica es la robótica de enjambre, la cual estudia la coordinación de un número grande de individuos (robots simples) para conseguir un objetivo. Está inspirado en el comportamiento de la naturaleza, en concreto de los insectos sociales. Estos son un ejemplo de cómo un número grande de individuos simples pueden interactuar

entre sí creando crear sistemas inteligentes colectivos. Este comportamiento colectivo emerge de una forma natural y autoorganizada a partir de las relaciones entre los individuos y el entorno y las interacciones entre los individuos.

El objetivo de este enjambre robótico va a ser la localización de señales de radiofrecuencia de alta intensidad en entornos urbanos. Estas señales suelen tener mayor intensidad en el centro de las ciudades, pero existe el riesgo de saturación. Las radiaciones RF son amplificadas o minimizadas dependiendo de la orografía y de las edificaciones. Los proveedores de señales necesitan ofrecer suficiente cobertura, configurando distintas antenas, con el fin de ofrecer un servicio mínimos de calidad.

En los sistemas de enjambre artificiales, a partir del comportamiento individual, la inteligencia emerge de forma natural, dando paso al comportamiento global del enjambre. Las tareas que lleva a cabo de manera global es denominada comportamiento macroscópico, mientras que el de los individuos es llamado comportamiento microscópico.

Los sistemas robóticos de enjambre deben cumplir una serie de características que los hacen diferentes de otros sistemas multirobóticos, como son el control descentralizado, el trabajar sólo con información local, la comunicación limitada entre agentes, la robustez y la emergencia de un comportamiento global. El enjambre estará formado por un gran número de robots, los cuales serán relativamente simples, tendrán sensores locales y capacidades de comunicación limitadas.

En la Ilustración 44 podemos ver la arquitectura del sistema. En el nivel alto e tenemos los agentes que realizan el comportamiento macroscópico del enjambre, mientras que en el bajo nivel tenemos los robots físicos del sistema. En el nivel intermedio tenemos agentes los agentes de secuenciación que se encargan de dirigir a pequeños grupos de robots.

A alto nivel, a parte de los agentes de comportamiento macro, tendremos los agentes de control de la plataforma de agentes típicos, AMS, DF y DS; un agente que actuará de interface con sistemas de modelado y simulación de emisión y propagación de señales RF; y un último que servirá para que un usuario supervise e interactúe con el sistema.

Cada agente de bajo nivel (microagente) estará asociado a un robot. En la práctica el software de microagente estará empotrado en cada uno de los robots. Cada uno de ellos recibirá señales del entorno a través de sus sensores y controlará los actuadores para moverse a través del entorno. Los robots tendrán un comportamiento muy sencillo moviéndose a través del entorno intentando localizar recursos y evitando colisiones, realizarán el comportamiento microscópico del enjambre.

Hacer notar al estar los robots organizados en pequeños grupos con canales de comunicación diferentes, será fácil escalar el sistema.

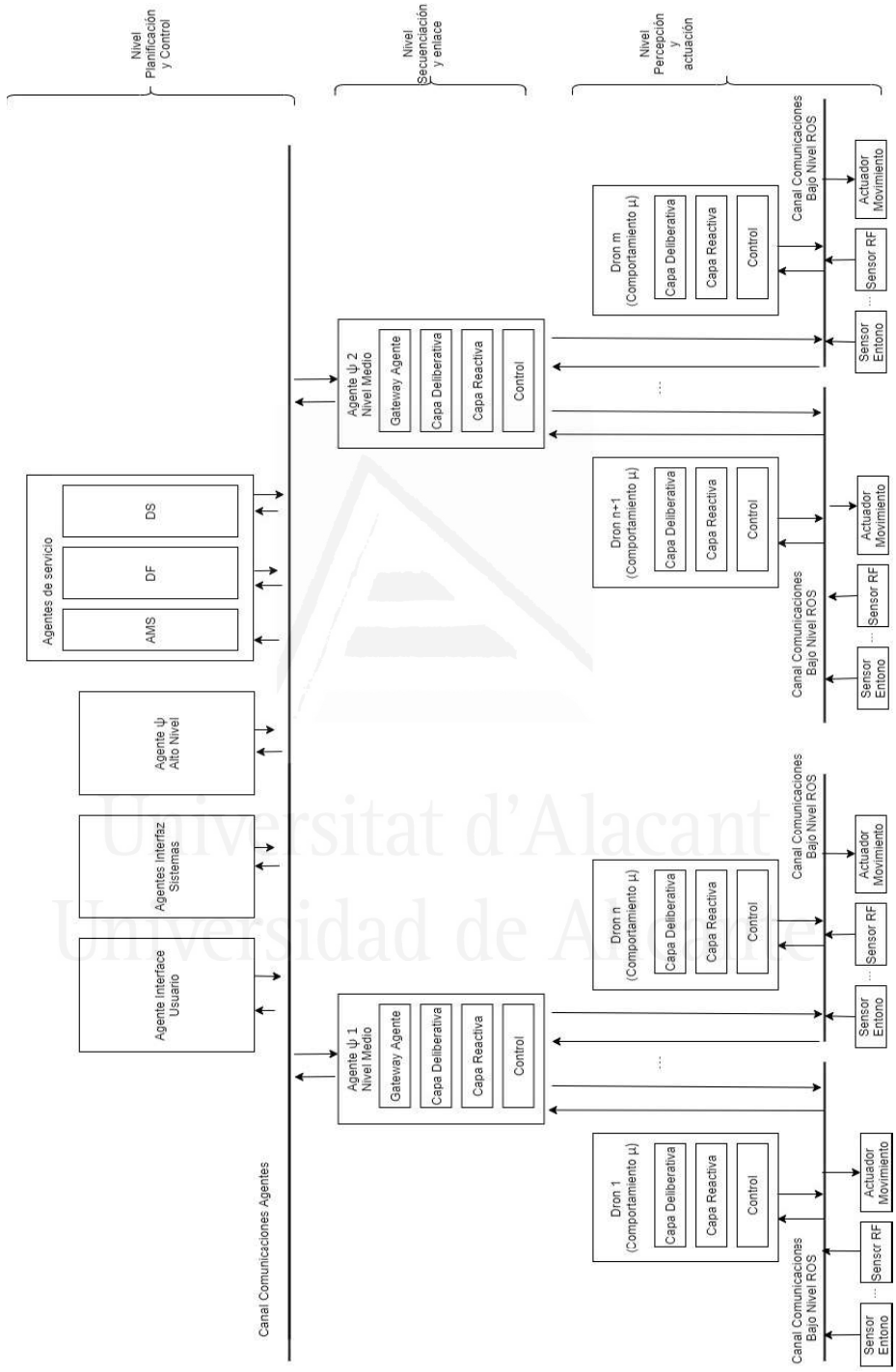


Ilustración 44. Arquitectura Híbrida Multinivel aplicada a la Detección de Señales de Radiofrecuencia a través de un Enjambre Robótico.

3.5.3 Aplicación de la Arquitectura sobre la Creación de una Herramienta Tecnológica para Medir y Registrar Movimiento de Hiperactividad.

No existe ninguna condición que determine de forma objetiva e inequívoca la existencia del trastorno por déficit de atención por hiperactividad TDAH [72] [73]. El diagnóstico viene determinado por la observación y la información que facilitan los profesores y los padres. Al no haber instrumentos, ni procedimientos de evaluación los resultados son subjetivos y dispares. Esta dificultad a determinar el diagnóstico y que uno de los síntomas principales sea la hiperactividad han provocado que en la última década se realicen estudios para registrar los movimientos de los sujetos a través de medidas objetivas.

A lo largo de los últimos 30 años se han ido aumentando las capacidades de los sistemas de información permitiendo hacer análisis de grandes volúmenes de datos aplicando sistemas estadísticos. E el caso que nos ocupa, al estudio del comportamiento humano [74].

En la actualidad se pueden utilizar tres tipos de tecnologías para registrar los movimientos: sistemas de video, sistemas de captura de movimiento basado en sensores o marcadores y mapas de profundidad o nubes de puntos. Cada una de ellas tiene unas ventajas y unos inconvenientes:

Los sistemas de video son más fáciles de utilizar, pero no mantienen el anonimato de los sujetos, al establecer manualmente el origen del sistema de coordenada ofrecen resultados distintos para un mismo video, el análisis fotograma a fotograma requiere un tiempo elevado de computación y solamente permiten trabajar en dos dimensiones.

La ventaja de los sistemas basados en sensores o marcadores es la precisión. Estos sistemas permiten capturar movimiento en tres dimensiones de forma bastante precisa. Sin embargo, tienen el inconveniente de tener que adquirir todos los sensores necesarios y lo peor tener que llevarlos sobre el cuerpo, algo que resulta artificial.

Los sistemas de detección de mapas de profundidad registran los movimientos sin tener que utilizar los sensores sobre el propio cuerpo o llevar algún tipo de traje, y pueden hacerlo en una gran variedad de entornos. Hoy en día disponemos de una gran variedad de dispositivos con diferentes precios y características.

Muchas veces estos estudios se complementan por la información recogida por un observador que va haciendo anotaciones sobre el comportamiento de los sujetos (veces que se levantan o giran en la silla, etc.).

También es necesario hacer los estudios con grupos de control que nos sirvan de referencia para realizar comparaciones y validar el estudio.

Entre los dispositivos que utilizan la tecnología de mapas de profundidad, encontramos a Microsoft Kinect. Es un dispositivo interesante en primer lugar, porque es capaz de hacer el seguimiento de 6 cuerpos diferentes. En segundo lugar mantiene el anonimato al no facilitar la información en formato de video. Y en tercer lugar, porque es un dispositivo económico y pequeño que apenas requiere instalación y se puede utilizar fácilmente en el entorno natural Los datos que nos ofrece el sensor Kinect incluyen información de medidas de profundidad, las coordenadas (x, y, z) de los objetos presentes en la escena y en concreto para un cuerpo humano, ofrece de cada uno de los esqueletos detectados, las coordenadas de cada una de las 25 articulaciones (Ilustración 45).

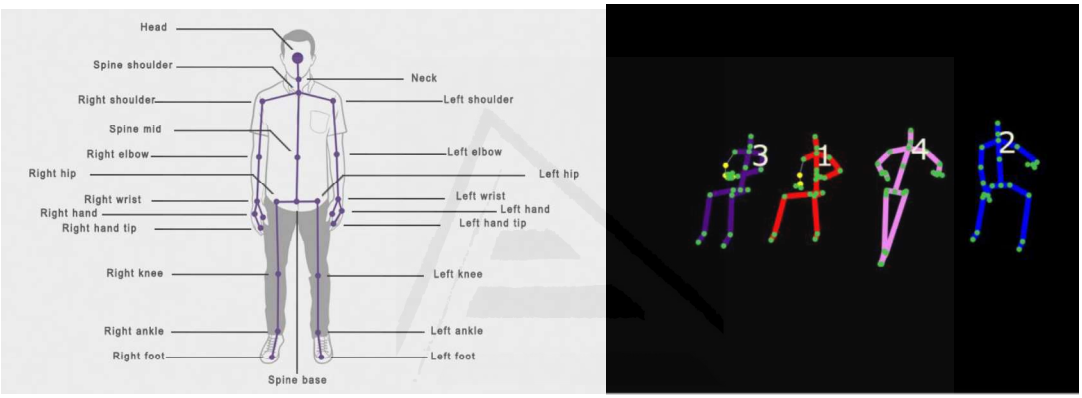


Ilustración 45. Segmentos detectados por Microsoft Kinect y representados en el módulo de supervisión [75].

En la Ilustración 46 podemos ver el problema de detección de TDAH modelado con nuestra arquitectura.

En el nivel superior tenemos los agentes de información, estadísticos y de mantenimiento de la plataforma y de segmentación de TDAH. Este último agente será el corazón del sistema ya que será el encargado de clasificar los distintos comportamientos utilizando el conjunto de datos recogidos y haciendo las valoraciones de los resultados estadísticos.

La capa intermedia está compuesta por agentes de monitorización de grupos. Por cada grupo habrá un agente de este tipo, el cual recopilará la información de cada individuo tanto automatizada como supervisada y la mandará la información hacía el nivel superior. Este agente intermedio estima la distancia recorrida por cada articulación del esqueleto durante cada x segundos de la sesión para cada uno de los individuos.

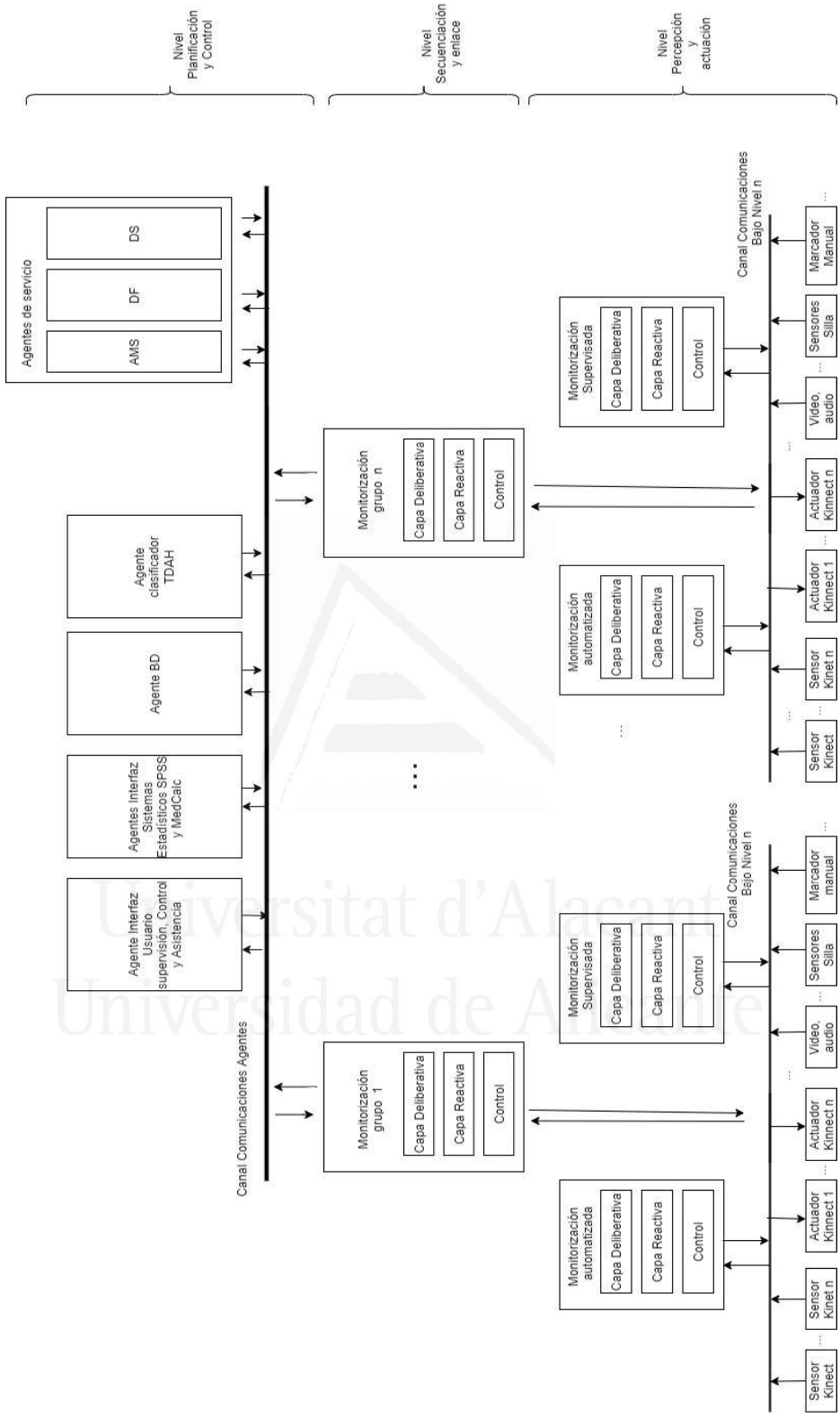


Ilustración 46. Arquitectura Híbrida Multinivel aplicada a la Detección de ADTH

A bajo nivel tenemos toda la recogida de datos. Para cada grupo vamos a tener microagente especializado en el manejo de los dispositivos Kinect. Su función es recibir la información de segmentos de los distintos individuos que detecta Kinect y pasarlos a su agente de monitorización de grupo. Aparte de este microagente, cada grupo tendrá otro microagente cuya función será la supervisión del grupo. En concreto recibirá video/audio, información de unos sensores en las sillas que indican el comportamiento de los asistentes y una información que introducirá manualmente un supervisor.

A diferencia de las otras aplicaciones, para este problema hemos diseñado un único canal de comunicación a bajo nivel debido a dos motivos fundamentales: El ancho de banda es suficiente y la información de video y video es captada y compartida en local por los dispositivos Kinect.

3.6 Resumen del Capítulo.

En este capítulo hemos hecho la propuesta de nuestra arquitectura y de una metodología para modelar y construir sistemas multiagente.

La arquitectura propuesta es una arquitectura híbrida con 3 niveles y dos capas de comunicación claramente diferenciados. El nivel superior será el encargado de la coordinación, encontrándose en ella los agentes deliberativos, de planificación y de control. El nivel inferior se encargará de la percepción y actuación, en pocas palabras es donde centralizaremos el uso de sensores y actuadores. Por último, el nivel intermedio actúa como Gateway entre la capa de alto y la de bajo nivel, además realiza funciones de secuenciación dividiendo los planes de actuación de la capa superior en planes más concretos distribuyéndolos entre los agentes del nivel inferior. El hecho de modelar el sistema con dos capas de comunicación diferenciadas es clave, ya que cada una de ellas tiene requisitos distintos en cuando a estructura y rendimiento.

El resultado es una arquitectura versátil, flexible y escalable. Esta arquitectura simplifica el diseño de los sistemas multiagente y permite construir los sistemas de una manera rápida y fiable utilizando los estándares que nos proporciona el mercado.

Acompañamos la propuesta de arquitectura con una metodología ágil e iterativa, la cual hemos estructurado en 5 fases y cada una de ellas en etapas, la cual nos permitirá el diseñar y desarrollar el sistema multiagente. Las fases son las siguientes: análisis de requerimientos del sistema, diseño de sociedad de agentes, diseño de implementación de agentes, codificación y despliegue Dentro de cada una de ellas indicaremos los artefactos que deberemos de ir generando

hasta completar la producción del sistema, siempre siguiendo los estándares disponibles, principalmente UML.

En este capítulo también hemos hablado de las características que deberán de tener las herramientas y los frameworks de desarrollo de bajo nivel, ya que generalmente en la literatura de sistemas multiagente se trata con los elementos de alto nivel, dejando a los desarrolladores el diseño e implementación de los elementos de bajo nivel.

Para terminar el capítulo, para probar la validez y flexibilidad de nuestra arquitectura hemos modelado 3 problemas aplicándola. El primero era un problema con gestión centralizada, el de un hotel inteligente, donde se quiere mejorar el comportamiento de ahorro energético e integrar elementos IoT de bajo coste. El segundo problema ha sido el de un problema con gestión descentralizada de detección de señales RF a través de un enjambre robótico de drones. Y por último, aplicamos la arquitectura a un problema totalmente distinto, el de la creación de una herramienta tecnológica que nos permita medir y registrar movimiento de hiperactividad (ADTH) de forma objetiva sobre estudiantes.

4 Modelado de un Hotel Inteligente para el Ahorro de Energía

En este capítulo, vamos a modelar el sistema de control energético de un establecimiento hotelero con la arquitectura y metodología propuestas en esta tesis. Este sistema multiagente nos ayudará a controlar el gasto energético desde las perspectivas de consumo, confort, seguridad y mantenimiento. La idea clave es demostrar como con nuestro modelo se pueden integrar soluciones de bajo nivel con sistemas multiagente de alto nivel. Aplicaremos la metodología propuesta para crear el sistema, viendo el aporte de la tesis desde el punto de vista de arquitectura y metodología, facilitando la integración y desarrollo de los sistemas IoT a bajo nivel y aportando mejoras desde el punto de vista de estandarización e implementación de sistemas inteligentes de alto nivel. Para finalizar el capítulo hablaremos de los objetivos conseguidos.

4.1 Análisis de Requerimientos.

4.1.1 Descripción del Dominio.

El objetivo es implementar un sistema que ayude a controlar el gasto energético de un hotel, nos ayude a aumentar el confort y la seguridad del mismo, así como ayudar en las labores de planificación sobre el mantenimiento del mismo. Para ello vamos a tener un sistema centralizado de control y planificación sobre todo el hotel, mientras que las instalaciones constarán de un sistema básico, pero con autonomía propia en cada habitación.

El consumo energético de un hotel por una parte es en forma de energía eléctrica. Esta es consumida por el alumbrado, los ascensores, el bombeo de agua, el aire acondicionado, los electrodomésticos de las cocinas, restaurante, lavandería, etc. Cada vez también es más habitual la implantación de bombas de calor eléctricas para climatizar las estancias en los meses de frío. Los hoteles también consumen otro tipo de combustible, generalmente gas o gasóleo para la producción de agua caliente para calefacción o para la producción de agua caliente sanitaria (ACS), para la calefacción de la piscina cubierta y también para el suministro de los fogones de la cocina.

Un estudio de la agencia valenciana de energía [76] muestra datos de consumo de establecimientos hoteleros desde varias perspectivas. Podemos ver en la Ilustración 47 como existe un ligero predominio del consumo eléctrico frente al térmico. Sin embargo, vemos que si el estudio lo hacemos atendiendo al coste (Ilustración 48) el porcentaje de gasto eléctrico es bastante mayor que el térmico.

Ilustración 47. Distribución de consumo energético [76].

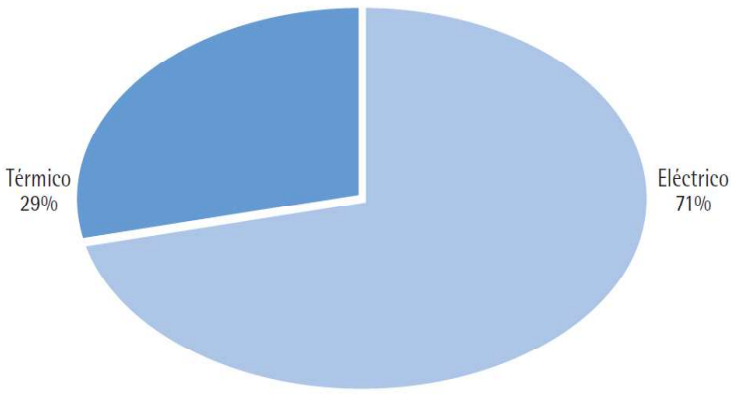


Ilustración 48. Distribución de costes energéticos [76].

El estudio de distribución de consumo energético en establecimientos hoteleros es difícil ya que el tipo de establecimientos es muy heterogéneo, teniendo distintas categorías, número de habitaciones, categoría, ubicación geográfica y fuentes de energía utilizadas. Dependiendo de estas características varían también los servicios ofrecidos complicando este estudio de distribución. A pesar de estas dificultades, de manera indicativa se consigue mostrar una distribución típica para un establecimiento hotelero de la Comunidad Valenciana (Ilustración 49).

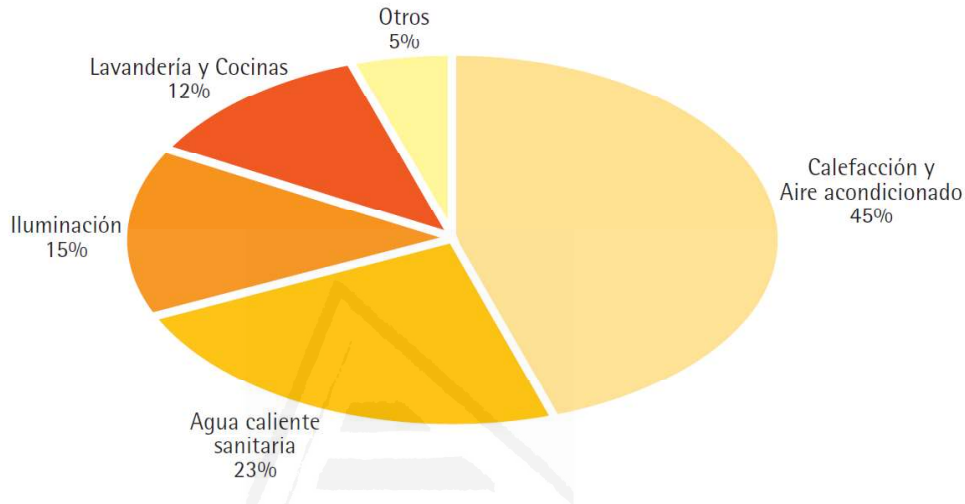


Ilustración 49. Distribución de consumo energético [76].

La partida de calefacción y aire acondicionado, es la principal consumidora de la energía de un hotel. Si queremos reducir el consumo tendremos que dirigir nuestros esfuerzos principalmente a esta partida.

Uno de los objetivos principales de un hotel es los huéspedes descansen y sea confortable. Crear un ambiente agradable requerirá el consumo de energía en distintas formas, pero no siempre un mayor consumo se traslada a un mayor confort. Habrá que buscar el equilibrio entre consumo y confort.

4.1.1.1 Calefacción y Aire Acondicionado.

Como hemos mencionado antes los sistemas de calefacción y climatización generalmente son los principales consumidores del consumo energético de un hotel. Esto, junto con el incremento de los costes energéticos, ha hecho que en el diseño de nuevos hoteles sean considerados aspectos desde la óptica energética, buscando combinar el ahorro energético con factores estéticos y de confort.

Podemos definir el confort como la sensación equilibrada entre temperatura, humedad, ruido, calidad y velocidad del aire. El

acondicionamiento térmico tendrá que buscar este equilibrio en función de la ocupación y de la actividad a desarrollar en la zona a climatizar.

Una primera medida que podemos tomar es en invierno reducir las pérdidas de calor y en verano las ganancias de calor, disminuyendo de este modo la energía necesaria para acondicionar térmicamente el edificio. Muchas de estas pérdidas dependen de las características constructivas del edificio (edificación, orientación, etc.).

Otro aspecto para evitar ganancias de calor, muchas veces olvidado hoy, es la existencia de protecciones solares (exteriores e interiores). Existen diferentes tipos de protecciones, siendo más recomendables unas u otras dependiendo de la orientación. Cuando la orientación es Sur son más adecuadas las protecciones semifijas o fijas. Sin embargo, cuando la orientación es Oeste o Noreste se recomiendan las protecciones con lamas móviles, ya sean horizontales o verticales. No hay que perder el punto de vista del confort, recomendando para orientaciones Este u Oeste las protecciones móviles, para poder disfrutar tanto del amanecer como del atardecer, así como la entrada de la luz solar en época frías o templadas.

La regulación y el control de la calefacción y el aire acondicionado es otra mejora importante para reducir la demanda de energía, permitiendo controlar el modo de funcionamiento según las necesidades específicas del momento y el lugar.

La sectorización por zonas nos permite ahorrar entre un 20 y un 30% de la energía de este apartado. Para ello contamos con el uso de sistemas individuales para el control de la temperatura en cada habitación o zona, la regulación de la velocidad de funcionamiento de los ventiladores o la regulación de las bombas de agua.

Otra medida que podemos emplear es el uso de la información que nos facilitan los sistemas de gestión centralizada. Estos nos permitirán un control de la temperatura en función del estado de la habitación (desocupada, reservada u ocupada). Con esta información se pueden adecuar los parámetros de temperatura y humedad, desde el momento de la reserva y manteniendo los equipos en espera hasta que la habitación se ocupa. Cuando el cliente no esté en la habitación, el sistema podría entrar de nuevo en espera. La temperatura desde el modo espera puede llevarse en pocos minutos y con un bajo coste a la temperatura de confort.

El consumo energético disminuye entre un 5-7% por cada grado que se disminuye la temperatura ambiente. De esta forma obtenemos importantes ahorros entre 20-30% del consumo de calefacción.

La introducción de aire exterior atendiendo a la ocupación de la sala, y a calidad y temperatura del aire también es una buena técnica de ahorro

energético. Es interesante la instalación de un sistema de freecooling, que aprovecha de forma gratuita el aire exterior para refrigerar la estancia, siempre y cuando las condiciones así lo permitan, consiguiendo de esta forma importantes ahorros energéticos.

En las piscinas cubiertas, la bomba de calor juega un buen papel, ya que reduce el caudal de ventilación necesario, y favoreciendo el ahorro energético. Cuando se utilizan calderas, en invierno, se requiere renovar el aire para evitar una excesiva humedad en el ambiente. Mediante la bomba de calor, el aire húmedo se enfría en el evaporador, produciendo la condensación del exceso de humedad del aire. El aire frío y seco se calienta en el condensador y es de nuevo introducido en el recinto de la piscina. El calor excedente de la bomba se puede utilizar para el calentamiento del agua del vaso o para la calefacción de los vestuarios o duchas.

Nuestro sistema recibirá información de sensores de temperatura, humedad, apertura de ventanas y puertas, ocupación y uso de las instancias, y actuará sobre los reguladores de ventiladores, bombas de calor, sistemas de free-cooling y persianas y toldos.

4.1.1.2 Agua Caliente Sanitaria.

El consumo derivado del agua caliente sanitaria (ACS) representa una parte significativa del consumo energético del hotel. Dependiendo de la categoría del hotel tenemos unos valores entre un 15 y un 25% del consumo total de energía del hotel.

Para producir el ACS se utilizan generalmente calderas, por lo que en este apartado hablaremos de mejoras para este tipo de aparatos. También hay que tener en cuenta que la temperatura de almacenamiento no sea inferior a 60ºC para evitar riegos de legionela.

El uso de sistemas de bajo consumo en baños y duchas, que reducen el caudal suministrado sin perjuicio de la calidad del suministro, además de ahorrar agua, también aportan ahorros energéticos al disminuir el caudal de agua a calentar. El valor de ahorro en consumo de agua puede estar en algunos equipos en un rango del 50-60%. Las válvulas termostáticas, que limitan y regulan la temperatura del ACS, son otra medida de ahorro energético.

En los últimos años se ha implantado una mejora importante: la producción ACS utilizando la energía solar, obteniendo unos ahorros notables en el consumo energético y una mejora medioambiental al reducir las emisiones de CO2. El posicionamiento (orientación e inclinación) de las placas solares, así como las sombras generadas por elementos en cercanos va a afectar al rendimiento de las placas. El rendimiento del panel es mayor cuanto más se inclina el panel perpendicularmente a los rayos del sol. La máxima captación de energía se conseguirá al mediodía, cuando el sol alcanza su máxima altura en el horizonte.

Por tanto necesitamos saber, durante todo el año, la altura del sol para saber cuánto inclinar nuestros paneles. Si por ejemplo estamos en Madrid, la latitud será de 40º, los paneles deberán de tener una inclinación entre 17º en verano y 63º en invierno. En la Ilustración 50 podemos ver las pérdidas que tenemos en función de la orientación e inclinación.

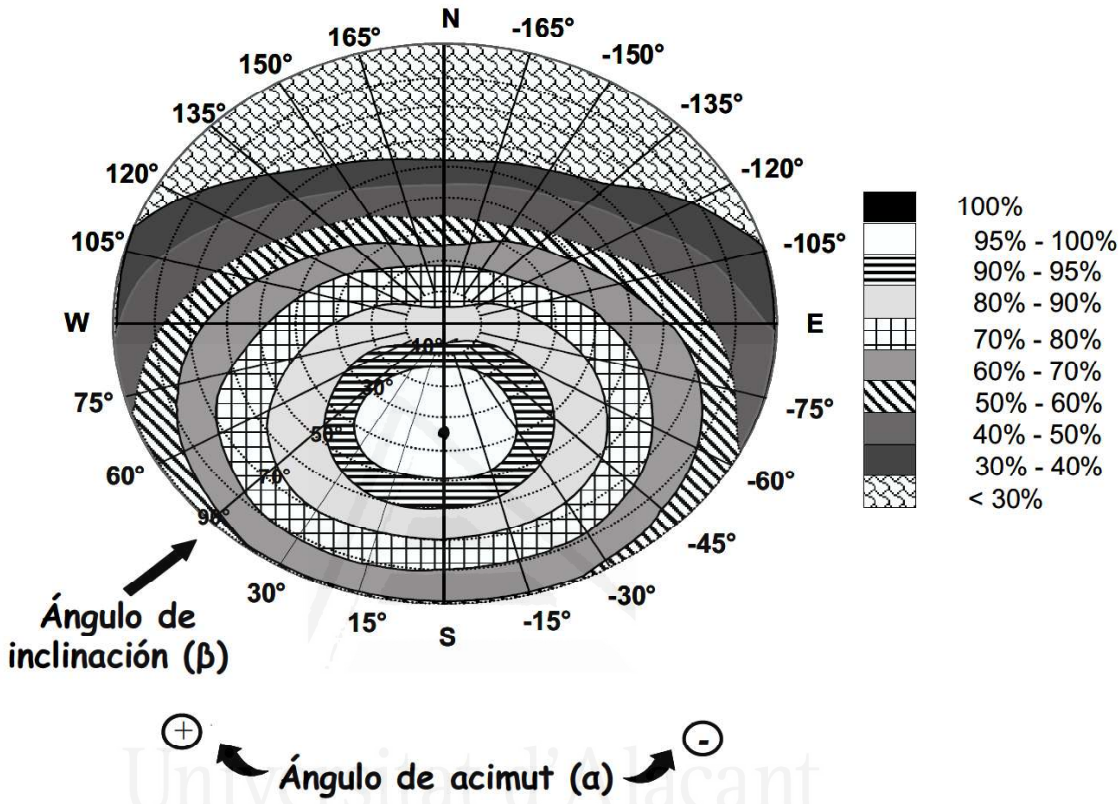


Ilustración 50. Porcentaje de energía respecto al máximo como consecuencia de las pérdidas por orientación e inclinación. [77]

En nuestra Comunidad el agua es un bien escaso y los hoteles son grandes consumidores de agua. Es todavía más importante minimizar su consumo, pero siempre sin que suponga un perjuicio en los servicios que presta el hotel.

