Coordinación y control de robots móviles basado en agentes.

UNIVERSIDAD
POLITECNICA
DE VALENCIA

etsinf

Escola Tècnica
Superior d'Enginyeria
Informàtica

PROYECTO FINAL DE CARRERA

AUTOR:

ADRIÁN CERVERA ANDÉS

DIRECTORES:

DR. ÁNGEL VALERA FERNÁNDEZ

DRA. MARINA VALLÉS MIQUEL

Valencia, julio de 2011

A los que quiero: a mi familia y amigos.

A mis padres y hermana, por su constante apoyo.

A mis abuelos, especialmente a mi abuelo Toni, por enseñarme tanto.

A mis directores, por su ayuda y colaboración.

"*Hay una fuerza motriz más poderosa*

que el vapor, la electricidad y la

energía atómica: la voluntad"

Albert Einstein

Índice:

1.- INTRODUCCIÓN_____9

1.1. Introducción……9

1.2. Objetivos……10

2.- DESARROLLO TEÓRICO I: ROBÓTICA_____12

2.1. Introducción a la robótica…… 12

2.2. Historia de la robótica antigua…… 12

2.3. Tipos de robots…… 19

2.3.1. Robots Industriales……19

2.3.1.1. Robots industriales de primera generación……19

2.3.1.2. Robots industriales de segunda generación……20

2.3.1.3. Robots industriales de tercera generación……21

2.3.1.4. Robots industriales de cuarta generación……21

2.3.1.5. Robots industriales de quinta generación……22

2.3.2. Robots móviles……23

2.3.2.1. Robots rodantes……23

2.3.2.2. Robots andantes……25

2.3.2.3. Robots reptadores……26

2.3.2.4. Robots nadadores……27

2.3.2.5. Robots voladores……27

2.4. Sistemas de control……28

2.4.1. Sistemas en bucle abierto……28

2.4.2. Sistemas en bucle cerrado……29

2.5. Comunicaciones inalámbricas……29

2.5.1. Wi-Fi……30

2.5.2. Bluetooth……31

2.5.2.1. Wibree……32
2.5.3. Zigbee……32
2.6. LEGO NXT……35
2.6.1. Introducción……35
2.6.2. Ladrillo NXT……35
2.6.2.1. Microcontrolador……36
2.6.2.2. Entradas y salidas……36
2.6.2.3. Comunicaciones……37
2.6.3. Motores……37
2.6.4. Sensores……38
2.6.4.1. Sensores originales de LEGO……38
2.6.4.2. Sensores de terceros……40
2.6.5. Programación……41
3.- DESARROLLO TEÓRICO II: TECNOLOGÍA AGENTE_____44
3.1. Historia……44
3.2. ¿Qué es una agente software? ……44
3.3. Tipos de agentes……47
3.4. ¿Por qué agentes? ……49
3.5. Comunicación entre agentes……50
3.5.1. Introducción……50
3.5.2. Organizaciones de estandarización……51
3.5.2.1. OMG……51
3.5.2.2. KSE……52
3.5.2.3. FIPA……52
3.6. FIPA (Foundation for Intelligent Physical Agents) ……53
3.6.1. Modelo de referencia de FIPA……53

3.6.2. Comunicación FIPA – FIPA ACL……55

3.7. JADE (Java Agent DEvelopment Framework) ……..57

3.7.1. La plataforma de agentes JADE……57

3.7.2. Comportamientos……58

3.7.2.1. Planificación y ejecución de comportamientos……58

3.7.2.2. Tipos de comportamientos……60

3.7.3. Directorio y servicios……61

3.7.3.1. Publicación de servicios……62

3.7.3.2. Busqueda de servicios……63

3.7.4. Comunicacion entre agentes……64

3.7.4.1. Envio de mensajes……64

3.7.4.2. Recepción de mensajes……65

4.- DESARROLLO PRÁCTICO_____66

4.1. Configuración inicial……66

4.1.1. Instalación del JAVA JDK……66

4.1.2. Instalación del driver USB de Lego……66

4.1.3. Instalación de Lego NXJ……66

4.1.4. Instalación y configuración de la plataforma JADE……67

4.1.5. Configuración del entorno de desarrollo (ECLIPSE) ……69

4.1.5.1. Compilación y ejecución para NXJ……69

4.1.5.2. Compilación y ejecución de agentes JADE……72

4.2. Herramientas de administración y depuración……75

4.2.1. Lanzamiento de agentes usando la interfaz……75

4.2.2. Envío de mensajes ACL……76

4.2.3. Dummy Agent……78

4.2.4. DF (Directory Facilator) GUI……79

4.2.5. Sniffer Agent……81
4.2.6. Instrospector Agent……82
4.3. Distribución de la aplicación en varios PCs……83
4.4. Java Threads: Concurrencia en JAVA……85
4.5. Aplicación I: Robots limpiadores……86
4.5.1. Introducción……86
4.5.2. Descripción del escenario……86
4.5.3. Captura y procesamiento de imágenes……87
4.5.4. Programación de los algoritmos……90
4.5.4.1. AgenteCámara……90
4.5.4.2. AgenteLimpiador……92
4.5.4.3. Código robots……93
4.6. Aplicación II: Generación de convoyes……93
4.6.1. Introducción……93
4.6.2. Descripción del escenario……93
4.6.2.1. Configuración del robot líder del convoy……94
4.6.2.2. Configuración del robot seguidor……95
4.6.3. Programación del algoritmo……96
4.6.3.1. AgenteLider……96
4.6.3.2. AgenteSeguidor……97
4.6.3.3. AgenteSolicitador……97
4.6.3.4. Programación de los robots……98
4.6.3.5. Programación de la cámara NXTCam……99
4.6.3.6. Algoritmos de evasión de obstáculos……101
4.6.3.7. División de convoyes……103
4.6.3.8. Abandono del convoy por batería agotada……105

4.7. Problemas encontrados y soluciones……106

4.7.1. Problemas por luz ambiental……106

4.7.2. Conexión Bluetooth y Threads……107

5.- CONCLUSIONES_____109

6.- BIBLIOGRAFÍA_____110

Anexo A. Programación de agentes con comportamientos……111

Anexo B. Programación y utilización del sensor NXTCam……116

1.- INTRODUCCIÓN

1.1. Introducción y justificación

Vivimos en una sociedad en la que las necesidades de comunicación se han incrementado de forma exponencial, hasta el extremo que necesitamos estar constantemente conectados entre nosotros. La utilización y proliferación de internet ha sido, sin duda, una de las causantes de este fenómeno. En concreto, en Europa, más del 50% de las personas disponen de una conexión con la que acceder a los contenidos de la red de redes. Esto, junto con la aparición de dispositivos que utilizan métodos de acceso inalámbrico (como portátiles, móviles, PDAs, tabletas y netbooks) hacen que cada día sean más las personas que deciden utilizar estas tecnologías.

Este crecimiento en su utilización ha dado lugar a un incesante trabajo por encontrar tecnologías de información cada vez más potentes y con mayores prestaciones, desarrollando protocolos de comunicaciones inalámbricas como Wi-Fi, Bluetooth y WiMax.

Al mismo tiempo, las tecnologías electrónicas y mecánicas también han evolucionado, incrementando la capacidad de cómputo a precios muy asequibles.

La unión de estas ciencias ha dado lugar a disciplinas como la mecatrónica, sinergia que une la ingeniería mecánica, la ingeniería electrónica y la ingeniería informática, y que tiene como objetivo utilizar un computador en la automatización y control de dispositivos mecánicos y electrónicos. La mecatrónica nos rodea en nuestro día a día, en casa (lavadora, televisión, microondas), en el trabajo (aire acondicionado, calefacción, fotocopiadora), mientras conducimos, etc. Nos permite incluso controlar nuestra casa de forma automática y remota (domótica) o solucionar problemas físicos en los seres humanos (biónica).

El uso masivo de Internet ha propiciado también la aparición de aplicaciones basadas en agentes, desarrolladas sobre distintas plataformas. La interoperabilidad entre distintas plataformas de agentes es una característica crítica para el desarrollo de aplicaciones heterogéneas. Para conseguir esta interoperabilidad es muy importante el uso de las especificaciones propuestas por FIPA, organización encargada de la producción de estándares en el área de los agentes.

Actualmente, el concepto de robótica ha evolucionado hacia los sistemas móviles autónomos, que son aquellos que son capaces de desenvolverse por sí mismos en entornos desconocidos y parcialmente cambiantes sin necesidad de supervisión. Uno de los grandes retos de la robótica se basa en la colaboración y autodeterminación de sus movimientos, buscando que los robots puedan trabajar de una forma lo más autónoma e independiente del ser humano posible. Para ello, deberán hacer uso de

todas sus posibilidades, siendo las más importantes la sensorización, el control y la comunicación.

Este proyecto engloba el uso de las nuevas tecnologías mecánicas y electrónicas, como los robots de Lego Mindstorms NXT, con el uso de últimas tecnologías en comunicación, como el Bluetooth, y con el estudio y utilización de las nuevas tecnología agente, a través de la cual podemos obtener soluciones para la colaboración y la coordinación de robots móviles, además de ayudar a la integración de distintos robots para la consecución de nuestros objetivos.

1.2. Objetivos

Este proyecto pretende dotar de capacidades de autonomía y cooperación a una holarquía de robots móviles, que permitan una respuesta dinámica y cooperativa frente a posibles situaciones cambiantes (en su entorno) por medio de la adaptación y/o evolución de la propia estructura organizativa, de tal manera que ésta sea capaz de detectar situaciones de interés (por ejemplo, necesidad de adaptación para maximizar el desempeño del sistema o para mejorar la solución), y que pueda manejarlo maximizando la flexibilidad y la capacidad de respuesta ante el cambio.

En este proyecto se trabajará en modelos basados en agentes para lograr la implementación de un modelo de cooperación o comunicación entre los distintos nodos de control locales, con el fin de lograr un objetivo común. Los agentes de control, diseñados específicamente para cumplir las restricciones de estabilidad, robustez y tiempo real en entornos de control distribuido, deben poder ser capaces de comunicarse de forma segura y tomar decisiones de forma conjunta para lograr objetivos comunes, no sólo entre nodos locales y un agente global supervisor, sino en un nivel horizontal, con canales de comunicación entre los propios agentes o nodos locales.

Además, se buscan métodos para poder integrar distintas plataformas, con distinto hardware, para la consecución de un objetivo común.

En concreto, en el proyecto se han resuelto las siguientes actividades:

- Se ha diseñado un modelo holónico adaptativo basado en agentes para entornos dinámicos cooperativos.
- Se han implementado varias soluciones en las cuales se establece la comunicación y la coordinación entre distintos agentes.

- Se han creado agentes que se encargan del control de los distintos robots móviles, los cuales permiten que éstos trabajen de forma coordinada y colaborativa.

- Se ha desarrollado el sistema de comunicaciones necesario para poder intercambiar información entre más de un robot simultáneamente sin colisiones, mediante la utilización de un único dispositivo Bluetooth.

- Se ha conseguido distribuir la aplicación en más de un computador, siendo capaces los agentes de intercambiar información a través de la red, facilitando la incorporación de robots y la escalabilidad de la solución.

2.- DESARROLLO TEÓRICO I: ROBÓTICA

2.1. Introducción a la robótica.

La imagen del robot como máquina a semejanza con el ser humano ha prevalecido en las culturas desde hace muchos siglos. El afán por fabricar máquinas capaces de realizar tareas independientes ha sido una constante en la historia, a través de la que se han descrito infinidad de ingenios, antecesores directos de los actuales robots.

El término "robot" fue utilizado por primera vez por Karel Capek en su obra de teatro "R.U.R." (*Rossum's Universal Robots*) la cual fue escrita en colaboración con su hermano Josef y publicada en 1920. La palabra "robot" viene del vocablo checo "robota" que significa "trabajo", en el sentido de la obligatoriedad, entendido como servidumbre, trabajo forzado o esclavitud, en referencia sobre todo a los llamados "trabajadores alquilados" que vivieron en el Imperio Austrohúngaro hasta 1848. Este concepto entronca con la terminología "amo-esclavo" de los robots actuales, cuando las unidades basan cada movimiento en una orden humana. En "R.U.R." se desarrolla el concepto de la fabricación en línea ejecutada por robots humanoides, tanto desde el punto de vista narrativo como filosófico. Años más tarde la novela fue adaptada al cine en la película "Metrópolis" y el término robot quedó fijado para ese significado.

2.2. Historia de la robótica antigua

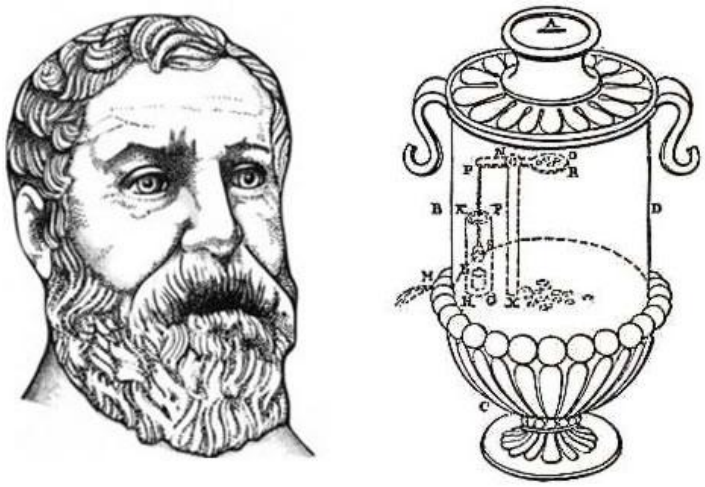
Se tiene constancia que ya, en el año 1300 a.C., Amenhotep, hijo de Hapu, hace construir una estatua de Memón, rey de Etiopía, que emite sonidos cuando la iluminan los rayos de sol al amanecer. Los egipcios fueron los primeros en desarrollar modelos matemáticos muy avanzados y en construir automatismos sofisticados, como el reloj de agua. Se tiene constancia de la existencia del ábaco ya entre el año 1000 y 500 a.C., aunque existen dudas de si fue en Babilonia o en China donde fue inventado. Este ingenio matemático permitió el desarrollo de la computación y la inteligencia artificial que fueron desarrollándose paralelamente al interés por los automatismos y el diseño de máquinas imitadoras del ser humano.

Todavía en el 500 a.C., King-su Tse inventa en China una urraca voladora de madera y bambú, y un caballo de madera capaz de dar saltos. Hacia el 200 a.C., Filón de Bizancio, inventor de la catapulta repetitiva, construye un autómata acuático. Años más tarde, en el año 206 a.C., durante el reinado del primer emperador Han, fue encontrado el tesoro de Chin Shih Huang Ti, consistente en una orquesta mecánica de muñecos que se movían de una forma independiente.

En la antigua Grecia, Arquitas de Tarento, filósofo, matemático y político coetáneo de Platón, considerado el padre de la ingeniería mecánica y precursor occidental de la

robótica, inventó el tornillo y la polea, entre otros muchos dispositivos. Fabricó el primer cohete autopropulsado de la historia, que usó con fines militares. Hacia el año 400 a. C. construyó un autómata consistente en una paloma de madera que rotaba por si sola gracias a un surtidor de agua o vapor y simulaba el vuelo. Hacia el 300 a. C., Ctesibio inventa un reloj de agua (o clepsidra) y un órgano que emite los sonidos por impulsos de agua.

En el año 62 de nuestra era Heron de Alejandría muestra, en su libro "Autómata", los diseños de juguetes capaces de moverse por si solos de forma repetida, como aves que vuelan, gorjean y beben; o ingenios que funcionaban a partir de la fuerza generada por aspas de molino o circuitos de agua en ebullición, precursores rudimentarios de la turbina de vapor.



Herón de Alejandría junto uno de sus inventos, un dispensador de agua sagrada operado por monedas.