Como hemos mencionado antes la disminución del consumo de agua redunda también en un importante ahorro energético debido a la disminución del combustible necesario para su calentamiento.

Otro factor que no debemos de olvidar son las pérdidas de agua en la instalación, ya que además de un mayor consumo de agua, provocan un mayor consumo de los equipos de bombeo al aumentar sus horas de funcionamiento. Esto se puede evitar a través de técnicas para la detección de fugas como el

reconocimiento riguroso de toda la instalación, la instalación de manómetros para detectar puntos de caída de presión, la utilización de equipos electro-acústicos para la detección de estas pérdidas, y el registro y estudio periódico de los consumos de agua.

En este sentido nuestro sistema llevará un control de gestión sobre las calderas, consumos atendiendo a la ocupación y demanda de las instalaciones. Así mismo llevará un control de averías y fugas. Desde el punto de vista de aprovechamiento solar el sistema podría actuar sobre la orientación e inclinación de las placas solares en conjunción con datos provenientes del sistema astrofísico y la información de rendimiento de sensores en la instalación concreta. Esta última medida no sólo mejoraría el aprovechamiento solar térmico, sino que también mejoraría el rendimiento de la energía fotovoltaica utilizada para consumo eléctrico.

1.1.1.1 Iluminación.

Dentro de un hotel, la iluminación es el apartado que representa un mayor consumo eléctrico. Dependiendo del tamaño del hotel, del uso principal a que se destina el hotel, y del clima de la región donde está ubicado, este consumo puede oscilar entre un 12% y un 18% del consumo energía total, y sobre un 40% del consumo de electricidad. Por esta razón cualquier medida de ahorro en iluminación tendrá un impacto importante en los costes de funcionamiento del hotel.

Utilizando componentes más eficientes, usando sistemas de control, integrando la luz natural y realizando un correcto mantenimiento, se estima que podrían lograrse reducciones de entre el 30% y el 50% en el consumo de iluminación.

Este ahorro puede generar un ahorro adicional en climatización, debido a que la iluminación de bajo consumo energético genera una menor emisión de calor.

Como en otros apartados, el ahorro energético no debe estar reñido con la calidad del servicio y los sistemas de iluminación del hotel deben proporcionar los niveles adecuados para cada actividad, creando ambientes agradables y confortables. Para conseguir esta meta se recomienda seguir las recomendaciones del Comité Español de Iluminación (CEI) sobre iluminación en hoteles.

Un buen sistema de control de alumbrado debe proporcionar una iluminación de calidad sólo cuando es necesario, para ello puede combinar sistemas de control de tiempo, sistemas de aprovechamiento de la luz natural y sistemas de control de ocupación.

Los sistemas de control de tiempo hacen funcionar la luminaria siguiendo un horario establecido, recomendándose la instalación de interruptores horarios en jardines y exteriores y de zonas comunes con horarios de uso limitados.

Por otro lado, los sistemas de control de la ocupación permiten, mediante detectores de presencia, el encendido y el apagado de la iluminación. Estos sistemas se suelen encontrar en zonas de servicio, servicios, en los pasillos de acceso a las habitaciones, etc.

La instalación de fotocélulas es usada para desconectar, conectar o regular la iluminación cuando la luz natural es suficiente, pudiéndose aplicar tanto a la iluminación interior como a la exterior.

Muchas veces unos simples interruptores que nos permitan la desconexión por zonas nos permiten ahorros importantes sobre todo cuando sólo es preciso en una pequeña parte de una gran zona.

Con estas sencillas medidas de control, con una inversión moderada, se pueden obtener ahorros energéticos entorno al 10% del consumo eléctrico en iluminación.

Otro detalle a tener en cuenta es la suciedad. Esta se acumula en las luminarias y en las paredes de los recintos, por lo que la luz emitida por las lámparas decrece o es absorbida por las paredes.

Nuestro sistema va a centrarse en la mejora a través del control de los elementos de nuestras instalaciones, optimizando su uso y aprovechando la luz natural. Conjugaremos sensores de iluminación, de ocupación e información astrofísica para actuar sobre reguladores/interruptores de iluminación, persianas y toldos. También podemos programar comportamientos de nuestros sistemas atendiendo a las tareas que vayamos a realizar, por ejemplo, para un comedor podríamos tener los modos: desayuno, comida, cena, limpieza o sin actividad; o en una habitación los modos (desocupada, ver tv, leer, dormir, ducha, etc…). Por otro lado, nuestro sistema ayudará en el sistema de planificación de mantenimiento de las instalaciones.

4.1.1.3 Ascensores.

Los ascensores son una partida pequeña dentro del consumo eléctrico de un hotel. Generalmente no es un criterio predominante en su elección la eficiencia energética.

Los arranques de los ascensores provocan la mayor parte del consumo de los mismos, siendo de tres a cuatro veces mayor que el valor de la potencia nominal. Por esta razón el funcionamiento del sistema de ascensores será un buen aliado para reducir su consumo energético.

Encontramos en los ascensores varios modos de funcionamiento:

• Modo “taxi”: donde no hay ninguna regulación y el ascensor va directamente desde el piso de partida al destino final seleccionado por un usuario, sin ninguna parada. Este modo presenta muy mala eficiencia energética y hoy en día nunca se utiliza.

• Modo “autobús”: el ascensor va parando en cada piso desde dónde es llamado, tanto cuando sube y como cuando baja.

• Modo mixto: En una dirección, el ascensor realiza una parada en cada piso desde

dónde es llamado, y no realiza ninguna parada en la otra dirección.

Cuando hay varios ascensores funcionando conjuntamente, es posible utilizar un sistema de control combinando los diferentes modos de funcionamiento. En nuestro caso podemos ajustar su comportamiento atendiendo a momentos concretos del día, a temporadas o a eventos.

4.1.1.4 El Internet de las Cosas (IoT: Internet of Things).

Podemos definir IoT como el “Concepto que se refiere a la interconexión de objetos cotidianos con Internet” [78] . Donde entendemos una “cosa” (Thing) como cualquier elemento de nuestras vidas cotidianas que hemos conectado a Internet, como por ejemplo, una pulsera inteligente con monitorización de frecuencia cardiaca, o una bombilla, enchufe o una cerradura inteligente.

Cuando hablamos de IoT no nos referimos a una tecnología, sino de un conjunto de soluciones y tecnologías hardware como software, donde los dispositivos se caracterizan por estar conectados a la red. Los dispositivos IoT suelen componerse de dos elementos: los sensores, que capturan los datos del entorno como pueden ser la temperatura, humedad, etc.; y los actuadores, que realizan alguna acción modificando el entorno. Esta acción puede venir dada por una orden del usuario o partir del comportamiento (programación) del dispositivo. Dependiendo de la funcionalidad y del grado de precisión del comportamiento del dispositivo IoT este puede estar compuesto de ninguno o varios de estos elementos. Por ejemplo, una bombilla inteligente puede ajustar su intensidad o color, funcionando únicamente con las órdenes del o tener algún sensor lumínico que desencadene su aumento de intensidad cuando empiece a oscurecer.

Gracias a estos dispositivos, con una tecnología punta y unos precios baratos, podrían crear un potencial casi sin límites de los sistemas multiagente.

En este apartado vamos a hacer una pequeña introducción a las características de estos dispositivos.

La conexión de los dispositivos, sobre todo cuando lo hacemos a Internet, nos abre un mundo de posibilidades y aplicaciones. Esta conexión se puede hacer de muchas formas dependiendo de los requerimientos que tengamos:

- Conexión de dispositivo a dispositivo: mediante una red local, habitualmente a través de WIFI o Bluetooth, ambos dispositivos se comunican entre sí.
- Dispositivo a Internet: se emplea direccionamiento IP para poder lograr comunicar a los dispositivos.
- Dispositivo a Gateway a Internet: se emplea un elemento intermedio, denominado Gateway, para conectar los dispositivos y la Internet. El Gateway aparte de comunicar al elemento IoT a Internet podría procesar la información de alguna manera.

Pero esta conexión a través de redes convencionales puede acarrear una serie de problemas. Podemos dividir las soluciones existentes en dos tipos: por un lado, las redes locales (LAN) o personales (PAN) y por otro las redes con una cobertura más amplia (WAN).

Las tecnologías de LAN y PAN más importantes son:

- Wi-Fi: Es uno de los estándares más utilizados. Se ha mejorado gracias a la aparición de tecnologías de cifrado y Mesh.
- Bluetooth: se emplea normalmente para redes personales. Los dispositivos interconectados suelen utilizar el teléfono móvil como nodo central.
- Zigbee: Es una tecnología nueva que apareció para tener un bajo consumo y velocidades de transferencia alta (utiliza mallas).
- Z-Wave: similar a Zigbee, también emplea tecnología de mallas, pero suele escogerse cuando se necesita mayor potencia de señal.
- Thread: es una alternativa para redes PAN de bajo consumo que emplea IPv6.

Tecnologías WAN:

- LoRaWAN: emplea frecuencias bajas, y fue diseñada para dar soporte a numerosos dispositivos, los cuales que mandan datos a la nube en entornos abiertos.
- NarrowBand-IoT: se trata de una tecnología que surgió para soportar un gran número de dispositivos de bajo coste en entornos cerrados.

- LTE M: estándar para redes móviles para el soporte de dispositivos IoT, Esta previsto que el 5G la sustituya.
- 5G: redes rápidas y flexibles, que permitirán multitud de aplicaciones IoT.

Al hablar de IoT tenemos que tener en cuenta unas características:

- Eficiencia energética: cómo vamos a tener múltiples dispositivos va a ser crucial la eficiencia energética de los mismos.
- Heterogeneidad: IoT es una tecnología reciente y no estandarizada, existen una heterogeneidad de dispositivos y tecnologías que habrá que tener en cuenta para lograr comunicar todos los dispositivos de un mismo sistema.
- Arquitectura para IoT [79]: Cada sistema IoT es diferente, pero la base de cada arquitectura de IoT, así como su flujo de proceso de datos general son los mismos.

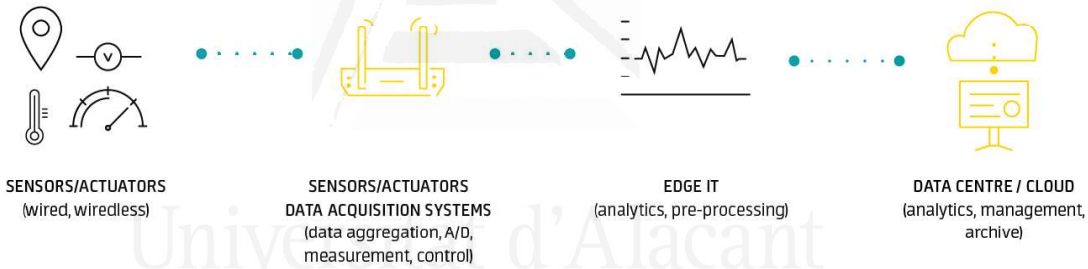


Ilustración 51. Arquitectura de IoT [79].

- Datos masivos: debido a la cantidad de dispositivos y a su trabajo casi interrumpido la cantidad de datos recopilada es gigantesca, haciendo necesario el uso de la tecnología Big Data.
- Naturaleza dinámica: cada vez se demanda un comportamiento más dinámico de los dispositivos, haciendo fundamental el uso de sus sensores.
- Escalabilidad: las aplicaciones deben de permitir aumentar el número de dispositivos fácilmente.

- Densidad de dispositivos: el uso de dispositivos IoT, su desarrollo comercial y abaratamiento está haciendo que la cantidad de dispositivos aumente a gran velocidad.
- Seguridad: La seguridad es uno de los factores donde no podrán bajar la guardia los desarrolladores de dispositivos IoT si quieren que triunfe esta corriente.
- Privacidad: el recelo de usuarios por la privacidad de los datos es otra de las trabas con las que se enfrenta el desarrollo de los dispositivos IoT.

4.1.1.5 ROS.

En el capítulo anterior veíamos la necesidad de un framework para cubrir la comunicación de los elementos de bajo nivel. Entre las características que necesitábamos era la de permitirnos abstraernos del hardware, ser rápido, eficiente y seguro y permitirnos trabajar con distintos lenguajes de programación a bajo nivel. ROS [80] (en inglés Robot Operating System) cumple con todos estos requisitos y es el framework que nosotros hemos elegido para trabajar a bajo nivel. A continuación, vamos a pasar a describir muy rápidamente este framework para el desarrollo de software para robots, el cual provee las librerías y las herramientas a los para el desarrollo de aplicaciones.

ROS, a pesar de su nombre, no es un sistema operativo, de hecho, funciona sobre Linux. Provee servicios estándar de un sistema operativo: abstracción del hardware, control de dispositivos de bajo nivel, facilidades de comunicación entre procesos, gestión de interface gráfica, etc. También nos proporciona herramientas para diseñar módulos individualmente y que se acoplen en tiempo de ejecución, incluso de forma distribuida.

ROS está basado en grafos, donde los nodos pueden recibir, mandar y multiplexar mensajes entre elementos de distintos tipos: sensores, actuadores, control, estados, planificaciones, etc. Estos nodos se apoyan en la infraestructura de ROS, pudiendo enviar y recibir mensajes al resto de nodos. Los nodos se pueden agrupar en paquetes que pueden ser compartidos con otros sistemas ROS. El concepto es similar al de plataforma en JADE.

Es independiente del lenguaje, estando actualmente implementado en C++, Python y Lisp. Es compatible con librerías como OpenCV, lo que nos posibilita implementar programas que trabajan a más bajo nivel que las herramientas JAVA, utilizadas en las plataformas de agentes.

ROS tiene dos niveles de conceptos: el nivel de Sistema de Archivos y el nivel de Gráficos de Computación.

A nivel de sistema de ficheros tenemos los conceptos que se encuentran en el disco. Entre ellos tenemos:

- Paquetes: son la unidad principal para organizar el software en ROS.
- Metapaquetes: representar un grupo de otros paquetes relacionados según su funcionalidad.
- Datos de los paquetes: son archivos (package.xml) que proporcionan los metadatos sobre un paquete (nombre, versión, descripción, licencia, dependencias, etc.).
- Tipos de mensajes (msg): aquí daremos las descripciones de los mensajes que definen las estructuras de datos para los mensajes.
- Tipos de servicio (srv): describirán los servicios definen las estructuras de datos de solicitud y respuesta.

ROS está basado en una arquitectura de grafos (Ilustración 52) basada en una arquitectura cliente-servidor. Los principales elementos de esta red de grafos son:

- Nodos: dentro de la red cada nodo es un proceso que realiza una tarea, así podemos tener varios nodos ejecutando distintas tareas. En un robot su sistema de control estará compuesto por muchos nodos: un nodo controla los motores de las ruedas, un nodo controla un láser, un nodo realiza la geolocalización, un nodo planifica las rutas, un nodo visualiza la interfaz gráfica, etc.
- Maestro: el ROS Master proporciona un registro de nombres, permitiéndonos también hacer búsquedas dentro del grafo. Sin este nodo maestro los otros nodos no podrían intercambiar mensajes o utilizar servicios. El nodo Maestro también informará a los nodos que solicitaron la información cuando esta información de registro cambie al ejecutan nuevos nodos, lo que proporciona gran dinamismo al sistema.
- Servidor de parámetros: el servidor de parámetros permite almacenar datos identificados por una clave (actualmente lo incorpora el Ros Master).
- Mensajes: la comunicación entre nodos se realiza intercambiando mensajes. Los cuales son estructuras de datos de distintos campos tipados (enteros, booleanos, float, etc.), incluso pueden incluir distintas estructuras o arrays anidados.
- Topic: los mensajes utilizan un sistema de publicación/suscripción. Para enviar un mensaje el nodo lo publicará en un topic determinado. El topic es el nombre que se utiliza para identificar el contenido de un mensaje. Para recibir los mensajes, los nodos se suscribirán al topic que les

interese. Puede haber múltiples editores y suscriptores, al mismo tiempo, para un mismo topic, y un nodo podrá publicar y/o suscribirse a múltiples topics.

- Servicios: facilitan las interacciones de solicitud/respuesta, ya que para esto el sistema de suscripciones no es válido. Los servicios se definen mediante un par de estructuras de mensajes: el de solicitud y el de respuesta (Cliente/Servidor).
- Bags: son un formato de almacenamiento de datos en ROS, los cuales permiten guardar y recuperar distintos tipos de datos.

Los nodos se registran en el nodo Maestro (ROS Master) y este les proporciona servicios de nombres y de registro al resto de los nodos. Los nodos se conectan directamente a otros nodos, el nodo maestro sólo les facilita la dirección. Cuando un nodo se suscribe a un topic, solicitará las direcciones de los nodos que publiquen en ese topic, y establecerá las conexiones oportunas. El protocolo más utilizado en ROS es TCPROS, el cual utiliza sockets TCP/IP estándar.

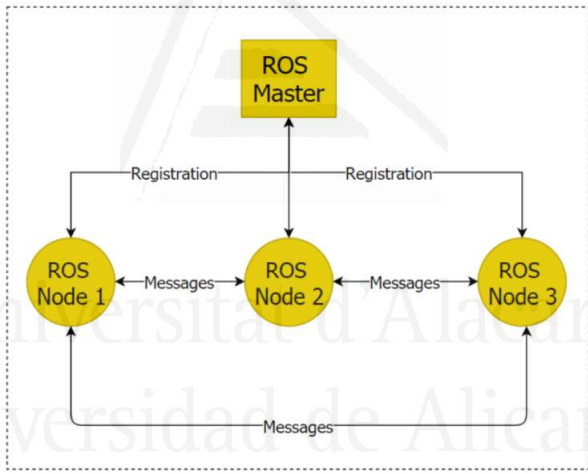


Ilustración 52. Arquitectura de conexión de ROS.

La arquitectura de conexión de ROS permite la operación desacoplada gracias al sistema de nombres permitiendo construir sistemas grandes y complejos. Incluso desde línea de comandos, los nodos de ROS admiten la redistribución de nombres permitiendo que un programa compilado pueda reconfigurarse en tiempo de ejecución.

4.1.2 Identificación de Agentes.

Uno de los primeros pasos que vamos a seguir va ser la identificación de los agentes del sistema. A continuación, vamos a explicar la Ilustración 53 donde

podemos observar los agentes de nuestro sistema, así como los sistemas externos.

Nuestro sistema se va a comunicar con tres sistemas externos: reservas, mantenimiento y astrofísica. El sistema de reservas y el sistema de mantenimiento son propios del establecimiento. En el primero de ellos se consultará la previsión de entradas y salidas de huéspedes con el fin de preparar las habitaciones para lograr un máximo confort a su llegada y una parada en el gasto energético a su salida. Nuestro sistema se comunicará con el sistema de mantenimiento para conocer las operaciones previstas, así como hacer peticiones de mantenimiento y limpieza. La comunicación de nuestro sistema con estos dos sistemas se va a centralizar en un agente (Agente de reservas y mantenimiento) que va a implementar los interfaces de comunicación de los mismos. El sistema de astrofísica es un sistema externo con el que nos vamos a comunicar para acceder a posiciones del sol atendiendo a fecha, hora, altitud y latitud. Esta información en conjunción con nuestros sensores será crucial para la orientación de las placas solares, el uso de los toldos, así como la iluminación en las distintas estancias. Siguiendo la misma estrategia que con los sistemas del establecimiento, se va a utilizar un agente (agente sistema astrofísica) como interface de comunicación entre nuestro sistema y el sistema de astrofísica.

A nivel más bajo hemos diseñado nuestro sistema con agentes específicos para gestionar las placas solares, las calderas y los ascensores; así como agentes genéricos de zona. Estos agentes de zona se encargarán de gestionar distintas estancias del hotel: habitaciones, pasillos, restaurante, gimnasio, piscina, etc. En siguientes secciones veremos el diseño interno de estos agentes, parte clave de esta tesis, para integrar los elementos de IoT.

Nuestro sistema también cuenta con un agente que vela por la seguridad en todas las áreas del establecimiento. Hemos preferido separar la parte de seguridad del agente de control para dotar de un comportamiento más reactivo y rápido al sistema en situaciones de emergencia.

Además de estos agentes contamos con los típicos agentes de páginas blancas y amarillas los cuales facilitarán el acceso a agentes y a servicios. De ellos ya hablamos en la sección 2.2 donde describíamos con detalle el standard FIPA.

A nivel de personas que actúan con nuestro sistema tenemos un operador, que va a ser la/s persona/s que gestionará y supervisará el sistema. El agente de control central será el que contenga la parte de interface de usuario con los operadores. Los huéspedes y personal del establecimiento interactuarán con nuestro sistema para pedir ajustes en iluminación, climatización, persianas y toldos. Todas están peticiones serán realizadas en los distintos agentes de zona y serán supervisadas atendiendo a permisos y programación planificada por el agente de control.

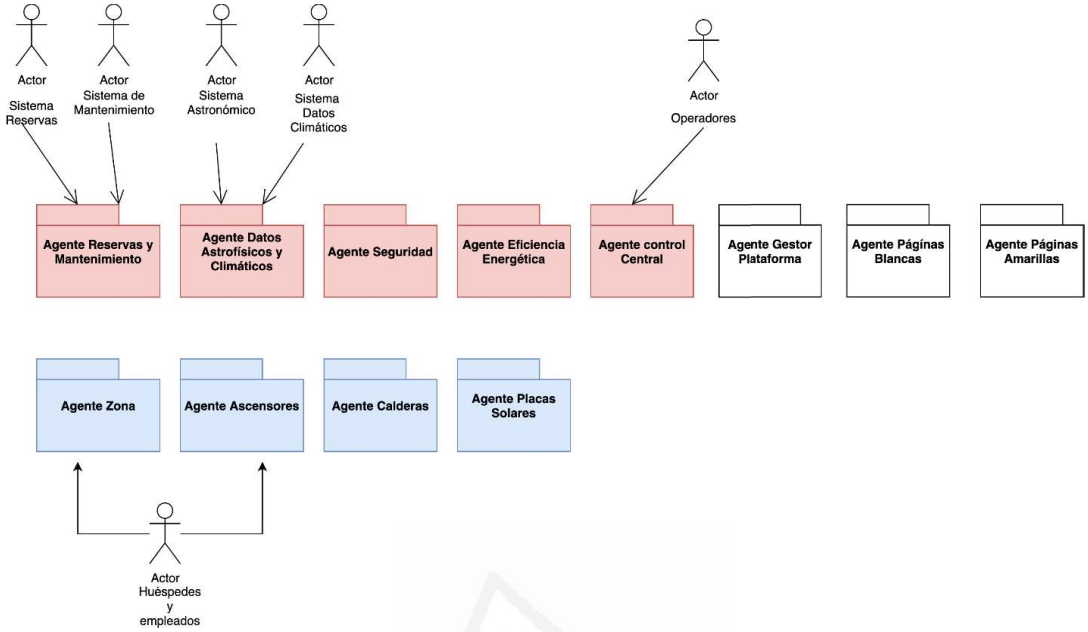


Ilustración 53. Diagrama de Agentes del Sistema de Ahorro Energético.

4.1.3 Identificación de Roles.

4.1.3.1 Rol de Control de Consumo Energético y Confort.

El rol principal de nuestro sistema y eje principal es de "control de consumo energético y confort". Como podemos ver en la Ilustración 54 este rol va a hacer uso de todos servicios del sistema: consultar reservar, gestión de mantenimiento y servicio, consultar información astrofísica, controlar la seguridad y la gestión de energía y control (a más bajo nivel). Este agente será el que haga de interfaz entre el sistema y los operadores del mismo, implementando el cuadro de mandos.

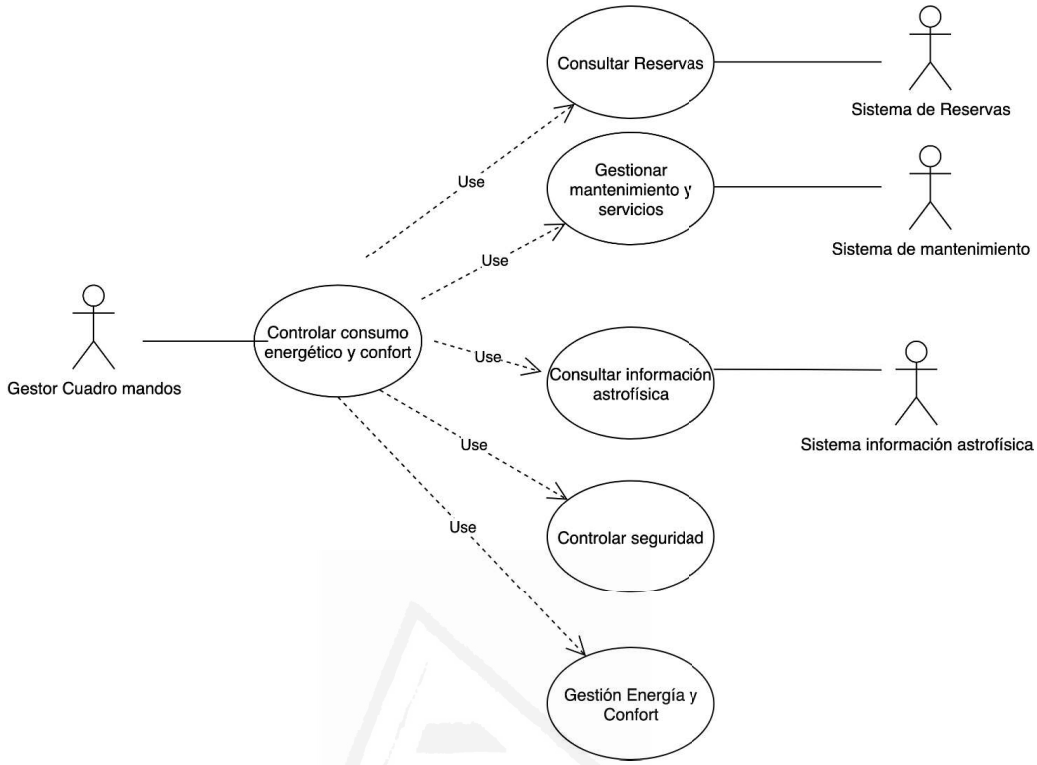


Ilustración 54. Casos de Uso de Control Consumo Energético y Confort.

4.1.3.2 Rol de Consultar Reservas.

Como hemos mencionado en secciones anteriores no vamos a tener un rol que se encargue de la gestión de reservas, sino que vamos a delegar esta función en otro subsistema ya existente. Lo que nosotros vamos a hacer a nivel de reservas simplemente va a ser recibir la información de este sistema, para ello este rol será el encargado de comunicar a ambos sistemas: el nuestro con el de gestión de reservas.

4.1.3.3 Rol de Gestionar Mantenimiento y servicios.

Igual que el Rol anterior el papel principal de este rol es de actuar como interface entre el subsistema ya existente de mantenimiento. La misión principal es recoger la información sobre la limpieza y el mantenimiento programado de las distintas zonas, pero su misión no será solamente recuperar la información, sino que será capaz de comunicar automáticamente incidencias al subsistema de mantenimiento, permitiendo acortar los tiempos de resolución de problemas.

4.1.3.4 Rol de Consultar Información Astrofísica.

Este es otro de los roles cuya misión es obtener información de un subsistema externo. En este caso obtiene los datos de las posiciones del sol (ángulo e inclinación) dependiendo de la situación geográfica y del día/hora del año, de la salida y puesta del sol, etc.

4.1.3.5 Rol de Seguridad.

Dentro de este rol podemos diferenciar dos ámbitos muy diferentes. Por un lado la seguridad desde el punto de vista de control de acceso a instalaciones y por otro lado seguridad desde el punto de vista de fugas de agua, gas, humo, incendio, etc.

En el caso de emergencias queremos que los agentes de bajo nivel tengan capacidad reactiva de actuar ante emergencias, pero también deseamos un rol a más alto nivel encargado de centralizar todos los temas de seguridad.

4.1.3.6 Rol de Gestión de Energía y Confort.

Este es uno de los roles principales del sistema. A diferencia del rol de control de consumo energético y confort, este rol se encarga de la gestión a bajo nivel de los distintos elementos. Este rol aglutina distintos casos de uso como podemos ver en Ilustración 55: Gestionar iluminación, gestionar climatización, gestionar presión de agua, gestionar caldera, gestionar paneles solares y gestionar ascensores.

4.1.3.7 Rol de Gestión de Iluminación.

Dentro de este rol tenemos funciones tanto de información como de actuación. Desde el punto de vista de la información comunicará los valores lumínicos de la estancia. Desde aquí se podrá regular la iluminación natural a través de los toldos, persianas y cortinas, así como la artificial a través de las bombillas de las distintas estancias.

4.1.3.8 Rol de Gestión de Climatización.

Dentro de este rol también nos encontramos como en el caso de la gestión de la iluminación sensores y actuadores. Los sensores recogerán información de temperatura, humedad y consumo energético. Desde el punto de visto de regulación podremos incidir sobre los elementos de free-cooling, toldos, persianas y cortinas, así como sobre los elementos activos de regulación de temperatura (conductos, split, radiadores, etc.).

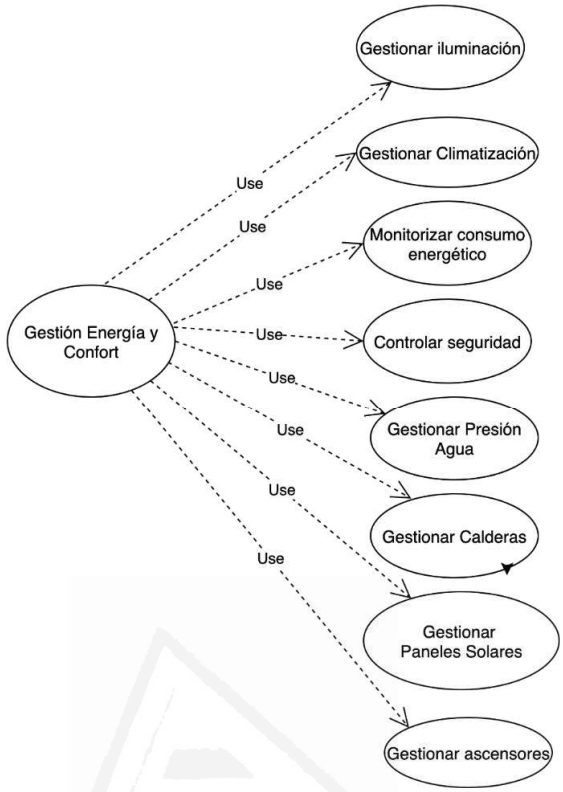


Ilustración 55. Casos de Uso de Rol Gestión de Energía y Confort.

4.1.3.9 Rol de Gestión Presión de Agua.

El objetivo de este rol es mantener una presión de agua suficiente para dotar a grifos y duchas de la presión suficiente, pero al mismo tiempo ahorrar manteniendo el régimen de trabajo de las bombas de presión.

4.1.3.10 Rol de Gestión de Caldera.

Este rol es el que se encargará de mantener e informar sobre la temperatura de las calderas. Este rol se tendrá que coordinar con el de gestión de las placas solares, el cual precalentará el agua con energía solar.

4.1.3.11 Rol de Gestión de Placas Solares.

Dentro de este rol nos vamos a encontrar con dos variantes. Por un lado placas solares cuyo objetivo va a ser el suministro de energía eléctrica, mientras que por otro va a ser el precalentamiento de agua corriente sanitaria (ACS).

4.1.3.12 Rol de Gestión de Gestión de Ascensores.

Como veíamos en el capítulo 4.1.1.3 uno de los gastos más representativos de un hotel era el consumo de los ascensores. Este rol será el que controle la posición, número de ocupantes, modo de funcionamiento (autobús, taxi y mixto), y uso de los ascensores, haciendo que las esperas sean lo menores posibles y al mismo tiempo se minimicen los movimientos y los consumos asociados.

4.1.4 Especificación de tareas.

A la hora de especificar las tareas lo vamos a hacer a través de diagramas de interacción. Dentro de las tareas podemos diferenciar unas más básicas donde los agentes de zona de manera autónoma pueden resolverlas, mientras que otras tareas van a necesitar del sistema multiagente para su resolución.

No vamos a poner los diagramas de interacción de todas las tareas ya que todas siguen los mismos esquemas y sería reiterativo.

En primer lugar, para mostrar cómo se comporta el sistema a alto nivel, vamos a utilizar el diagrama de interacción de la tarea "Regular temperatura zona" donde intervienen varios agentes del sistema. Para conseguir este objetivo el agente de control energético consulta los datos de reservas del agente de reservas y a continuación manda las instrucciones a seguir a los distintos agentes de zona. Como establecimos en la arquitectura de agentes, el agente de reservas es el que va a hacer de interface con el sistema externo de reservas. De este modo por ejemplo se puede preparar la habitación para aumentar el confort a la llegada del huésped a la habitación, o parar el consumo energético de la misma cuando este la abandona. En este diagrama hemos obviado por simplificación la comunicación con los agentes de páginas amarillas y páginas blancas los cuales son los encargados de facilitar los datos de comunicación con el resto de agentes.

Para la ejecución a bajo nivel de la tarea "Regular una zona" vamos a ver el diagrama de interacción de la Ilustración 57. Aquí el agente de zona mediante un plan precargado va a ser capaz de conseguir el objetivo de la tarea "mantener las condiciones de climatización utilizando la menor cantidad de energía. Para ello el agente de zona se comunica con los sensores climáticos y de presencia y atendiendo a los objetivos establecidos, manda las órdenes a los actuadores de la zona pasivos (toldos, persianas y free-cooling) en los cuales el gasto energético es insignificante, y a los actuadores del aire acondicionado y/o calefacción, donde el gasto energético ya es considerable. En este esquema hay dos elementos clave: la arquitectura de comunicación entre el agente y los elementos de bajo nivel que se hará a través de un componente interno del agente de zona (implementado en nuestro caso con un nodo ROS) y el comportamiento autónomo del agente mediante un plan que tendremos que programar donde se

tendrá en cuenta los datos climáticos a alcanzar, minimizar el mínimo de energía y manteniendo la seguridad del sistema.

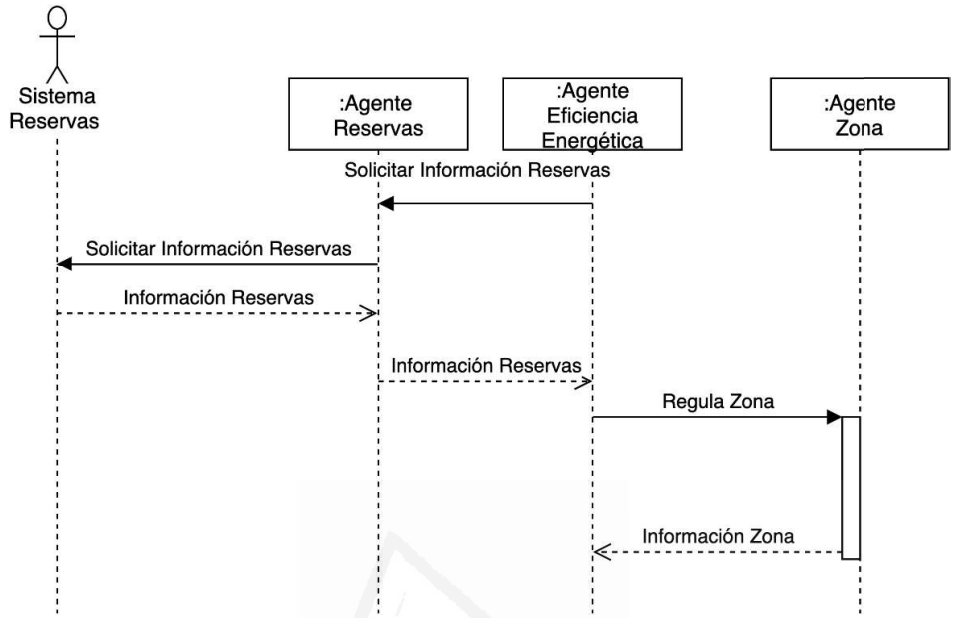


Ilustración 56. Diagrama de interacción para Regular una Zona.

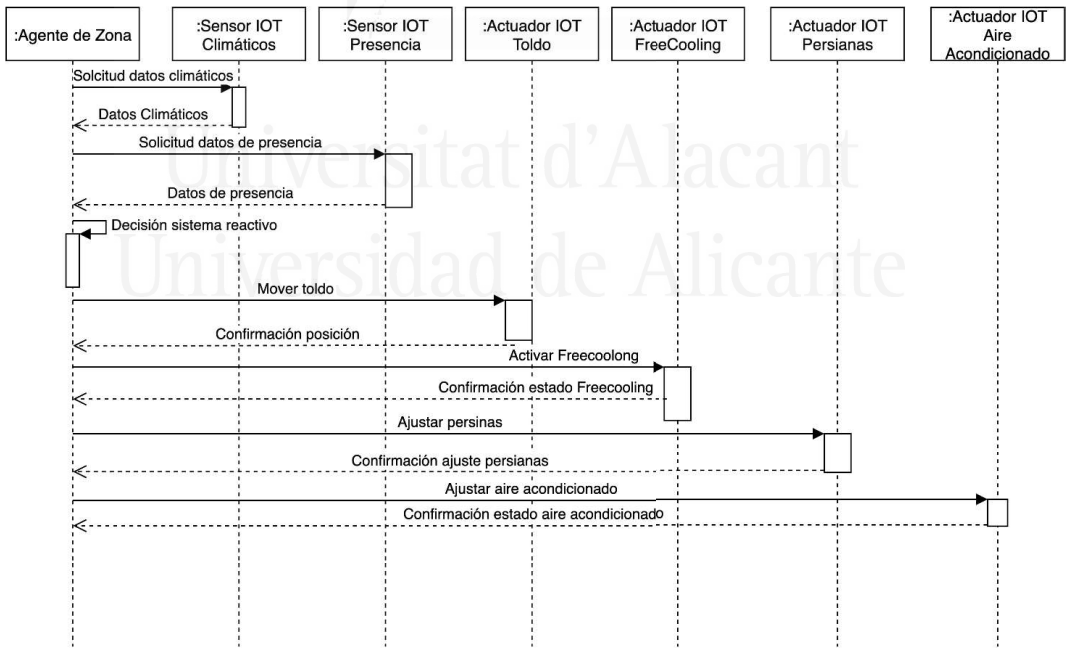


Ilustración 57. Diagrama de Interacción de la tarea "Mantener condiciones climáticas".

Para ilustrar la comunicación del sistema multiagente vamos a ver el diagrama de interacción (Ilustración 58) donde describimos el comportamiento para la tarea de orientación de placas solares. El agente de eficiencia energética va a consultar la información astrofísica a través del agente que actúa como interface con el sistema externo que nos facilita los datos de fecha, hora y posición idónea de las placas. Una vez obtenidos los datos astrofísicos idóneos el agente de ahorro energético manda las órdenes de posicionamiento al agente encargado de las placas solares y verifica la posición de las mismas. Este agente también se encarga de monitorizar el rendimiento de los paneles y de la temperatura del agua tanto en los paneles como en la caldera, donde el agua llega precalentada desde los paneles.

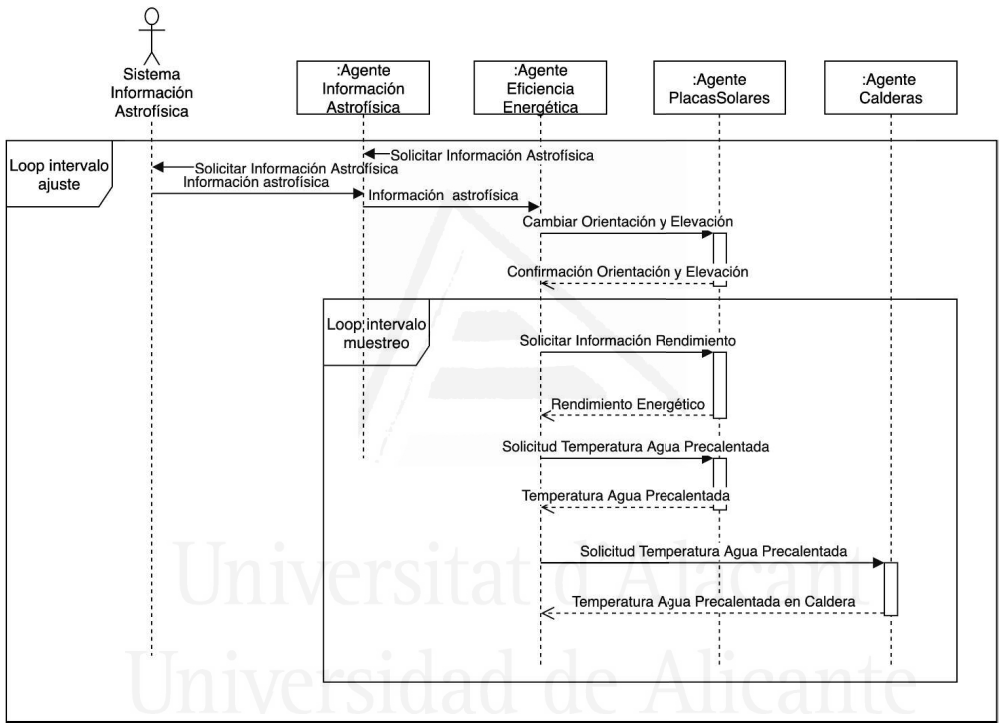


Ilustración 58. Diagrama de interacción regulación placas solares.

4.2 Sociedad de agentes. Ontologías y Protocolos.

4.2.1 Diseño Ontología de Dominio

En esta etapa es donde vamos a aplicar la arquitectura propuesta. Ya en el apartado 3.5.1 avanzábamos en la aplicación y diseño de la arquitectura de la

arquitectura (Ilustración 43). En la Ilustración 59 vamos a repetir dicha ilustración solamente por simplificar la lectura y no obligar al lector a moverse a través del documento.

Tal como marca la propuesta tenemos 3 niveles. El nivel más alto lo componen el sistema multiagente propiamente dicho. A bajo nivel tenemos los elementos encargados de la obtención de datos (temperatura, humedad, presencia, etc.) y actuación (reguladores, accionadores, etc.). El nivel central se ha utilizado para comunicar los dos niveles y para dividir los planes en grupos.

A alto nivel nos encontramos los agentes encargados de mantener la plataforma multiagente; un agente que va desempeñar el rol de cuadro de mandos, desde el cual los controladores podrán supervisar la gestión de los distintos elementos del hotel; los agentes de interfaz usados para comunicar con los sistemas externos de reservas, mantenimiento, datos astrofísicos y climáticos; un agente cuya misión es velar por la seguridad en el hotel, seguridad desde el punto de vista de presencia de personas en zonas donde tenga permisos y seguridad entendida como el buen funcionamiento de los elementos; y para terminar de describir este nivel tendríamos el agente de control energético que es el encargado de coordinar al sistema multiagente.

En el nivel intermedio encontramos agentes de secuenciación y comunicación, los cuales agrupan a los elementos de bajo nivel para facilitar la planificación de tareas y comunican a los elementos de alto y bajo nivel. El agente coordinador de zonas coordinará el estado y las acciones sobre las estancias (salas comunes, habitaciones, etc.). El coordinador de placas solares y calderas gestionará tanto las placas solares fotovoltaicas como las placas ACS y la caldera. La gestión de las placas solares se ha agrupado por su similitud en datos necesarios y comportamiento, mientras que la caldera tiene un funcionamiento muy estrecho con las placas solares ACS. Para completar este nivel intermedio contamos con un agente encargado del sistema de ascensores, coordinando el funcionamiento de los distintos ascensores.

En el nivel más bajo tenemos los microagentes. Entre ellos tenemos los de zona, los cuales recogerán los datos climáticos, de iluminación y de presencia de las estancias y serán los encargados de regular el aire acondicionado, la iluminación, los toldos, las persianas, las ventanas, etc. Dentro de este nivel tenemos los de gestión de placas solares los cuales permitirán regular la orientación e inclinación de las placas y facilitarán información de rendimiento, temperatura del agua precalentada, etc. Lo mismo ocurre con el agente de caldera el cual facilitará la información de temperatura y presión del agua y permitirá aumentar la temperatura de la misma. En este último grupo tenemos microagentes de ascensor que facilitarán la información de posición, dirección, número de pasajeros y será el que mande las peticiones a los motores físicos.

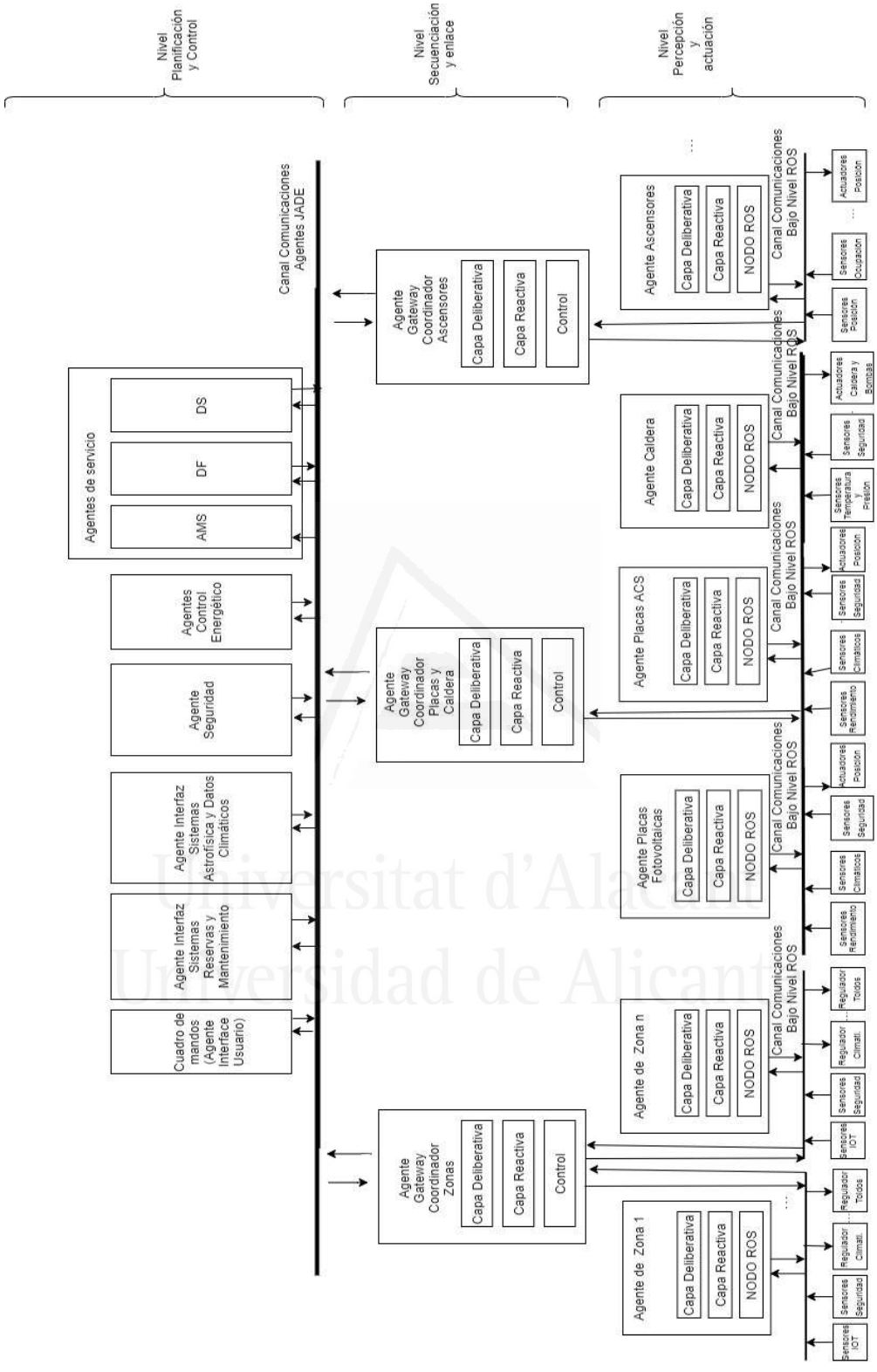


Ilustración 59. Arquitectura Híbrida Multinivel aplicada a Establecimiento Hotelero.

En Ilustración 60 y Ilustración 61 podemos ver los diagramas de clases que define la ontología del dominio para los agentes de alto nivel y bajo nivel. En ellas podemos observar perfectamente que los agentes de alto nivel heredan de una clase padre "agente", mientras que los de bajo nivel heredan de otra clase denominada "microagente". De la clase "agente" heredan las siguientes clases:

- Agentes de Interfaz con otros Subsistemas: son agentes especializados en comunicarse con sistemas externos. En nuestro caso están los sistemas de reservas, mantenimiento, datos astrofísicos y climáticos.
- Agentes de Interfaz de usuario: hemos preferido dejar un tipo de agente para encapsular toda la comunicación con el usuario, englobando en él el cuadro de mandos del sistema.
- Agentes de Deliberativos: son los agentes encargados planificar y coordinar los objetivos a alto nivel del sistema. Irán haciendo peticiones al resto de agentes, pero ellos serán los que vayan orientando al sistema hacia los objetivos deseados. También serán los responsables de recoger los resultados y de incorporar mejoras.
- Agentes de Control de la Plataforma: dentro de esta categoría nos encontramos a los agentes que gestionan y facilitarán la plataforma de agentes.

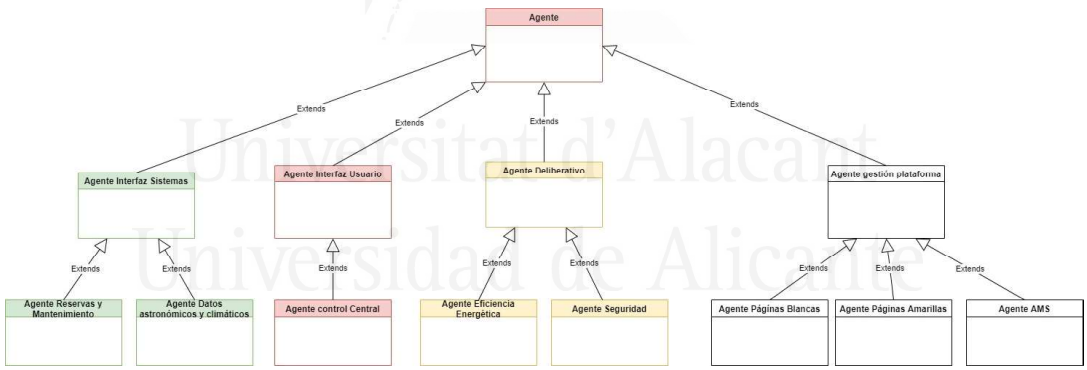


Ilustración 60. Ontología de Dominio de Agentes Alto Nivel.

Para el control de los elementos de bajo nivel hemos creado clases que cuelgan de la clase padre "microagente". Estos microagentes serán los encargados de agrupar los elementos de bajo nivel (sensores y actuadores) relacionados con cada rol: agente de zona, agente calderas, agente placas solares ACS, agente placas solares fotovoltaicas, ascensores.

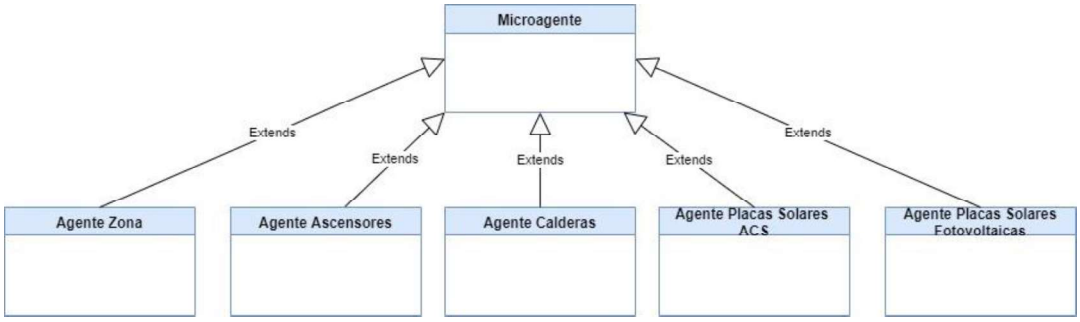


Ilustración 61. Ontología de Dominio de Agentes de Bajo Nivel (microagentes).

La capa intermedia tendría una clase "agente Gateway" que sería una composición de las clases "agente" y "microagente". De esta manera podríamos utilizan las características de cada clase agregada. Para nuestro problema hemos diseñado 3 agentes Gateway atendiendo a los roles: agentes coordinadores de zona, agentes coordinadores de placas solares y calderas, y agentes coordinadores de ascensores.

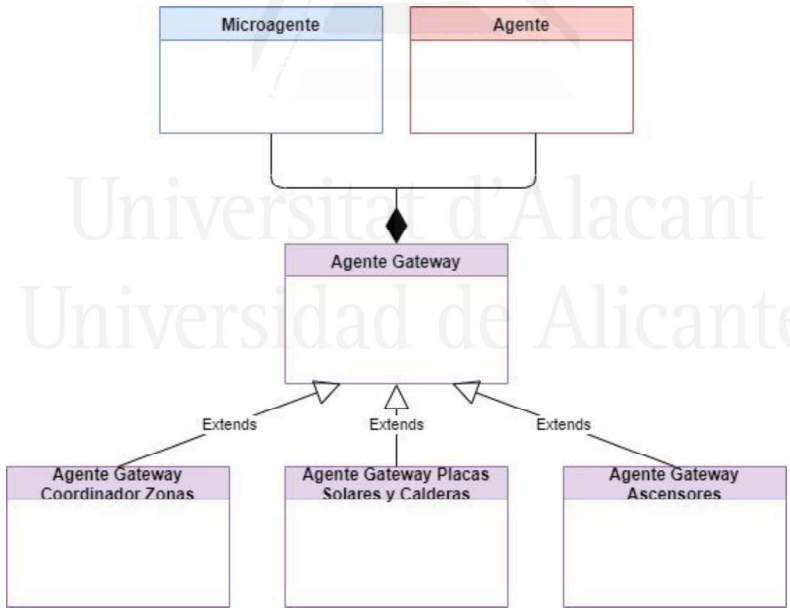


Ilustración 62. Ontología de Dominio Agentes de nivel medio.

4.2.2 Diseño Ontología de Comunicaciones

La comunicación entre los agentes se va a realizar mediante las especificaciones FIPA utilizando los lenguajes: SL ("Semantic Language"), CCL ("Constraint Choice Language"), KIF ("Knowledge Interchange Format") y RDF ("Resource Descripction Framework"). En nuestro caso el framework utilizado (JADE) nos facilitará la implementación del sistema de mensajes.

La comunicación a bajo nivel de los agentes de zona con los elementos IoT requieren una menor complejidad, pero al mismo tiempo necesitan una mayor velocidad. Para realizar esta comunicación nos tocará definir los "Topics" a utilizar en ROS.

Gracias a los estándares en comunicación el trabajo de comunicación entre agentes y de agentes con elementos de bajo nivel los protocolos se simplificarán bastante. Sin embargo, para la comunicación de agentes con sistemas externos con tocará ir cubriéndola adhoc. Dependiendo de la complejidad, tecnología y arquitectura utilizada en estos sistemas externos la implementación de estos agentes será más o menos compleja. Muchas veces este apartado se complica debido a la antigüedad de estos sistemas ya existentes. Destacar que tener encapsulados estos interfaces en agentes concretos con va a facilitar el cambio a otros sistemas externos si fuese necesario.

4.3 Diseño de Implementación de Agentes.

4.3.1 Definición de Estructura de Agentes.

Nos vamos a centrar en la estructura de los agentes de bajo nivel que es la parte donde incide esta tesis, así como en la parte de comunicación con el sistema multiagente. En este caso concreto, vamos a utilizar JADE a alto nivel, y ROS a bajo nivel con el fin de unificar la forma de controlar y comunicar los elementos IoT. ROS nos permitirá manera casi transparente el uso de elementos IoT que utilizan distintas tecnologías gracias a los paquetes, entre los cuales son muy utilizados "ROS Package Home Automation" y "Ros Serial para Arduino", tendremos un sistema flexible y fácilmente ampliable. Los agentes de nivel medio son los que permitirán la comunicación de los dos niveles realizando la función de Gateway. En la Ilustración 63 podemos ver mediante un diagrama de clases como los agentes de alto nivel heredan del agente base de "JADE", mientras que los "microagentes" heredan de la clase "nodo ROS". Al estar implementados los protocolos y métodos de comunicación en estas clases padre permitirá sus clases hijas comunicarse a través de las plataformas

correspondientes (el canal de comunicación de JADE y a través de los nodos ROS, respectivamente). El agente Gateway agrupa en una única clase las funcionalidades de ambas plataformas.

El siguiente paso que debemos de dar es definir la arquitectura interna de los agentes. En la Ilustración 64 podemos ver como el agente de zona está formado por varios módulos de competencia organizados en capas. La información se recibe a través de módulo "comunicaciones ROS". Esta información puede provenir de los sensores o del módulo Gateway. El módulo de "seguridad" se encarga de comprobar que no hay ninguna alerta por la cual tenga que ejecutar una acción y pasa el control a los módulos de competencia propiamente dichos, "regulación de climatización" y "regulación iluminación". Los comportamientos de estos módulos definirán las acciones a realizar, las cuales serán transferidas a los actuadores a través del módulo de "comunicación ROS". Si tuviésemos que enviar algún tipo de información al Gateway lo haríamos también a través del módulo de "comunicaciones ROS".

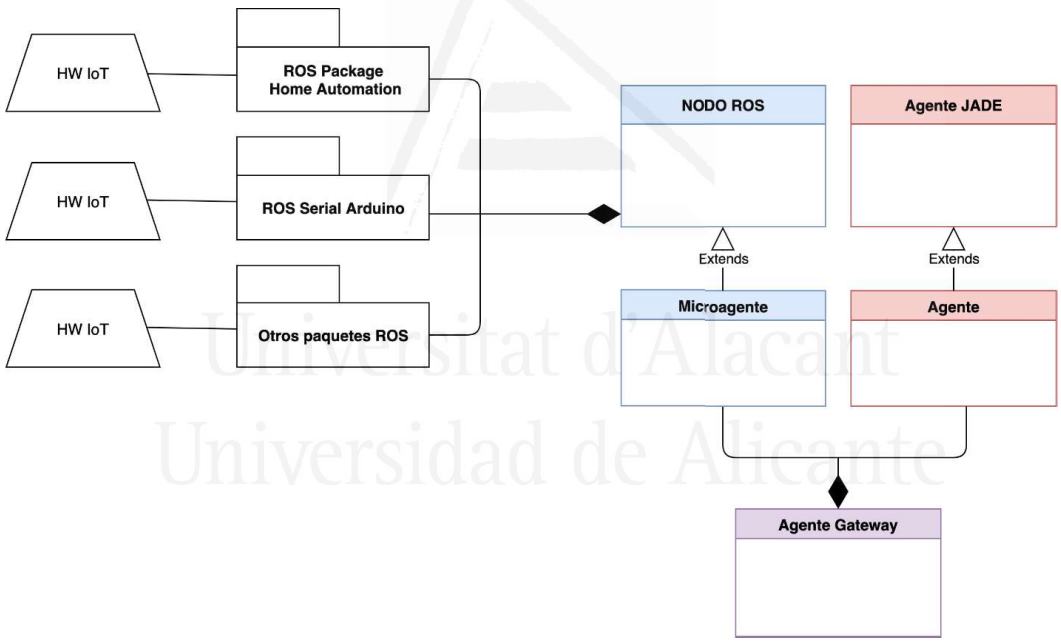


Ilustración 63. Diagrama de Clases de Agentes donde se muestra las relaciones con las clases de los Frameworks utilizados JADE y ROS.

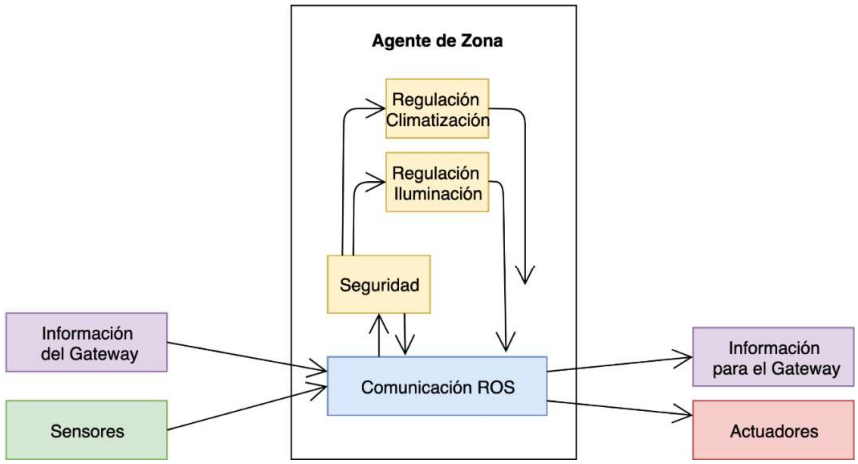


Ilustración 64. Arquitectura interna de Agente de Zona.

La función del agente Gateway como hemos visto es doble, por un lado la de secuenciación de planes provenientes de la capa de alto nivel, y por otro la de la comunicación entre capas. En la Ilustración 65 vemos la arquitectura interna que hemos diseñado para dotar al agente de estas funcionalidades. A través de los módulos de competencia de comunicaciones nos comunicaremos con las capas de alto (JADE) y bajo nivel (ROS). El módulo de traducción será el encargado de convertir la información del mensaje s a un formato interno que el agente puede utilizar y viceversa, haciendo posible el dialogo entre "agentes" y "microagentes" así como poder utilizar la información internamente. El módulo de secuenciación será el encargado de multiplexar y demultiplexar la secuenciación de los planes.

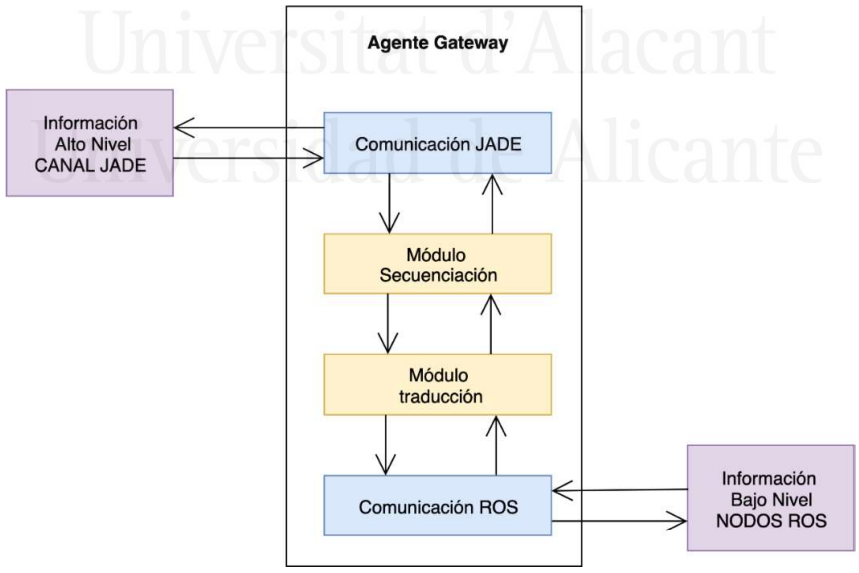


Ilustración 65. Arquitectura interna de Agente Gateway.

4.3.2 Descripción de Comportamientos.

Para ilustrar los comportamientos vamos a utilizar un algoritmo Fuzzy en un agente de zona para definir su comportamiento respecto a la regulación de climatización. El agente de control de energía es el que tiene toda la información (equipamiento, reserva/ocupación y será el encargado de facilitar las etiquetas lingüísticas y la matriz de comportamiento (fuzzy matrix). Fijémonos lo fácil que sería cambiar el comportamiento del agente de zona simplemente variando los valores asociados a las etiquetas lingüísticas o la fuzzy matrix. Siendo de esta manera un cambio de comportamiento cuando la habitación este vacía, la tarea en la zona cambie o simplemente el perfil del cliente sea diferente.

En concreto en la zona vamos a contar con los siguientes elementos:

- Sensor de temperatura. Tendrá un rango de funcionamiento entre 0 y 40ºC y una precisión de centésimas.
- Sensor de humedad. Cuenta con un rango de funcionamiento entre 0% y 100% de humedad relativa, con una precisión de centésimas.
- Actuador sobre la bomba de calor. Se puede controlar mediante incrementos/decrementos de temperatura, desde -15ºC a +15ºC.

En la Ilustración 66 podemos ver las que hemos creado 5 etiquetas lingüísticas para cada atributo:

- Temperatura: muy baja (MB), baja(B), normal (N), alta (A) y muy alta (MA).
- Humedad: muy baja (MB), baja(B), normal (N), alta (A) y muy alta (MA).