Figura 1: Herón de Alejandría junto a uno de sus inventos

Los relojes pueden considerarse como las máquinas antiguas más perfectas, muy cercanas al concepto de automatismo y, consecutivamente, a la de robótica. Es frecuente hallar relojes que incluyen figuras humanas móviles que se mueven con el orden de las horas. El reloj de la catedral de Múnich y el reloj del Ánker de Viena son buenos ejemplos. El Gallo de Estrasburgo, el robot más antiguo que se conserva en la actualidad, funcionó desde 1352 hasta 1789. Formaba parte del reloj de la catedral y, al dar las horas, movía el pico y las alas20. En España, el Papamoscas de la catedral de Burgos, construido en el siglo XVI, consiste en un hombre mecánico que se mueve con los cambios horarios y funciona aún hoy día.

Figura 2: Papamoscas de la catedral de Burgos

Los autómatas más famosos del Medievo son el hombre de hierro de Alberto Magno (1204-1282) o la cabeza parlante de Roger Bacon (1214-1294). Leonardo da Vinci (1452-1519) construyó para el rey Luis XII de Francia un León Mecánico, que se abría el pecho con la garra y mostraba el escudo de armas real.

Figura 3: León mecánico de Leonardo Da Vinci

En la España del siglo XVI Juanelo Turriano (1500?-1585), relojero del emperador Carlos V, construye el "Hombre de Palo", un monje autómata capaz de andar y mover la cabeza.

En el siglo XVII G. W. von Leibniz (1646-1716) abogó por el empleo del sistema binario como base para el cálculo automático, sentando definitivamente las bases de la computación actual. Los materiales empleados para la construcción de autómatas eran la madera (partes formes), el hierro (estructura fija, soportes, goznes), el cobre

(que es moldeable y permite construir partes más finas), el cuero (cables, calzado) y los tejidos. Los primeros modelos utilizaban la aplicación de fuerza directa para realizar los movimientos, facilitados con juegos de poleas, engranajes y palancas.

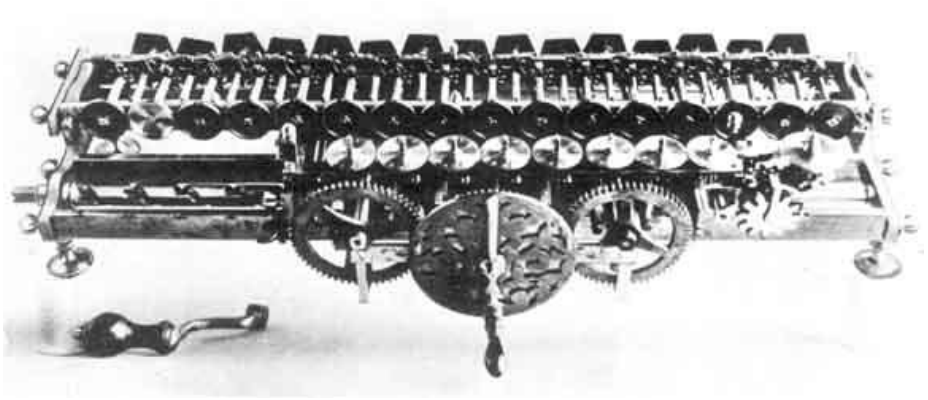


Figura 4: Step Reckoner de G.W. Leibniz

El desarrollo de modelos matemáticos con operativa mecánica, como el de George Boole (1815-1864), permitió pasar de la robótica clásica a la moderna tomando la computación como base. Jacques Vaucanson (1709-1782) es uno de los más famosos y completos constructores de androides automatizados de la historia. Persona de gran ingenio recorrió toda Europa presentando sus artefactos en las cortes de época. En 1738 montó un autómata flautista capaz de ejecutar melodías barrocas. El muñeco realizaba la digitación sobre el instrumento y seguía con los ojos la partitura. Además consiguió uno de los hitos más sonados de la historia de la robótica al construir un pato mecánico de más de 400 piezas móviles, capaz de graznar y comer de la mano del público, completando de forma total la digestión. En el museo de autómatas de Grenoble existe una copia del pato de Vaucanson.

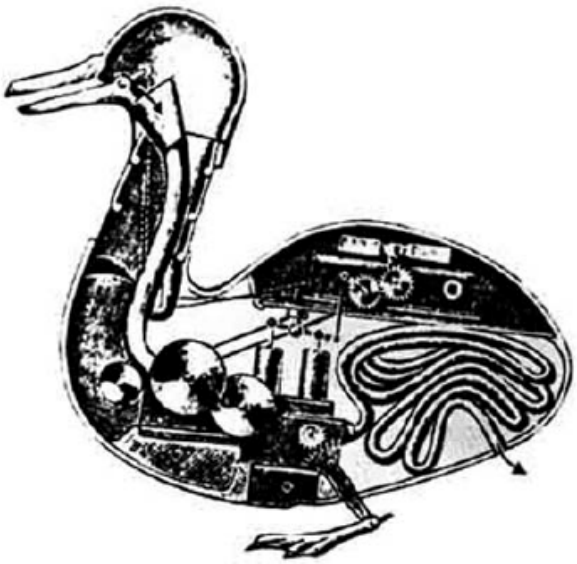


Figura 5: Pato de Vaucanson

La construcción de miembros artificiales se remonta a Ambroise Paré (1510-1590) que desarrolló modelos de sorprendente complejidad.

En 1769, el ingeniero húngaro Johann Wolfgang Ritter von Kempelen (1734-1804), construye uno de los autómatas más famosos de la historia: una máquina para jugar al ajedrez. Se trataba de un dispositivo puramente mecánico. Por su aspecto la máquina era conocida como “el turco”. Sobre la mesa había un tablero de ajedrez, y en el interior unos finos engranajes y resortes que imprimían movimiento a sus manos, que iban cambiando las fichas de posición a medida que transcurría la partida. El maniquí ganaba las partidas más complicadas y se hizo famoso en toda Europacuando derrotó por tres veces a Napoleón Bonaparte. Nadie consiguió descubrir el secreto de esta máquina y, por supuesto, todos ignoraron que se habían enfrentado en realidad al campeón de ajedrez Johann Allgaier, oculto dentro del cajón.

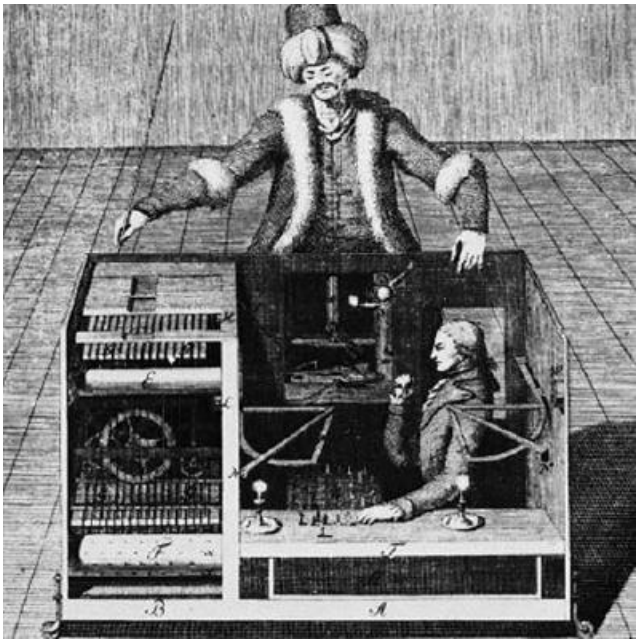


Figura 6: Máquina para jugar al ajedrez conocida como "El Turco"

En esta época los autómatas tenían sobre todo una intención lúdica y eran exhibidos en ferias y circos. Sólo después de unos años, a partir de la revolución industrial, empezaron a ser usados para funciones productivas.

En 1898 Nicola Tesla (1856-1943), inventor del motor eléctrico de corriente alterna, presenta el que algunos consideran el primer robot de la historia moderna, un barco teledirigido, a partir del que patentó el Teleautomation, un torpedo teledirigido para uso militar.

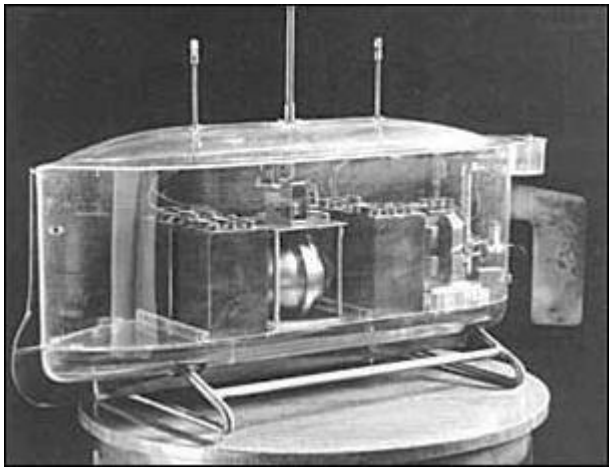


Figura 7: Barco teledirigido de Nicola Tesla

Sin embargo, no fue hasta 1921 cuando se escuchó por primera vez la palabra robot, utilizada por el escritor checo Karel Capek en su obra de teatro R.U.R (Rossum´s Universal Robots). La palabra robot viene del vocablo checo 'Robota' que significa "trabajo", entendido como servidumbre, trabajo forzado o esclavitud. Más tarde, Isaac Asimov (1920-1992) postuló las tres leyes de la robótica en su libro I Robot (Yo robot) publicado en 1950, coincidiendo con el apogeo de la robótica moderna.

Los primeros robots industriales empezaron a producirse a principios de los años 60 y estaban diseñados principalmente para realizar trabajos mecánicos difíciles y peligrosos. Las áreas donde estos robots tuvieron su aplicación fueron trabajos laboriosos y repetitivos, como la carga y descarga de hornos de fundición. En 1961 el inventor norteamericano George Devol patentaba el primer robot programable de la historia, conocido como Unimate, estableciendo las bases de la robótica industrial moderna.

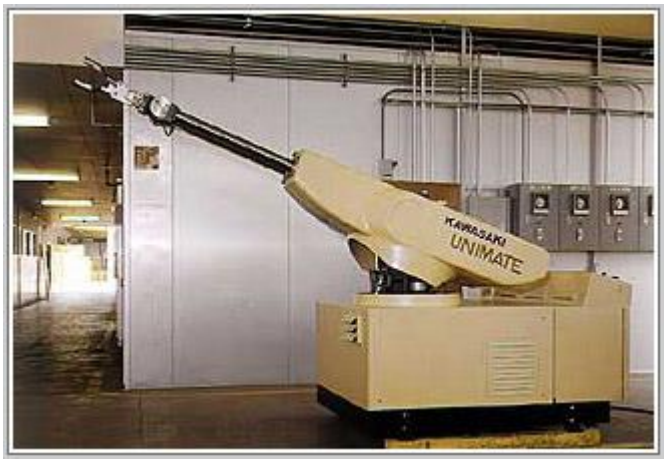


Figura 8: Unimate

Este robot industrial era un manipulador que formaba parte de una célula de trabajo en la empresa de automóviles Ford Motors Company, diseñado para levantar y apilar grandes piezas de metal caliente, de hasta 225 kg, de una troqueladora de fundición por inyección.

Debido a los continuos avances en la informática y la electrónica, a partir de 1970 fueron desarrollados diversos robots programables, siendo de gran importancia en la industria mecánica, tanto en las líneas de ensamblaje como en aplicaciones como la soldadura o pintura.

En los últimos años, los robots han tomado posición en todas las áreas productivas industriales. La incorporación del robot al proceso productivo ha representado uno de

los avances más espectaculares de la edad moderna. En poco más de cuarenta años, se ha pasado de aquellos primeros modelos, rudos y limitados, a sofisticadas máquinas capaces de sustituir al hombre en todo tipo de tareas repetitivas o peligrosas, y además, hacerlo de forma más rápida, precisa y económica que el ser humano. Hoy en día, se calcula que el número de robots industriales instalados en el mundo es de un millón de unidades, unos 20.000 en España, siendo Japón el país más tecnológicamente avanzado, con una media de 322 robots por cada 10.000 trabajadores.

2.3. Tipos de robots

Aquí lo que se pretende es describir las posibles categorías en las que se pueden clasificar los robots actuales. Aunque se pueden realizar distintas clasificaciones (por grado de autonomía, por tipo de propósito, por función, por medio, por tamaño y peso, anatomía e inteligencia), aquí vamos a describir los principales tipos de robots que se utilizan en la actualidad.

2.3.1. Robots Industriales

Se entiende por Robot Industrial a un dispositivo de maniobra destinado a ser utilizado en la industria y dotado de uno o varios brazos, fácilmente programable para cumplir operaciones diversas con varios grados de libertad y destinado a sustituir la actividad física del hombre en las tareas repetitivas, monótonas, desagradables o peligrosas. Por su parte, la Organización Internacional de Estándares (ISO) que define al robot industrial como un manipulador multifuncional reprogramable con varios grados de libertad, capaz de manipular materias, piezas, herramientas o dispositivos especiales según trayectorias variables programadas para realizar tareas diversas. Una clasificación del grado de complejidad del Robot puede establecerse de la siguiente forma:

2.3.1.1. Robots de primera generación

Dispositivos que actúan como "esclavo" mecánico de un hombre, quien provee mediante su intervención directa el control de los órganos de movimiento. Esta transmisión tiene lugar mediante servomecanismos actuados por las extremidades superiores del hombre, caso típico manipulación de materiales radiactivos, obtención de muestras submarinas, etc.

El sistema de control usado en la primera generación de robots esta basado en la “paradas fijas” mecánicamente. Esta estrategia es conocida como control de lazo abierto o control “bang bang”.

Figura 9: Robot de primera generación

2.3.1.2. Robots de segunda generación

El dispositivo actúa automáticamente sin intervención humana frente a posiciones fijas en las que el trabajo ha sido preparado y ubicado de modo adecuado ejecutando movimientos repetitivos en el tiempo, que obedecen a lógicas combinatorias, secuenciales, programadores paso a paso, neumáticos o Controladores Lógicos Programables. Utiliza una estructura de control de ciclo abierto, pero en lugar de utilizar interruptores y botones mecánicos utiliza una secuencia numérica de control de movimientos almacenados en un disco o cinta magnética. El mayor número de aplicaciones en los que se utilizan los robots de esta generación son de la industria automotriz, en soldadura, pintado, etc.

Figura 10: Robot de segunda generación de ABB

2.3.1.3. Robots de tercera generación

La tercera generación de robots utiliza las computadoras para su estrategia de control y tiene algún conocimiento del ambiente local a través del uso de sensores, los cuales miden el ambiente y modifican su estrategia de control, con esta generación se inicia la era de los robots inteligentes y aparecen los lenguajes de programación para escribir los programas de control. La estrategia de control utilizada se denomina de "ciclo cerrado".

Figura 11: Robot de tercera generación

2.3.1.4. Robots de cuarta generación

La cuarta generación de robots, ya los califica de inteligentes con más y mejores extensiones sensoriales, para comprender sus acciones y el mundo que los rodea.

Incorpora un concepto de “modelo del mundo” de su propia conducta y del ambiente en el que operan. Utilizan conocimiento difuso y procesamiento dirigido por expectativas que mejoran el desempeño del sistema de manera que la tarea de los sensores se extiende a la supervisión del ambiente global, registrando los efectos de sus acciones en un modelo del mundo y auxiliar en la determinación de tareas y metas.

Figura 12: Robot de cuarta generación de ABB

2.3.1.5. Robots de quinta generación

La quinta generación, actualmente está en desarrollo esta nueva generación de robots, que pretende que el control emerja de la adecuada organización y distribución de módulos conductuales, esta nueva arquitectura es denominada arquitectura de subsunción, cuyo promotor es Rodney Brooks, donde la toma de decisiones del agente (función acción) se realiza mediante un conjunto de módulos de comportamiento que realizan tareas.

Un modulo de comportamiento suele implementarse como un autómata finito sin ningún tipo de representación o razonamiento simbólico. Casi siempre el comportamiento se implementa como reglas del tipo: Si (situación) entonces (acción), donde situación se toma directamente de la percepción, sin ningún tipo de transformación a representación simbólica.

La selección del comportamiento se basa en la jerarquía de subsunción. Los comportamientos están ordenados por capas y los de las capas mas bajas (máxima prioridad) inhiben a los de las capas superiores (comportamiento más abstracto). Este proyecto final de carrera estaría clasificado en esta generación de robots, ya que sigue

la investigación de la toma de decisiones por parte de uno o varios agentes, el cual se encarga de la activación de los comportamientos necesarios.

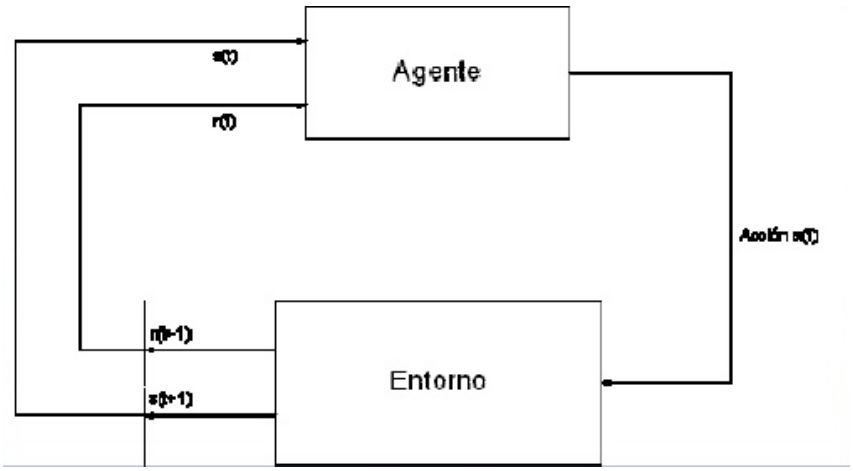


Figura 13: Esquema de funcionamiento de robot de quinta generación

2.3.2. Robótica Móvil

En el apartado anterior se ha realizado un desglose de los diferentes tipos de robots existentes atendiendo a su aplicación, pero más allá de este aspecto práctico hay otro hecho característico de los robots modernos que les confiere un mayor grado de libertad y utilidad. Esta característica es el movimiento en el espacio físico, es decir, la posibilidad de desplazarse por el entorno para observarlo e interactuar con él, y de esta forma emular con mayor fidelidad las funciones y capacidades de los seres vivos.

2.3.2.1. Robots rodantes

Son aquellos que, como su nombre indica, se desplazan haciendo uso de ruedas. Podemos encontrar varias configuraciones para la posición y el número de ruedas. La primera de ellas sería la configuración de Ackerman, la cual se usa casi exclusivamente en la industria del automóvil. Es la configuración que llevan los coches: dos ruedas con tracción traseras, y dos ruedas de dirección delanteras. Esta configuración está diseñada para que la rueda delantera interior en un giro tenga un ángulo ligeramente más agudo que la exterior, y evitar así el derrape de las ruedas.

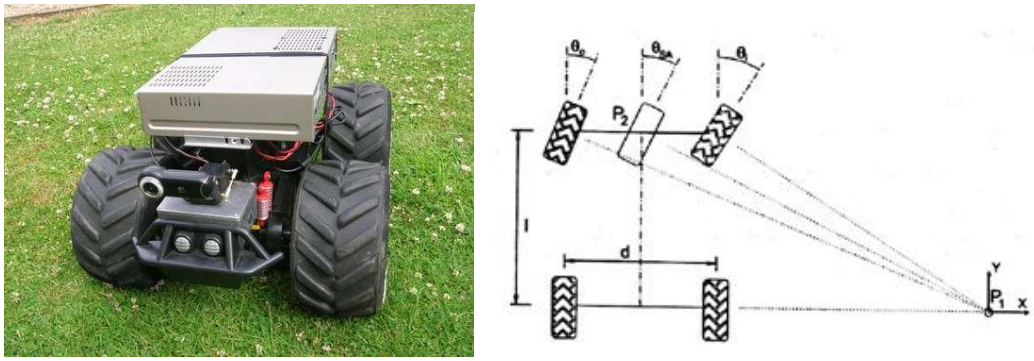


Figura 14: Robot con configuración de Ackerman

También es frecuente encontrar distribuciones de ruedas montadas en configuración diferencial (triciclo) se presenta como la más sencilla de todas. Consta de dos ruedas situadas diametralmente opuestas en un eje perpendicular a la dirección del robot más una rueda loca. Estas ruedas no llevan asociadas ningún motor, giran libremente según la velocidad del robot. Además, pueden orientarse según la dirección del movimiento, de forma análoga a como lo hacen las ruedas traseras de los carritos del supermercado

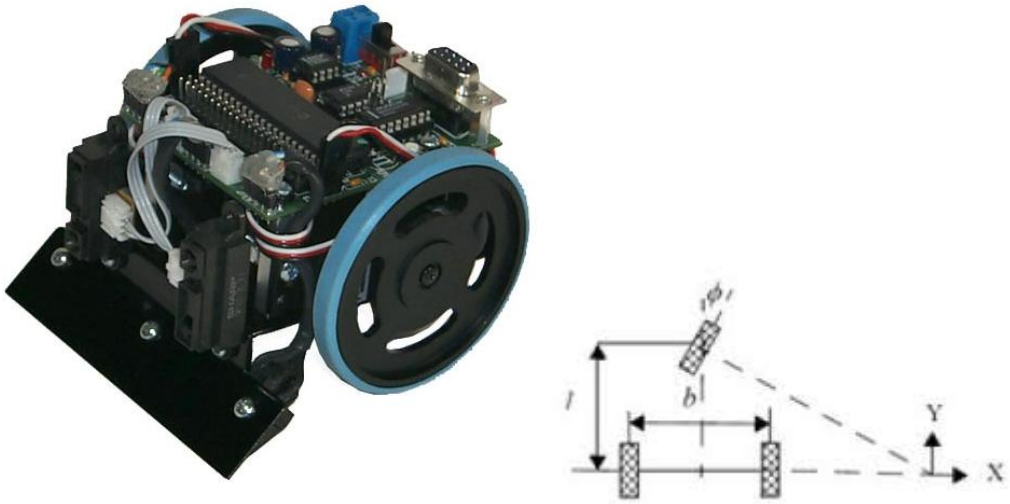


Figura 15: Robot en configuración triciclo

Existen algunos casos especiales en los que se usan otras configuraciones que dotan al robot de mejor adaptación a terrenos difíciles. En estos casos los algoritmos de control de movimiento adquieren una mayor complejidad, proporcional al número de elementos direccionales de forma independiente.

Figura 16: Robot Koala

El último grupo de robots que aquí convendría nombrar es el de los robots que se desplazan mediante la utilización de cadenas. Para la realización de los giros es necesario que ambas cadenas giren en sentidos opuestos dependiendo de cual sea el sentido de giro final del robot, de modo parecido al que se utiliza en la configuración diferencial.

Figura 17: Robot rodante Meccano Spykee

2.3.2.2. Robots andantes

Los robots andantes son aquellos que basan su movilidad en la simulación de los movimientos realizados por los seres humanos al andar. Estos robots están provistos de dos patas con varios grados de libertad, a través de las cuales son capaces de

desplazarse manteniendo el equilibrio. No obstante, este tipo de robots son los más inestables que se encuentran en la actualidad, y los esfuerzos se aúnan en intentar conseguir técnicas de estabilidad y equilibro que eviten que el robot se desestabilice y vuelque en situaciones adversas, como al correr, subir una cuesta, sortear obstáculos, etc.

Sin embargo, podemos encontrar modelos avanzados de este tipo de robot que son capaces de caminar, gatear, bailar e incluso trotar a velocidades de 5 kilómetros/hora.

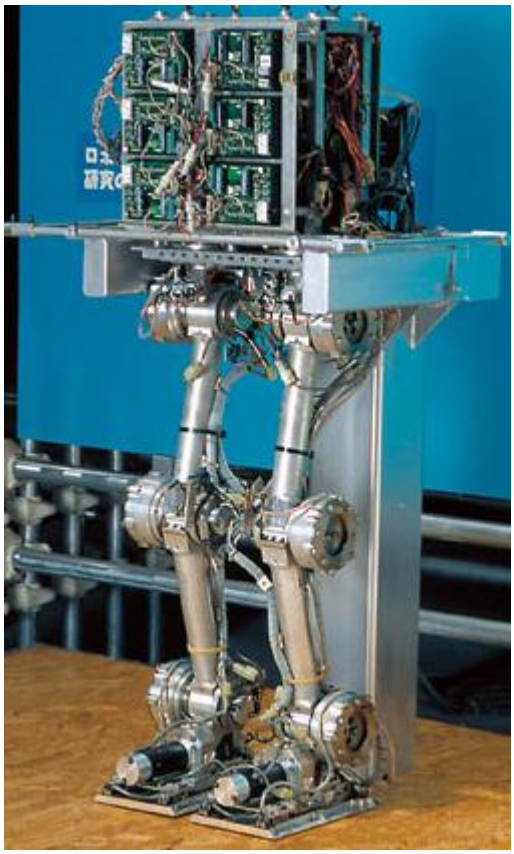


Figura 18: Robot andante de Honda.

2.3.2.3. Robots reptadores

Las características que se buscan son: flexibilidad, versatilidad y adaptabilidad. Están formados por módulos en conexión de viraje-viraje (yaw-yaw) con ruedas pasivas, logrando una propulsión similar a la de las serpientes biológicas.

Figura 19: Robot reptador

2.3.2.4. Robots nadadores

Estos robots son capaces de desenvolverse en el medio acuático, generalmente enfocados a tareas de exploración submarina en zonas donde no es posible llegar pos ser de difícil acceso o estar a profundidades que el cuerpo humano no tolera.

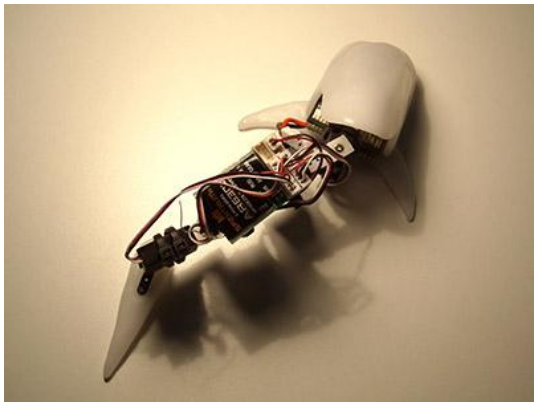


Figura 20: Robot ballena

2.3.2.5. Robots voladores

Este tipo de robots son capaces de desplazarse por el aire, del mismo modo que un avión o un helicóptero. Para ello, incorporan una serie de hélices que se encargan de generar la fuerza necesaria para elevar el robot y de realizar los giros pertinentes para seguir una determinada trayectoria.

Figura 21: Cuadricóptero Ar Drone de Parrot

2.4. Sistemas de control

La aplicación del computador en el control de procesos supone un salto tecnológico enorme que se traduce en la implantación de nuevos sistemas de control en el entorno de la Industria. Desde el punto de vista de la aplicación de las teorías de control automático el computador no esta limitado a emular el cálculo realizado en los reguladores analógicos, el computador permite la implantación de avanzados algoritmos de control mucho más complejos como pueden ser el control óptimo o el control adaptativo. El controlador se encarga de almacenar y procesar la información de los diferentes componentes del robot industrial.

La definición de un sistema de control es la combinación de componentes que actúan juntos para realizar el control de un proceso. Este control se puede hacer de forma continua, es decir en todo momento o de forma discreta, es decir cada cierto tiempo. Si el sistema es continuo, el control se realiza con elementos continuos. En cambio, cuando el sistema es discreto el control se realiza con elementos digitales, como el ordenador, por lo que hay que digitalizar los valores antes de su procesamiento y volver a convertirlos tras el procesamiento.

Existen dos tipos de sistemas, sistemas en lazo abierto y sistemas en lazo cerrado.

2.4.1. Sistemas en bucle abierto

Estos toman la información acerca de las condiciones de operación que reciben de varios sensores y entonces usan esa información para determinar –sea por medios mecánicos o usando medios electrónicos programados- exactamente que acción debe aplicarse para alcanzar la situación deseada. La precisión en la medición depende

enteramente de qué tan bien el sistema –de cualquier tipo que sea- puede predecir las necesidades del motor basado en su "conocimiento" de las condiciones de operación

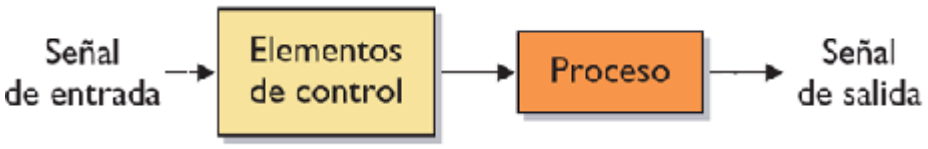


Figura 22: Esquema bucle abierto

2.4.2. Sistemas en bucle cerrado

En un sistema de lazo cerrado ó de retroalimentación, la información acerca de cualquier cosa que esté siendo controlada es continuamente retro-alimentada al sistema como un dato de entrada. La operación de un termostato en un sistema de calefacción automático es un ejemplo de control de lazo cerrado.

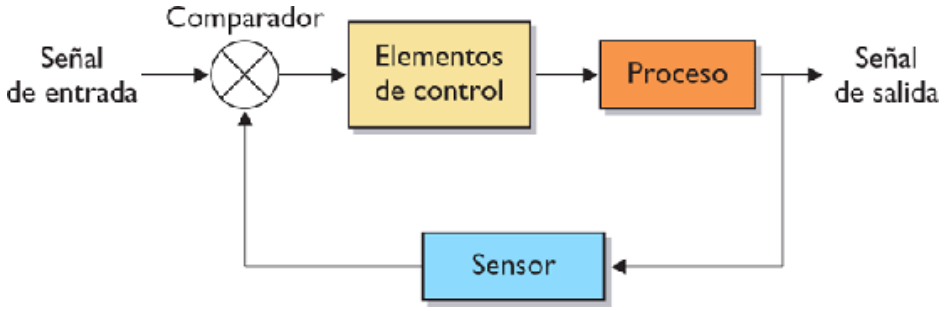


Figura 23: Esquema bucle cerrado

Para el ejemplo del termostato, conforme baja la temperatura, el termostato siente el descenso y le indica al horno que añada calor. Tan pronto como la temperatura sube más allá de lo previsto, el termostato siente el resultado de su propia acción de control –el calor producido por el horno- y le indica a éste que corte el calor.

Un sistema de lazo abierto puede, por ejemplo, sentir la baja de temperatura y simplemente encender el calor por un predeterminado lapso de tiempo; sin embargo el control de lazo cerrado es automático, la temperatura permanece relativamente constante, y el consumo de energía probablemente se reduce. De todas maneras, el resultado es un control mejor y más preciso.

2.5. Comunicaciones inalámbricas

La comunicación inalámbrica o sin cables es aquella en la que extremos de la comunicación (emisor/receptor) no se encuentran unidos por un medio de propagación físico, sino que se utiliza la modulación de ondas electromagnéticas a través del espacio. En este sentido, los dispositivos físicos sólo están presentes en los emisores y receptores de la señal. A continuación se detallarán los principales protocolos de comunicación inalámbrica en el ámbito de la robótica.