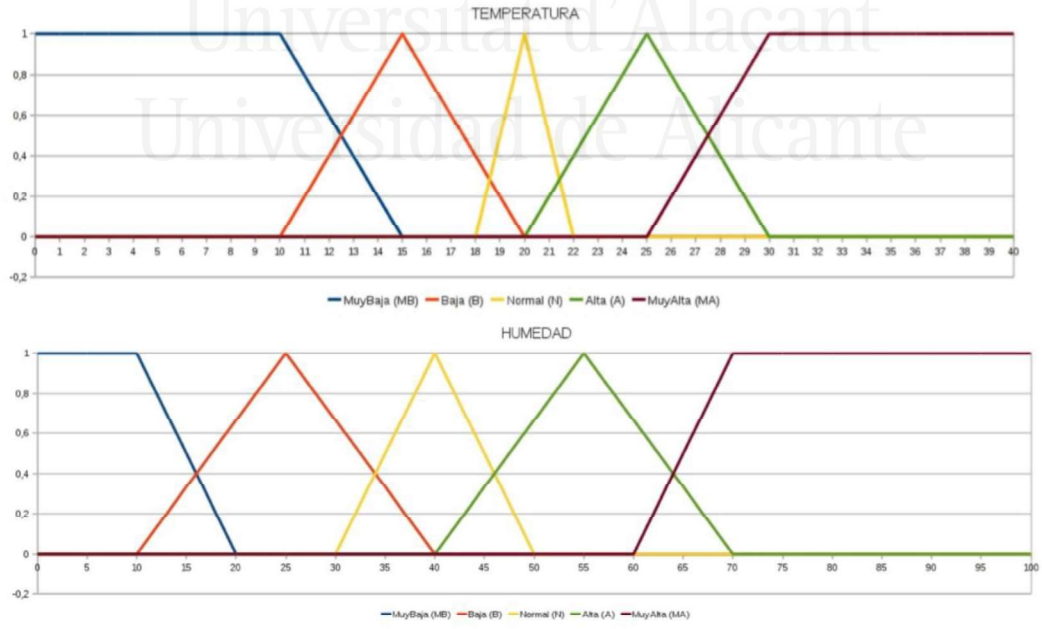


Ilustración 66. Etiquetas lingüísticas para temperatura y humedad.

Podríamos actuar sobre varios elementos (ventana, humificador, bomba de calor, etc.). En nuestro caso para simplificar actuaremos sobre la bomba de calor vamos y vamos a establecer 7 etiquetas lingüísticas (Ilustración 67): bajada grande (BG), bajada normal (BN), bajada pequeña (BP), mantener (M), subida pequeña (SP), subida normal (SN), subida grande (SG).

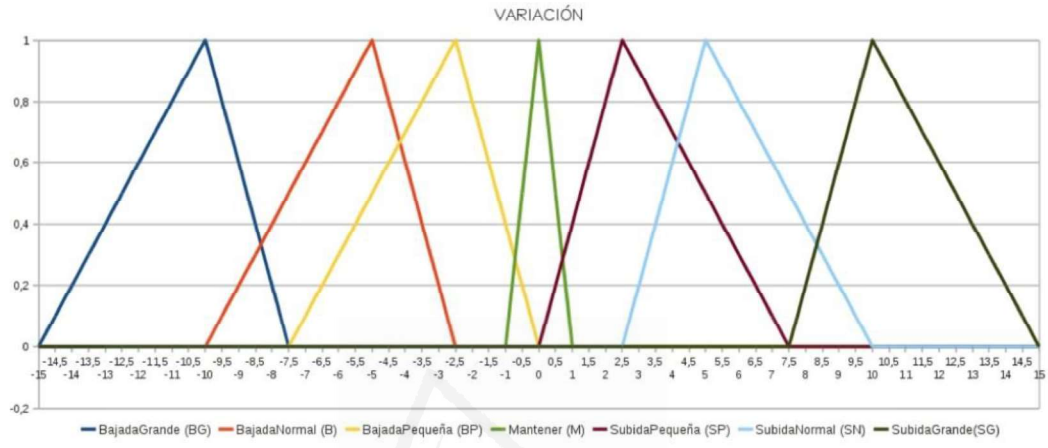


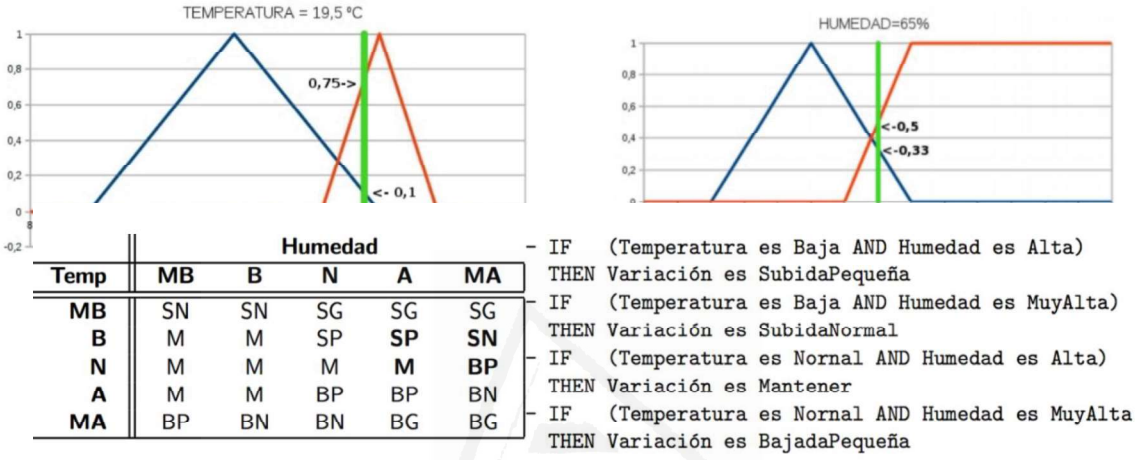
Ilustración 67. Etiquetas lingüisticas de actuación.

La parte clave del comportamiento vendrá dada por la Fuzzy Association Matrix (la tabla de aplicación de reglas difusas), Ilustración 68 , que como hemos comentado será proporcionada por el agente de control energético.

	Humedad				
Temp.	**MB**	**B**	**N**	**A**	**MA**
MB	SN	SN	SG	SG	SG
B	M	M	SP	SP	SN
N	M	M	M	M	BP
A	M	M	BP	BP	BN
MA	BP	BN	BN	BG	BG

Ilustración 68. Fuzzy Matrix (matriz de comportamiento).

En la Ilustración 69 podemos ver como se comportaría el agente de zona ante unos valores de temperatura y humedad. El valor singleton "Temperatura=19,5º C" se corresponde con un grado de verdad 0,1 para el valor difuso Temperatura Baja(B) y con un grado de verdad 0, 75 para el valor difuso Temperatura Normal(N). El valor singleton "Humedad=65%" se corresponde con un grado de verdad 0,5 para el valor difuso Humedad Alta(A) y con un grado de verdad 0,33 para el valor difuso Humedad MuyAlta(MA). Ante estos valores la matriz nos recomendaría la acción "mantener".



Temp	Humedad MB	B	N	A	MA
MB	SN	SN	SG	SG	SG
B	M	M	SP	SP	SN
N	M	M	M	M	BP
A	M	M	BP	BP	BN
MA	BP	BN	BN	BG	BG

- IF (Temperatura es Baja AND Humedad es Alta) THEN Variación es SubidaPequeña
- IF (Temperatura es Baja AND Humedad es MuyAlta) THEN Variación es SubidaNormal
- IF (Temperatura es Nornal AND Humedad es Alta) THEN Variación es Mantener
- IF (Temperatura es Nornal AND Humedad es MuyAlta THEN Variación es BajadaPequeña

Ilustración 69. Valores ejemplo para actuación de la tabla de comportamiento.

4.4 Codificación y Pruebas.

El entorno de desarrollo lo hemos montado sobre una máquina virtual con Ubuntu 16.04 en la cual hemos instalado los siguientes elementos:

- JADE versión 4.5.0.
- ROS versión Kinetic, a la cual se le ha añadido el módulo ROSJava.
- Java 1.8.

Las versiones escogidas nos han venido un poco impuestas por razones de compatibilidad entre módulos ROS y JADE, y más en concreto por el paquete de ROS "ROSJava". JADE funciona en Java y para que ROS pueda trabajar con Java tuvimos que usar dicho paquete. La última versión de ROS que admite "ROSJava" es ROS Kinetic, y este funciona bajo Ubuntu 16.04.

Se ha planificado el desarrollo del proyecto mediante un modelo ágil basado en iteraciones. En concreto para la parte que nos atañe se crearon 4 iteraciones con los siguientes objetivos:

- Desarrollo de la capa de alto nivel en JADE y probar las comunicaciones entre los agentes.
- Desarrollo de la capa de bajo nivel, implementando los nodos de ROS necesarios para la gestión de los elementos IoT y comprobando las comunicaciones entre los nodos y los elementos IoT.
- Desarrollo del agente Gateway para lograr comunicar los 2 niveles.
- Verificación del sistema a través de todos los niveles.

En las tres primeras iteraciones, se fueron haciendo pruebas unitarias con el fin de ir probando cada uno de los niveles de manera independiente. En la cuarta se hicieron pruebas a nivel de sistema con el fin de comprobar la interacción entre los niveles y comprobar las funcionalidades del sistema.

4.4.1 Implementación de Agentes con JADE.

Como veíamos en el capítulo 2.4.1, JADE proporciona un framework para el desarrollo de agentes software, y proporciona una implementación que cumple con los estándares de FIPA, así como herramientas gráficas que nos permiten depurar e inspeccionar el sistema.

4.4.1.1 Agente Básico.

En JADE la implementación de un agente se hace a través de la creación de una clase que herede de la clase jade.core.Agent, proporcionada por framework. Cuando un agente se inicializa se llama al método setup(), el cual es un método abstracto definido la clase padre y por tanto será necesario implementarlo. Las acciones que tiene que realizar el agente se definirán a través de los comportamientos, definidos por la clase JADE jade.core.behaviours. Y estos comportamientos se añaden a la clase agente creada a través del método addBehaviour(), ya sean comportamientos de una sola ejecución, múltiples ejecuciones o ejecuciones cíclicas.

Para comprenderlo mejor vamos a poner un ejemplo. Un agente tiene como objetivos atender peticiones y atenderlas. Para ello se inicializa con el método setup() y se queda a la espera de recibir peticiones. Este comportamiento de espera lo realizaremos definiendo el método waitForMessages() de forma que el agente se queda esperando mensajes de otros agentes; una vez que llegan los verifica y si son correctos, llama comportamiento correspondiente para atender y responder la petición.

4.4.1.2 Agentes de la Plataforma (AMS, DF).

Como vimos en el capítulo 2.4.1 JADE nos proporciona los agentes y mecanismos necesarios para la creación, mantenimiento y uso de la plataforma.

En la Ilustración 70 podemos ver un esquema de los pasos seguidos para la construcción de la plataforma de agentes de alto nivel. Primero se crea el contenedor de la plataforma. Luego se crean identificadores de la plataforma y de los agentes de servicio AMS y DF. Crea los agentes y por último los lanza, permitiendo a partir de este momento el registro de otros y la utilización de los servicios.

Para nosotros todos los agentes son importantes, pero desde el punto de vista de funcionalidad del sistema y de interacción será muy importante el Directory Facilitator (DF). La función del agente DF es recopilar los servicios ofrecidos por los agentes de la plataforma y facilitar los datos de los agentes que proveen este servicio cuando otros agentes lo demanden. Por ejemplo, el agente de "datos climáticos" registra el servicio de "obtener_datos_climáticos" de forma que cuando otro agente demande proveedores de este servicio, el agente DF le devolverá el identificador del agente que ha registrado el servicio. Si un agente ofrece varios servicios registraría cada uno de ellos en el agente DF.

Hacer notar la tolerancia a fallos, flexibilidad y escalabilidad de este esquema, al poder tener varios proveedores capaces de ofrecer el servicio.

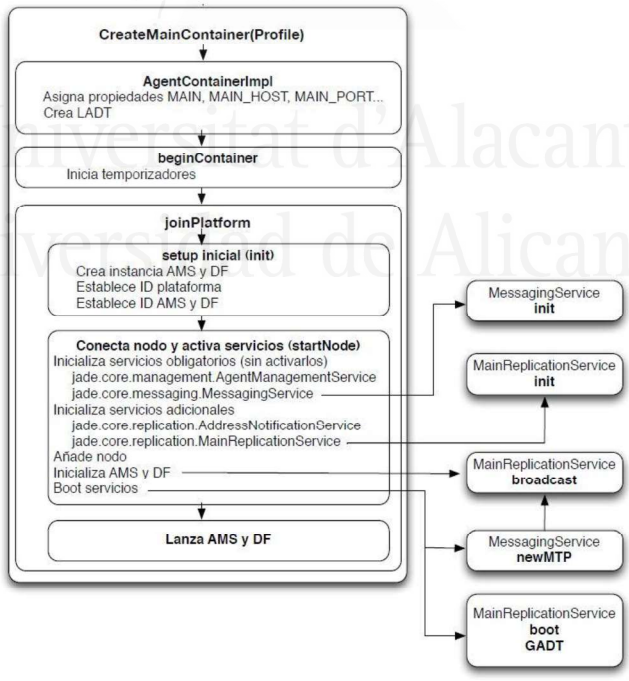


Ilustración 70. Pasos seguidos para la creación de la plataforma JADE y los agentes de servicio.

4.4.2 Implementación de Microagentes con ROS e Integración de elementos IOT.

4.4.2.1 Agente de Zona.

Los agentes de zona son "microagentes" que representan las distintas estancias (habitaciones y zonas comunes) del hotel dentro del sistema multiagente. Cada uno de estos es instanciado como un nodo ROS. Para comunicarse con los agentes del sistema lo hará a través del agente Gateway asociado.

La gran ventaja de utilizar ROS es que es el propio ROS el que nos hace transparente el uso de dispositivos que utilizan distintos protocolos. Nosotros vamos a tratar con los distintos dispositivos como nodos ROS independientemente de si utilizan OpenHAB, Rosserial Arduino o cualquier otro tipo de variante.

4.4.3 Gateway ROS-JADE.

La implementación del Gateway entre ROS y JADE ha consistido en una parte clave en la prueba del modelo de este trabajo, permitiendo la comunicación entre el nivel superior e inferior de la arquitectura (agentes y los elementos IoT).

Una decisión un poco forzada ha sido la utilización del paquete de ROSJava para poder facilitar la integración de ambas plataformas. JADE está disponible únicamente en Java, mientras que ROS permite varios lenguajes de forma nativa (sin introducir ningún paquete añadido) entre ellos C++ o Python.

El siguiente problema que nos encontramos fue en como lanzar los agentes y los nodos ROS mediante código o mediante instrucciones por terminal. En el diseño inicial la idea era crear el agente de zona en la plataforma de agentes y que este creará el nodo ROS principal de su zona. Pero debido a la estructura de ROS y gracias a las facilidades de Java lo tuvimos que hacer al revés: creamos el nodo principal y desde él, haciendo uso de la funcionalidad que tiene JADE de lanzar agentes desde programas Java externos.

ROSJava es un paquete de ROS que permite crear nodos utilizando el lenguaje JAVA. De esta forma creamos el nodo ROS que va integrado en cada coordinador de zonas y este creará el agente para comunicarse con la plataforma del sistema multiagente.

Hemos definido la clase `AbstractRosjavaNode` para encapsular todo esto: de la instanciación del agente, al que hemos llamado JADEGateway, y la comunicación con él cuándo sea necesario En primer lugar, la clase instanciará un nuevo agente dentro del sistema multiagente empleando las clases que ofrece JADE para crear agentes de mediante código, inicializa también el nodo ROS e

implementa un comportamiento que es el que se encargará de ejecutar el JADEGateway para comunicarse con el agente de zona correspondiente y de devolver el mensaje de respuesta al finalizar.

En la Ilustración 71 podemos ver el diagrama de flujo del comportamiento del agente Gateway ante peticiones del sistema multiagente. Para crear distintas zonas que se comuniquen con dispositivos IoT distintos correspondería con implementar una clase que herede de AbstractRosjavaNode y que implemente los comportamientos abstractos, definidos en procesar peticiones y respuestas.

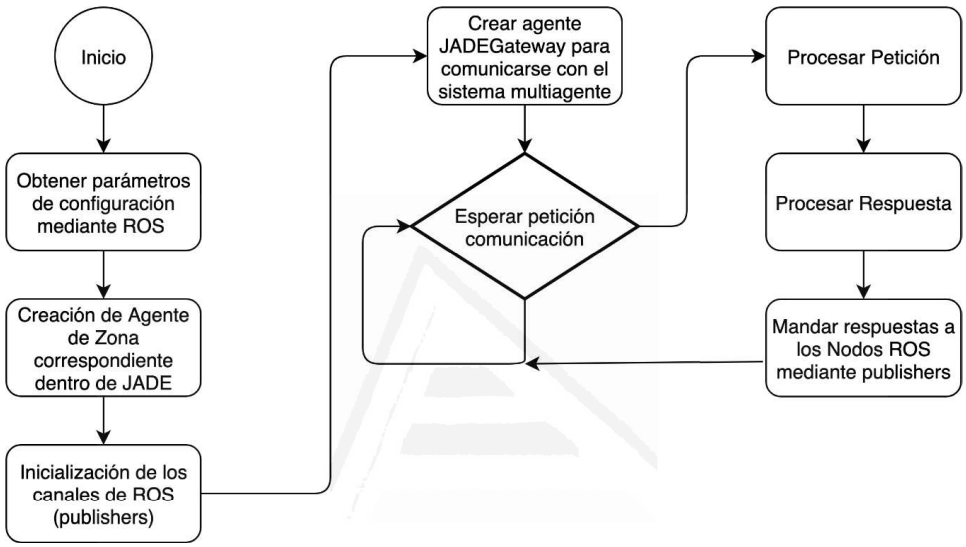


Ilustración 71. Diagrama de flujo recepción peticiones del sistema multiagente y paso a los elementos IoT a través de nodos ROS.

4.4.4 Instanciación y Creación de Clases del Sistema Completo.

Por último, vamos a explicar cómo vamos a lanzar y ejecutar cada parte del sistema. Desde terminal lanzamos un script auxiliar llamado "launch_jade_core.sh" el cual crea la instancia de la plataforma JADE, lanzando todos los agentes de alto nivel. Para lanzar los nodos ROS tanto de dispositivos IoT como agentes de zona la herramienta Roslaunch, donde se han descrito las diferentes zonas del hotel. Se instanciará un nodo ROS para cada zona utilizando un namespace diferente y se instanciará un agente dentro del sistema multiagente con el mismo nombre que el namespace. Por último, se crearán los nodos ROS de los diferentes elementos IoT para la zona, utilizando de nuevo el mismo namespace. Repetiremos este proceso para cada uno de los microagentes que necesitemos tener.

4.4.5 Pruebas Realizadas.

Como comentamos al principio de este apartado, dividimos el desarrollo y las pruebas en varias iteraciones. Al trabajar con diferentes niveles, frameworks y tecnologías pensamos la mejor forma de abordarlo era desarrollar y probar las comunicaciones dentro de cada uno de los niveles por separado, luego realizar pruebas de interconexión entre niveles y para finalizar hacer las pruebas de funcionamiento del sistema completo.

Para realizar las pruebas del sistema, utilizamos los siguientes elementos con el fin de realizar pruebas con distintos periféricos IoT y hacerlo en un entorno distribuido:

- Plataforma de agentes: ordenador Thinkpad E490 (Core i7 a 4,60GHz, 8GB de memoria, disco de 512GB SSD) el cual albergará JADE y la plataforma de agentes.
- Agente de Zona (microagente): Raspberry Pi 3B+.
- Dispositivos IoT: Placa Arduino para simular dispositivos IoT, tanto sensores como actuadores (temperatura, humedad, regulador temperatura) y bombilla Xiaomi Lightbulb, como dispositivo IoT con protocolo OPENHUB.

4.4.5.1 Prueba de Comunicaciones de Agentes en JADE.

En estas primeras pruebas comprobamos que los distintos agentes del sistema se creaban y comunican correctamente. Para ello utilizamos las herramientas de depuración que proporciona JADE, como estudiamos en el apartado 2.4.1.1.

A través el agente "Introspector" comprobamos el estado interno del agente, observando su comportamiento actual, los mensajes enviados y recibidos, e incluso pudimos hacer pruebas cambiando el estado del agente. En la Ilustración 72 podemos ver una captura de la creación de un agente coordinador de zona.

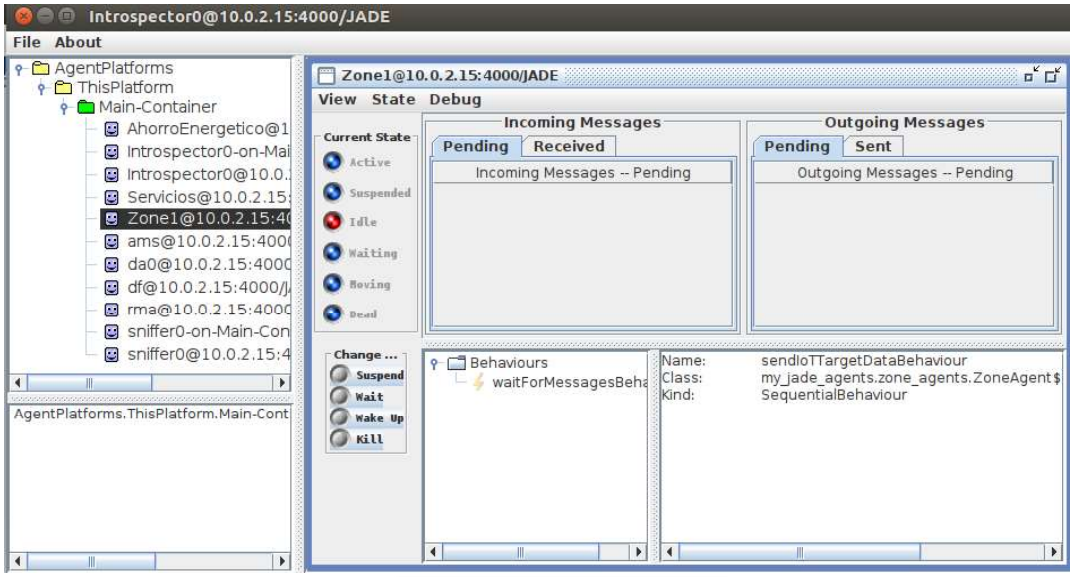


Ilustración 72. Agente Introspector sobre agente coordinador de zona.

También utilizamos otra herramienta de JADE, el DummyAgent, que nos sirvió para generar o editar mensajes FIPA y ver que ocurría en el sistema. Una herramienta muy útil que nos ofrece JADE es el "Sniffer". A través de este agente pudimos seleccionar los agentes activos en el sistema, viendo las conversaciones y mensajes intercambiados entre todos los agentes.

En la Ilustración 73, podemos ver que con un agente dummy (other) se generan y envían dos mensajes a un agente de coordinador de zona llamado "JADAD". El primero de ellos se formó incorrectamente y el agente lo ignoró, mientras que el segundo, que sí que está formado correctamente y desencadena las acciones siguientes:

1. El agente de zona solicita al agente páginas amarillas (DF) un agente de Ahorro Energético (conversación rosa).
2. El agente DF le facilita la dirección del agente de Ahorro Energético.
3. El agente de zona hace una petición al agente de Ahorro energético para recibir los parámetros de funcionamiento (request gris).
4. Como el agente de ahorro energético necesita información de ocupación y datos climáticos solicita al agente DF los datos del agente que puede ofrecerle estos servicios (conversación azul).
5. El agente de ahorro energético consulta la ocupación del hotel al agente de servicios (conversación amarilla).

6. El agente de ahorro energético solicita los datos climáticos (conversación roja).
7. El agente de ahorro energético crea la respuesta y le contesta al agente de zona (inform negro).
8. El agente de zona procesa lo obtenido y responde al agente desencadenador (other) con un INFORM (inform rosa final).

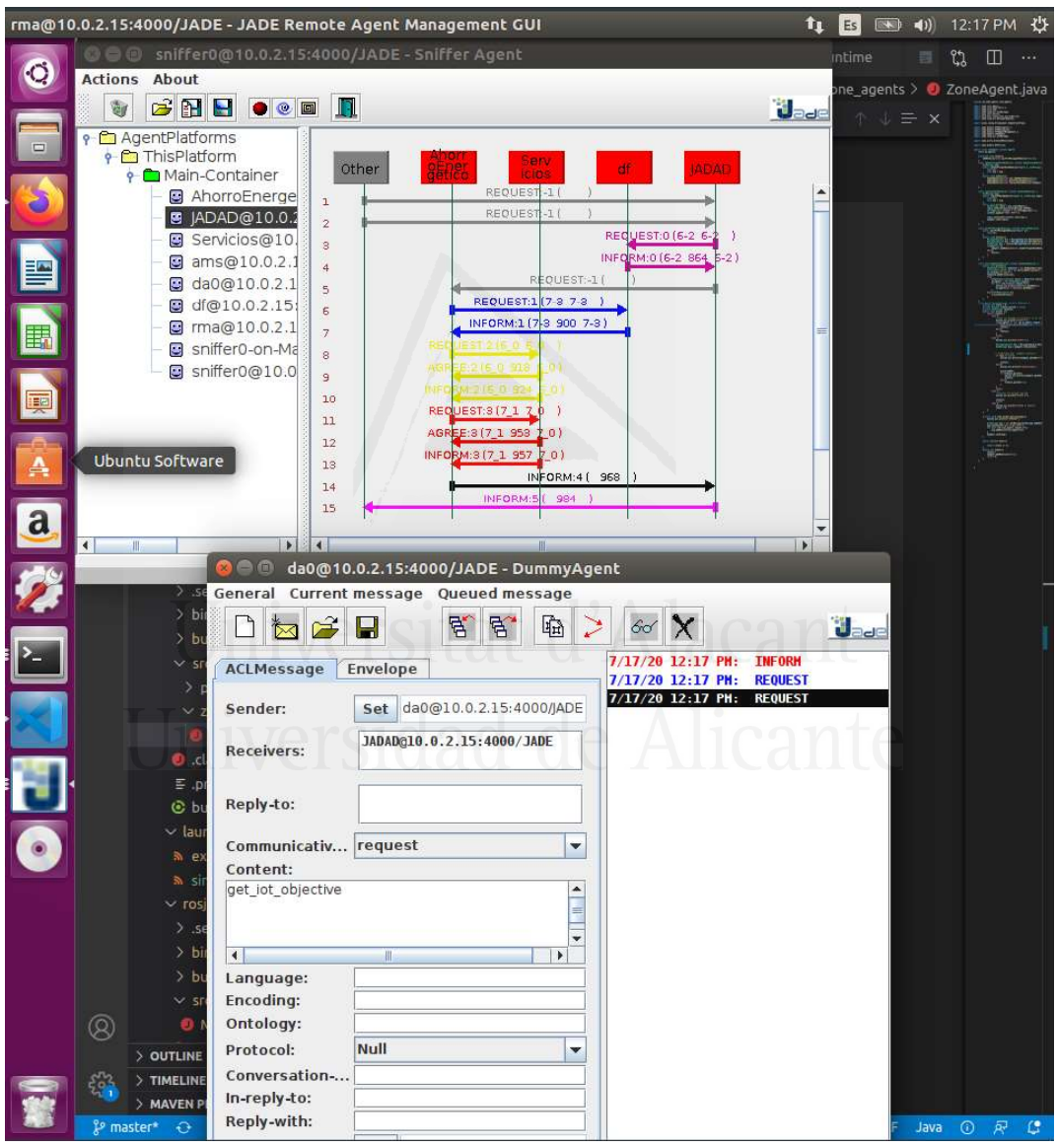


Ilustración 73. Ejemplo del proceso de depuración.

Con esto hemos visto como efectivamente los agentes del sistema pueden localizar a los agentes que dan los servicios, comunicarse con ellos solicitando información y enviarla a otros agentes del sistema.

4.4.5.2 Pruebas de Comunicaciones en ROS.

Al igual que JADE, ROS nos proporciona herramientas de depuración. Para validar la comunicación entre nodos nosotros empleamos las herramientas "rqt_graph", "rqt_plot" y la siempre socorrida salida de terminal. Rqt_graph nos ha permitido ver el estado actual del grafo computacional de ROS y observar los valores que se transmiten a través de los topics. Rqt_plot crea y visualiza el grafo al completo del sistema, permitiéndonos visualizar si se han creado los nodos correctamente.

En la Ilustración 74 podemos ver la creación varios zonas y nodos y el envío de un mensaje entre 2 elementos de la zona 1. La terminal se empleó para ver los mensajes se formaban y enviaban correctamente. Tras confirmar que ROS estaba funcionando como se esperaba, el siguiente paso consistió en crear el nodo del agente coordinador de zona y ver que las comunicaciones funcionaban perfectamente.

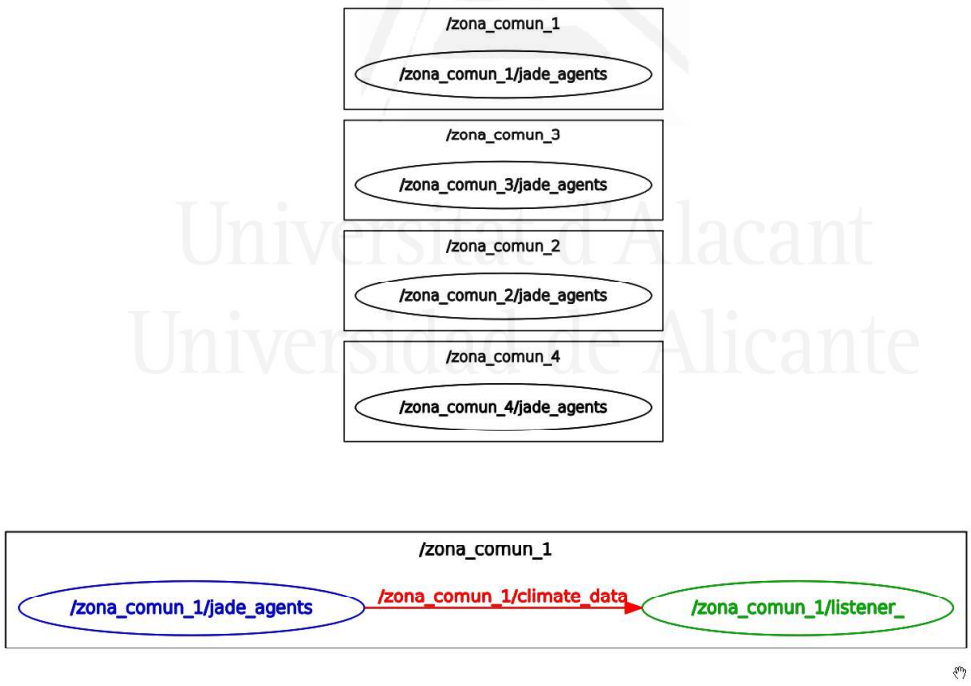


Ilustración 74. Captura de rqt_plot donde se han creado varias zonas y nodos, y se transmite un mensaje entre dos nodos.

4.4.5.3 Prueba de Comunicaciones Middleware, prueba End-to-End.

La siguiente prueba que nos planteamos fue verificar que el sistema podía enviar información de un extremo a otro, pasando de JADE a ROS y viceversa.

Para esta prueba se lanzó por un lado la plataforma de agentes JADE y se utilizó la herramienta "roslaunch" en otra terminal para crear dos nodos (un microagente de zona y dispositivo IoT).

En la Ilustración 75, podemos observar una captura de la herramienta "Sniffer" de JADE donde vemos que un agente externo envía una petición request a el agente coordinador de zona "name1", y como se realiza el proceso completo de obtención de datos y de confirmación al agente que realizó la petición.