2.5.1. Wi-Fi

Wi-Fi es un estándar para la conexión de dispositivos electrónicos mediante tecnología inalámbrica. Hoy en día, la mayor parte de los dispositivos que utilizamos incorporan esta tecnología (teléfonos, consolas, portátiles, etc.) con la que podemos conectarnos a una red (o Internet) sin la necesidad de utilizar cables. Para la conexión tenemos dos modos de acceso posibles:

- (AP) Access point: Permite al resto de dispositivos conectarse a una red inalámbrica, centralizando las conexiones en un único dispositivo. Normalmente está conectado a un encaminador (router) que se encarga de conectar la red inalámbrica con Internet.

- Ad-hoc: Permite el intercambio de información entre varios dispositivos de forma descentralizada, sin la necesidad de utilizar una infraestructura ajena para la conexión.

Wi-Fi es una marca de la Wi-Fi Alliance, la organización comercial que adopta, prueba y certifica que los equipos cumplen los estándares 802.11 relacionados a redes inalámbricas de área local. El alcance de una red Wi-Fi está alrededor de los 100 metros al aire libre, reduciéndose esta distancia con la presencia de obstáculos como paredes.

Los estándares de Wi-Fi IEEE 802.11b, IEEE 802.11g e IEEE 802.11n disfrutan de una aceptación internacional debido a que la banda de 2.4 GHz está disponible casi universalmente, con una velocidad de hasta 11 Mbps , 54 Mbps y 300 Mbps, respectivamente.

En cuanto a los protocolos de seguridad que utiliza existen varias alternativas para garantizar la seguridad de estas redes. Las más comunes son la utilización de protocolos de cifrado de datos para los estándares Wi-Fi como el WEP, el WPA, o el WPA2 que se encargan de codificar la información transmitida para proteger su confidencialidad, proporcionados por los propios dispositivos inalámbricos. La mayoría de las formas son las siguientes:

- WEP, cifra los datos en su red de forma que sólo el destinatario deseado pueda acceder a ellos. Los cifrados de 64 y 128 bits son dos niveles de seguridad WEP.

WEP codifica los datos mediante una “clave” de cifrado antes de enviarlo al aire. Este tipo de cifrado no está muy recomendado, debido a las grandes vulnerabilidades que presenta, ya que cualquier cracker puede conseguir sacar la clave.

- WPA: presenta mejoras como generación dinámica de la clave de acceso. Las claves se insertan como de dígitos alfanuméricos, sin restricción de longitud

Sin embargo, su uso esta más encaminado a la conectividad de dispositivos que buscan un largo alcance y gran ancho de banda, independientemente de otros factores tan importantes en los robots móviles como puede ser el consumo de las baterías.

Figura 24: Logotipo Wi-Fi

2.5.2. Bluetooth

Bluetooth es una especificación industrial para Redes Inalámbricas de Área Personal (WPANs) que posibilita la transmisión de voz y datos entre diferentes dispositivos mediante un enlace por radiofrecuencia en la banda ISM de los 2,4 GHz. Los principales objetivos que se pretenden conseguir con esta norma son:

- Facilitar las comunicaciones entre equipos móviles y fijos.
- Eliminar cables y conectores entre éstos.
- Ofrecer la posibilidad de crear pequeñas redes inalámbricas y facilitar la sincronización de datos entre equipos personales.

Se denomina Bluetooth al protocolo de comunicaciones diseñado especialmente para dispositivos de bajo consumo, con baja cobertura y basados en transceptores de bajo coste.

Gracias a este protocolo, los dispositivos que lo implementan pueden comunicarse entre ellos cuando se encuentran dentro de su alcance. Las comunicaciones se realizan

por radiofrecuencia de forma inalámbrica. Estos dispositivos se clasifican como "Clase 1", "Clase 2" o "Clase 3" en referencia a su potencia de transmisión, siendo totalmente compatibles los dispositivos de una clase con los de las otras.

En cuanto al ancho de banda que nos proporciona una conexión Bluetooth, la versión 1.2 de la especificación consigue transmisión de datos a 1Mbps, la versión 2.0 alcanza los 3Mbps, mientras que la última versión 3.0 ofrece hasta 24Mbps, buscando la compatibilidad con Wi-Fi.

Figura 25: Logotipo Bluetooth

2.5.2.1. Wibree

A partir de la especificación Bluetooth apareció el 12 de junio de 2007 la especificación Wibree. Fue anunciada como una parte de la especificación de Bluetooth encaminada a ser una versión de muy bajo consumo. Sus aplicaciones son principalmente dispositivos sensores o mandos a distancia. Puede resultar interesante para equipamiento médico.

Wibree es una nueva tecnología digital de radio interoperable para pequeños dispositivos. Es la primera tecnología abierta de comunicación inalámbrica, que ofrece comunicación entre dispositivos móviles o computadores y otros dispositivos más pequeños (de pila de botón), diseñada para que funcione con poca energía.

Figura 26: Logotipo WiBree

2.5.3. Zigbee

ZigBee es el nombre de la especificación de un conjunto de protocolos de alto nivel de comunicación inalámbrica para su utilización con radiodifusión digital de bajo consumo, basada en el estándar IEEE 802.15.4 de redes inalámbricas de área personal (wireless personal area network, WPAN). Su objetivo son las aplicaciones que requieren comunicaciones seguras con baja tasa de envío de datos y maximización de la vida útil de sus baterías.

En principio, el ámbito donde se prevé que esta tecnología cobre más fuerza es en domótica, como puede verse en los documentos de la ZigBee Alliance. La razón de ello son diversas características que lo diferencian de otras tecnologías:

- Su bajo consumo.
- Su topología de red en malla.
- Su fácil integración (se pueden fabricar nodos con muy poca electrónica).

Uno de los objetivos de la especificación de ZigBee es ser más simple y barato que otras tecnologías encaminadas a WPANs como puede ser Bluetooth. Si principal campo de integración son aplicaciones que necesitan un reducido tráfico de datos, un bajo consumo de sus baterías y unas conexiones lo más seguras posibles. Estas características hacen que sea ampliamente utilizado en las siguientes aplicaciones:

1. Automoción
2. Aplicaciones industriales
3. Aviónica
4. Entornos inteligentes
5. Domótica y seguridad
6. Monitorización del medio ambiente
7. Identificación de productos
8. Etc.

Este estándar define dos capas físicas: a 868/915 MHz DSSS PHY (espectro con propagación de secuencia directa) y otra a 2450 MHz DSSS PHY. La capa de 2450 MHz soporta un rango de datos de 250 Kb/s y la de 868/915 un rango de datos entre los 20 Kb/s y 40 Kb/s. La elección de cada una de las capas depende de las regulaciones locales y de las preferencias del usuario.

Las principales diferencias respecto a Bluetooth son las siguientes:

- Una red ZigBee puede constar de un máximo de 65535 nodos distribuidos en subredes de 255 nodos, frente a los 8 máximos de una subred (Piconet) Bluetooth.

- Menor consumo eléctrico que el de Bluetooth. En términos exactos, ZigBee tiene un consumo de 30 mA transmitiendo y de 3 uA en reposo, frente a los 40 mA transmitiendo y 0,2 mA en reposo que tiene el Bluetooth. Este menor consumo se debe a que el sistema ZigBee se queda la mayor parte del tiempo dormido, mientras que en una comunicación Bluetooth esto no se puede dar, y siempre se está transmitiendo y/o recibiendo.

- Tiene una velocidad de hasta 250 Kbps, mientras que en Bluetooth es de hasta 3 Mbps.

- Debido a las velocidades de cada uno, uno es más apropiado que el otro para ciertas cosas. Por ejemplo, mientras que el Bluetooth se usa para aplicaciones como los teléfonos móviles y la informática casera, la velocidad del ZigBee se hace insuficiente para estas tareas, desviándolo a usos tales como la Domótica, los productos dependientes de la batería, los sensores médicos, y en artículos de juguetería, en los cuales la transferencia de datos es menor.

- Existe una versión que integra el sistema de radiofrecuencias característico de Bluetooth junto a una interfaz de transmisión de datos vía infrarrojos desarrollado por IBM mediante un protocolo ADSI y MDSI.

Figura 27: Logotipo ZigBee Alliance

Actualmente ya podemos encontrar un dispositivo ZigBee para la transferencia de datos entre robots Lego Mindstorms NXT. Dicho hardware se conecta al ladrillo del NXT como un sensor más y es capaz de conectar entre si una gran cantidad de robots NXT. Su alcance le permite enviar datos hasta una distancia de 1200 metros en exterior y más de 90 metros en interior.

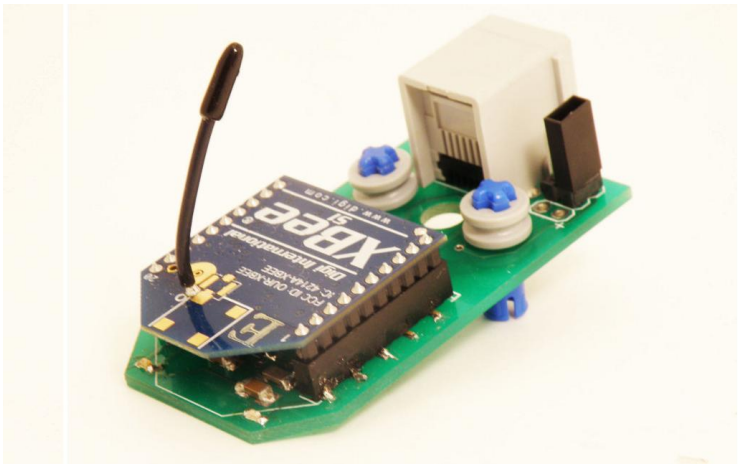


Figura 28: Dispositivo ZigBee para Lego NXT

2.6. LEGO NXT

2.6.1. Introducción

Lego Mindstorms es un kit de robótica fabricado por la empresa Lego, el cual posee elementos básicos de las teorías robóticas, como la unión de piezas y la programación de acciones, en forma interactiva. Este robot fue comercializado por primera vez en septiembre de 1998. Comercialmente se publicita como «Robotic Invention System», en español Sistema de Invención Robotizado (RIS). También se vende como herramienta educacional, lo que originalmente se pensó en una sociedad entre Lego y el MIT. La versión educativa se llama «Lego Mindstorms for Schools», en español Lego Mindstorms para la escuela y viene con un software de programación basado en la GUI de Robolab. Lego Mindstorms puede ser usado para construir un modelo de sistema integrado con partes electromecánicas controladas por computador. Prácticamente todo puede ser representado con las piezas tal como en la vida real, como un elevador o robots industriales.

Lego Mindstorms fue uno de los resultados de la fructífera colaboración entre Lego y el MIT. Esta asociación se emplea como ejemplo de relación entre la industria y la investigación académica que resulta muy beneficiosa para ambos socios. La línea Lego Mindstorms nació en una época difícil para Lego, a partir de un acuerdo entre Lego y el MIT. Según este trato, Lego financiaría investigaciones del grupo de epistemología y aprendizaje del MIT sobre cómo aprenden los niños y a cambio obtendría nuevas ideas para sus productos, que podría lanzar al mercado sin tener que pagar regalías al MIT. Un fruto de esta colaboración fue el desarrollo del MIT Programmable Brick (Ladrillo programable).

2.6.2. Ladrillo NXT

El bloque NXT es una versión mejorada a partir de Lego Mindstorms RCX, que generalmente se considera la predecesora y precursora de los bloques programables de Lego.

Debido a la comercialización de los bloques programables, Lego vendió la generación NXT en dos versiones: Retail Version y Education Base Set. Una ventaja de la versión Educacional es que se incluía las baterías recargables y el cargador, pero esta misma versión debía comprar el software según el tipo de licencia: Personal, Sala de clases, Sitio.

Figura 29: Ladrillo del Lego NXT

2.6.2.1. Microcontrolador

El microcontrolador que posee es un ARM7 de 32 bits, que incluye 256 Kb de memoria Flash y 64 Kb de RAM externa, la cual a diferencia del bloque RCX, posee mayores capacidades de ejecución de programas, evitando que los procesos inherentes de varios paquetes de datos colisionen y produzcan errores y un posible error en la ejecución del software. Su presentación es similar al Hitachi H8 ya que se encuentra en el circuito impreso del bloque, junto a la memoria FLASH.

2.6.2.2. Entradas y salidas

En el bloque de NXT existen cuatro entradas para los sensores, pero los conectores son distintos de los del RCX, lo que impide la conexión de sus motores o sensores, sin embargo, el kit de NXT incluye el adaptador para que los sensores de RCX sean compatibles con NXT.16

Las salidas de energía son tres, localizadas en la parte posterior del bloque, haciendo que la conexión para los motores y partes móviles sean de más fácil acceso.

2.6.2.3. Comunicaciones

El bloque de NXT puede comunicarse con el computador mediante la interfaz de USB que posee, la cual ya viene en la versión 2.0. Además, para comunicarse con otros robots en las cercanías posee una interfaz Bluetooth. Esta conectividad con Bluetooth no tan sólo permite conectarse con otros bloques, sino también con computadores, teléfonos móviles, y otros aparatos con esta interfaz de comunicación.

Dentro de las posibilidades de conexión se encuentran:

- Conectar hasta tres dispositivos distintos
- Buscar y conectarse a otros dispositivos que posean Bluetooth
- Recordar dispositivos con los cuales se ha conectado anteriormente para conectarse más rápidamente
- Establecer el bloque NXT como visible o invisible para el resto de los dispositivos.

2.6.3. Motores

Los motores integrados al bloque son menos versátiles, pero no dependen de conexiones externas, lo cual visualmente ayuda al robot en su presentación. El modelo NXT usa servo motores, los cuales permiten la detección de giros de la rueda, indicando los giros completos o medios giros, que es controlado por el software.

El *encoder* o tacómetro, por su parte, permite conocer el número de vueltas que ha dado el eje del motor con una precisión de 1 grado. Es decir, si el motor realiza un giro completo el *encoder* verá incrementada su cuenta en 360 unidades. Esto es cierto sea cual sea el sentido de movimiento del motor, incrementándose en un sentido y decrementándose al girar en sentido contrario.

Además, el control de la velocidad de los motores se realiza con una señal PWM (Pulse Width Modulation o modulación por ancho de pulso) en la que, dependiendo del ciclo de trabajo de la señal de entrada, podemos incrementar o decrementar la velocidad de los motores.

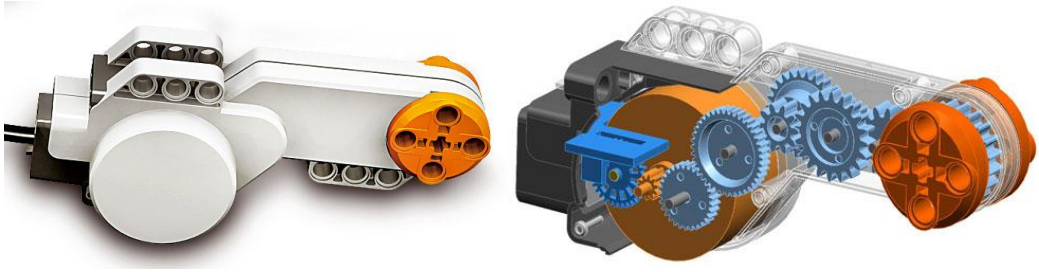


Figura 30: Motor del Lego NXT

2.6.4. Sensores

El robot Lego puede utilizar dos tipos distintos de sensores, tanto los originales de Lego como los que han sido construidos por terceros.

2.6.4.1. Sensores originales Lego

Algunos de los sensores originales de Lego que podemos encontrar son los siguientes:

- Sensor de luz: El sensor de luz permite tomar una muestra de luz mediante un bloque modificado que un extremo trae un conductor eléctrico y por el otro una cámara oscura que capta las luces. Esta cámara es capaz de captar luces entre los rangos de 0,6 a 760 lux. Este valor lo considera como un porcentaje, el cual es procesado por el bloque lógico, obteniendo un porcentaje aproximado de luminosidad. La principal utilización de este sensor es la detección de líneas, en aplicaciones como el seguidor de línea, lucha de sumos, robots limpiadores, etc.