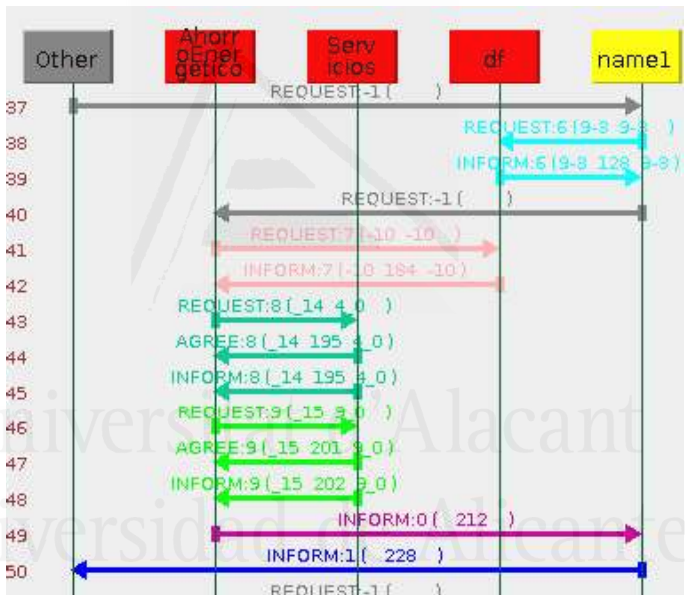


Ilustración 75. Snifer en prueba end-to-end

Por otro lado, en la Ilustración 76, observamos como se crea el node CORE de ROS y el nodo del agente coordinador de zona, así como que el nodo crea y registra un agente dentro del sistema multiagente, dentro de la plataforma "zona común 1".

```
started roslaunch server http://ubuntu-vm:45791/

SUMMARY
========

PARAMETERS
 * /rosdistro: kinetic
 * /rosversion: 1.12.14
 * /zona_comun_1/jade_agents/agent: name1
 * /zona_comun_1/jade_agents/zone_name: zona comun 1

NODES
  /zona_comun_1/
    jade_agents (rosjava_pkg_a/execute)
    listener_ (ros_iot/listener.py)

auto-starting new master
process[master]: started with pid [3559]
ROS_MASTER_URI=http://localhost:11311

setting /run_id to e316350c-cd42-11ea-a151-0800272925e5
process[rosout-1]: started with pid [3572]
started core service [/rosout]
process[zona_comun_1/jade_agents-2]: started with pid [3575]
process[zona_comun_1/listener_-3]: started with pid [3579]
Jul 24, 2020 2:16:16 AM org.ros.internal.node.client.Registrar <init>
INFO: MasterXmlRpcEndpoint URI: http://localhost:11311
Jul 24, 2020 2:16:16 AM org.ros.internal.node.client.Registrar onPublisherAdded
INFO: Registering publisher: Publisher<PublisherDefinition<PublisherIdentifier<NodeIdentifier</zona_c
omun_1/jade_agents, http://127.0.0.1:43243/>, TopicIdentifier</rosout>>, Topic<TopicIdentifier</rosou
t>, TopicDescription<rosgraph_msgs/Log, acffd30cd6b6de30f120938c17c593fb>>>>
Jul 24, 2020 2:16:16 AM org.ros.internal.node.client.Registrar callMaster
INFO: Response<Success, Registered [/zona_comun_1/jade_agents] as publisher of [/rosout], [http://ubu
ntu-vm:45291/]>
Jul 24, 2020 2:16:16 AM org.ros.internal.node.topic.DefaultPublisher$1 onMasterRegistrationSuccess
INFO: Publisher registered: Publisher<PublisherDefinition<PublisherIdentifier<NodeIdentifier</zona_co
mun_1/jade_agents, http://127.0.0.1:43243/>, TopicIdentifier</rosout>>, Topic<TopicIdentifier</rosout
>, TopicDescription<rosgraph_msgs/Log, acffd30cd6b6de30f120938c17c593fb>>>>
Jul 24, 2020 2:16:16 AM jade.core.Runtime beginContainer
INFO: ----------------------------------
    This is JADE 4.5.0 - revision 6825 of 23-05-2017 10:06:04
    downloaded in Open Source, under LGPL restrictions,
    at http://jade.tilab.com/
----------------------------------------
Jul 24, 2020 2:16:16 AM jade.imtp.leap.LEAPIMTPManager initialize
INFO: Listening for intra-platform commands on address:
- jicp://10.0.2.15:1099

Jul 24, 2020 2:16:16 AM jade.core.BaseService init
INFO: Service jade.core.management.AgentManagement initialized
Jul 24, 2020 2:16:16 AM jade.core.BaseService init
INFO: Service jade.core.messaging.Messaging initialized
Jul 24, 2020 2:16:16 AM jade.core.BaseService init
INFO: Service jade.core.resource.ResourceManagement initialized
Jul 24, 2020 2:16:16 AM jade.core.BaseService init
INFO: Service jade.core.mobility.AgentMobility initialized
Jul 24, 2020 2:16:16 AM jade.core.BaseService init
INFO: Service jade.core.event.Notification initialized
Jul 24, 2020 2:16:17 AM jade.core.AgentContainerImpl joinPlatform
```

Ilustración 76. Salida de terminal de roslaunch con creación nodo CORE, nodo de zona y de creación de agente coordinador de zona.

En la Ilustración 77 vemos como el nodo crea los publishers necesarios para transmitir los datos a otros nodos ROS (dispositivos IoT) de su zona, la cual ha sido definida por su namespace.

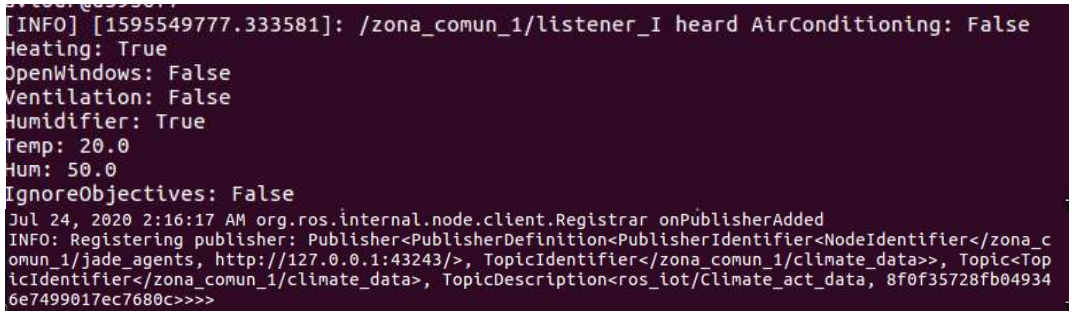


Ilustración 77. Creación de publishers.

Con esta prueba hemos podido verificar la creación de los elementos, así como la generación de peticiones dentro de ROS, su transformación en una petición para los agentes de la plataforma JADE, la devolución de la información solicitada al microagente ROS y así poder realizar las acciones a través del envío de órdenes a los nodos IoT.

4.4.5.4 Pruebas de Sistema.

El objetivo de la prueba era verificar el sistema al completo, desde el punto de vista de comunicación y comportamiento. Se simplificaron las reglas de actuación para facilitar la observación del funcionamiento de los distintos elementos. Para el agente de zona se fijaron 3 valores objetivo: temperatura, humedad y luminosidad. Estos objetivos serán mandados por el agente de planificación. El agente de zona solicitará los datos climáticos del exterior al agente de datos astronómicos y climáticos, y con todo ello actuará sobre los elementos los IoT de su zona. Tendremos 2 categorías de nodos IoT: climáticos (calefacción, apertura de ventanas, ventilación mecánica) y de iluminación. (toldos, persianas, bombillas). Además de esto, existirá un mecanismo de respuesta a una alerta de seguridad. En esta prueba los elementos actuadores de climatización se simularon en la placa Arduino con leds que indicaban las acciones solicitadas, mientras que para los elementos de iluminación se utilizaron bombillas Xioami con protocolo OpenHAB.

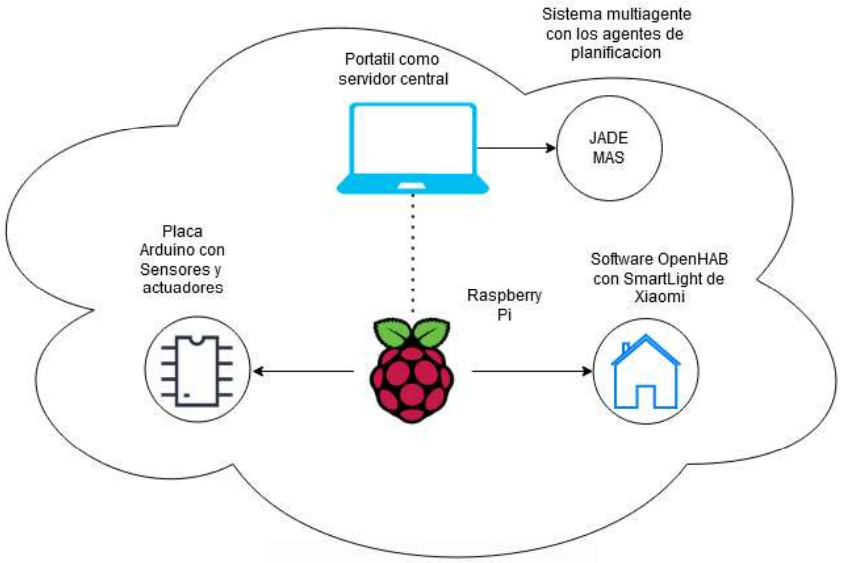


Ilustración 78. Entrono de pruebas del sistema con Raspberry Pi como agente de zona.

El agente de alto nivel de ahorro energético será el que decida si se pueden utilizar los elementos del exterior atendiendo a la ocupación de la habitación y a las condiciones externas (velocidad del viento, lluvia, etc).

El mecanismo de seguridad se puede disparar por los sensores locales o por peticiones del agente de alto nivel encargado de la seguridad. Para simular un mensaje de alerta hemos utilizado un DummyAgent, pudiendo comprobar que efectivamente el mensaje de alerta llega al agente de zona, y esta manda la respuesta de actuación de forma instantánea elementos IoT que controla. De esta forma demostramos que capacidad reactiva del agente y la prioridad del módulo de competencia de seguridad frente al resto.

Para empezar, ejecutamos el sistema multiagente (desde el PC), y el nodo de zona desde la Raspberry. Si miramos dentro de JADE, podemos ver que se ha creado una nueva plataforma con un agente que correspondería al generado agente coordinador de zona, donde cada una tiene un segmento de red distinto (10.0.2.0/24 y 192.168.1.0/24) respectivamente. También podemos observar como el agente de zona desencadena un flujo de información entre los distintos agentes del sistema (Ilustración 79).

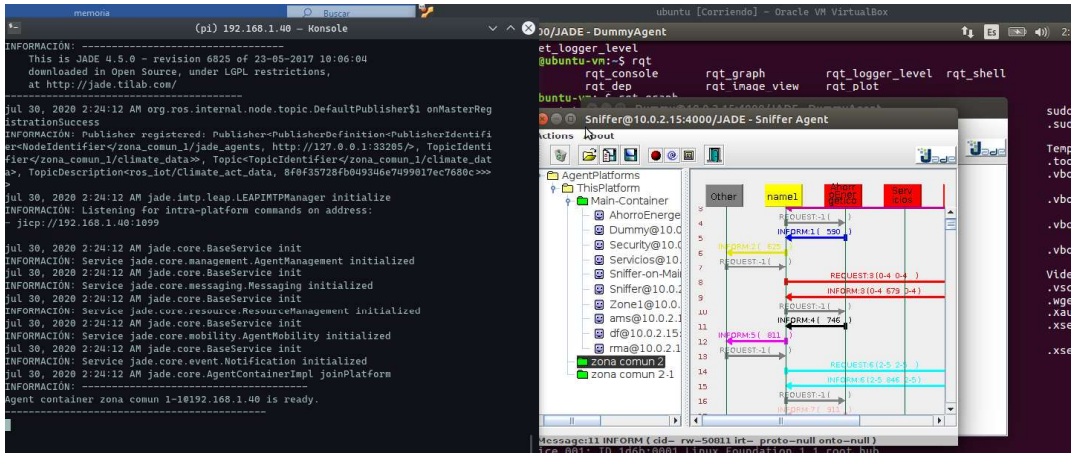


Ilustración 79. Resultado del lanzamiento de la plataforma de agentes.

A bajo nivel observamos como el agente de zona recibe la información desde el Gateway y opera sobre distintos dispositivos IoT a través de nodos ROS consiguiendo los objetivos previstos. En el grafo generado por ROS de la Ilustración 80, podemos ver los nodos creados en un mismo espacio de nombres (zona_comun_2) que genera ROS, podemos ver como en esta ocasión todos los nodos existen dentro del mismo espacio de nombres, y son gestionados por un único agente.

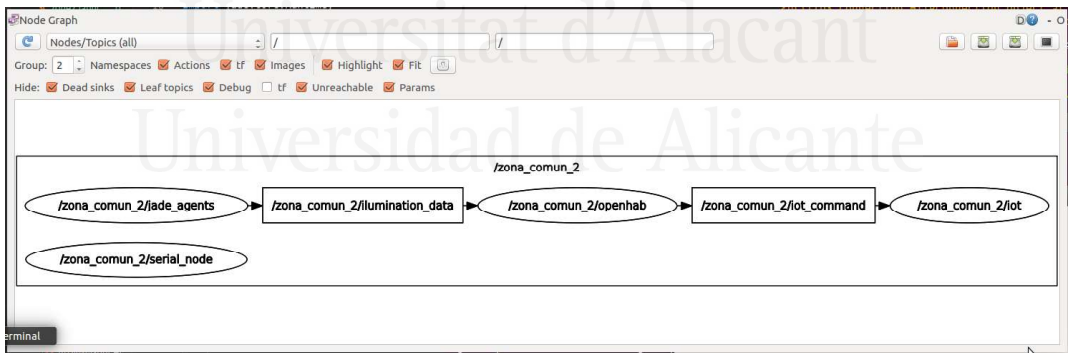


Ilustración 80. Grafo de ROS de una zona con varios nodos.

Por último, en la Ilustración 81, vemos como existen los dos topics y podemos ver por terminal como los mensajes se están transmitiendo por un mismo agente, demostrando así que es posible controlar varios tipos de nodos IoT con un único agente.

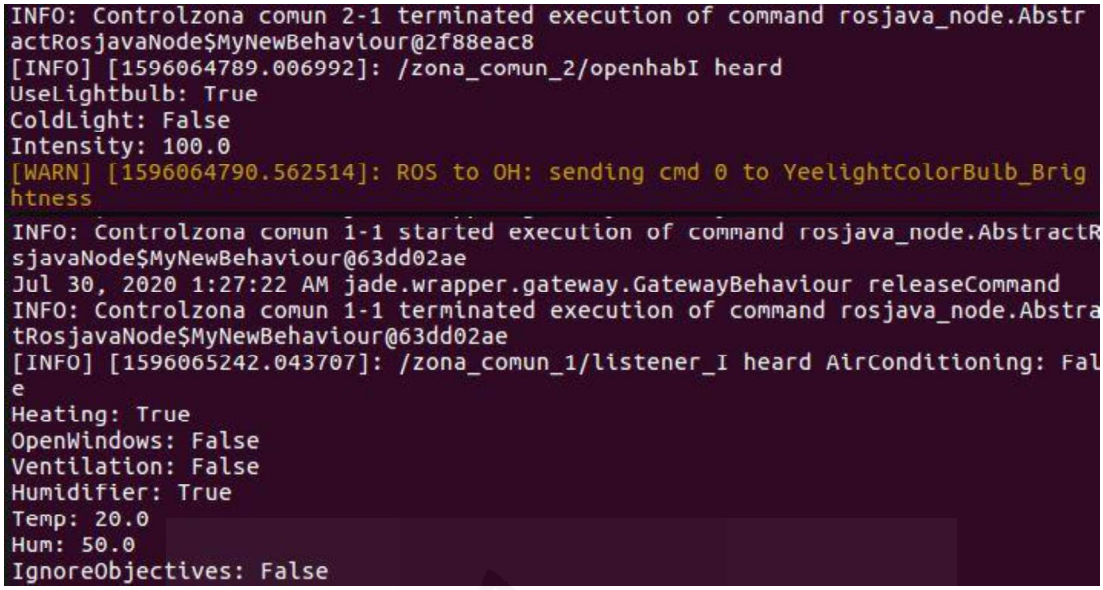


Ilustración 81. Comunicación con varios elementos IoT dentro de un agente de zona.

4.5 Despliegue.

En nuestro caso el despliegue en JADE es sencillo ya que lo vamos a hacer sobre una única plataforma de agentes y el propio framework nos facilita la tarea a través de un "launch" y ofrece una interfaz gráfica de gestión de la plataforma (Remote management Agent GUI). Si el despliegue queremos hacerlo en distintas plataformas (contenedores) sería posible porque JADE lo permite, pero ya sería algo más laborioso.

ROS incluye la herramienta Roslaunch para facilitar la ejecución de sistemas con múltiples nodos. La configuración se hace a través de lenguaje XML, pudiendo así configurar los nombres y los parámetros de los nodos de una manera sencilla.

Siguiendo el diagrama de arquitectura, hemos configurado una plataforma ROS para cada zona que integra los distintos elementos IOT de esa zona., aunque en las pruebas iniciales creamos una única plataforma ROS donde independizábamos las distintas zonas a través del uso de namespaces" de ROS.

En la Ilustración 82 podemos ver un sencillo ejemplo donde definimos una zona del hotel con un nodo IoT encargado de mostrar por pantalla información para la depuración del programa.

```
<launch>
    <group ns="zona_comun_1">
        <node pkg="rosjava_pkg_a" type="execute" name="jade_agents" args="rosjava_node.RosjavaNodeArd">
            <param name="agent" value="name1"/>
            <param name="zone_name" value="zona comun 1"/>
        </node>
        <node pkg="ros_iot" type="listener.py" name="listener_" output="screen">

        </node>
    </group>
</launch>
```

Ilustración 82. Roslaunch simple con dos nodos.

4.6 Resultados.

Una vez realizadas las pruebas de viabilidad del sistema podemos concluir que se han conseguido los objetivos propuestos.

Se ha creado un modelo que permite diseñar y construir sistemas multiagente con las siguientes características:

- Fácil de entender e implementar: tiene 2 niveles claramente diferenciados el alto nivel (agentes) y el bajo (sensores, actuadores y comportamiento primarios).
- Cumple con la normativa FIPA.
- Abierto. Compatible a bajo nivel con sensores y actuadores genéricos.
- Escalable. Hay que destacar que el modelo es altamente escalable no solamente en número de elementos, sino en el tipo de los mismos.
- La construcción de los sistemas es rápida, de bajo coste y alta calidad.

Desde el punto de vista de ahorro energético [76] [77] [81] nuestro modelo va a hacer que los porcentajes de ahorro para cada una de las medidas se sitúen en las cotas más altas o las incremente.

En la Ilustración 83, hemos utilizado la clasificación de gasto energético que veíamos en el apartado 4.1, donde el gasto energético se repartía entre climatización (45%), agua corriente sanitaria (23%), iluminación (15%), lavandería y cocina (12%) y otros usos (5%), mostrando cada uno de ellos en un color distinto. Se ha añadido una tercera columna donde indicamos si estos

consumos corresponden a energía eléctrica o a térmica. En la columna 4 y 5 indicamos las medidas de ahorro utilizadas en la actualidad con los rangos de ahorro que se pueden conseguir, mientras que en la 6 y 7 mostramos las mejoras estimadas que incorporaría nuestro sistema multiagente. En los siguientes párrafos vamos a ir explicando estas mejoras.

Dentro de climatización en la última década se ha reformado el sistema de calefacción mediante el precalentamiento de agua a través de placas solares. La posición de estas placas es fija, orientando e inclinando las mismas a su posición de maximización de ahorro. Nuestra mejora viene dada en dos sentidos: hacer que las placas no sean fijas y que su movimiento venga dirigido por la información de otros sistemas. El sistema de información clave sería el astronómico que nos iría indicando la orientación e inclinación óptima de las placas atendiendo a la estación, a la hora y a la ubicación. Así mismo, también contaríamos con un sistema de supervisión del rendimiento ya que dependiendo de los elementos estructurales el rendimiento podría cambiar debido a las sombras. Se estima que de esta forma no sólo estaríamos incrementando sobre un 5% el rendimiento de los paneles, sino que las probabilidades de cumplir con este rango de mejora serían mayores.

Otra de las mejoras en climatización esta la regulación por sectores. Tradicionalmente estos sectores se regulan manualmente o atendiendo a franjas horarias. La idea clave en la mejora propuesta ha sido introducir el concepto de tarea o uso de la estancia. Atendiendo al uso que se vaya a hacer podemos regular automáticamente distintos elementos. En este caso también estimamos que el ahorro puede estar en torno a un 5% adicional suponiendo que las zonas se están regulando correctamente manualmente y/o por horarios, cosa que muchas veces no ocurre de forma que de nuevo podemos estar hablando de un ahorro real mucho mayor; entre un 20-35% respecto al hacerlo sin ningún tipo de regulación o de un 15% adicional respecto a una mínima regulación.

Otro factor que hemos explotado ha sido a través del sistema de reservas y del sistema de detección de presencia. El sistema de reservas nos informa del estado de la habitación (desocupada, reservada, ocupada) no solamente en el momento actual sino que nos permite hacer previsiones. Aquí también jugamos con una información importante el número de huéspedes. Toda esta información va a permitir regular de una manera más eficiente nuestros sistemas frente a sistemas tradicionales los cuales detectan simplemente la presencia en la habitación o no a través de mecanismos simples como puede ser la típica tarjeta de presencia que funciona como un interruptor. En este caso podríamos tener ahorros de hasta un 10%.

Para terminar con el apartado de climatización hemos automatizado elementos de protección (toldos, persianas, cortinas, ventanas, etc.). Gracias a los datos de ocupación, presencia, climáticos podemos hacer que estos sistemas

tradicionalmente pasivos y manuales adquieran un carácter dinámico consiguiendo otro 5% adicional de ahorro frente a los tradicionales.

Respecto a las filas en color azul, agua corriente sanitaria, de nuevo interviene el posicionamiento dinámico de las placas solares como veíamos en los párrafos anteriores. Aquí sin embargo podemos observar que el ahorro es mayor que en climatización ya que su uso es a lo largo de todo el año y es exclusivamente térmico, mientras que la climatización muchas veces es realizada o apoyada por bombas de calor eléctricas.

La segunda fila azul corresponde al ahorro estimado respecto a la ocupación. En este caso y utilizando de nuevo el sistema de reservas y presencia podemos ajustar la potencia de las bombas de agua, consiguiendo un funcionamiento más ajustado a las necesidades reales. De esta manera podemos hacer utilizar bombas de menores prestaciones y/o que trabajen a niveles más bajos, consiguiendo ahorros de un 5% sobre el 23% que teníamos.

En el tercer apartado, en color amarillo, nos centramos en los gastos derivados de la iluminación, la cual supone en torno a un 15% del consumo total. Aquí volvemos a utilizar la regulación a través de tareas y al uso de elementos tradicionalmente pasivos como persianas y cortinas.

La penúltima fila hace referencia a los elementos de lavandería y cocina. Para el ahorro de los mismos vamos a realizar ajustes mayores gracias a tener mejores previsiones y datos reales sobre la ocupación, así como hábitos y uso de las instalaciones. Aquí es difícil estimar la mejora introducida, pero fácilmente podríamos hablar de un 10% sobre el 12% de esta partida.

Respecto a los ascensores podemos incorporar comportamientos más inteligentes en lugar de la programación fija tipo bus o mixta utilizada actualmente. De esta manera podemos hacer por ejemplo que en determinados horarios y dependiendo de la ocupación del hotel el modo de los ascensores cambie. También se podría hacer uso del número de personas que están en el ascensor para evitar paradas y arranques innecesarios. Además de esto, se puede hacer un uso coordinado de los distintos ascensores de los que consta el establecimiento. Con todo esto podríamos estar hablando de un ahorro entorno a un 10% sobre el gasto en ascensores.

Además de los datos recogidos en Ilustración 83, utilizando técnicas similares, podemos incidir en los sistemas de generación de energía, como por ejemplo con las medidas de orientación e inclinación de placas solares fotovoltaicas utilizadas para proporcionar energía eléctrica al establecimiento.

Categoría Gasto	% Consumo	Fuente de energía	Técnica utilizada	Mejora incorporada	% Ahorro con técnicas tradicionales	% Ahorro estimado con sistema multiagente
Climatización	45%	Térmico y eléctrico	Placas solares para climatización con calderas, rendimiento maximizado para posición fija.	Inclinación y orientación dinámica atendiendo a fecha, hora y elementos estructurales.	20-30%	20-35%
		Térmico y eléctrico	Sectorizando, sistemas autónomos, regulación ventiladores y bombas.	Regulación atendiendo a tareas	20-30%	20-35%
		Eléctrico-Térmico	Tarjeta de control de presencia.	Control de habitaciones (desocupada, reservada u ocupada)	5%	10%
		Eléctrico-Térmico	Protecciones solares.	Uso dinámico de protecciones y free-cooling (toldos, persianas, ventanas) atendiendo a climatología, datos astronómicos y ocupación.	10% sobre climatización	15%
Agua Caliente Sanitaria	23%	Térmico	Utilización de placas solares para producción ACS. Rendimiento maximizado para posición fija.	Inclinación y orientación dinámica atendiendo a fecha, hora y elementos estructurales.	80%	90%
		Térmico y Eléctrico (bombas presión)	Datos estadísticos de ocupación.	Datos reserva y ocupación (número huéspedes). Comportamiento dinámico de bombas.	20-30%	20-35%
Iluminación	15%	Eléctrico	Sistemas de regulación y control.	Regulación atendiendo a tareas	20-30%	20-35%
		Eléctrico	-	Apertura de persianas y cortinas. Uso de protecciones atendiendo a climatología, datos astronómicos y ocupación.	-	5%
Lavandería y cocina	12%	Térmico y Eléctrico	Ocupación	Datos de reservas	-	10%
Otros	5%	Eléctrico	-	Comportamiento dinámico ascensores	-	10%

Ilustración 83. Ahorro energético estimado aplicando distintas estrategias.

Además de tener mejoras desde el punto de vista energético, también obtenemos muchas ventajas más como:

- Ahorro en los costes de inversión de los elementos de bajo nivel, permitiéndose dispositivos más sencillos al delegarse los comportamientos en los agentes de zona y o agentes de control.
- Reducción de costes y facilidad de implantación en los sistemas de gestión y BMS.
- Conocimiento más preciso de las demandas sobre los sistemas.
- Información más precisa sobre la ocupación (número de personas en las distintas estancias)
- Detección precoz de averías y creación automática de partes de mantenimiento.
- Mejor Confort para el cliente. El ambiente se puede personalizar más al cliente y tener preparada la habitación a sus gustos.
- Configuración más sencilla de los espacios respecto a actividades (ver tv, leer, dormir, arreglarse, etc.), actuando sobre los diversos elementos de las estancias.
- Cambio de comportamiento dinámico en elementos atendiendo a circunstancias externas. Por ejemplo, en el caso del sistema de ascensores podríamos cambiar del modo autobús al modo taxi de forma inmediata y sin costes adicionales, adaptándonos a situaciones especiales como ha ocurrido con el COVID-19.

4.7 Resumen del Capítulo.

En este capítulo hemos aplicado el modelo defendido en esta tesis a un sistema multiagente de un establecimiento hotelero con el fin de ahorrar energía, mejorar el confort y la seguridad, y facilitando las tareas de mantenimiento. Hemos seguido la metodología propuesta e introducido nuestro, dividiendo así la implementación en dos capas claramente diferenciadas: una de alto nivel a nivel de agentes y otra de bajo nivel con los elementos IoT (sensores, actuadores y comportamiento primarios). Desde el punto de vista de implementación hemos utilizado el framework JADE para implementar la parte del sistema multiagente y ROS para unificar la parte de los elementos IoT y hemos creado un Gateway ROSJADE para comunicar ambas capas.

Gracias al modelo tenemos un sistema que cumple con los estándares de FIPA y compatible a bajo nivel con sensores y actuadores de distintos fabricantes, obteniendo un sistema escalable. Todo ello acompañado de construcción rápida, de bajo coste y alta calidad.

Desde el punto de vista de ahorro energético nuestro sistema integra mayor información (reservas, presencia, astrofísica, climáticos …) y gracias a la colaboración de agentes, nuestro modelo va a permitir realizar comportamientos más dinámicos e inteligentes de los sistemas, haciendo que los porcentajes de ahorro se incrementen. Y las mejoras no acaban aquí, sino que también mejoramos la generación de energía con las medidas de orientación e inclinación de placas solares fotovoltaicas.

5 Modelado de un Sistema Multiagente para la Detección de Señales RF de Alta Intensidad

En este capítulo vamos a detallar la arquitectura del sistema de enjambre robótico que anticipamos en el capítulo 3. Como en el capítulo anterior, con sistema del ahorro energético del hotel, aplicaremos la metodología propuesta para crear el sistema. Veremos cómo gracias a la arquitectura se pueden reutilizar muchos conceptos y componentes, facilitando en este caso el desarrollo de los sistemas robóticos de bajo nivel y los comportamientos microscópicos y macroscópicos del enjambre robótico.

Una característica de los sistemas de enjambre que lo diferencian de los sistemas multiagente, propiamente dichos, es que están compuestos de elementos simples, que no tienen que estar conectados explícitamente. Este esquema elimina las dependencias de fallos de elementos centrales, pero añade complejidad a los algoritmos, teniendo que definir comportamientos microscópicos a nivel de robots y un comportamiento macroscópico a nivel de sistema.

El objetivo de este enjambre robótico va a ser la localización de señales de radiofrecuencia de alta intensidad en entornos urbanos. Estas señales suelen tener mayor intensidad en el centro de las ciudades, pero existe el riesgo de saturación. Las radiaciones son amplificadas o minimizadas dependiendo de la orografía y de las edificaciones. Los proveedores de señales necesitan ofrecer suficiente cobertura, configurando distintas antenas, con el fin de ofrecer un servicio mínimos de calidad.

5.1 Análisis de requerimientos.

5.1.1 Descripción del Dominio.

5.1.1.1 La robótica de Enjambre.

Dentro del área de la robótica tenemos el campo de investigación de la robótica de enjambre que estudia la coordinación de un conjunto de muchos robots simples. Se inspira en la observación de la naturaleza, y en concreto en el comportamiento de los insectos sociales, donde un gran número de individuos simples interactúan creando sistemas colectivos inteligentes. A partir de las interacciones de los individuos, y de los individuos con el entorno, emerge un comportamiento colectivo de forma autoorganizada.

De forma similar a la naturaleza, en los sistemas de enjambre artificiales, la inteligencia emerge a partir del comportamiento global del enjambre. En estos sistemas, el enjambre es capaz de realizar tareas, de una forma global, que estarían fuera del alcance de un robot individual [82].

Los sistemas robóticos de enjambre se diferencian de otros sistemas multirobóticos en una serie de características, de las cuales algunas son compartidas con los sistemas multiagente. Debido a esto, los sistemas multiagente pueden ser una alternativa para la construcción de enjambres artificiales, permitiendo, lo que conlleva una serie de ventajas como la reutilización de código, la conectividad y la estandarización.

La robótica de enjambre pone el foco en aspectos como el control descentralizado, la comunicación limitada entre agentes, información local, la aparición de comportamiento global y la robustez. Los sistemas multirobóticos de enjambre se diferencian del resto de sistemas multirobóticos por las siguientes características:

- Los robots que conforman el enjambre son robots autónomos situados en el entorno.
- Un gran número de robots forma el enjambre.
- El enjambre se compone de pequeños grupos de robots homogéneos.
- Los robots serán relativamente simples.
- Los robots tendrán sensores locales.
- Los robots tendrán capacidades de comunicación limitadas.

Estas características aseguran una coordinación entre los robots de forma distribuida y una alta tolerancia a fallos del sistema a través de la redundancia

de robots, siendo cada uno de los agentes prescindible o pudiendo ser sustituido por otro agente. En este mismo sentido, el sistema es fácilmente escalable, permitiendo añadir o eliminar más agentes según sea necesario. Debido a estas características es complicado desarrollar una arquitectura que sea capaz, por una parte, de modelar de manera correcta un sistema de enjambre y por otra, que sea capaz de coordinar el enjambre para llevar a cabo tareas complejas.

En los sistemas de enjambre hay que destacar dos tipos de comportamientos, el comportamiento microscópico y macroscópico. El enjambre se compone de un gran grupo de individuos, especializados en tareas sencillas, los cuales tienen un comportamiento microscópico definido en términos locales. A través de este comportamiento, microscópico de los individuos de un grupo, emergen patrones de comportamiento en el nivel más alto del sistema, que es lo que denominamos comportamiento macroscópico. En otras palabras, definimos el comportamiento de partes simples del sistema y esperamos a que el resultado se muestre en un comportamiento global deseable.

Es estos sistemas es muy habitual contar con simuladores de entorno, individuos y comportamiento, y en nuestro caso concreto necesitaremos hacer uso de modelos y simuladores de RF.

En la Ilustración 84 podemos ver los casos de uso identificados durante el análisis de requerimientos. Hacer notar, aunque en principio no identificamos la funcionalidad "entrenar entorno" al hacer la iteración de comportamientos vimos la necesidad de la misma y la incorporamos.

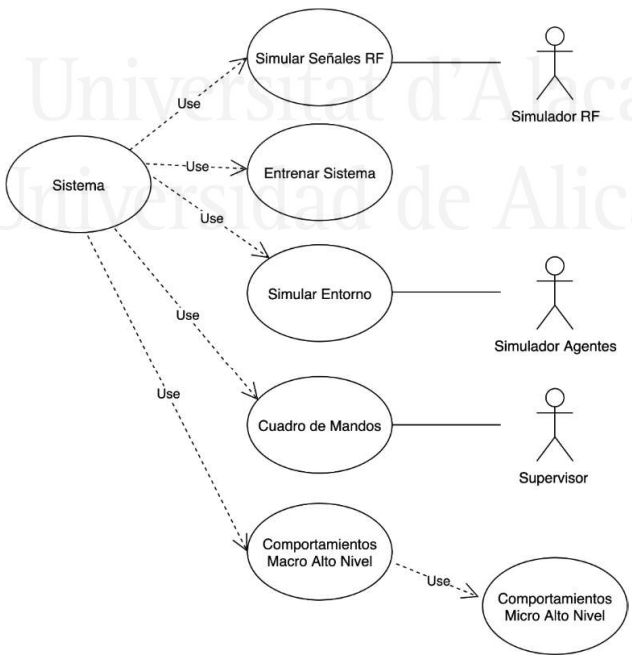


Ilustración 84. Casos de uso Enjambre Robótico detección señales RF.

5.1.1.2 Presencia de Señales RF en Entornos Urbanos.

En los entornos urbanos, hay muchas ondas electromagnéticas de alta y baja frecuencia. Las señales de baja frecuencia corresponden a transformadores, líneas eléctricas, etc., mientras que las de alta frecuencia corresponden a redes de telefonía, redes inalámbricas, radio, televisión entre otras. A pesar de que las señales de radiofrecuencia (RF) trabajan en distintas longitudes de onda, se producen interferencias entre las señales, bajando su velocidad de transmisión. Además de esto cuando varios usuarios utilizan el mismo canal simultáneamente, como por ejemplo en una red inalámbrica, el rendimiento baja ya que el acceso al medio empieza a ser ineficiente. La calidad de servicio de las señales RF está influenciada por diferentes circunstancias (geometría de la atenuación, condiciones atmosféricas, construcciones, etc.), existiendo diferentes modelos para predecir la atenuación [83], [84] . En todos los casos los modelos teóricos para elegir las antenas y sus características deben de ser validados utilizando test de campo para evaluar efectos no deseados [85]. En general, encontramos que las redes RF están expuestas a una gran fuente de ruidos producidos por distintos elementos, los cuales pueden causar un gran impacto en el nivel de interferencias de la red. En consecuencia, para mantener el rendimiento de la red debemos de utilizar determinados niveles de potencia para asegurar los enlaces, pero muchas veces utilizamos niveles más altos de lo necesario, consumiendo mayores recursos y generando mayores ruidos en otras señales. Nuestro trabajo va a ir orientado a determinar la cobertura de calidad de una señal RF móvil en una zona urbana, identificando las zonas donde la señal llega con mayor potencia. Con estos resultados podremos determinar si el nivel de señal es excesivo o no en las áreas identificadas.

5.1.2 Identificación de Agentes.

Al igual que en el capítulo anterior, vamos a cubrir uno de los primeros pasos de la nuestra metodología a través de un diagrama (Ilustración 85) que muestra los agentes identificados en el sistema.

A nivel más bajo hemos diseñado nuestro sistema con agentes específicos (microagentes) para gestionar comportamiento básico (comportamiento microscópico) de los elementos del enjambre. En siguientes secciones veremos el diseño interno de estos agentes microscópicos los cuales serán los encargados finales de actuación en los drones.

Los agentes macroscópicos que se encargarán del comportamiento deseado del enjambre a alto nivel (comportamiento macroscópico).

Debido al requerimiento de descentralización de los sistemas de enjambre se ha pensado en utilizar unos agentes a los cuales hemos denominado "agentes comportamiento de macroscópico de nivel medio" cuya misión será independizar

en la medida posible los microagentes del resto del sistema. Los agentes macroscópicos de alto nivel dividirán el problema en partes, es decir lo serializarán, y pasarán estos subproblemas a los de nivel medio los cuales organizarán por grupos a los microagentes con el fin de cumplir cada uno de los subobjetivos.

Además de estos agentes contamos con los típicos agentes de páginas blancas y amarillas, y del agente encargado de gestionar la plataforma de agentes. Estos facilitarán el acceso a agentes y a servicios, y mantendrán la operativa del conjunto de agentes. De ellos ya hablamos en la sección 2.2 donde describíamos con detalle el standard FIPA.

Igual que en el caso del Ahorro Energético, se ha incorporado al sistema un agente que hace de interface de usuario con el fin de encapsular toda la interacción y supervisión del sistema.

Por último, hemos pensado en dotar al sistema con un agente de simulación que encapsulará la simulación de elementos del sistema por un lado y de simulación de comportamientos de las ondas RF.

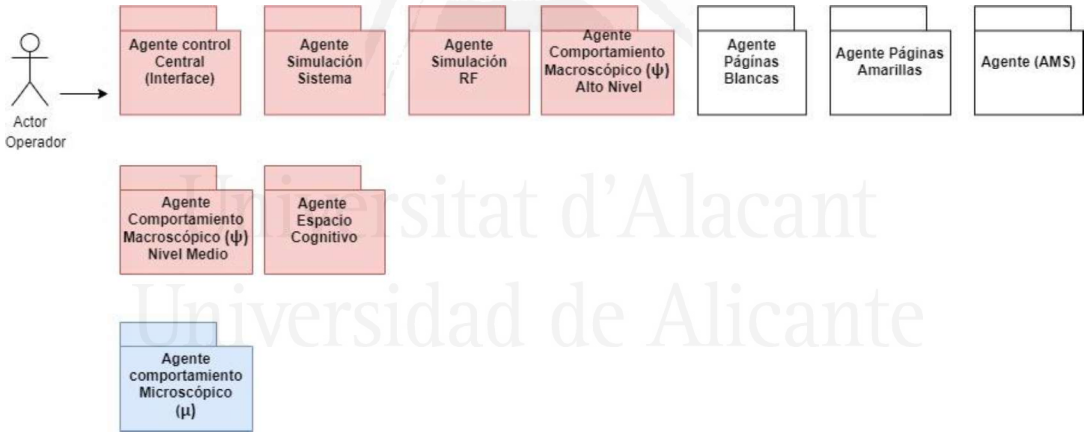


Ilustración 85. Diagrama de Identificación de Agentes para un Sistema de Detección de Señales de Radiofrecuencia.

5.1.3 Identificación de Roles.

5.1.3.1 Rol Microscópico (μ)

El rol microscópico (μ) lo desarrollan aquellos agentes que se encargan de interactuar con los robots físicos y generalmente irá embebido en el propio robot. Se tratarán por tanto de agentes que posibilitarán el desarrollo de tareas básicas

de enjambre. Podemos por tanto considerar que este tipo de agentes desarrollarán tareas de enjambre, cercanas a la reactividad. Este tipo de agentes, una vez conocida su dirección y la de los agentes μ con los que interactúe, puede comunicarse de manera directa, sin pasar por ningún agente de control. Esto puede ser útil en situaciones en las cuales no se tenga acceso por completo a la plataforma de agentes o cuando no se quiera sobrecargar el sistema, estableciendo únicamente comunicaciones punto a punto.

Los agentes de rol μ sólo se podrán comunicar con agentes de su mismo tipo. No obstante, proporcionarán acceso a los datos básicos de la tarea de enjambre desarrollada y al estado de la misma, de manera que los agentes de rol macroscópico (ψ) puedan monitorizar y dirigir dichas tareas.

En la Ilustración 86 podemos ver el diagrama de casos de uso para el rol de comportamiento microscópico de los robots del enjambre robótico, básicamente recibirá parámetros de configuración, realizará funciones de moverse por el entorno evitando obstáculos atendiendo a un comportamiento programado y devolverá información recogida del entorno.

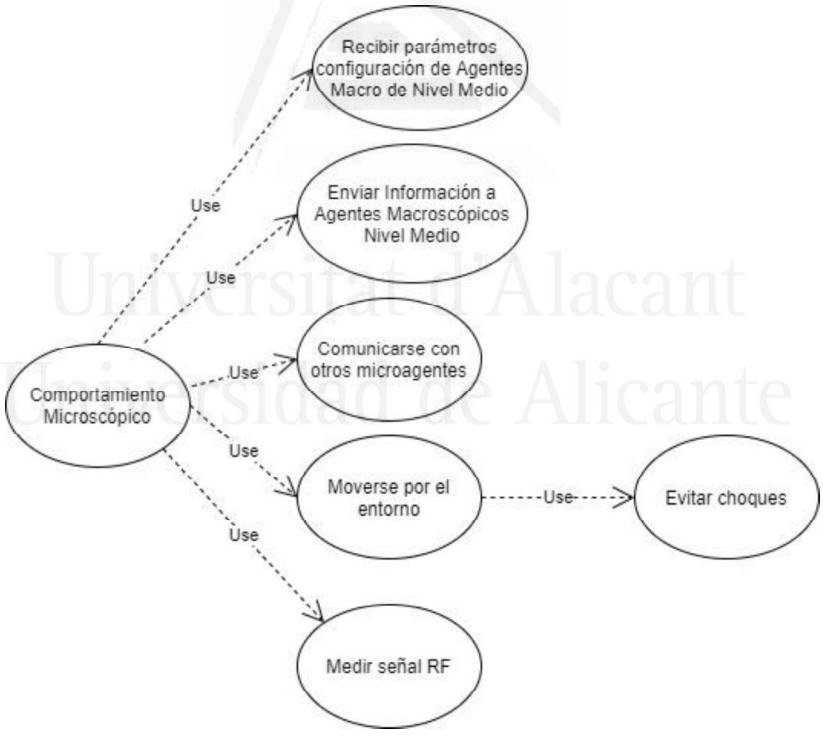


Ilustración 86. Casos de Uso Rol Comportamiento Microscópico.

5.1.3.2 Rol Macroscópico (ψ)

El rol macroscópico (ψ) lo desarrollarán agentes cognitivos encargados de los comportamientos macroscópicos del enjambre y de dirigir al enjambre hacia su meta. Los agentes cognitivos son móviles, y pueden situarse físicamente en cualquiera de los robots de enjambre disponibles. La suma de todos los agentes ψ debe ser mucho menor que la de los agentes μ, ya que la coordinación recaerá en unos pocos agentes, comparado con el montante del enjambre.

Un sistema de enjambre puro no debe tener ningún agente coordinador, ni ningún módulo centralizado que se encargue del control de los agentes. No obstante, cuando se desean desarrollar tareas de complejidad media o alta es necesario proveer mecanismos que permitan dirigir al enjambre de manera que su interacción lleve al sistema a realizar la tarea requerida.

Los agentes de rol ψ pueden ser de medio o alto nivel. Los agentes ψ de medio nivel son los encargados de dirigir al enjambre de agentes μ para realizar una determinada tarea a nivel macroscópico. Para ello se modificarán los parámetros de los agentes μ individuales, de manera que la interacción entre ellos genere el nuevo comportamiento macroscópico requerido. Estos agentes se pueden comunicar entre agentes del mismo rol o con agentes de rol μ, para monitorizar y dirigir su funcionamiento.

Los agentes ψ también pueden desarrollar tareas de alto nivel, que coordinen a los agentes macroscópicos de nivel medio hacia su meta. Este tipo de agentes únicamente se comunicarán entre ellos para desarrollar sus funciones (no podrán comunicarse con agentes de tipo μ).

Por último, algunos de los agentes del rol verificarán continuamente la existencia de un agente especial, llamado espacio cognitivo. Será su misión el recrear este agente en caso de que desaparezca del sistema.

En las ilustraciones Ilustración 87 y Ilustración 88 podemos ver los diagramas de casos de uso asociados a los roles de comportamiento macroscópico de nivel alto y medio respectivamente.

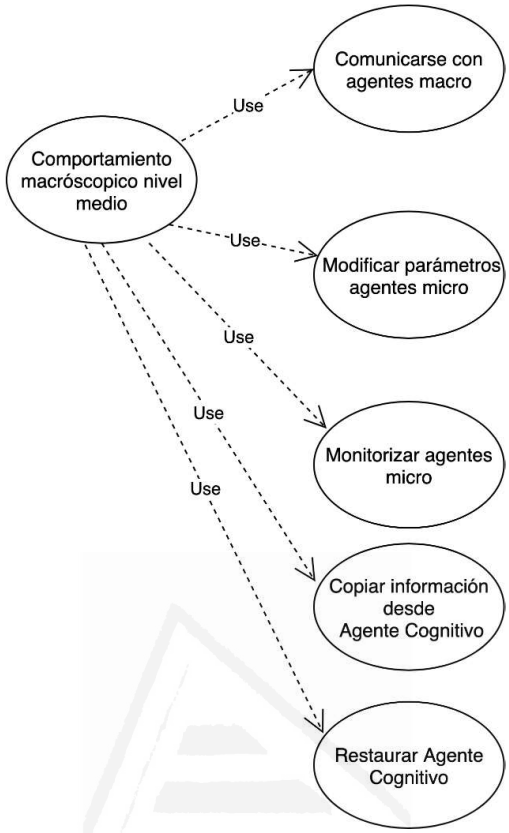


Ilustración 87. Diagrama de Casos de Uso de Rol de Comportamiento Macroscópico de Alto Nivel.

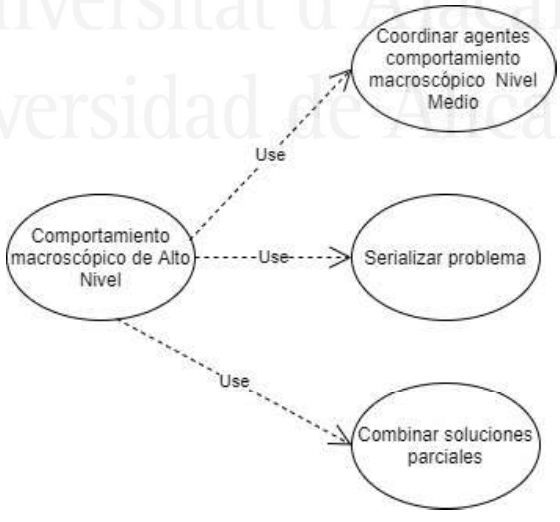


Ilustración 88. Diagrama de Casos de Uso del Rol Comportamiento Macroscópico Nivel Medio.

5.1.3.3 Rol Espacio Cognitivo (CS).

Dada la importancia de los agentes macroscópicos (ψ), se debe asegurar su permanencia y evitar a toda costa su desconexión. No se puede establecer un control centralizado, ya que un fallo en el mismo provocaría la pérdida de todo el enjambre. Existen varios mecanismos, como la replicación que proporciona una solución para la tolerancia a fallos de partes concretas del sistema. No obstante, dadas las características especiales de la robótica de enjambre, incluso la replicación de todos los agentes del sistema puede ser insuficiente, ya que, si un agente ψ y su réplica fallan, el sistema de enjambre dejaría de funcionar.

Es por ello, que se plantea la creación de un agente de espacio cognitivo, que se encargue de almacenar y gestionar el estado de todos los agentes ψ del sistema. La finalidad de este agente es triple:

- Por una parte, provee a todos los agentes cognitivos de un espacio común donde acceder a información compartida por todos. Esta información es tanto el modelo del mundo como el modelo cognitivo del sistema. Las arquitecturas robóticas orientadas a modelo suelen utilizar esta estructura.
- Por otra parte, realiza tareas de verificación para la tolerancia a fallos, verificando la existencia de todos los agentes ψ. Si un agente de este tipo cae, el agente espacio cognitivo se encarga de volverlo a crear. Como el contenido fundamental de todos los agentes cognitivos se encuentra actualizado en el CS la pérdida de información es mínima y la restauración del agente es inmediata. Este agente además deberá verificar la duplicidad de agentes que realicen la misma función para gestionar su eliminación.
- Además, realiza tareas de proxy, distribuyendo la información almacenada en él por la red de agentes cognitivos (gestiona una base de datos distribuida). De esta manera, en el caso más extremo, todos los agentes cognitivos dispondrán de una copia del contenido del agente CS, es decir, de toda la información importante de los agentes cognitivos, pudiendo restaurar todo el sistema cognitivo sin pérdidas de información, en caso de ser necesario.
- Todos los agentes de rol ψ sincronizados con el CS verificarán continuamente el estado del agente cognitivo. En caso de que este agente caiga, un agente ψ (por ejemplo, el de mayor Id) lo restaurará utilizando su copia interna del CS.

Para terminar de modelar los roles hemos creado el diagrama de casos de uso asociados al rol de mantener espacio cognitivo (Ilustración 89).

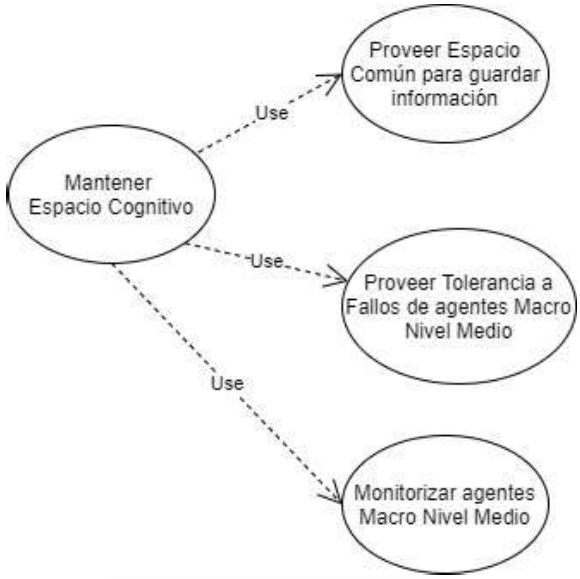


Ilustración 89. Diagrama de Casos de Uso de Rol Mantener Espacio Cognitivo.

A nivel de implementación es clave el soporte de replicación y recuperación de errores a la hora de la selección de los frameworks tanto de las plataformas de agentes como de robots. Es una de las razones por las cuales hemos elegido JADE y ROS.

1.1.1.1 Roles de Simulación.

Dentro de los sistemas de enjambre es muy habitual encontrar simuladores. Nosotros hemos preferido dividir la simulación en 2 roles: uno general para el sistema y otro para la simulación de radiofrecuencia específica de este problema. La idea es que estos roles vayan asociados a dos agentes diferentes, con lo cual se simplificará y encapsulará mejor la implementación, y se facilitará su reutilización en otros proyectos.

En la Ilustración 90 podemos ver el diagrama de casos de uso del Rol Simulador del Sistema. En el podemos ver que este rol será el encargado de simular el entorno por el que van a moverse los agentes y simular los agentes, en nuestro caso los microagentes de bajo nivel.

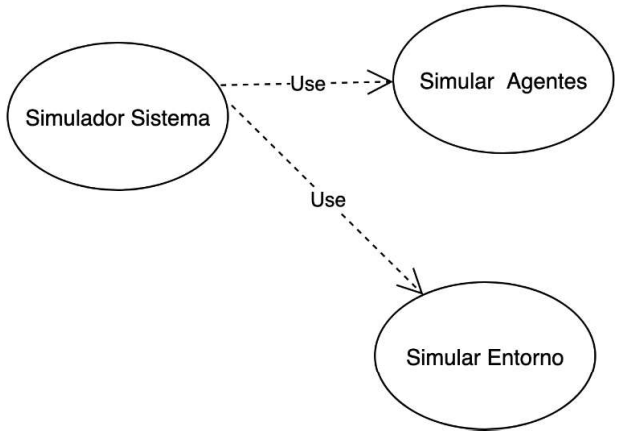


Ilustración 90. Diagrama de Casos del Rol de Simulador de Sistema.

Por otro lado, en la Ilustración 91 vemos los casos de uso asociados al rol de Simular señales de Radiofrecuencia. Al principio no detectamos la funcionalidad entrenar sistema, la cual fue identificada en la fase de codificación. Esto nos obligó a ampliar el rol, aunque más tarde tomamos la decisión de dividir el rol en dos y asociar cada funcionalidad a un agente diferente. En el apartado 5.4.2 se describe esta reflexión sobre el diseño.

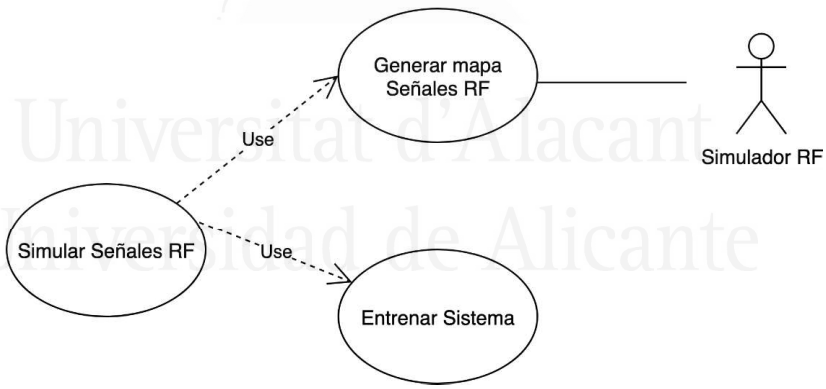


Ilustración 91. Diagrama de Casos del Rol de Simulación Radiofrecuencia.

5.1.4 Especificación de Tareas.

A la hora de especificar las tareas lo vamos a hacer a través de diagramas de interacción.

En la Ilustración 92, podemos ver cómo es la comunicación entre los agentes para la detección de señales de radio frecuencia (comportamiento

macroscópico de alto nivel), es decir el comportamiento del enjambre para la detección de las señales de radiofrecuencia. En el vemos que el agente ψ de alto nivel solicita al agente de páginas amarillas (DF) la dirección del agente simulador de radiofrecuencia. Una vez obtenida su dirección, le solicita que generé el mapa de recursos de señales necesarios para las simulaciones. Lo mismo ocurre con el con el agente entrenador: primero se solicita su dirección y a continuación se demandan sus servicios. El entrenador devuelve los valores de probabilidad para cada uno de los estados deambular, descubrir y agrupar $(p_{deambular}, p_{descubrir}, p_{agrupar})$. El agente macro de alto nivel solicita las direcciones de los agentes macro de nivel medio, divide (serializa) el problema y delega la resolución a los agentes macro de nivel intermedio. Estos realizan las acciones sobre los microagentes y devuelven los resultados parciales, los cuales son unificados por el agente macro de alto nivel para obtener los resultados finales.

El agente de entrenamiento previamente ha sido entrenado usando varios mapas teóricos RF, generados con diferentes tamaños de entorno y diferente número de antenas con el fin de calcular los valores de probabilidad para la máquina de estados finitos probabilísticos que maximizaban la convergencia del enjambre.

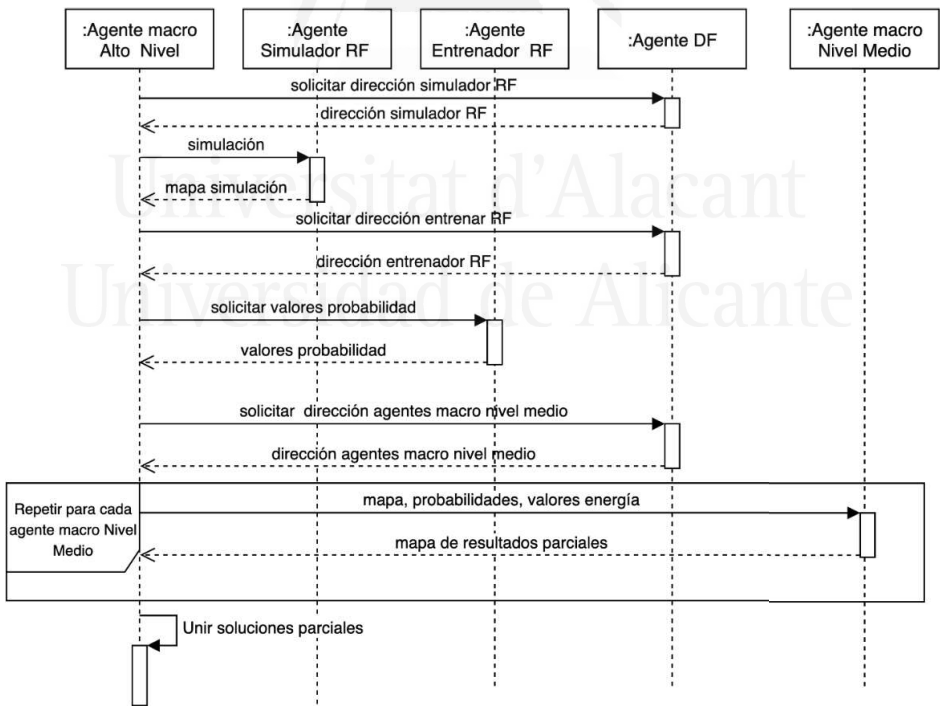


Ilustración 92. Diagrama de Interacción del Comportamiento Macroscópico Detección Señal Radiofrecuencia.

El detalle de interacción entre los agentes macroscópicos y los microagentes podemos verlos en la Ilustración 93. En el diagrama de la Ilustración 92, veíamos como el agente macroscópico de alto nivel dividía el problema en partes y se lo mandaba a los agentes macroscópicos de nivel medio. En la Ilustración 93 llega una de esas peticiones a uno de los agentes macroscópicos de nivel medio el cual realiza la tarea de resolver el subproblema (detectar las señales RF en una zona). Para ello envía a los microagentes que coordina la posición inicial, el estado inicial, las probabilidades para cada uno de los estados y los valores de energía. Con esta información los microagentes ejecutaran su comportamiento (microscópico) y devuelven la posición final, estado final y señal RF al agente macroscópico de nivel medio que lo coordina. Este a su vez recopila la información de los microagentes a su cargo y se la envía al agente macroscópico de alto nivel.

Ilustración 93. Diagrama de Interacción de Comunicación entre agentes Macroscópicos y Microagentes.

5.2 Diseño de sociedad de agentes.

1.1.1 Diseño de la Ontología del Dominio.

En esta etapa es donde vamos a aplicar la arquitectura propuesta. Ya en el apartado 3.5.2 avanzábamos en la aplicación y diseño de la arquitectura (Ilustración 44). En la Ilustración 94 vamos a repetir dicha ilustración solamente por simplificar la lectura y no obligar al lector a moverse a través del documento.

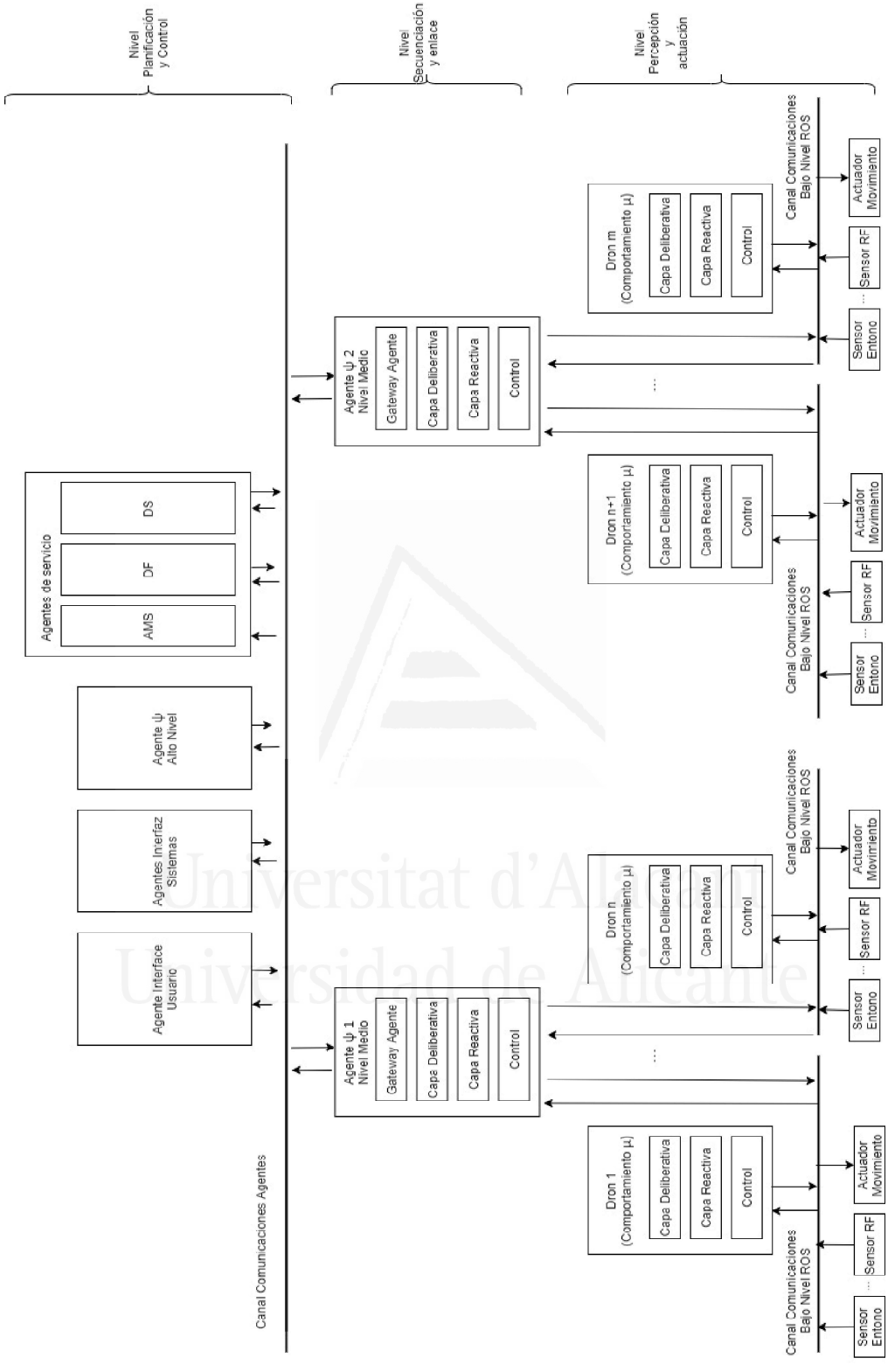


Ilustración 94. Arquitectura Híbrida Multinivel aplicada a la Detección de Señales de Radiofrecuencia a través de un Enjambre Robótico.

En el nivel alto tenemos los agentes que realizan el comportamiento macroscópico del enjambre, mientras que en el bajo nivel tenemos los robots físicos del sistema. En el nivel intermedio tenemos agentes los agentes de secuenciación que se encargan de dirigir a pequeños grupos de robots.

A alto nivel, a parte de los agentes de comportamiento macro, tendremos los agentes de control de la plataforma de agentes típicos, AMS, DF y DS; un agente que actuará de interface con sistemas de modelado y simulación de emisión y propagación de señales RF; y un último que servirá para que un usuario supervise e interactúe con el sistema.

Cada agente de bajo nivel (microagente) estará asociado a un robot (dron). En la práctica el software de microagente estará empotrado en cada uno de los drones. Cada uno de ellos recibirá señales del entorno a través de sus sensores y controlará los actuadores para moverse a través del entorno. Los robots tendrán un comportamiento muy sencillo moviéndose a través del entorno intentando localizar recursos y evitando colisiones, realizarán el comportamiento microscópico del enjambre.

Como agente de "espacio cognitivo" se usa el proporcionado por la plataforma JADE, encargándose de las funciones asociadas a ese ROL, proveyendo el canal de comunicación, la supervisión de los agentes de la plataforma (en este caso agentes macro de nivel medio) y la regeneración automática ante la caída de un agente macro.

Hacer notar que al estar los robots organizados en pequeños grupos con canales de comunicación diferentes, será fácil escalar el sistema.

A nivel de diagramas de clases, vamos a tener un esquema muy similar al del ahorro energético. Es interesante destacar la potencia de la arquitectura, permitiéndonos una reutilización de las clases definidas en el problema anterior. Las clases padre (agente, agente Gateway y microagente) las tendríamos ya implementadas y probadas, con lo cual todas las comunicaciones las tenemos operativas, teniendo que preocuparnos básicamente de los comportamientos.

En las ilustraciones 95 y 96 podemos ver los diagramas de clases que definen la ontología del dominio para los agentes de alto nivel y bajo nivel. En ellas podemos observar perfectamente que los agentes de alto nivel heredan de una clase padre "agente" mientras que los de bajo nivel heredan de otra clase denominada "microagente". De la clase "agente" heredan las siguientes clases:

- Agentes de Simulación: son agentes especializados en comunicarse con sistemas externos de simulación. En nuestro caso utilizaremos un sistema de simulación de ondas RF y un simulador de entorno.

- Agentes de Interfaz de usuario: hemos preferido dejar un tipo de agente para encapsular toda la comunicación con el usuario, englobando en él el cuadro de mandos del sistema.
- Agentes de Deliberativos: son los agentes encargados planificar y coordinar la consecución del comportamiento macroscópico.
- Agentes de Control de la Plataforma: dentro de esta categoría nos encontramos a los agentes que gestionar y facilitarán la plataforma de agentes.

Para el control de los elementos de bajo nivel hemos creado clases que cuelgan de la clase padre "microagente". Estos microagentes serán los encargados de realizar el comportamiento microscópico. Cada dron del enjambre será instanciado como un microagente en el sistema, el cual manejará los sensores y actuadores, al igual que lo hacían los microagentes del sistema de ahorro energético del hotel.

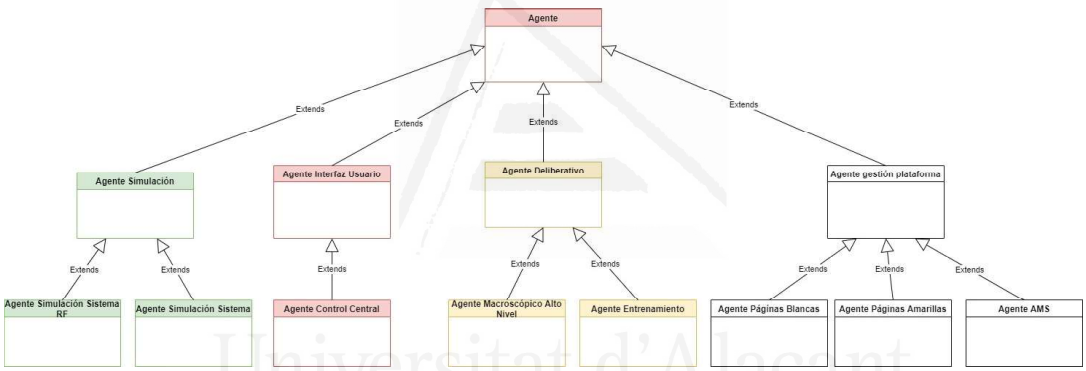


Ilustración 95. Ontología de Dominio de Agentes Alto Nivel.

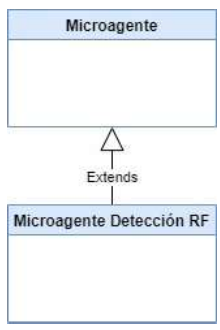


Ilustración 96. Ontología de Dominio de Agentes de Bajo Nivel (microagentes).

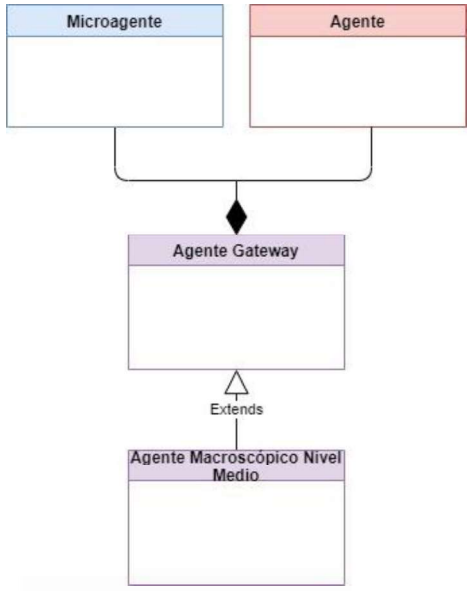


Ilustración 97. Ontología de Dominio Agentes de nivel medio.