Figura 31: Sensor de luz del Lego NXT

- Sensor de temperatura: El sensor de temperatura permite leer el valor aproximado de la temperatura, mediante la interacción de un termistor en uno de los extremos, generando un campo magnético que permite la detección aproximada de la temperatura del bloque que lo contiene. El bloque original de

Lego posee un termistor de 12.2 Kohms a 25 °C con un coeficiente de corrección aproximado de un -3,7%/°C.

Figura 32: Sensor de temperatura del Lego NXT

- Sensor de contacto: El sensor de contacto permite detectar si el bloque que lo posee ha colisionado o no con algún objeto que se encuentre en su trayectoria inmediata. Al tocar una superficie, una pequeña cabeza externa se contrae, permitiendo que una pieza dentro del bloque cierre un circuito eléctrico comience a circular energía, provocando una variación de energía de 0 a 5 V.

Figura 33: Sensor de contacto del Lego NXT

- Sensor de giro: El sensor de giro permite conocer la posición del robot en cualquier instante.

Figura 34: Sensor de giro del Lego NXT

- Sensor ultrasónico: su principal función detectar las distancias y el movimiento de un objeto que se interponga en el camino del robot, mediante el principio de la detección ultrasónica. Este sensor es capaz de detectar objetos que se encuentren desde 0 a 255 cm, con una precisión relativa de +/- 3 cm. Mediante el principio del eco, el sensor es capaz de recibir la información de los distintos objetos que se encuentren en el campo de detección.

Figura 35: Sensor de ultrasonidos del Lego NXT

2.6.4.2. Sensores de terceros

A parte de los sensores creados por Lego, podemos encontrar varios sensores creados por fabricantes externos a lego y que permiten ampliar la funcionalidad y sensorización de nuestro robot en una gran medida. Algunos de estos sensores se pueden conseguir en:

- HiTechnic (*www.HiTechnic.com*)
- MindSensors (*www.MindSensors.com*)

Los sensores más importantes que podemos destacar en este apartado son los siguientes:

- Sensor de brújula: permite conocer cual es la orientación del robot respecto al norte magnético.
- Sensor de aceleración: Es capaz de medir las aceleraciones en los ejes de coordenadas.
- Sensor cámara: Este es uno de los más importantes y que se han utilizado a lo largo del proyecto. Este sensor es capaz de obtener imágenes del medio en el que opera y obtener su información. Su funcionamiento se detallará en secciones posteriores.

Figura 36: Sensor cámara NXTCam V2

2.6.5. Programación

Para la programación del robot existen varias alternativas a la programación por diagramas de cajas o arbol de decisiones que se proporciona con el robot, el cual tiene una funcionalidad bastante limitada. Para una programación utilizando lenguajes de programación de alto nivel se utilizan principalmente los siguientes lenguajes:

- RobotC: Entorno de desarrollo integrado que tiene como objetivo la programación del NXT o RCX bajo el lenguaje de programación C. Una de sus ventajas es que no necesita sustituir el firmware original del robot, funcionando bajo el original. Entre sus principales características tenemos una interfaz parecida a la de VisualBasic, herramientas para realizar la depuración de nuestras aplicaciones, tutoriales para aprender la sintáxis del lenguaje y permite la ejecución de distintas tareas de forma concurrente.

- LejOS: para la programación en este lenguaje es necesario sustituir el firmware original del brick por éste. Actualmente se puede instalar tanto en el brick RCX como en el NXT. Incluye una máquina virtual de JAVA, la cual permite al Lego Mindstorms ejecutar aplicaciones que se han implementado bajo este lenguaje. Incluye funciones para el control de prácticamente todos los sensores y actuadores, incluso de aquellos que han sido desarrollados por terceros como la brújula, el acelerómetro y la cámara. Este es el firmware que se ha utilizado para el desarrollo de nuestro proyecto.

- BricxCC: Bricx Command Center es un conocido IDE que soporta programación del RCX con NQC, C, C++, Pascal, Forth, y Java utilizando brickOS, pbForth y LeJOS. Con BricxCC se pueden desarrollar programas en NBC y NXC. Tanto NBC como NXC utilizan el firmware estándar del NXT. Este software está disponible en código abierto.

- **NXC:** NXC es un lenguaje de alto nivel similar a C. Utiliza el firmware original de LEGO y está disponible para Windows, Mac OSX y Linux (ia32). Ha sido desarrollado por John Hansen. Hay disponible una guía del programador y un tutorial en ingles (http://bricxcc.sourceforge.net/nbc) y se puede utilizar como editor BricxCC.

En la siguiente tabla podemos ver cuales son los principales entornos de desarrollo disponibles para implementar nuestras aplicaciones y las características que poseen cada uno de ellos.

Características	NXT-G Retail	NXT-G Educational	RoboLab 2.9	NBC	NXC	Robot C	NI LabVIEW Toolkit	leJOS NXJ	pbLua	LEJOS OSEK
Tipo de lenguaje	Gráfico	Gráfico	Gráfico	Ensamblador	Como C	C	Gráfico	Java	Lua	ANSI C
Firmware	Estándar	Standard	Estándar(#1)	Estándar	Estándar	Estándar (#1)	Estándar	Modificado	Modificado	Modificado
IDE (¿incluido?)	Si	Si	Si	Si	Si	Si	No (#6)	plugins para Eclipse y NetBeans	No (#7)	Eclipse CDT(GCC+ ATMEL SAM-BA)
Windows	Si	Si	Si	Si	Si	Si	Si	Si	Si (#7)	Si
Mac OSX	Si	Si	Si	Si	Si	Aún no	Si	Si	Si (#7)	No
Linux	No	No	No	Si	Si	No	No	Si	Si (#7)	No (puede)
Eventos	No	No	Si	No	No	Si	No	Eventos estándar de java		Si (OSEK RTOS)
Multitarea	Si	Si	Si	Si	Si	Si	Si	Si		Si (OSEK RTOS)
Bluetooth Brick hacia PC	Si	Si	No	Si	Si	Si	Si	Si	Si	Si
Bluetooth Brick hacia Brick	Si	Si	No	Si	Si	Si	Si	Si	Si	Aún no
Bluetooth Brick hacia otro dispositivo	No	No	No	No	No	Si	No	Si	Si	No
I2C Support	(#5)	(#5)	Si	Si	Si	Si	Si	Si	Si	Si (EEUU sólo)
File System	Si	Si	Si	Si	Si	Si	Si	Si	Aún no	Sin planear
Floating Point	No	No	Si	No	No	Si	¿No?	Si	(#8)	Si
Datalog	No	No	Si	No	No	Si	¿No?	Si	No	Aún no

Tabla 1: Comparación de lenguajes de programación

3.- DESARROLLO TEÓRICO II: TECNOLOGÍA AGENTE

3.1. Historia

Nwana (1996) prefiere separar la investigación de la tecnología agente en dos hilos principales, el primero de ellos se situaría sobre 1977, mientras que el segundo y último lo haría alrededor de 1990.

La primera línea de investigación tiene sus raíces en la inteligencia artificial distribuida (DAI), dando lugar a que el concepto de agente aparezca en los primeros estudios sobre inteligencia artificial en la década de los 70. De hecho, surge en el momento en el que Carl Hewitt define el modelo de Actor (1977). En este modelo, Hewitt propone el concepto de objeto auto-contenido, interactivo y concurrente. Además, dicho objeto encapsula estados internos y puede responder a mensajes de otros objetos similares.

"*Is a computational agent which has a mail address and a behaviour. Actor communicate by message-passin and carry out theis actions concurrently*" (Hweitt, 1977, p. 131).

En esta primera línea de investigación, el trabajo de los investigadores estaba concentrado en resolver grandes cuestiones básicas, tales como la interacción y la comunicación entre los agentes, la descomposición y distribución de tareas, la coordinación y cooperación, la resolución de conflictos mediante negociación, etc. Su objetivo era especificar, analizar, diseñar e integrar sistemas que comprendían la colaboración de múltiples agentes. Este trabajo se caracterizo también por investigar el desarrollo teórico, arquitectónico y de aspectos del lenguaje.

Sin embargo, desde 1990, y contrastando con la anterior, surge la actual línea de investigación, la cual es más reciente y está en constante movimiento, estudiando un rango mas amplio de tipos de agentes. La gran diversidad de aplicaciones e investigaciones son solo una demostración de que los agentes software se están convirtiendo en una corriente principal.

3.2. ¿Qué es un agente software?

Actualmente se está produciendo un gran uso de la palabra agente sin tener un pleno conocimiento de lo que ello significa. Algunos programas son llamados agentes simplemente porque son capaces de planificarse para la consecución de un objetivo o tarea en una maquina remota; algunos porque logran controlar tareas de bajo nivel mientras utilizan lenguajes de programación de alto nivel; algunos porque son una abstracción o encapsulación de fuentes de información o servicios; otros porque

implementan funciones cognitivas, o porque tienen funciones parecidas a la inteligencia distribuida, etc.

El término agente describe una abstracción de software, una idea o concepto, similar a los métodos, funciones y objetos en la programación orientada a objetos. El concepto de un agente provee una forma conveniente y poderosa de describir una compleja entidad de software, que es capaz de actuar con cierto grado de autonomía, para cumplir tareas en representación de personas. Sin embargo, y a diferencia de los objetos (que son definidos por métodos y atributos), un agente es definido por su propio comportamiento.

Los agentes software surgen dentro de la Inteligencia Artificial y, a partir de los trabajos desarrollados en el área de la Inteligencia Artificial Distribuida (DAI), surge el concepto de sistemas multiagente.

Podemos encontrar otra definición de agente software mucho mas especifica y que seguramente los investigadores de esta tecnología encuentran mas aceptable: Un agente software es una entidad la cual funciona continua y autónomamente en un entorno particular, a menudo inhabitado por otros agentes o procesos. La necesidad de continuidad y autonomía deriva de nuestra necesidad de que un agente sea capaz de llevar a cabo actividades de forma flexible e inteligente como respuesta de los cambios del entorno, pero sin la intervención de un humano. Idealmente, un agente que funciona continuamente en un entorno por un largo periodo de tiempo, debe ser capaz de aprender de esta experiencia. Además, se busca que un agente habite en un determinado ambiente con otros agentes y tenga las habilidades necesarias para comunicarse y cooperar entre ellos, moviéndose de un lado a otro como consecuencia de ello.

El termino agente software engloba a una gran cantidad de tipos de agentes, mucho más específicos y concretos. Coherente con los requerimientos de un problema en particular, cada agente puede poseer un mayor o menos grado de atributos. Dichos atributos. Franklin y Graesser (1997) discutieron sobre estos atributos clave que definen a cualquier agente, los cuales son:

- Persistencia: el código no es ejecutado bajo demanda sino que se ejecuta continuamente y decide por si mismo cuando debería llevar a cabo alguna actividad.

- Autonomía: pueden trabajar sin intervención directa del usuario y tienen cierto control sobre sus acciones y estado interno. Los agentes tienen la capacidad de seleccionar tareas, priorizarlas, tomar decisiones sin intervención humana, etc.

- Capacidad o habilidad social: tienen la habilidad de sincronizarse con personar y otros agentes, a través de coordinación y comunicación, para colaborar en la consecución de una determinada tarea.
- Reactividad: pueden percibir su entorno (que puede ser el mundo físico, un usuario detrás de una interfaz gráfica o vocal, aplicaciones en la red, u otros agentes) y responder oportunamente a cambios que se produzcan en el mismo.
- Iniciativa: el comportamiento de los agentes esta determinado por los objetivos (metas) que persiguen y por tanto pueden producir acciones no sólo como respuesta al entorno.
- Continuidad temporal: persistencia de su identidad y estado durante largos periodos de tiempo.
- Adaptabilidad: deben ser capaces de aprender e improvisar a partir de su experiencia.
- Movilidad: deben ser capaces de desplazarse de una posición a otra de forma autodirigida.

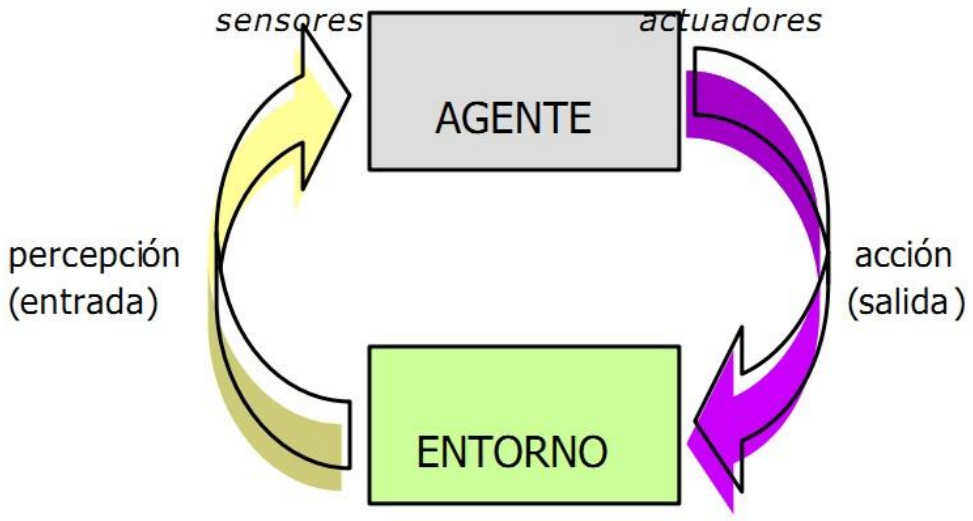


Figura 37: Relación entre el agente y el entorno

Es bastante común no comprender qué es y qué no es un agente. Por esto, debemos aclarar que un agente no es:

a) Un programa o un método. La principal diferencia la encontramos en cuatro de las cualidades que hemos descrito anteriormente: reactividad, autonomía, persistencia y búsqueda de objetivos.

b) Un objeto. Los agentes son entidades mucho más autónomas que los objetos, ya que tienen comportamientos flexibles, reactivos, proactivos y sociales. Además, un agente tiene al menos un hilo de ejecución que se encarga del control, aunque puede tener incluso más.

c) Un sistema experto. Los sistemas expertos no están acoplados a su entorno, además de no tener designado ningún comportamiento de tipo reactivo o proactivo, además de no poseer habilidades sociales.

3.3. Tipos de agentes

Nwana propone, en 1996, una tipología de agentes que identifica otras dimensiones de clasificación, con lo que clasifica a los agentes acorde a:

- Movilidad: estático o móvil.
- Presencia de un modelo de razonamiento simbólico: deliberativo o reactivo.
- Exhibición de atributos ideales y primarios: Autonomía, cooperación, aprendizaje. Para estas características, Nwana deriva cuatro tipos de agentes: colaborativos, colaborativo con aprendizaje, interface y smart.
- Roles: información o Internet.
- Filosofías hibridas: si combina dos o mas enfoques en un agente concreto.
- Atributos secundarios: versatibilidad, benevolencia, veracidad, integridad, continuidad temporal, cualidades emocionales, etc.