La capa intermedia tendría una clase "agente Gateway" que sería una composición de las clases "agente" y "microagente". En nuestro caso el número de agentes intermedios dependerá del número de microagentes total que tenemos y del número de microagentes que queramos que controlen.

5.2.1 Diseño Ontología de Comunicaciones.

La comunicación entre los agentes se va a realizar mediante las especificaciones FIPA utilizando los lenguajes: SL ("Semantic Language"), CCL ("Constraint Choice Language"), KIF ("Knowledge Interchange Format") y RDF ("Resource Descripction Framework"). En nuestro caso el framework utilizado (JADE) nos facilitará la implementación del sistema de mensajes.

La comunicación a bajo nivel tanto de los agentes macroscópicos de nivel medio con los microagentes, como de los microagentes entre sí, y de los microagentes con los sensores y actuadores se realizará a través de ROS. Para ello definiremos los "Topics" a utilizar en ROS. Los topics para manejar los sensores y actuadores llevarán un identificador único del microagente con el fin de independizarlos (privatizarlos) del resto. Para declarar las estructuras de datos pasadas en los mensajes hemos utilizado "custom messages" tanto para el intercambio con el Gateway, como otro con otros microagentes, otro para los sensores y una última con los actuadores.

5.3 Diseño Implementación de Agentes.

1.1.2 Definición de Estructura de Agentes.

De nuevo vamos a utilizar JADE a alto nivel y ROS a bajo nivel. Para ello a nivel de implementación utilizaremos una herencia del "Agente base" de JADE" a nuestro "Agente" y de la clase "Nodo ROS" a nuestro "Microagente", pudiendo ver en la Ilustración 98 estas relaciones de herencia. Los agentes de nivel medio son los que permitirán la comunicación de los dos niveles realizando la función de Gateway. Nos vamos a centrar en la estructura de los agentes nivel medio y bajo que es la parte donde incide esta tesis.

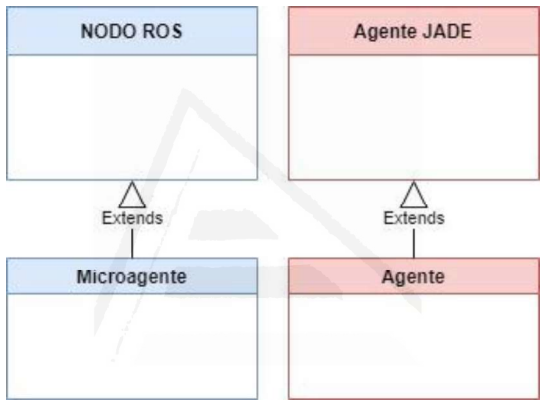


Ilustración 98. Herencia desde JADE y ROS a nuestras clases "Agente" y "Microagente" respectivamente.

5.3.1.1 Estructura de Agente Macroscópico de Nivel Medio.

En la Ilustración 99, podemos ver mediante un diagrama de clases como la estructura es igual a los agentes y microagentes implementados en el sistema de ahorro energético, apareciendo una de las grandes ventajas de este modelo, la reutilización de componentes. La clase "Gateway" al ser una composición de "Agente" y "Microagente" heredará todas las características y funcionalidades de las clases padre, permitiendo la comunicación tanto en la plataforma JADE como en la de ROS.

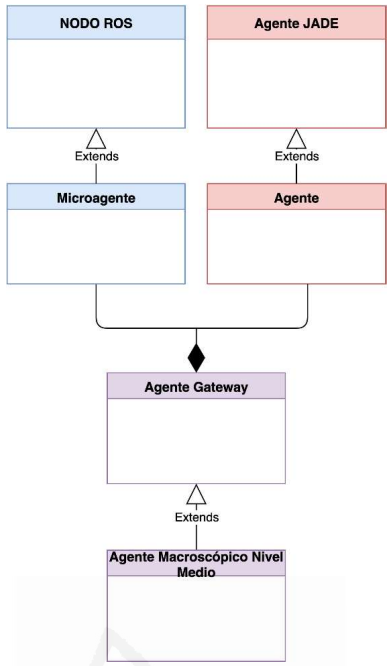


Ilustración 99. Diagrama de Clases de Agentes donde se muestra las relaciones con las clases de los Frameworks utilizados JADE y ROS.

La función del agente macroscópico de nivel medio es triple en este caso. Por un lado, llevar a cabo la de secuenciación de planes provenientes de la capa de alto nivel, por otro la de la comunicación entre capas y por último la supervisión de los agentes a su cargo. En la Ilustración 100 vemos la arquitectura interna que hemos diseñado para dotar al agente de estas funcionalidades. A través de los módulos de competencia de comunicaciones nos comunicaremos con las capas de alto nivel (JADE) y de bajo nivel (ROS). El módulo de traducción será el encargado de convertir la información de los mensajes a un formato interno que el agente puede utilizar y viceversa, haciendo posible el dialogo entre "agentes" y "microagentes" así como poder utilizar la información internamente. El módulo de secuenciación será el encargado de multiplexar y demultiplexar la secuenciación de los planes. Y nos quedaría el módulo de supervisión los microagentes que coordina.

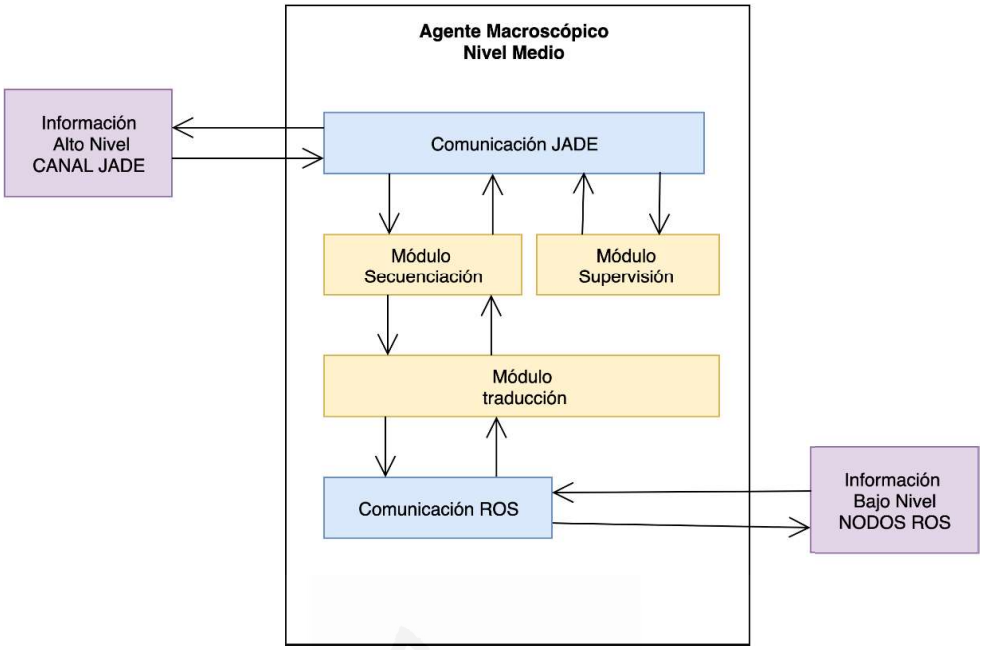


Ilustración 100. Arquitectura Interna Agente Macroscópico Nivel Medio.

5.3.1.2 Estructura de Microagente.

En la Ilustración 101 podemos ver como el microagente está formado por varios módulos de competencia organizados en capas. La información se recibe a través de módulo "comunicaciones ROS". Esta información puede provenir de los sensores o del módulo Gateway. El módulo de "seguridad" se encarga de comprobar que no hay ninguna alerta (generalmente colisiones) por la cual tenga que ejecutar una acción y pasa el control al módulo de actuación que realizará el comportamiento propiamente dicho para el microagente (está descrito en la sección 5.3.1.4) transfiriendo las órdenes a los actuadores a través del módulo de "comunicación ROS". Si tuviésemos que enviar algún tipo de información al Gateway lo haríamos también a través del módulo de "comunicaciones ROS".

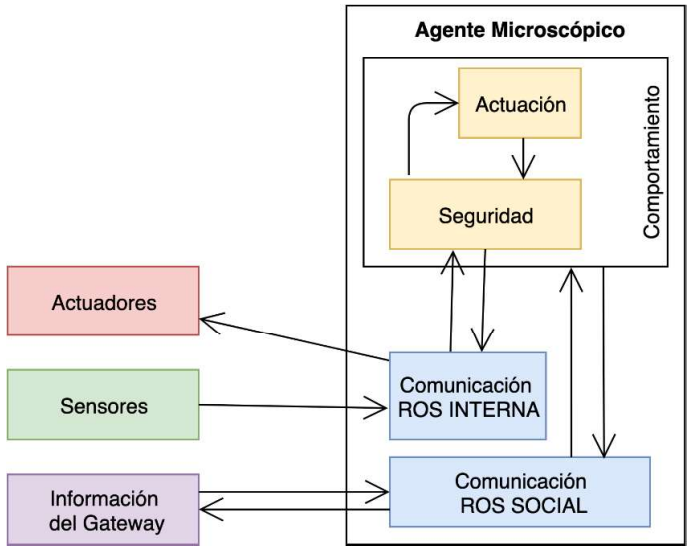


Ilustración 101. Arquitectura Interna de Agente Comportamiento Microscópico (microagente).

Cada microagente se registra en un par de nodos Master de ROS, uno que es por el que se comunica con los microagentes del mismo grupo (ROS social) y al agente macroscópico de nivel medio que coordina a ese grupo; y otro (ROS interno) con el que se comunica con sus sensores y actuadores. Esto permite una comunicación local por un lado y otra conexión interna privatizando el uso de sus dispositivos.

En el nodo Master propio, "privado", se subscribirá para recibir la información de los sensores, mientras que a nivel de actuadores se registrará como publicador con el fin de poder mandar comandos a los motores. Este entorno ROS, lo hemos decidido diseñar así para que el dron tenga autonomía propia, y no ser dependiente de las comunicaciones como ocurriría si utilizásemos el entorno ROS común al agente macroscópico coordinador. Nos ha tocado definir "custom messages" para declarar las estructuras de datos pasadas en los mensajes y una función node_handle para sincronizar el envío y recepción y hemos establecido el rate loop de la rutina de control a de 100 milisegundos, tiempo más que suficiente para la atención de los publicadores y la actuación sobre los motores.

5.3.1.3 Estructura de Agente Cognitivo.

En este caso concreto, tenemos una necesidad de robustez que describíamos con el rol de agente de espacio cognitivo. Para cubrir esta necesidad hemos tomado una decisión de diseño, la cual consiste en utilizar dos contenedores de agentes JADE, uno a alto nivel y otro a medio nivel, ambos intercomunicados. Con esto logramos tener una sincronización local de los agentes macroscópicos

de nivel medio y mayores capacidades de replicación. Al gestionar la propia plataforma JADE la replicación nos vamos a ahorrar el tiempo de desarrollo del agente cognitivo.

1.1.3 Descripción de Comportamientos.

5.3.1.4 Comportamiento Microscópico.

Dentro del comportamiento microscópico vamos a utilizar tres comportamientos diferentes dependiendo del estado del individuo: los agentes agresivos persiguen a otros agentes, los agentes de reconocimiento que deambulan por el entorno y los agentes elusivos cuyo propósito es evitar a los agresivos. Si cada agente persigue a otro, el comportamiento del enjambre tiende a centralizarse en un punto del espacio. Por otro lado si el comportamiento de los agentes es elusivo el enjambre se expande infinitamente.

Nosotros vamos a definir un agente $r_i \in R$ como un vector $r_i = (s_i, p_i, a_i)$ el cual contiene los estados $s_i \in S$, los sensores $p_i \in P$ y los actuadores $a_i \in A$.

Cada agente va a utilizar tres sensores $p_i = (m_i, c_i, t_i)$ donde:

- m_i: Es un sensor que permite identificar a su pareja en el enjambre. Cada robot tiene su pareja $\forall m_i \in M$, $\forall m_j \in M$, $i \neq j$, $m_i \neq m_j$.
- $c_i \in [0,1]$: Es un sensor que nos indica la intensidad de la señal RF en la posición actual del robot.
- $t_i \in [0, \infty[$:Es la hora que marca un reloj interno del robot.

A nivel de actuadores los agentes utilizan cada uno dos actuadores $a_i = (v_{tras}, v_{rot})$ los cuales le permiten seleccionar las velocidades de traslación y rotación.

Cada agente va a poder estar en tres estados $s_i \in$ {expansión, deambulando y agrupación}. Expansión se refiere al comportamiento elusivo de los agentes, donde el agente r_i tiene que calcular el vector de repulsión respecto al robot m_i. Deambular define un comportamiento de navegación aleatorio. Y por último, agrupar corresponde a un comportamiento de atracción, donde el vector de atracción a mj, debe de ser calculado. Para el cambio de estados vamos a utilizar una máquina de estados finitos probabilísticos ( ), donde el estado cambia en base a las reglas definidas por la quíntupla (Σ, S, s0, δ, F):

- Σ: Es una letra del alfabeto que será creada en cada nuevo ciclo.
- S: es un conjunto de estados no vacío y finito

- s_0: es un estado inicial dentro de los tres posible. Vamos a considerar es "expansión".
- δ: Va a ser la función de transición de estado δ: S x Δ ⟶P(S).
- F: Es un conjunto de estados finales. En nuestro caso no va a ser utilizado, F = ∅.

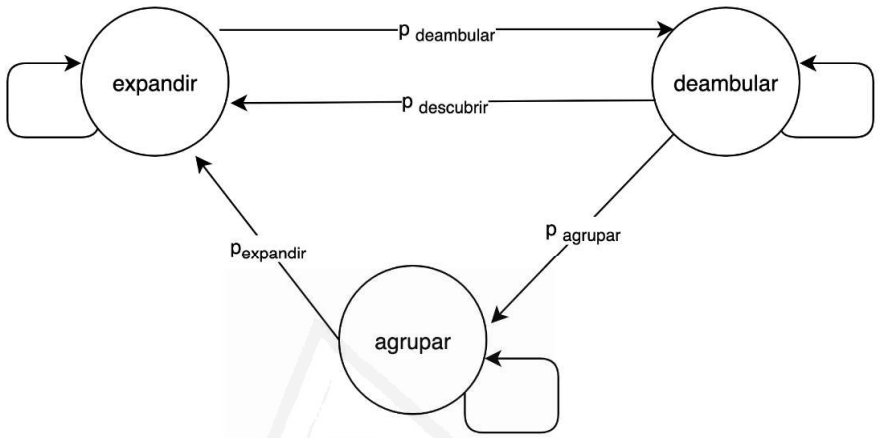


Ilustración 102. Máquina de Estados Finitos Probabilísticos.

Las probabilidades de $p_{descubrir}$, $p_{deambular}$ y $p_{agrupar}$ van a ser constantes a lo largo de la ejecución del sistema. Podríamos ajustar sus valores en base al tiempo que queremos que se mantenga en cada estado. Los agentes empezarán en el estado "expandir" y no podrán estar en varios estados simultáneamente. Cada agente calculará su nuevo estado a la hora t, cada t_{ciclos} de reloj. Este número de ciclos viene condicionado por el tiempo de ejecución de los comportamientos, siendo constante y homogéneo para todos los agentes.

Además, $p_{expandir}$ permitirá al enjambre expandirse una vez agrupado, pero necesitaremos un tiempo para explorar esa área. Esta probabilidad dependerá de la energía de cada uno de los agentes. Cuando el sistema empieza a operar su energía será mayor y el enjambre comenzará a explorar el entorno. Cuando la energía disminuye debemos de asegurarnos la agrupación del agente con otros miembros del enjambre. Para ello vamos a proponer en la función de energía (Ilustración 103), donde e_i es la energía inicial del agente r_i, Δe es el decremento de la energía del agente y Φ las unidades de tiempo transcurridas.

$$\left(\frac{e_i - \Delta e}{e_i}\right)^{\Phi}$$

Ilustración 103. Función de energía.

Además de esto el sistema será diseñado para que cuando el agente se sitúe sobre un recurso, su velocidad disminuya, el agente la pueda inspeccionar e incluso marcarla para el resto del enjambre. En base a esto nosotros proponemos definir el vector de velocidad de la siguiente forma: $\vec{v} = \vec{v}_b \times c^k$, donde k es un modificador de la intensidad de atracción del recurso, $\vec{v}_b$ es el vector de desplazamiento obtenido del comportamiento desarrollado por el robot en el estado si y c es la información recibida por el sensor de RF. Del $\vec{v}$ derivamos los valores de las velocidades traslacionales y rotacionales del robot (v_{tras}, v_{rot}).

También debemos de tener en cuenta en el comportamiento frente a colisiones. En este caso generaremos un vector de velocidades aleatorio $\vec{v}_{aleatorio}$hasta que el peligro de colisión desaparezca. Esto lo vamos a hacer así debido a las limitaciones de percepción de nuestros robots. En otros robots con sensores más complejos podíamos haber utilizado otras estrategias.

La última cosa que nos quedaría por hacer es establecer la posición inicial de los agentes. Para ello para va utilizar una matriz de 10 columnas y un número indeterminado de filas que dependerá del número de elementos del enjambre. Definiremos la posición inicial como $pos_{inicial}(r_i) = \{i\ (mod\ 10), \lfloor i/10 \rfloor$. Para cada par de agentes $m_i = r_a, a = i + 1\ (mod\ |R|)$, donde $|R|$ es el número de robots del enjambre.

1.1.3.1 Comportamiento Macroscópico.

El comportamiento macroscópico va a definir el comportamiento del enjambre a nivel global, definiendo la tendencia del enjambre en términos de predicción. A partir de la PFSM (Máquina de Estados Finitos Probabilística) y el estado de los agentes en un momento dado vamos a obtener las ecuaciones de recurrencia.

$$\begin{pmatrix} expandir_{n+1} \\ deambular_{n+1} \\ agrupar_{n+1} \end{pmatrix} = \begin{pmatrix} 1 - p_{deambular} & p_{descubrir} & \left(\frac{e_i - \Delta e}{e_i}\right)^{\Phi} \\ p_{deambular} & 1 - (p_{descubrir} + p_{agrupar} & 0 \\ 0 & p_{agrupar} & 1 - \left(\frac{e_i - \Delta e}{e_i}\right)^{\Phi} \end{pmatrix} \begin{pmatrix} expandir_n \\ deambular_n \\ agrupar_n \end{pmatrix}$$

Es sencillo ver la estabilidad del sistema desde los valores propios de la matriz de transición, donde el sistema tiende a estabilizarse con todos los robots inicializados con un estado de agregación. Este es el resultado esperado, después de que la energía de los robots decrezca y en consecuencia la probabilidad de estar en estado de expansión disminuye.

En [86] demuestra que el comportamiento de agregación origina un comportamiento estabilizado en un límite de tiempo y la conexión entre los agentes constituye un gráfico de conexión. La tendencia de nuestro enjambre será tener a nuestros agentes posicionados y agrupados. Diremos que nuestro enjambre está agrupado cuando el número de agentes en estado de expansión tienda a 0.

5.4 Codificación y Pruebas.

En este caso nos vamos a centrar en la codificación y pruebas del sistema, ya que las pruebas de comunicación entre capas son similares a las del sistema de ahorro energético, se sigue la misma arquitectura y los mismos componentes.

La prueba del sistema tiene dos objetivos: Por un lado comprobar el funcionamiento de los comportamientos macroscópicos y microscópicos, y por otro ver como el enjambre es capaz de indicar las zonas con señal y las zonas de máxima cobertura.

5.4.1 Entorno de Pruebas.

Cuando nosotros queremos cubrir un área con señales de radio frecuencia lo hacemos situando diferentes antenas con distintas intensidades generando distintas zonas de cobertura y de superposición de señales. Aparte de las antenas se utilizan amplificadores y atenuadores de señal.

Al no tener un número elevado de drones reales para realizar el comportamiento de enjambre, dada la dificultad de la prueba y a los requisitos requeridos por AESA para desarrollarla, lo que vamos a hacer es simularlos. A diferencia de otras pruebas que requieren de una recodificación de módulos, en nuestro caso, gracias a la arquitectura y al uso de ROS para comunicar los microagentes con los macroagentes de nivel medio, el paso de este sistema simulado a uno real sería muy sencillo sustituyendo simplemente los microagentes simulados por reales y manteniendo los nombres de los nodos.

Como zona a cubrir vamos a utilizar el campus de la Universidad de Alicante. Para la prueba del modelo vamos a utilizar dos simuladores: uno para simular antenas y su comportamiento, y otra para simular robots del enjambre.

Ilustración 104. Zona a cubrir por las pruebas: Campus de la Universidad de Alicante (38º 23’ 13”N, 0º 30’ 45"O).

Para la simulación de la señal de cobertura hemos usado el simulador CloudRF (http://cloudrf.com), situando tres antenas en el perímetro, las cuales generan diferentes señales de intensidad en el Campus y utilizando el modelo de propagación irregular terrestre (ITM), también conocido como “Longley Rice”, utilizado en la planificación de señales de televisión de broadcast. Este modelo cubre muchas aplicaciones en un amplio rango de frecuencias que van desde los 20 a los 20,000 MHz. En la Ilustración 105 podemos ver el mapa de intensidad de señal generado, donde a tono más claro de verde mayor es la intensidad de señal. Estas zonas también coinciden con las de mayor superposición de señales. En la Ilustración 106 hemos marcado el perímetro de la zona de cobertura para nuestra red de radiofrecuencia. El agente “Simulador RF” es el que va a hacer de interfaz entre este simulador y el sistema multiagente.

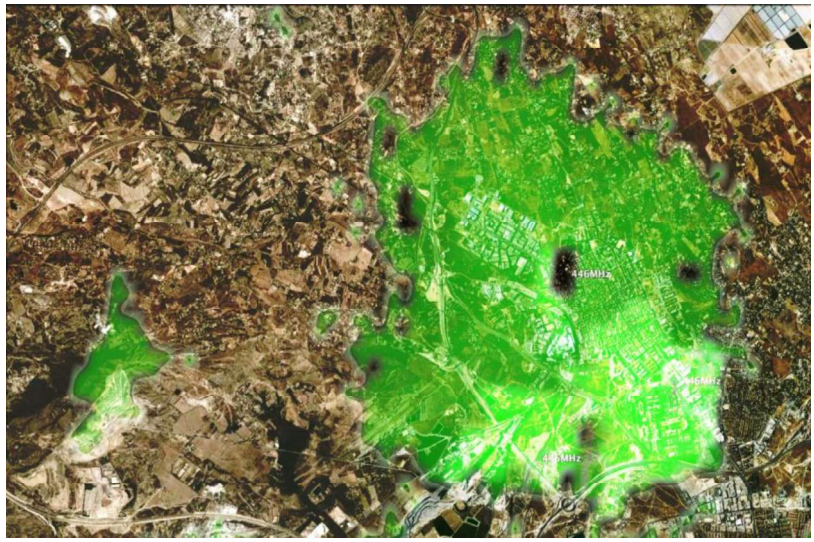


Ilustración 105. Zonas de cobertura generadas con CloudRF para el Campus de la Universidad de Alicante.

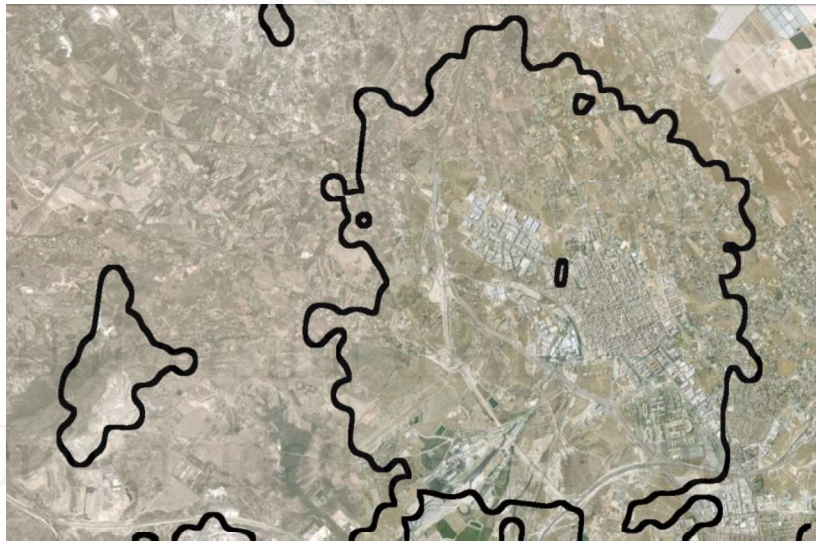


Ilustración 106. Perímetro de la zona de cobertura del Campus de la Universidad de Alicante generado con CloudRF.

Para la simulación del sistema hemos utilizado el simulador de sistemas multiagente MASON. Apoyándonos en este simulador hemos desarrollado nuestro entorno de movimiento en 2D, al fijar la altitud conseguimos simplificar de 3D a 2D. Hemos discretizado el entorno dividiéndolo en celdas de 1m x 1m que es el espacio que ocupa cada robot junto con su zona de seguridad aproximadamente. En la Ilustración 107 podemos ver el mapa de recursos de señal RF utilizado en nuestro simulador, donde en negro tenemos las zonas sin

cobertura y en blanco las zonas de con máxima intensidad de señal. Dentro de este simulador también hemos creado los microagentes necesarios para las simulaciones, utilizando la arquitectura y los comportamientos diseñados en etapas anteriores. Para las simulaciones nos hemos limitado el entorno al mapa creado y hemos hecho simulaciones para enjambres de 10, 50, 100 y 500 robots.

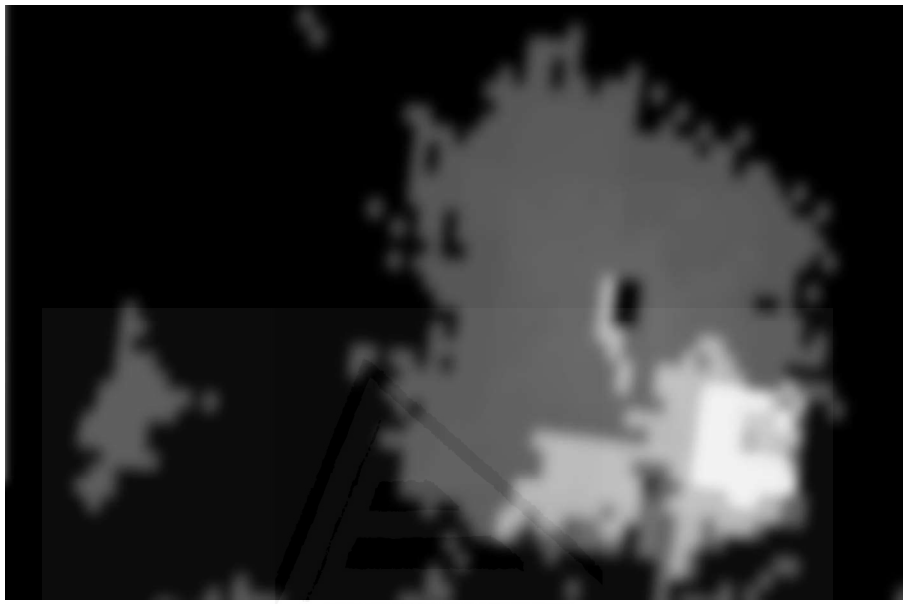


Ilustración 107. Mapa de recursos de intensidad de señal utilizado en el simulador.

5.4.2 Pruebas de Sistema

Para la realización de las pruebas hemos necesitado hacer un paso adicional, y ello ha conllevado a tener que introducir un nuevo agente en el sistema. A la hora de diseñar el sistema hemos pensado en si incorporábamos esa funcionalidad en el agente de simulación RF, pero por las características de su propósito y para poder reutilizarlo en otros sistemas, hemos preferido implementarlo aparte, es decir incorporamos un nuevo rol (entrenamiento del sistema) y asociado a él, un nuevo agente de alto nivel, el agente de entrenamiento.

El agente de entrenamiento obtendrá las probabilidades con metaheurísticas utilizando la convergencia del enjambre como medida de calidad. Después de entrenar al sistema con varios mapas teóricos RF, generados con diferentes tamaños de entorno y diferente número de antenas obtuvimos los valores de probabilidad para la máquina de estados finitos probabilísticos que maximizaban la convergencia del enjambre. De esta forma

los test sobre el sistema los realizamos con los siguientes valores $(p_{deambular}, p_{descubrir}, p_{agrupar}) = (0.274, 0.154, 0.222)$ y para cada agente utilizamos los siguientes valores de energía $(e_i, \Delta e) = (500, 35)$.

Con estos valores realizamos 50 ejecuciones independientes para cada uno de los enjambres, de los diferentes tamaños, comparando los resultados del modelo teórico con los del modelo actual.

5.5 Resultados.

En la Ilustración 108, podemos ver que el modelo teórico predice de una manera precisa el estado de los cien agentes para un estado de tiempo dado. En las líneas discontinuas se muestra el modelo teórico y en colores los datos experimentales (en azul agentes agrupados, en verde en expansión y en rojo deambulando).

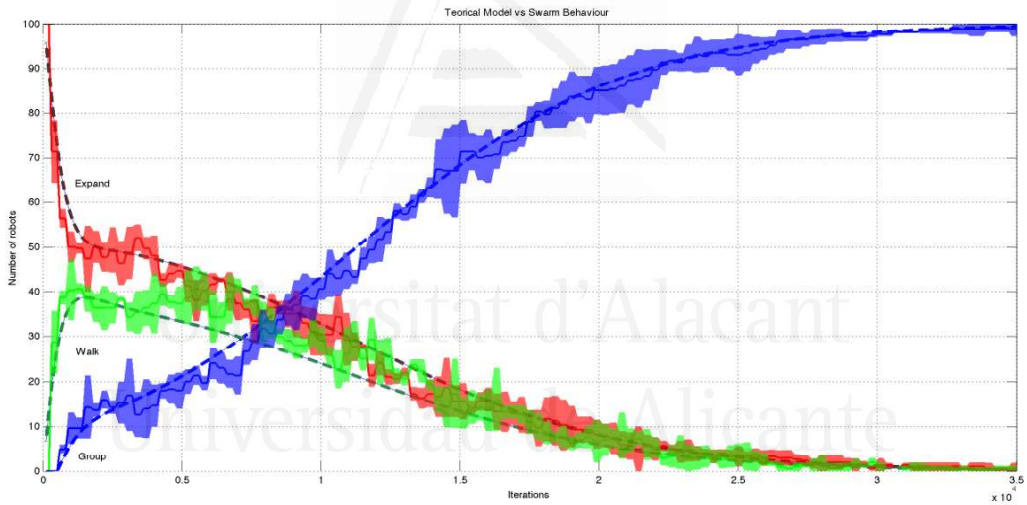


Ilustración 108. Comparación modelo teórico con datos experimentales.

También hemos podido verificar la convergencia hacia el área de mayor cobertura de señal RF. La Ilustración 109 muestra la distancia Euclídea desde el centroide del enjambre al área de mayor intensidad de señal RF. Para las simulaciones con cada tamaño de enjambre hemos mostrado los cinco valores más representativos, mostrando la media y la desviación estándar para todos los casos. A pesar de que la desviación es grande para tamaños pequeños de

enjambre se consiguen marcar las grandes áreas de intensidad de señal utilizando un número de iteraciones similar al de enjambres de mayor tamaño.

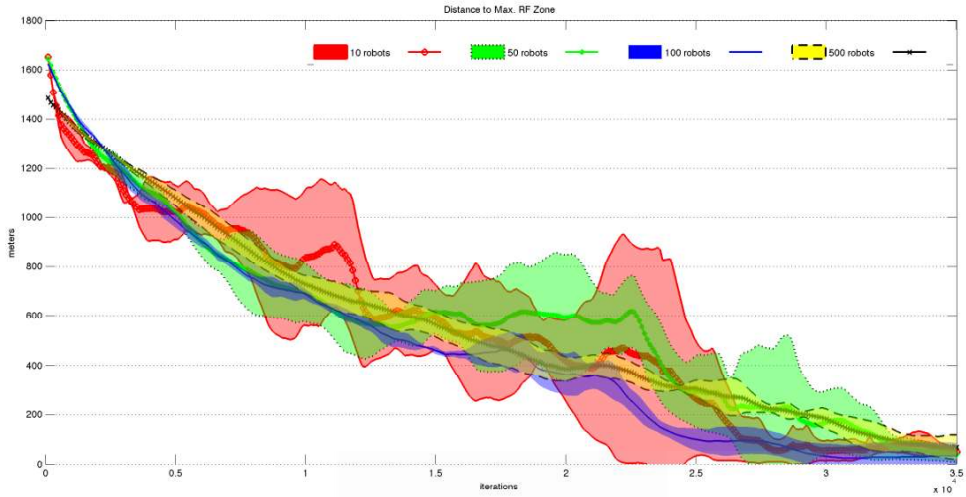


Ilustración 109. Convergencia del enjambre sobre el área de alta intensidad de señal.

Una vez que el proceso de localización termina, los miembros del enjambre están situados en las zonas de mayor intensidad de señal RF. Esto permitirá a los ingenieros distribuir la señal más eficientemente a través de dos alternativas: posicionando las antenas en otros puntos o modificando la potencia de emisión. En la Ilustración 110 podemos ver la convergencia de un enjambre de 100 individuos sobre las 2 áreas de máxima intensidad de señal. Se necesitaron 25.000 iteraciones para obtener la convergencia.

Nosotros hemos mostrado que el modelo teórico de señales RF puede ser validado empíricamente a través de enjambres los cuales consiguen agregarse en zonas de máxima intensidad de señales de radiofrecuencia. Esta aplicación de los enjambres puede establecer nuevas formas para determinar la ampliación o reducción de la intensidad en distintas zonas. Así mismo podemos aplicar estos modelos a otras áreas como sistemas de rescate o delimitación de recursos.

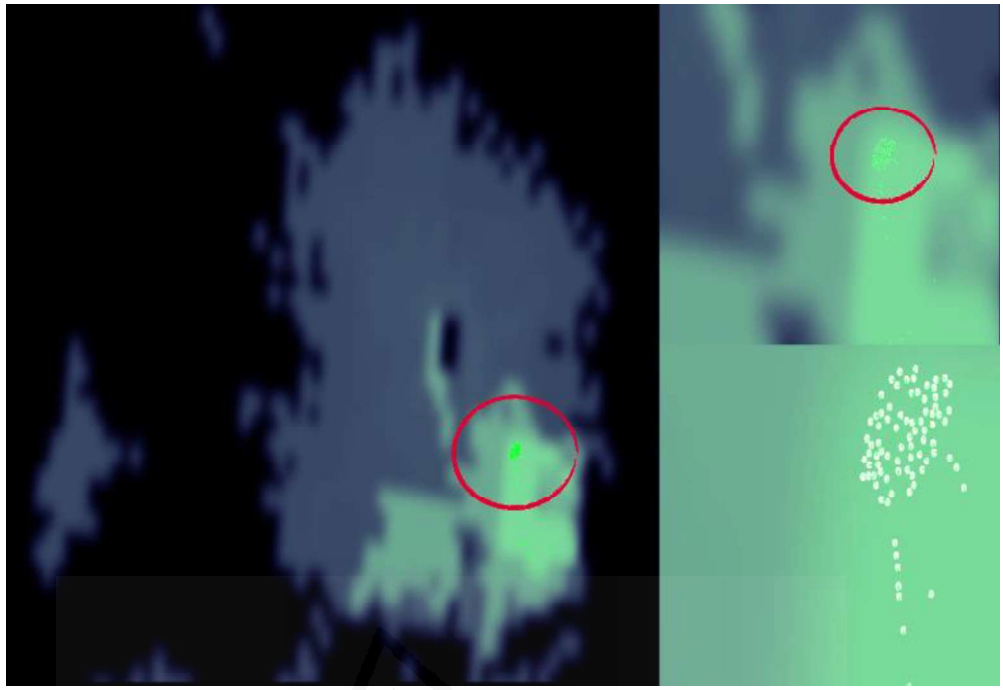


Ilustración 110.Convergencia de un enjambre de 100 individuos sobre el área de mayor intensidad de radiofrecuencia.

5.6 Resumen del Capítulo.

En este capítulo hemos mostrado la aplicación de nuestra arquitectura sobre un enjambre de drones. Con esta aplicación hemos podido comprobar la versatilidad, flexibilidad y escalado que permite la arquitectura, adaptándose fácilmente a la forma de resolver estos problemas a través de la definición de comportamientos macroscópicos y microscópicos.

También hemos visto como ha sido fácil incorporar simulaciones dentro de la arquitectura, tanto a alto nivel como a bajo nivel. Gracias al diseño de la arquitectura y a los frameworks elegidos (JADE y ROS), pasar de la simulación a la prueba real del sistema es un paso bastante rápido y transparente.

A la hora de aplicar diseñar y construir el sistema de enjambre lo hemos hecho siguiendo la metodología propuesta. En algunas etapas se han detectado nuevos elementos, como el sistema de entrenamiento, pero gracias a la vuelta atrás, a las iteraciones y a la flexibilidad de la arquitectura ha sido posible incorporar estos elementos no detectados en fases tempranas.

Gracias a todo lo anterior, arquitectura y metodología, hemos podido crear un sistema de enjambre de robots capaz de detectar las zonas de cobertura y

máxima cobertura de señales RF. Este problema puede ser extendido a otras problemáticas reales como detección y seguimiento de incendios, nubes tóxicas, vertidos petrolíferos, control de plagas y a otros muchos controles de seguridad y vigilancia.

Pero no todo está hecho, y queda mucho trabajo, como veremos en el capítulo siguiente. En el hablaremos de las líneas futuras de investigación referentes tanto a arquitecturas, metodologías como a herramientas.

6 Conclusiones y Líneas Futuras

Este capítulo resume el trabajo realizado en esta tesis. En primer lugar, presentaré las conclusiones del trabajo realizado y expondré las aportaciones realizadas. A continuación, indicaré las publicaciones derivadas y ya para terminar, haré una descripción de las líneas futuras de investigación asociadas al desarrollo de esta tesis.

6.1 Conclusiones.

Los sistemas multiagente son un tipo de sistemas cada vez más demandados y utilizados. Su aplicación la podremos encontrar en un amplio número de campos como Big Data, aplicaciones de transporte y logística, sistemas de información geográfica, balanceo de redes y móviles, diagnóstico médico, aplicaciones gráficas, juegos, así como en muchos otros campos. Al igual ocurre con los sistemas robóticos que ya no son exclusivos de entornos industriales, ocupando un lugar más cercano cada día en nuestros hogares.

Los problemas que resuelven ambos paradigmas tienen muchos puntos en común, solapándose cada día más, pero a la vez tienen unas necesidades específicas que los distinguen como la velocidad de ejecución o los requerimientos hardware, entre otros. En ambos casos son problemas complejos donde un grupo de individuos software/hardware colaborarán, combinando sus habilidades y su conocimiento, permitiendo la resolución del problema de manera conjunta.

A lo largo de los años se ha trabajado mucho para facilitar el diseño y la construcción de sistemas multiagente, han aparecido diversas arquitecturas, metodologías y herramientas, incluso se ha creado un estándar, el estándar IEEE-FIPA (Foundation for Intelligent Physical Agents). Algo parecido a pasado

con los sistemas robóticos, donde han aparecido sobre todo arquitecturas y herramientas para facilitar su construcción.

Tras un minucioso estudio del estado del arte, en esta tesis se ha aportado una arquitectura versátil y una metodología que permite avanzar en el diseño y construcción de sistemas multiagente y robóticos, uniendo lo mejor de ambos, cubriendo las necesidades que cada una de ellas tenía, pero que a su vez la otra ya resolvía.

En concreto se han realizado las siguientes aportaciones:

- **Arquitectura híbrida multinivel** versátil, flexible, robusta y escalable. Se ha definido una arquitectura híbrida multinivel de 3 capas y dos canales de comunicación bien diferenciados, siendo la parte clave que aporta este modelo la diferenciación del sistema a bajo nivel, el cual tendrá su propio diseño, sus propios estándares y canales de comunicación, así como una implementación diferenciada del resto del sistema multiagente. Esta arquitectura simplifica el diseño de los sistemas multiagente tanto a alto como a bajo nivel y permite construir los sistemas de una manera ágil utilizando los estándares que nos proporciona el mercado. En nuestro caso lo hemos hecho a través de FIPA, ROS y UML principalmente.
- Hemos establecido junto con la arquitectura, **una metodología ágil e iterativa**, estructurada en fases y etapas, la cual nos permitirá el diseñar y desarrollar el sistema multiagente. Dentro de cada una de estas etapas se indican las actividades a realizar y los artefactos a desarrollar. Recoge conceptos de otras metodologías de diseño y construcción de sistemas multiagente, principalmente PASSI y Message, pero con un carácter más rápido, informal y actual.
- **Integración de elementos IoT** en sistemas multiagente. Para ello se ha visto la aplicación de la arquitectura y metodología en el contexto de un hotel inteligente donde se persigue la mejora del ahorro energético.
- **Bases y medidas para mejorar los ahorros energéticos** de establecimientos gracias a la cooperación e intercambio de información (conocimiento) de diferentes sistemas. Nuestra arquitectura facilita la comunicación de sistemas existentes y nuevos.
- **Arquitectura capaz de modelar sistemas de enjambre robóticos.** donde se distinguen claramente los comportamientos macroscópicos y microscópicos. Gracias a esa arquitectura se han podido integrar fácilmente una simulación de los robots y hacer que el paso de simulación a entorno real sea inmediato.

- **Arquitectura flexible** que facilita su uso en **entornos** tan variopintos como el de la **salud**, modelando un sistema que sirve como herramienta tecnológica para medir y registrar movimiento de hiperactividad con un enfoque práctico y novedoso, el cual hace uso de dispositivos Microsoft Kinect de forma similar a como se usan dispositivos IoT o robóticos.

6.2 Publicaciones Derivadas del Trabajo de Tesis.

Se han realizado las siguientes publicaciones:

- Objective Analysis of Movement in Subjects with ADHD. Multidisciplinary Control Tool for Students in the Classroom. *Int. J. Environ. Res. Public Health* 2020, *17*, 5620. ISSN: 1660-4601. Sempere Tortosa, Mireia; Fernández Carrasco, Francisco; Mora Lizán, Francisco José; Rizo Maestre, Carlos.
- Intelligent Buildings: Foundation for Intelligent Physical Agents. Int. Journal of Engineering Research and Application (IJERA) 2017. ISSN: 2248-9622, Vol. 7, Issue 5, (Part -2) pp21-25. Mora Lizán, Francisco José; Rizo Maestre, Carlos.
- Intelligent Buildings: Considerations for its. Design using Multiagent Systems. International Journal of Engineering Research & Technology (IJERT). 2017. Vol. 6 Issue 04 pp 38-41. ISSN: 2278-0181. Mora Lizán, Francisco José; Rizo Maestre, Carlos.
- Formalization of a multi-agent system using Z notation: Application to a system for oil spill location. Mathematical Modelling in Engineering & Human Behaviour 2015. Páginas 228-235. Mora lizán, Francisco José; Rizo Aldeguer, Ramón; Pujol López Mar; Aznar Gregori, Fidel; Sempere Tortosa, Mireia.
- A Formal Specification of Autonomous Agent Systems with Subsystems Inspired by Biology. Mathematical Modelling in Engineering and Human Behaviour 2014. Páginas 90-95. Mora lizán, Francisco José; Rizo Aldeguer, Ramón; Pujol López Mar; Aznar Gregori, Fidel; Sempere Tortosa, Mireia.
- Agents for Swarm Robotics: Architecture and Implementation. Revista: Advances in Intelligent and Soft Computing ISBN 978-3-642-19916-5. Páginas 117-124. Springer, Berlin, Heidelberg. 2011. Aznar Gregori, Fidel; Sempere Tortosa, Mireia; Mora Lizán, Francisco José; Rizo Aldeguer, Ramón; Pujol López Mar.

- Bus Service Control System with Multiagent Architecture. 3rd International Workshop on Practical Applications of Agents and Multiagent Systems. ISBN: 84-96394. Páginas 124-134. Servicio de Publicaciones de la Universidad de Burgos. 2004. Llorens Largo, Faraón; Mora Lizán, Francisco José; Rizo Aldeguer, Ramón; Pujol López, Mar; Suau, Pérez, Pablo.
- Autonomous System of Teleoperation and Control through the Internet based on Intelligents Agents. The 8th World Multi-Conference on Systemics, Cybernetics and Informatics (SCI 2004). Aznar Gregori, Fidel; Mora Lizán, Francisco José; Llorens Largo, Faraón; Pujol López, Mar; Rizo Aldeguer, Ramón; Sempere Tortosa, Mireia; Suau Pérez, Pablo.
- Broadcasting of Image Sequences by Internet for the Telecontrol of a Movile Robot. The 8th World Multi-Conference on Systemics, Cybernetics and Informatics (SCI 2004). Aznar Gregori, Fidel; Mora Lizán, Francisco José; Llorens Largo, Faraón; Pujol López, Mar; Rizo Aldeguer, Ramón; Sempere Tortosa, Mireia; Suau Pérez, Pablo.
- Working with OpenCV and Intel Image Processing Libraries. Processing image Data Tools. 6th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2002). ISBN: 980-07-8150-1. Edit. Nagib Callaos, William Lesso Páginas 51-56. Llorens Largo, Faraón; Mora Lizán, Francisco José; Pujol Lopéz, Mar; Rizo Aldeguer, Ramón; Villagra Arnedo, Carlos.

6.3 Líneas Futuras.

En este apartado final y para terminar este trabajo, realizaré una aproximación a distintas líneas de investigación que parten de las aportaciones de esta tesis.

La verdad que es complicado concretar sólo unas líneas, ya que el trabajo ha sido bastante trasversal y ha desvelado necesidades en distintas áreas. De una forma resumida podríamos enmarcar el trabajo futuro dentro de las siguientes categorías: metodologías, modelos y herramientas multiagente; y arquitecturas para plataformas de bajo nivel.

Una línea de investigación vendría de la mano de las **metodologías para la construcción de sistemas multiagente**. En concreto habrá que investigar más sobre:

- Fases y etapas a seguir en el diseño y construcción de los sistemas: hemos definido un marco de proceso, pero habrá que ir refinando los pasos a seguir así como su organización. Habrá que formalizar el cómo se realizan las iteraciones, así como sus objetivos.
- Artefactos: en nuestra metodología hemos definido una serie de artefactos (diagramas y documentos) a completar en cada una de las etapas, pero habrá que revisar como complementar los actuales y sobre todo el como ir creándolos de una manera iterativa.
- Incorporación de flujos de soporte y gestión del proyecto: es necesaria la incorporación de disciplinas de gestión (estimación, planificación, métricas, seguimiento) dentro de la metodología.
- Herramientas de apoyo a la aplicación de la metodología: solemos utilizar herramientas de diseño y construcción generales o en todo caso apoyadas en un framework como vimos en capítulo 2. Pero cada día vamos siendo más conscientes de la necesidad de herramientas integrales que también incorporen la integración con las metodologías y los modelos de proceso.

Otra línea importante con la que tenemos que seguir trabajando es la de los **modelos y frameworks de los sistemas multiagente**. Se ha avanzado mucho en los últimos años, pero siguen existiendo grandes campos de trabajo, entre ellos.

- Robustez de las plataformas: prácticamente todos los sistemas multiagente necesitan ser tolerantes a fallos. A pesar de existir técnicas y métodos para prevenir las caídas e incluso de recuperación es necesario mejorar y facilitar su implementación.
- Distribución de elementos: un sistema multiagente es un sistema distribuido. Nuestra arquitectura establece varios niveles, pero dentro de cada nivel habrá que trabajar en facilitar y mejorar la distribución en ejecución de los elementos.
- Reutilización: hemos creado una base, para empezar sobre ella a crear tanto agentes como microagentes que se puedan ir reutilizando en distintos problemas. Tener una colección de tipos de agentes y comportamientos que podamos heredar será un área clave para que este paradigma se consolide.
- Estandarización: es una línea siempre abierta y al mismo tiempo muchas veces olvidada. Necesitamos trabajar por la estandarización si queremos ser productivos y crear sistemas de calidad.

Y para terminar hablaremos de la línea de **investigación de los elementos de bajo nivel**, que en muchas ocasiones son robóticos, como es el caso del problema del enjambre robótico del capítulo 5, pero en otras ocasiones, como en el caso del ahorro energético del capítulo 4, son dispositivos que vamos a encontrar habitualmente en nuestro entorno cotidiano. Dentro de esta categoría vamos a encontrar las siguientes líneas:

- Integración de elementos de bajo y alto nivel: a pesar de haber logrado crear una arquitectura que facilita el modelado en varios niveles, permitiendo construir soluciones de una manera racional, tenemos que trabajar mucho en facilitar las comunicaciones entre ambos niveles.
- Estandarización a bajo nivel: en los contextos planteados en esta tesis la capa de bajo nivel se ha cubierto con ROS. Ha sido una manera de solucionar nuestra implementación, pero no es ni la ideal, ni es única. Es complicado querer estandarizar elementos de muchos fabricantes donde las restricciones de velocidad y precio de los dispositivos, así como la competencia dificultan esta meta, pero el único camino del éxito será investigar en cómo lograr una estandarización de estos elementos.
- Mejora de la arquitectura interna y de los comportamientos: una línea de investigación importante, que ya existía, pero en la que habrá que segur trabajando, es la de mejorar los comportamientos de los agentes tanto de alto como de abajo nivel. Internamente los agentes pueden tener varios módulos de competencia que entren en conflicto y esto hará que tengamos que investigar en soluciones ante esos problemas. Por ejemplo, en nuestro agente de zona, el comportamiento climático y de iluminación puede entrar en conflicto a la hora de actuar sobre las persianas y toldos. Lo mismo ocurría en el sistema de enjambre donde aún queda mucho por estudiar respecto a los comportamientos microscópicos y macroscópicos.
- Privacidad y seguridad: un tema con el que nos estamos enfrentando y cada día vamos a tener que trabajar e investigar más es cómo recopilar datos a través de los dispositivos de bajo nivel, pero al mismo tiempo garantizar la privacidad y la seguridad de los usuarios.

Bibliografía

[1] P. Hípola y B. Vargas-Quesada, «Agentes inteligentes: definición y tipología. Los agentes de información», *El Prof. Inf.*, vol. 8, pp. 13-21, 1999.

[2] J. Silvestre y Ramos, Esmeralda, «Agentes Inteligentes. Lecturas en Ciencias de la Computación. Universidad Central de Venezuela», 2002.

[3] H. S. Nwana, «Software agents: an overview», *Knowl. Eng. Rev.*, vol. 11, n.º 3, pp. 205-244, sep. 1996, doi: 10.1017/S026988890000789X.

[4] J. He, H. Lai, y H. Wang, «A commonsense knowledge base supported multi-agent architecture», *Expert Syst Appl*, vol. 36, n.º 3, pp. 5051-5057, 2009.

[5] I. Cil y M. Mala, «A multi-agent architecture for modelling and simulation of small military unit combat in asymmetric warfare», *Expert Syst Appl*, vol. 37, n.º 2, pp. 1331-1343, 2010.

[6] G. Weiss, Ed., *Multiagent systems: a modern approach to distributed artificial intelligence*. Cambridge, MA, USA: MIT Press, 1999.

[7] M. Wooldridge, *An Introduction to MultiAgent Systems*. John Wiley & Sons, 2002.

[8] J. R. Searle, *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, 1969.

[9] J. R. Searle, *Mente, lenguaje y sociedad : la filosofía en el mundo real*. New York: Basic, 2001.

[10] A. Omicini, F. Zambonelli, y M. Klusch, «Coordination of Internet Agents: Models, Technologies, and Applications», 2001.

[11] B. H. Roth, «A Blackboard Architecture for Control», 1985.

[12] H. P. Nii, «Blackboard Application Systems, Blackboard Systems and a Knowledge Engineering Perspective», 1986.

[13] D. D. Corkill, «Blackboard Systems», 1991.

[14] R. Neches *et al.*, «Enabling Technology for Knowledge Sharing», *AI Mag.*, vol. 12, pp. 36-56, 1991.

[15] A. G. Pérez y V. R. Benjamins, «Overview of Knowledge Sharing and Reuse Components: Ontologies and Problem-Solving Methods», 1999.

[16] J. S. Rosenschein y G. Zlotkin, «Rules of Encounter - Designing Conventions for Automated Negotiation among Computers», 1994.

[17] A. B. Loyall y J. Bates, «A Reactive , Adaptive Architecture for Agents», 1991.

[18] A. Newell, «Physical Symbol Systems», *Cogn. Sci.*, vol. 4, pp. 135-183, 1980.

[19] F. Bellifemine, G. Caire, y D. Greenwood, *Developing Multi-Agent Systems with JADE*. John Wiley & Sons, Ltd, 2007.

[20] N. R. Jennings, «On agent-based software engineering», *Artif Intell*, vol. 117, pp. 277-296, 2000.

[21] M. J. Wooldridge, G. Weiß, y P. Ciancarini, «Agent-Oriented Software Engineering II», 2002.

[22] J. J. Gómez-Sanz y R. Fuentes-Fernández, «Understanding Agent-Oriented Software Engineering methodologies», *Knowl. Eng Rev.*, vol. 30, pp. 375-393, 2015.

[23] M. Wooldridge, N. R. Jennings, y D. Kinny, «The Gaia Methodology for Agent-Oriented Analysis and Design», *Auton. Agents Multi-Agent Syst.*, vol. 3, pp. 285-312, 2000.

[24] F. Zambonelli, N. R. Jennings, y M. J. Wooldridge, «Multi-Agent Systems as Computational Organizations: The Gaia Methodology», 2005.

[25] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, y J. Mylopoulos, «Tropos: An Agent-Oriented Software Development Methodology», *Auton. Agents Multi-Agent Syst.*, vol. 8, pp. 203-236, 2004.

[26] P. Giorgini, H. Mouratidis, y N. Zannone, «Modelling security and trust with Secure Tropos», 2006.

[27] M. Cossentino, «PASSI : a Process for Specifying and Implementing Multi-Agent Systems Using UML», 2001.

[28] M. Cossentino, «From Requirements to Code with PASSI Methodology», 2005.

[29] M. Winikoff y L. Padgham, «The Prometheus Methodology», 2004.

[30] S. A. DeLoach, «The MaSE methodology», 2006.

[31] C. A. Iglesias, M. Garijo, J. C. González, y J. R. Velasco, «Analysis and Design of multiagent systems using MAS-CommonKADS», 1998.

[32] P. Kearney *et al.*, «Technical Information MESSAGE : Methodology for Engineering Systems of Software Agents Methodology for Agent-Oriented Software Engineering».

[33] J. Pavón y J. J. Gómez-Sanz, «Agent Oriented Software Engineering with INGENIAS», 2003.

[34] J. J. Gómez-Sanz y J. Pavón, «Implementing Multi-agent Systems Organizations with INGENIAS», 2005.

[35] Z. Guessoum, J.-P. Briot, N. Faci, y O. Marin, «Towards Reliable Multi-Agent Systems - An Adaptive Replication Mechanism», *Multiagent Grid Syst. - Int. J.*, vol. 6, n.º 1, pp. 1-24, 2010.

[36] C. Dony, C. Tibermacine, C. Urtado, y S. Vauttier, «Specification of an exception handling system for a replicated agent environment», en *Proceedings of the 4th international workshop on Exception handling*, New York, NY, USA, 2008, pp. 24-31.

[37] A. De Luna Almeida, J.-P. Briot, S. Aknine, Z. Guessoum, y O. Marin, «Towards autonomic fault-tolerant multi-agent systems», 2007.

[38] A. de Luna Almeida, S. Aknine, J.-P. Briot, y J. Malenfant, «Plan-based replication for fault-tolerant multi-agent systems», en *Proceedings of the 20th international conference on Parallel and distributed processing*, Washington, DC, USA, 2006, pp. 347-347.

[39] S. Kumar y P. R. Cohen, «Towards a fault-tolerant multi-agent system architecture», en *Proceedings of the fourth international conference on Autonomous agents*, New York, NY, USA, 2000, pp. 459-466.

[40] P. Xu y R. Deters, «Fault-Management for Multi-Agent Systems», en *Proceedings of the The 2005 Symposium on Applications and the Internet*, Washington, DC, USA, 2005, pp. 287-293.

[41] L. Wang, H. Li, D. Goswami, y Z. Wei, «A Fault-Tolerant Multi-agent Development Framework», en *Parallel and Distributed Processing and Applications*, vol. 3358, J. Cao, L. Yang, M. Guo, y F. Lau, Eds. Springer Berlin / Heidelberg, 2005, pp. 126-135.

[42] Z. A. Khan, S. Shahid, H. F. Ahmad, A. Ali, y H. Suguri, «Decentralized architecture for fault tolerant multi agent system», en *Autonomous Decentralized Systems, 2005. ISADS 2005. Proceedings*, abr. 2005, pp. 167-174.

[43] H. S. Nwana, D. T. Ndumu, L. C. Lee, y J. C. Collis, «Zeus: A collaborative agents toolkit», 1997.

[44] Y. Labrou, T. W. Finin, y Y. Peng, «Agent communication languages: the current landscape», 1999.

[45] D. T. Ndumu y H. S. Nwana, «Research and development challenges for agend-based systems», *Softw. Eng. - IEE Proc.*, vol. 144, pp. 2-10, 1997.

[46] R. Titmuss, I. B. Crabtree, y C. S. Winter, «Agents, Mobility and Multimedia Information», 1996.

[47] J. L. Posadas Yague J. L. ;. Poza Lujan, «Revisión de las arquitecturas de control distribuido», Universidad Politécnica de Valencia, 2009.

[48] A. Oreback y HenrikI. Christensen, «Evaluation of Architectures for Mobile Robotics», *Auton. Robots*, vol. 14, n.º 1, pp. 33-49, 2003.

[49] M. Mtshali y A. Engelbrecht, «Robotic architectures», *Def. Sci. J.*, vol. 60(1), pp. 15-22, 2010.

[50] D. Nakhaeinia, S. H. Tang, S. B. Mohd Noor, y O. Motlagh, «A review of control architectures for autonomous navigation of mobile robots», *Int. J. Phys. Sci.*, vol. 6(2), pp. 169-174, 2011.

[51] R. Murphy, *An introduction to AI robotics*. The MIT Press, 2000.

[52] R. C. Arkin y T. Balch, «AuRA: Principles and Practice in Review», *J. Exp. Theor. Artif. Intell.*, vol. 9, pp. 175-189, 1997.

[53] R. R. Murphy y R. C. Arkin, «Sfx: An Architecture For Action-oriented Sensor Fusion», en *Intelligent Robots and Systems, 1992., Proceedings of the 1992 lEEE/RSJ International Conference on*, jul. 1992, vol. 2, pp. 1079-1086.

[54] H. Yavuz y A. Bradshaw, «A New Conceptual Approach to the Design of Hybrid Control Architecture for Autonomous Mobile Robots», *J. Intell. Robot. Syst.*, vol. 34, n.º 1, pp. 1-26, 2002.

[55] G. Kim y W. Chung, «Tripodal Schematic Control Architecture for Integration of Multi-Functional Indoor Service Robots», *IEEE Trans. Ind. Electron.*, vol. 53, n.º 5, pp. 1723-1736, 2006.

[56] R. P. Bonasso, R. Kerri, K. Jenks, y G. Johnson, «Using the 3T architecture for tracking Shuttle RMS procedures», en *Intelligence and Systems, 1998. Proceedings., IEEE International Joint Symposia on*, may 1998, pp. 180-187.

[57] M. Lindstrom, A. Oreback, y H. I. Christensen, «BERRA: a research architecture for service robots», en *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, 2000, vol. 4, pp. 3278-3283.

[58] J. H. Connell, «SSS: a hybrid architecture applied to robot navigation», en *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on*, may 1992, vol. 3, pp. 2719-2724.

[59] K. Konolige, K. Myers, E. Ruspini, y Alessandro Saffiotti, «The Saphira Architecture: A Design for Autonomy», *J. Exp. Theor. Artif. Intell.*, vol. 9, pp. 215-235, 1997.

[60] R. G. Simmons, «Structured control for autonomous robots», *Robot. Autom. IEEE Trans. On*, vol. 10, n.º 1, pp. 34-43, feb. 1994.

[61] F. Aznar, «FSR-BAY: Modelo probabilístico para la Fusión Sensorial Robótica», PhD Thesis, Universidad de Alicante, 2006.

[62] M. J. Xie Wei y Y. Mingli, *Research on Building Mechanism of System for Intelligent Service Mobile Robot*. Dr. Janusz Bedkowski, 2011.

[63] D. Busquets, C. Sierra, y R. L. de Mantaras, «A Multiagent Approach to Qualitative Landmark-Based Navigation», *Auton. Robots*, vol. 15, n.º 2, pp. 129-154, 2003.

[64] B. Innocenti, B. Lopez, y J. Salvi, «A multi-agent architecture with cooperative fuzzy control for a mobile robot», *Robot. Auton. Syst.*, vol. 55, n.º 12, pp. 881-891, 2007.

[65] J. L. Posadas, J. L. Poza, J. E. Simo, G. Benet, y F. Blanes, «Agent-based distributed architecture for mobile robot control», *Eng. Appl. Artif. Intell.*, vol. 21, n.º 6, pp. 805-823, 2008.

[66] X. Wang, H. Zhang, H. Lu, y Z. Zheng, «A New Triple-Based Multi-robot System Architecture and Application in Soccer Robots», en *Intelligent Robotics and Applications*, vol. 6425, H. Liu, H. Ding, Z. Xiong, y X. Zhu, Eds. Springer Berlin Heidelberg, 2010, pp. 105-115.

[67] L. E. Parker, «ALLIANCE: an architecture for fault tolerant multirobot cooperation», *Robot. Autom. IEEE Trans. On*, vol. 14, n.º 2, pp. 220-240, 1998.

[68] M. Piaggio, «HEIR - A Non-hierarchical Hybrid Architecture for Intelligent Robots», en *Intelligent Agents V: Agents Theories, Architectures, and Languages*, vol. 1555, J. P. Muller, A. S. Rao, y M. P. Singh, Eds. Springer Berlin Heidelberg, 1999, pp. 243-259.

[69] D. Goldberg, V. Cicirello, M. B. Dias, Reid Simmons, S. Smith, y A. (Tony) Stentz, «Market-Based Multi-Robot Planning in a Distributed Layered Architecture», en *Multi-Robot Systems: From Swarms to Intelligent Automata: Proceedings from the 2003 International Workshop on Multi-Robot Systems*, 2003, vol. 2, pp. 27-38.

[70] Y. Lei, Q. Zhu, y Z. Feng, «Research on architecture of multiple robots system based on the dynamic networks», en *Information and Automation (ICIA), 2010 IEEE International Conference on*, 2010, pp. 93-98.

[71] A. Marino, LynneE. Parker, G. Antonelli, y F. Caccavale, «A Decentralized Architecture for Multi-Robot Systems Based on the Null-Space-Behavioral Control with Application to Multi-Robot Border Patrolling», *J. Intell. Robot. Syst.*, pp. 1-22, 2012.

[72] J. Bastardas Sandans , J. Ortiz Guerra, V. Sánchez Gistau, J. Sabaté Chueca, «Diagnóstico del TDAH», *Rev. Esp. Pediatría*, vol. 71, n.º 2, pp. 69-64, 2015.

[73] American Psychiatric Association, *Trastorno por déficit de atención con hiperactividad. Manual diagnóstico y estadístico de los trastornos mentales*, 5.ª ed. Panaamericana, 2018.

[74] Cohen, Jacob, *Statistical Power Analysis for the Behavioral Sciences*. Academic Press, 2013.

[75] M. L. Sempere Tortosa, F. Fernández Carrasco, F. J. Mora Lizán, y C. Rizo Maestre, *Objective Analysis of Movement in Subjects with ADHD. Multidisciplinary Control Tool for Students in the Classroom*. 2020.

[76] Agencia Valenciana de Energía, «Guía de Ahorro y Eficiencia Energética en Establecimientos Hoteleros de la Comunidad Valenciana». .

[77] Código Técnico Edificación, «Contribución fotovoltaica mínima de energía eléctrica». 2019.

[78] M. Mohammadi Zanjireh y H. Larijani, *A Survey on Centralised and Distributed Clustering Routing Algorithms for WSNs*, vol. 2015. 2015.

[79] «What is IoT architecture?» 2019, [En línea]. Disponible en: https://www.avsystem.com/blog/what-is-iot-architecture/.

[80] «Ros. Conceptos generales». [En línea]. Disponible en: http://wiki.ros.org/ROS/Concepts.

[81] «Documento Básico HE Ahorro de Energía». .

[82] M. Dorigo, E. Tuci, R. Grob, V. Trianni, T. H. Labella, y S. Nouyan, «The SWARM-BOTS Project», *Swarm Robot. Lect. Notes Comput. Sci.*, vol. 3342, 2005.

[83] A. Akbulut, H. G. Ilk, y F. Ari, «Design, availability and reliability analysis on an experimental outdoor FSO/RF communication system», en *Proceedings of 2005 7th International Conference Transparent Optical Networks, 2005.*, 2005, vol. 1, pp. 403-406 Vol. 1, doi: 10.1109/ICTON.2005.1505831.

[84] F. Nadeem, E. Leitgeb, M. S. Awan, y G. Kandus, «Comparison of different models for prediction of attenuation from visibility data», en *2009 International Workshop on Satellite and Space Communications*, 2009, pp. 269-273, doi: 10.1109/IWSSC.2009.5286374.

[85] O. Bouchet, T. Marquis, M. Chabane, M. Alnaboulsi, y H. Sizun, «FSO and quality of service software prediction», *Proc. SPIE - Int. Soc. Opt. Eng.*, 2005, doi: 10.1117/12.614912.

[86] I. Gravagne y II Marks, «Emergent Behaviors of Protector, Refugee, and Aggressor Swarms», *Syst. Man Cybern. Part B Cybern. IEEE Trans. On*, vol. 37, pp. 471-476, 2007, doi: 10.1109/TSMCB.2006.883427.