Después de desarrollar esta tipología, Nwana describe las investigaciones en curso en siete categorías:

1. Collaborative agentes
2. Interface agentes
3. Mobile agentes
4. Information/Internet agentes
5. Reactive agentes

6. Hybrid agentes
7. Smart agentes

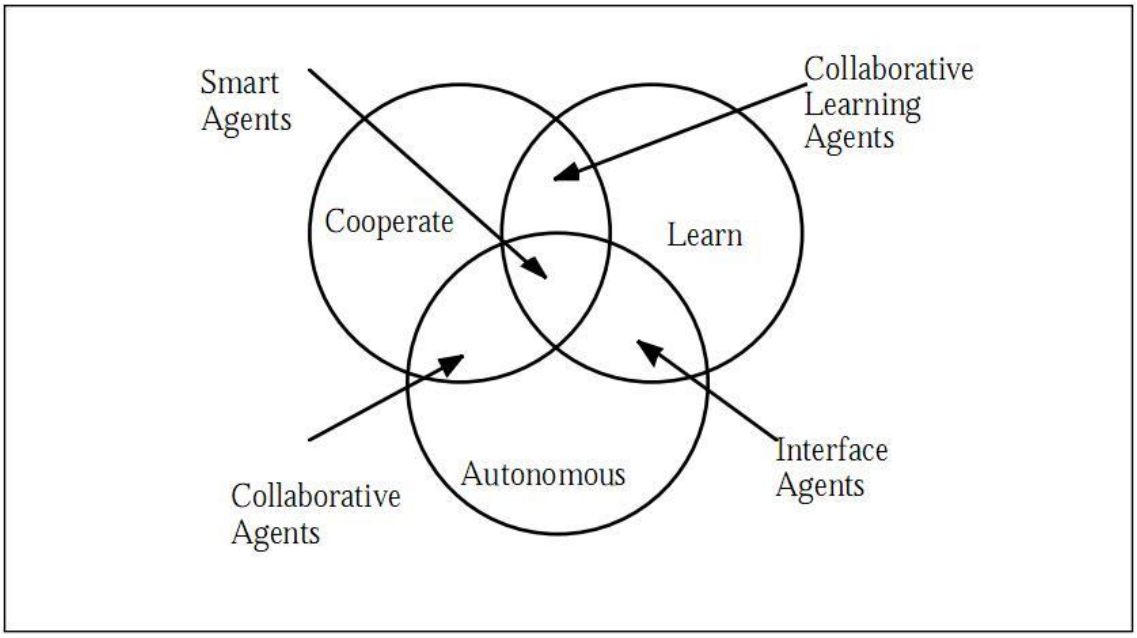


Figura 38: Tipos de agentes por Nwana

Por su parte, y después de las definiciones dadas por Nwana, Franklin y Graesser, en 1996, dan la suya propia:

"un agente autónomo es un sistema situado dentro de una parte del entorno que sensoriza, actuando sobre el todo el tiempo, buscando en todo momento cumplir con sus objetivos."

Al observar que, después de esta definición, incluso un termómetro podría ser considerado un agente, discuten una nueva clasificación, sugiriendo que los agentes pueden ser categorizados por estructuras de control, entornos (bases de datos, sistemas de ficheros, Internet), por el leguaje en el que han sido escritos, y sus aplicaciones. Por eso, finalmente definen la taxonomía que podemos ver en la siguiente imagen.

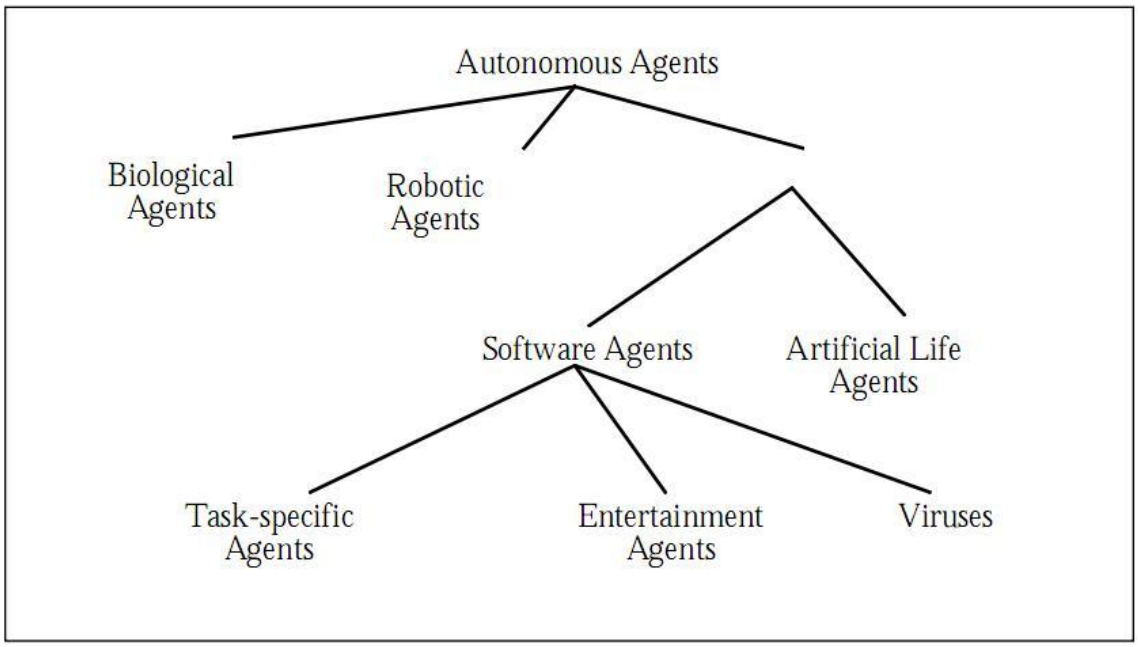


Figura 39: Tipos de agentes por Franklin y Graesser

El tiempo y la experiencia acabaran determinando cual es el largo significado de la palabra agente. Como muchos otros términos en uso común como “desktop”y “mouse”, la palabra agente terminará denotando un artefacto software en concreto.

3.4. ¿Por qué agentes?

El trabajo original en agentes instigado por investigadores trata de estudiar modelos computacionales de inteligencia distribuida. En el diseño de sistemas distribuidos los agentes proporcionan aspectos sociales, lenguajes y protocolos de comunicación de agentes y distribución de datos, control, conocimiento y recursos.

En cuanto al análisis del sistema, un sistema multi-agente tiene un mayor grado de abstracción que un sistema programado mediante la utilización de objetos. Al utilizar la tecnología agente, nuestra aplicación tiene una mayor autonomía y capacidad de decisión, componentes heterogéneos que mantienen relaciones entre ellos y con escalas de tiempo diferentes.

Además, facilitan la evolución del sistema, ya que son capaces de adaptarse al entorno y las modificaciones que éste va realizando. Proporcionan escalabilidad, pudiendo añadir más agentes si es necesario llevar a cabo acciones con una mayor carga de trabajo. Podemos también añadir funcionalidad en tiempo de ejecución, con un desarrollo incremental, con capacidad de aceptar nuevos elementos.

Los principales motivos que nos han llevado a seleccionar a los agentes son la autonomía de decisión, punto muy importante en nuestro caso en concreto, en el que queremos distribuir una serie de robots para que realicen movimientos de forma colaborativa. También son ventajas el flujo de control propio, la encapsulación de la activación de un comportamiento, la autonomía y la organización mediante las relaciones sociales entre los agentes.

Sin embargo, los agentes puede que no siempre sean la solución ideal para un determinado caso de estudio. La tecnología agente, debido a su autonomía y flexibilidad, puede acarrear una ausencia de control/visión global del sistema, lo que puede no ser beneficioso en determinados escenarios.

3.5. Comunicación entre agentes

3.5.1. Introducción

Comunicar, según la Real Academia Española, consiste en "*hacer a otro partícipe de lo que uno tiene*". La comunicación es uno de los actos más importantes en un sistema multiagente, ya que es la llave para obtener todo el potencial de dicho paradigma. Los sistemas multiagente, al estar orientados a la resolución distribuida de problemas, necesitan de la capacidad de comunicación mediante la cual establecer estrategias de cooperación.

Los modelos de comunicación se dividen en dos grandes grupos.

- Arquitectura de pizarra: La pizarra es una zona de trabajo común donde se encuentra la información a compartir. Esta pizarra puede ser consultada por todos los agentes, del mismo modo que todos pueden dejar información en ella. Pueden existir agentes con tareas de control específicas sobre la pizarra, así como varias pizarras. De este modo, no hay comunicación directa entre los agentes, teniendo toda la información centralizada en cada una de las pizarras.

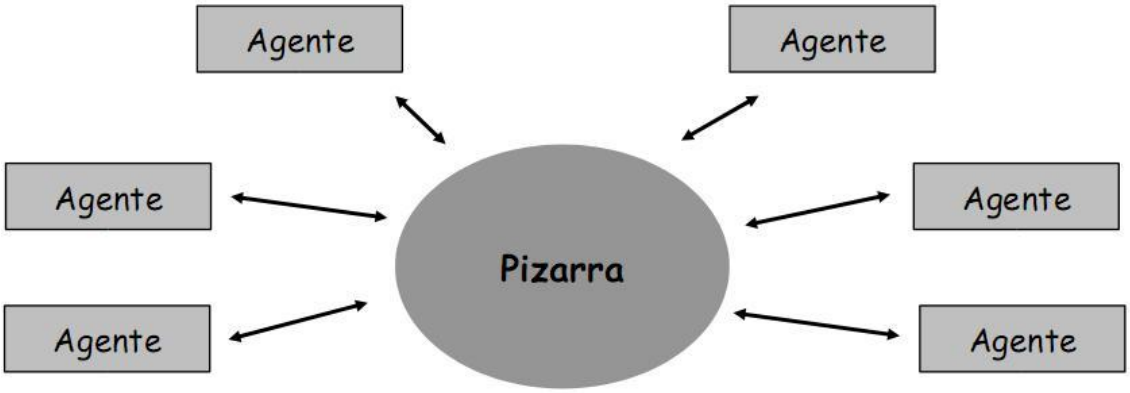


Figura 40: Comunicación utilizando una pizarra

- Paso de mensajes: La comunicación se realiza del mismo modo que se realiza entre dos seres humanos, mediante el establecimiento e intercambio directo de mensajes entre dos agentes (emisor y receptor). Como ventaja encontramos que es más flexible que la anterior, además de que no es necesario tener toda la información centralizada.

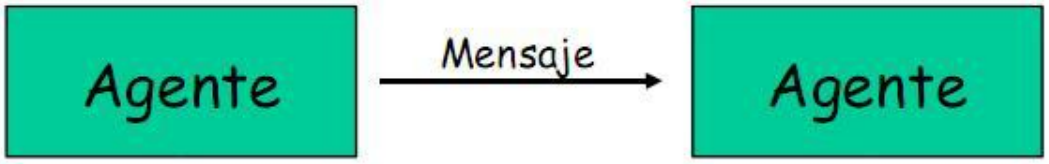


Figura 41: Comunicación utilizando mensajes

3.5.2. Organizaciones de estandarización

Las organizaciones de estandarización se encargan de la redacción y aprobación de las normas que rigen cual debe ser el formato de la información. En concreto, podemos hablar de tres organizaciones de estandarización para el caso de los agentes software.

3.5.2.1. OMG

Asociación de empresas e instituciones formada a finales de los 80. Su objetivo era la reutilización, portabilidad e interoperabilidad de sistemas distribuidos de componentes software orientados a objetos. Sus resultados fueron UML (Unified

Modeling Languaje), CORBA (Common Object Requies Broker Architecture) y MASIF (Mobile Agent System Interoperability Facility).

3.5.2.2. KSE

La iniciativa fue tomada por varias empresas a las cuales interesaba el software agente: ARPA (Advanced Research Projects Agency), ASOFR (Air Force Office of Scientific Research), NRI (Corporation for National Research Initiative) y NSF (National Science Foundation).

Su objetivo principal fue facilitar la compartición y reutilización de bases y sistemas basados en conocimiento. Además, para conseguir una interacción eficaz de agentes software eran necesarias tres componentes fundamentales:

1. Un leguaje común
2. Una comprensión común del conocimiento intercambiado
3. Una habilidad para intercambiar todo lo relativo al lenguaje y la comunicación.

De sus trabajos surgen varios lenguajes como resultado, los cuales son:

- KIF (Knowledge Interchange Format): lenguaje para el intercambio de conocimiento.
- Ontolingua: Lenguaje para la definición de ontologías
- KQML (Knowledge Query and Manipulation Language): Lenguaje para la comunicación entre agentes e interoperabilidad entre agentes en un entorno distribuido.

3.5.2.3. FIPA

Organización internacional dedicada a la promoción de la industria de los agentes inteligentes mediante el desarrollo de especificaciones que soporten la interoperabilidad entre agentes y aplicaciones basadas en agentes.

Su especificación cubre todos los aspectos de un entorno de agentes: aplicaciones, arquitectura, comunicación, protocolos de interacción, actos comunicativos, lenguajes de contenidos, gestión de agentes y transporte de mensajes.

Hasta la fecha ha publicado 3 conjuntos de especificaciones: FIPA 97, FIPA 98 y FIPA 2000.

3.6. FIPA (Foundation for Intelligent Physical Agents)

Una de las formas de alcanzar la deseada interoperabilidad entre plataformas es la definición y uso de estándares aprobados por un comité internacional. En el área de los agentes, FIPA es la organización encargada de producir especificaciones para la interacción de agentes y sistemas de agentes heterogéneos.

En la siguiente sección describiremos brevemente el modelo de referencia propuesto por FIPA para las plataformas de agentes, así como el formato e intercambio de los mensajes.

3.6.1. Modelo de referencia de FIPA

Uno de los objetivos principales de FIPA es especificar una arquitectura de agentes que permita la utilización de un amplio número de mecanismos y servicios, como varios protocolos de transporte o servicios de directorio. Dada la posibilidad elegir entre varios de estos mecanismos y protocolos, los sistemas de agentes construidos de acuerdo a esta arquitectura deberían de ser capaces de interoperar a través de pasarelas de transporte, ya que la especificación para una arquitectura abstracta de FIPA permite la creación de diferentes implementaciones y ofrece también las transformaciones tanto para transporte de mensajes, esquemas de codificación de mensajes y localización de agentes y servicios a través de directorios de servicios. Sin embargo FIPA no cubre otros aspectos que surgen o dependen de la implementación, bien por que están fuera de ámbito o porque no están incluidos en las especificaciones.

Una plataforma de agentes FIPA se define como el software que implementa un conjunto de especificaciones FIPA. Para que se considere que sigue las normas de FIPA, una plataforma debe implementar al menos la especificación sobre la gestión de agentes y las relativas al lenguaje de comunicación de agentes (ACL). La primera se ocupa del control y gestión de agentes dentro de y a través de plataformas de agentes. Las relativas al ACL se encargan del formato de los mensajes, los protocolos de interacción y de intercambio de mensajes entre agentes, la descripción de actos comunicativos que definen la semántica de los mensajes intercambiados y los diferentes lenguajes para expresar el contenido de un mensaje (lenguaje de contenido).

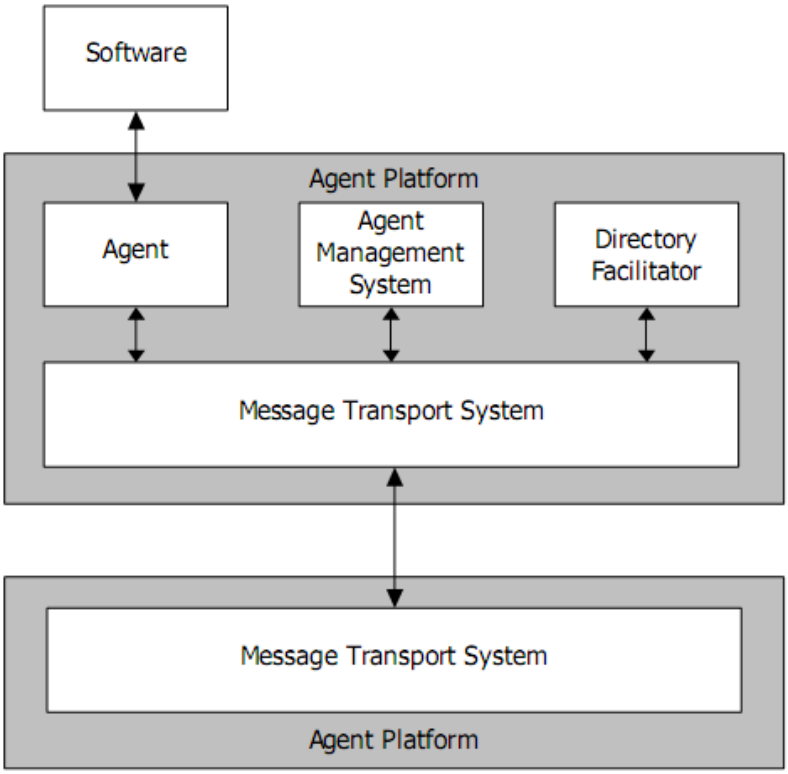


Figura 42: Intercambio de mensajes en FIPA

El objetivo de la especificación de gestión de agentes es ofrecer un marco de trabajo estándar donde los agentes FIPA existan y operen, estableciendo un modelo de referencia lógico para la creación, registro, localización, comunicación, migración y baja de agentes. Este modelo de referencia se compone de un conjunto de entidades que ofrecen diferentes servicios. Estas entidades son:

- Agente: un agente software es un proceso computacional que implementa la funcionalidad autónoma y comunicativa de la aplicación, ofreciendo al menos un servicio. Los agentes se comunican a través de un ACL. Un agente debe tener al menos un propietario y un identificador de agente (AID) que lo identifica de forma unívoca. Además, un agente puede disponer de varias direcciones de transporte a través de las cuales puede ser contactado. Dependiendo de la implementación de la plataforma un agente puede tratarse de un componente Java o un objeto CORBA.

- DF (Directory Facilitator): un facilitador de directorio que ofrece un servicio de páginas amarillas para localizar agentes. Los agentes deben registrarse previamente en este servicio para ser localizados como proveedores de un servicio, y recurren a él en busca de agentes que ofrezcan un determinado servicio.

- AMS (Agent Management System): un sistema de gestión de agentes que controla el acceso y uso de la plataforma de agentes. Sólo puede existir un agente AMS por plataforma, y se encarga de generar AIDs válidos además de ofrecer un servicio de búsqueda de agentes por nombre.

- MTS (Message Transport Service): un servicio de transporte de mensajes ofrece un servicio de comunicación entre agentes, encargándose del transporte de mensajes entre agentes. FIPA especifica para este componente un modelo de referencia y varias especificaciones sobre distintos protocolos y mecanismos de transporte que favorece la interoperabilidad entre plataformas.

- AP (Agent Platform): la plataforma de agentes ofrece una infraestructura física sobre la que desplegar los agentes y está compuesta de un software de soporte, los componentes para la gestión de agentes (DF, AMS, y MTS) y los agentes.

Además, FIPA también define y especifica una serie de elementos y conceptos importantes necesarios para conseguir la interoperabilidad. Estos conceptos son: un servicio (para definir un conjunto de acciones incluidas en una ontología), un ACL (lenguaje para construir mensajes y actos comunicativos) y un AID (identifica un agente incluyendo denominaciones, roles y direcciones).

3.6.2. Comunicación FIPA – FIPA ACL

La comunicación con FIPA ACL es un servicio de transporte utilizado por los agentes que implementan la especificación FIPA, con el fin de intercambiar información entre ellos. Está basado en la teoría de los actos del habla y con él somos capaces de enviar un mensaje, codificarlo y transmitirlo como una secuencia de bytes. Un mensaje en FIPA ACL representa la intención de realizar alguna acción (acto comunicativo). Este servicio tiene las siguientes características:

1. Servicio de confianza, donde los mensajes que están bien creados/formados no tienen inconveniente para llegar a su destino.

2. Fiable, ya que los mensajes se reciben tal y como se envían, sin modificaciones.

3. Ordenado. El orden de llegada de los mensajes es el mismo que el orden de salida.

El elemento básico para la comunicación en la especificación FIPA ACL es el mensaje. Estos mensajes tienen la siguiente estructura:

1. Identificador del acto comunicativo (obligatorio).

accept-proposal	not-understood
agree	propose
cancel	query-if
cfp	query-ref
confirm	refuse
disconfirm	reject-proposal
failure	request
inform	request-when
inform-if	request-whenever
inform-ref	subscribe

Figura 43: Actos comunicativos en FIPA

2. Pares parámetro-valor sin un orden predefinido (opcionales). Los posibles parámetros a definir en un mensaje ACL son:

    a. Content: Contenido del mensaje.

    b. Sender: Identidad del emisor del mensaje.

    c. Receiver: Identidad del receptor del mensaje, puede ser un agente o una lista de agentes.

    d. Language: El nombre del lenguaje de representación empleado en el atributo content.

    e. Ontology: El nombre de la ontología utilizada en el atributo content.

    f. Reply-With: Etiqueta para la respuesta (si es que el emisor la espera).

    g. In-Reply-To: Etiqueta esperada en la respuesta.

    h. Protocol: identificador del protocolo de interacción que se está utilizando.

    i. Conversation-Id: identificador de una secuencia de actos comunicativos que forman parte de una misma conversación.

j. Reply-To: Agente al que han de ser enviadas las respuestas (si no es el emisor).

k. Reply-By: Indicación del tiempo en el que se quiere que se responda al mensaje.

Además, el modelo de comunicación está basado en la asunción de que dos agente que quieren conversar han de compartir una ontología común que describa el universo del discurso, de forma que podamos asegurar que estos agentes atribuyan el mismo significado a los símbolos utilizados en los mensajes. Una ontología es una descripción formal de los conceptos y relaciones que pueden existir en una determinada comunidad de agentes.

3.7. JADE (Java Agent DEvelopment Framework)

Java Agent DEvelopment Framework, o JADE, es una plataforma software para el desarrollo de agentes, implementada en JAVA, que ha estado en desarrollo al menos desde 2001. La plataforma JADE soporta la coordinación de múltiples agentes FIPA y proporciona una implementación estándar del lenguaje de comunicación FIPA-ACL. Jade fue desarrollado originalmente por Telecom Italia y se distribuye como software libre. Podemos descargar su última versión desde su sitio (*http://jade.tilab.com*)

3.7.1. La plataforma de agentes JADE

La plataforma JADE es 100% compatible con el JDK 1.4 o superiores, e incluye la funcionalidad necesaria para la creación básica de agentes, la programación de comportamiento de los agentes en base a Behaviours, implementación de la especificación FIPA ACL para el envío y recepción de mensajes, clases útiles para programación de protocolos FIPA, manejo de información usando ontologías, etc.

Además, proporciona la plataforma FIPA (AMS, Facilitador de directorio y MTS), puede ejecutarse en una o varias JVM (donde cada JVM es vista como un entorno en donde los agentes pueden ejecutarse concurrentemente e intercambiar mensajes) y se organiza en contenedores (un contenedor principal que contiene el AMS, DF y RMA y más contenedores no principales conectados al principal).

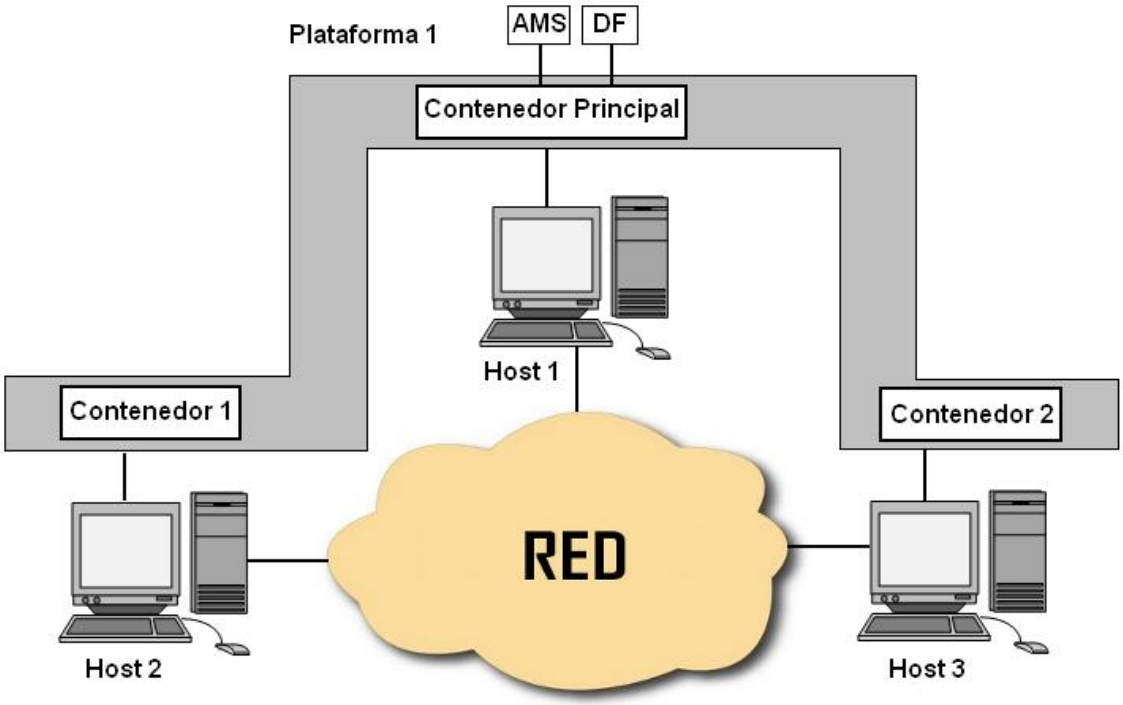


Figura 44: Esquema de plataforma JADE

3.7.2. Comportamientos

El trabajo que tiene que realizar cada uno de los agentes se encuentra dentro de sus comportamientos (behaviours). Un comportamiento representa una tarea que un agente puede llevar a cabo, implementada como un objeto que extiende de la clase *jade.core.behaviours.Behaviour*. A fin de que un agente ejecute una tarea implementada en un comportamiento es suficiente con que el agente añada su comportamiento a la pila de comportamientos con la instrucción *addBehaviour()* de la clase Agent. Además, un comportamiento puede ser añadido en cualquier momento: cuando el agente se crea o desde dentro de otros comportamientos.

Cada vez que queremos definir un comportamiento debemos extender de la clase Behaviour y darle código al metodo *action()*, el cual define las operaciones y acciones que el comportamiento debe realizar mientras se ejecuta. También podemos dar código al método *done()*, que especifica cuando un comportamiento se ha ejecutado por completo y debe ser eliminado de la pila de comportamientos de un determinado agente.

3.7.2.1. Planificación y ejecución de comportamientos

Un agente puede ejecutar varios comportamientos concurrentemente. Sin embargo, es importante avisar de que la planificación de comportamientos en un

agente no es preventiva, sino cooperativa. Esto significa que cuando un comportamiento es planificado para ejecutarse, su método *action()* es ejecutado hasta que termina.

Por lo tanto, es el programador el que decide cuando un agente cambia de la ejecución de un comportamiento a la ejecución de otro. Este enfoque tiene varias ventajas:

- Permite tener un único hilo de ejecución por cada agente (importante especialmente en entornos con recursos limitados).
- Proporciona mejores rendimientos, ya que la ejecución de un nuevo comportamiento es mucho más rápida que la planificación de hilos en Java.
- Elimina todas las cuestiones de sincronización entre comportamientos concurrentes para el acceso a recursos compartidos, ya que todos los comportamientos son ejecutados por el mismo hilo de ejecución.

El hilo de ejecución que sigue cada uno de los hilos de ejecución de los agentes se muestra en la siguiente figura:

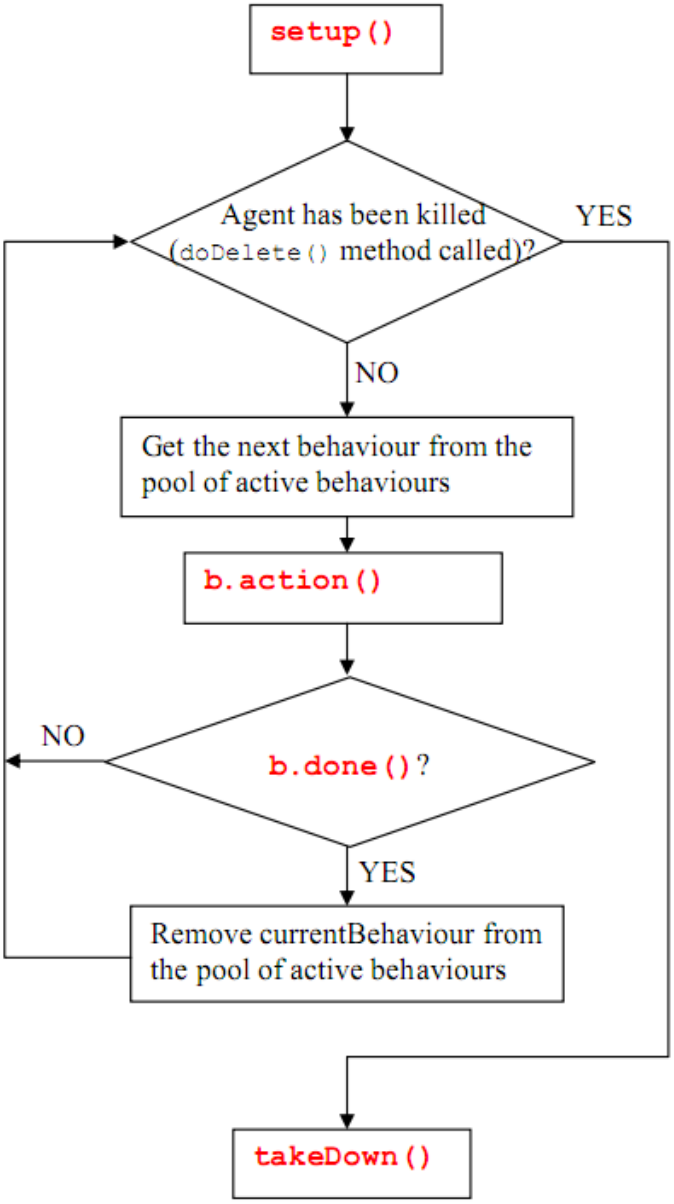


Figura 45: Ejecución de comportamientos en FIPA

3.7.2.2. Tipos de comportamientos

Jade proporciona una clase abstracta (Behaviour) que se utiliza como base para modelar los comportamientos y tareas que debe realizar cada uno de estos comportamientos. Dependiendo de las necesidades, podemos crear distintos tipos de comportamientos:

1. OneShotBehaviour: Esta clase abstracta modela comportamientos atómicos que deben ser ejecutados únicamente una vez.

2. CyclicBehaviour: Esta clase abstracta modela un comportamiento atómico que debe ser ejecutado constantemente. Su método *done()* siempre devolverá false.

3. CompositeBehaviour: Son comportamientos que no definen funcionalidad por si mismos, pero que realizan sus operaciones como composición de otros comportamientos hijos.

4. SequentialBehaviour: Es un *CompositeBehaviour* que ejecuta sus sub-comportamientos secuencialmente y termina cuando todos los sub-comportamientos finalizan. Se puede utilizar este comportamiento cuando una tarea puede ser expresada como una secuencia de pasos atómicos.

5. ParallelBehaviour: Es un *CompositeBehaviour* que ejecuta sus sub-comportamientos concurrentemente y termina cuando una condición particular en sus sub-behaviours es alcanzada (cuando todos los sub-comportamientos terminan, cuando un número de sub-comportamientos finaliza o cuando se han dado un número determinado de iteraciones).

6. WakerBehaviour: Esta clase abstracta implementa un comportamiento que se ejecuta una única vez justo después de transcurrir un *timeout*.

7. TickerBehaviour: Esta clase abstracta implementa una tarea cíclica que se ejecuta periódicamente.

En función de cuales sean nuestras necesidades u objetivos a alcanzar podemos utilizar el comportamiento que mejor se ajuste a nuestras necesidades.

3.7.3. Directorio y Servicios

El servicio de “yellow pages” o servicio de directorio permite a los agentes publicar uno o más servicios que ellos proveen, para que otros agentes puedan buscar por un determinado servicio.

El servicio de directorio en JADE (de acuerdo a la especificación FIPA) es previsto por un agente llamado DF (directory Facilitator). Cada plataforma FIAP dispone de un agente DF por defecto, aunque se pueden activar más agentes DF con el fin de facilitar un catálogo de páginas amarillas distribuido por toda la red.

Aunque es posible interactuar con el agente DF a través del intercambio de mensajes ACL como con cualquier otro agente, JADE simplifica estas interacciones mediante el uso de la clase *jade.domain.DFService,* con la que es posible publicar, modificar y buscar distintos servicios.

3.7.3.1. Publicación de servicios

Cuando un determinado agente desea publicar uno o más servicios debe de proporcionar al DF información acerca de su AID, la lista de lenguajes y ontologías que los otros agentes deben conocer para poder interactuar y la lista de servicios a publicar. Para cada servicio publicado debemos de proporcionar una descripción donde se incluye el tipo de servicio, el nombre del servicio y los lenguajes y ontologías necesarios para poder explotar el determinado servicio. Las clases *DFAgentDescription* y *ServiceDescription*, incluidos en el *paquete jade.domain.FOPAAgentManagement.*

El código necesario para publicar un servicio debe establecer una descripción (como instancia de la DFAgentDescription) y realizar la llamada al método *register()* de la clase DFService. A continuación podemos ver un ejemplo, en el que el agente registra un servicio de tipo "AyudanteEsperando":

```
protected void setup() {
  ...
  //Descripción del agente
  DFAgentDescription dfd = new DFAgentDescription();
  dfd.setName(getAID());
  //Descripción del servicio
  ServiceDescription sd = new ServiceDescription();
  sd.setType("AyudanteEsperando");
  sd.setName("JADE-AyudanteEsperando");
  dfd.addServices(sd);
  //Acción de registro del servicio
  try {
   DFService.register(this, dfd);
  }
  catch (FIPAException fe) {
   fe.printStackTrace();
```

```
    }
    ...
}
```

3.7.3.2. Búsqueda de servicios

Un agente que desea buscar por un determinado servicio, debe proporcionar al DF una plantilla de descripción. Como resultado de la búsqueda obtenemos una lista con todas las descripciones que concuerdan con lo buscado. Una descripción hace matching con una plantilla de descripción si todos los campos especificados en la plantilla están presentes en al descripción del servicio con los mismo valores.

El método estático *search()* de la clase *DFService* puede ser usado para encontrar todos los agentes que buscan un determinado servicio. A continuación vemos un ejemplo de su uso:

```
DFAgentDescription template = new DFAgentDescription();
ServiceDescription sd = new ServiceDescription();
sd.setType("AyudanteEsperando");
template.addServices(sd);
try {
  DFAgentDescription[] result = DFService.search(myAgent, template);
  /* En result se almacenan todas las descripciones de agente con el servicio AyudanteEsperando */
}
catch (FIPAException fe) {
  fe.printStackTrace();
}
```

Ahora, partiendo de la descripción de los agentes que proporcionan un determinado servicio, podemos acceder a su nombre y enviar un mensaje para activar un determinado comportamiento. Sin embargo, cada vez que necesitemos un agente con el servicio *AyudanteEsperando* deberemos buscar de nuevo, ya que posiblemente

alguno de los agentes del vector ha cambiado su estado y ya no tiene disponible dicho servicio.

3.7.4. Comunicación entre agentes

Una de las características mas importantes que los agentes JADE poseen es la habilidad para comunicarse. El paradigma de comunicación adoptado es el paso de mensajes asíncrono. Cada agente tiene una pequeña pila de entrada de mensajes donde el runtime de JADE almacena los mensajes enviados por otros agentes. Así mismo, cada vez que un mensaje es añadido a la cola de mensajes, el agente que lo recibe es notificado. Sin embargo, el instante en el que el robot coge el mensaje de la cola y lo procesa depende únicamente del programador.

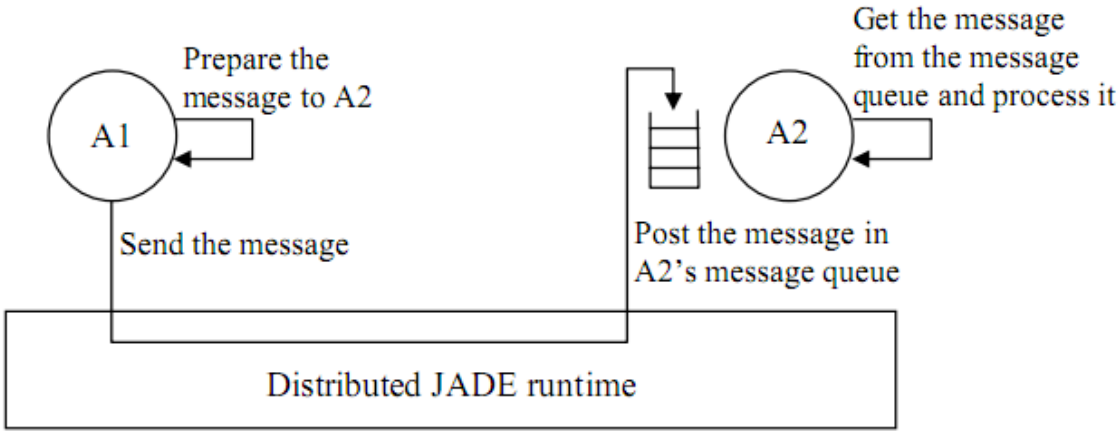


Figura 46: Cola de mensajes en JADE

3.7.4.1. Envío de mensajes

Enviar un mensaje a otro agente es tan simple como rellenar las valores de un objeto de tipo *ACLMessage* y llamar al metodo *send()* de la clase *Agent.* A continuación vamos a enviar un mensaje de información al agente Ayudante1 avisándole de que necesitamos su ayuda:

```
ACLMessage msg = new ACLMessage(ACLMessage.INFORM);

msg.addReceiver(new AID("Ayudante1", AID.ISLOCALNAME));

msg.setLanguage("Castellano");

msg.setContent("Necesitamos ayuda");
```

```
send(msg);
```

3.7.4.2. Recepción de mensajes

Como hemos dicho anteriormente, el runtime de JADE automáticamente almacena los mensajes recibidos en la cola de llegada tan pronto como sean recibidos. Un agente puede obtener un mensaje de su propia cola de mensajes utilizando el método *receive().* Este método devuelve el primer mensaje en la cola de mensajes (y lo elimina) o devuelve null en caso de que la cola de mensajes esté completamente vacía. Del mismo modo, disponemos de una función bloqueante para la recepción de mensajes, que detendrá el comportamiento que la invoque hasta que un mensaje con las características necesarias sea recibido. Para ello, debemos invocar a la función *blockingReceive().* A continuación mostramos un ejemplo de aplicación de la primera de estas funciones:

```
ACLMessage msg = receive();

if (msg != null) {

  // Hemos recibido un mensaje y pasamos a procesarlo.

}
```

El caso de la espera bloqueante para un mensaje sería muy similar al caso anterior, con la única diferencia de que no es necesario comprobar que el mensaje no es null, ya que siempre se ejecuta dicha instrucción tenemos un mensaje listo para procesar.

4.- DESARROLLO PRÁCTICO

En este apartado, y después de explicar todos los conceptos teóricos necesarios, nos centraremos en la descripción práctica que se ha llevado a cabo en el proyecto.

4.1. Configuración inicial

4.1.1. Instalación del JAVA JDK (Java Development Kit).

Para poder trabajar con el lenguaje de programación Java es necesario instalar lo que se conoce como JDK o Java Development Kit, el cual provee herramientas de desarrollo para la creación de programas en Java. Puede instalarse en un computador local o una unidad de red una vez ha sido descargado del siguiente enlace:

http://www.oracle.com/technetwork/java/javase/downloads/jdk-6u26-download-400750.html

Una vez descargado, solo tenemos que lanzar el ejecutable y seguir los pasos del asistente para instalar el JDK en nuestro ordenador.

4.1.2. Instalación del driver USB de Lego

Para que nuestro ordenador pueda comunicarse con todos los robots Lego NXT, independientemente de la versión de firmware que tengan instalada, es necesario instalar los drivers del conector Bluetooth USB de lego. Podemos descargarlos desde el siguiente enlace:

http://mindstorms.lego.com/en-us/support/files/default.aspx

Una vez instalado el driver, será necesario agregar todos los robots como dispositivos conocidos por el interfaz Bluetooth, ya que en caso contrario no podremos realizar la conexión entre ellos. Para ello, vamos a “Panel de control”->”Dispositivos Bluetooth”->”Agregar…”, con lo que podemos agregar cada uno de nuestros robots introduciendo el PIN cuando nos sea solicitado.

4.1.3. Instalación de LeJOS NXJ.

Dado que LeJOS NXJ cuenta con su propio firmware, es necesario reemplazar el original que tienen los robots. Para ello se debe descargar el software desde: http://LeJOS.sourceforge.net/nxj-downloads.php e instalarlo en el ordenador.

El software incluye una biblioteca de clases de Java (Classes.jar) que implementan la LEJOS NXJ Application Programming Interface (API), herramientas de PC para actualizar el firmware del robot, cargar los programas, depurarlos, un API de PC para escribir programas de PC que se comunican con los programas de LEJOS NXJ a través de Bluetooth o USB y otras muchas funciones y programas de ejemplo.

Se conecta el robot al ordenador por USB y se utiliza la aplicación nxjflash para actualizar el firmware a la versión de LeJOS NXJ, y con esto los robots quedan preparados.

Para realizar las aplicaciones, pruebas y rutinas, se ha utilizado el entorno de trabajo de ECLIPSE, por su popularidad entre programadores en Java y por la familiaridad personal con la que se contaba anteriormente.

4.1.4. Instalación y configuración de la plataforma JADE

En el presente apartado vamos a explicar la configuración/instalación de la plataforma.

Para poder utilizar el framework de JADE es necesario instalar y configurar el equipo. Para ello, lo primero que debemos hacer es descargar las librerías necesarias desde la página web del proyecto (http://jade.tilab.com). La última versión disponible es la 4.0.1, aunque nosotros, por motivos de estabilidad, hemos utilizado la versión 3.4. Para configurar el equipo, es necesario añadir la siguiente línea a la variable de entorno CLASSPATH:

CLASSPATH=/ruta/jade/lib/jade.jar:/ruta/jade/lib/iiop.jar:/ruta/jade/lib/http.jar

donde ruta indica la ruta donde tenemos las librerías de JADE que acabamos de descargar.

Para comprobar que funciona correctamente, abrimos un terminal y ejecutamos el siguiente código

java jade.Boot –gui

y nos aparece la siguiente ventana:

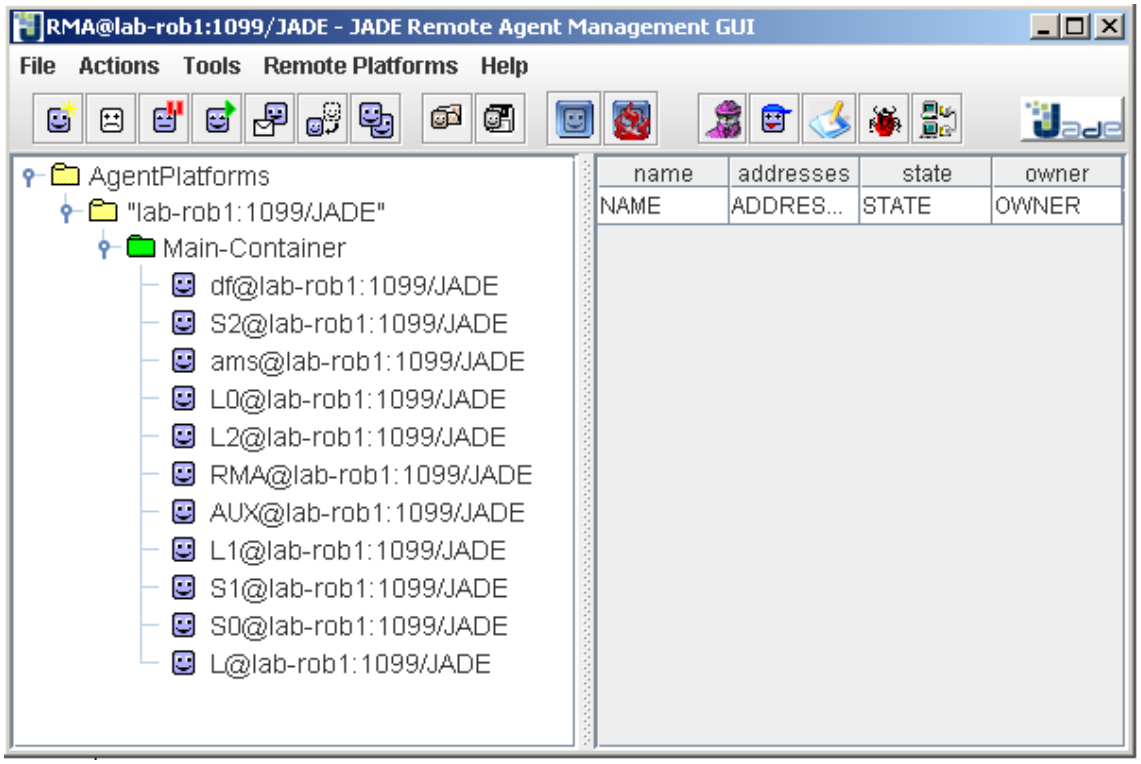


Figura 48: Interfaz de administración JADE

Esta es la interfaz de administración de JADE. En ella podemos realizar distintas acciones, desde consultar los agentes en ejecución, lanzar nuevos agentes, consultar sus estados, hacer debug, enviar mensajes ACL, etc. Algunas de estas funciones las explicaremos en el siguiente apartado.

Además, como podemos observar, al lanzar la interfaz ya aparecen creados los dos agentes que hemos descrito en el apartado anterior. En concreto, estos dos agentes son:

- DF (Directory Facilator): agente que se encarga de proveer el servicio de páginas amarillas, donde un agente puede buscar a otros agentes que proveen el servicio que necesita para conseguir sus objetivos.

- AMS (Agent Management System): provee el servicio de nombres (ya que cada agente debe tener un nombre único en la plataforma) y representa la autoridad en la plataforma, estando dotado de la funcionalidad necesaria para crear/eliminar agentes en su propio contenedor o incluso en contenedores remotos.

- RMA (Remote Monitoring Agent): Aunque no pertenece a la especificación de gestión de agentes, este agente se encarga del manejo y administración de la interfaz de JADE.

4.1.5. Configuración del entorno de desarrollo (Eclipse)

Desde http://www.eclipse.org/downloads/ se descarga la última versión de ECLIPSE IDE para desarrolladores Java y hay que descomprimirla en el disco duro.

4.1.5.1. Compilación y ejecución para NXJ

Aunque LeJOS NXJ proporciona una aplicación para descargar los programas en el NXT, el hecho de compilar con ECLIPSE y luego usar otra aplicación para descargar los datos al NXT resulta algo engorroso. Por tanto, ya que se va a trabajar en ECLIPSE, se configurará para tener la opción de descargar los programas directamente en el ladrillo, sin necesidad de la aplicación de LeJOS. Para esto hay que configurarlo de la siguiente manera:

Con un proyecto abierto, se hace clic en “Run” -> “External Tools” -> “External Tools Configurations…”

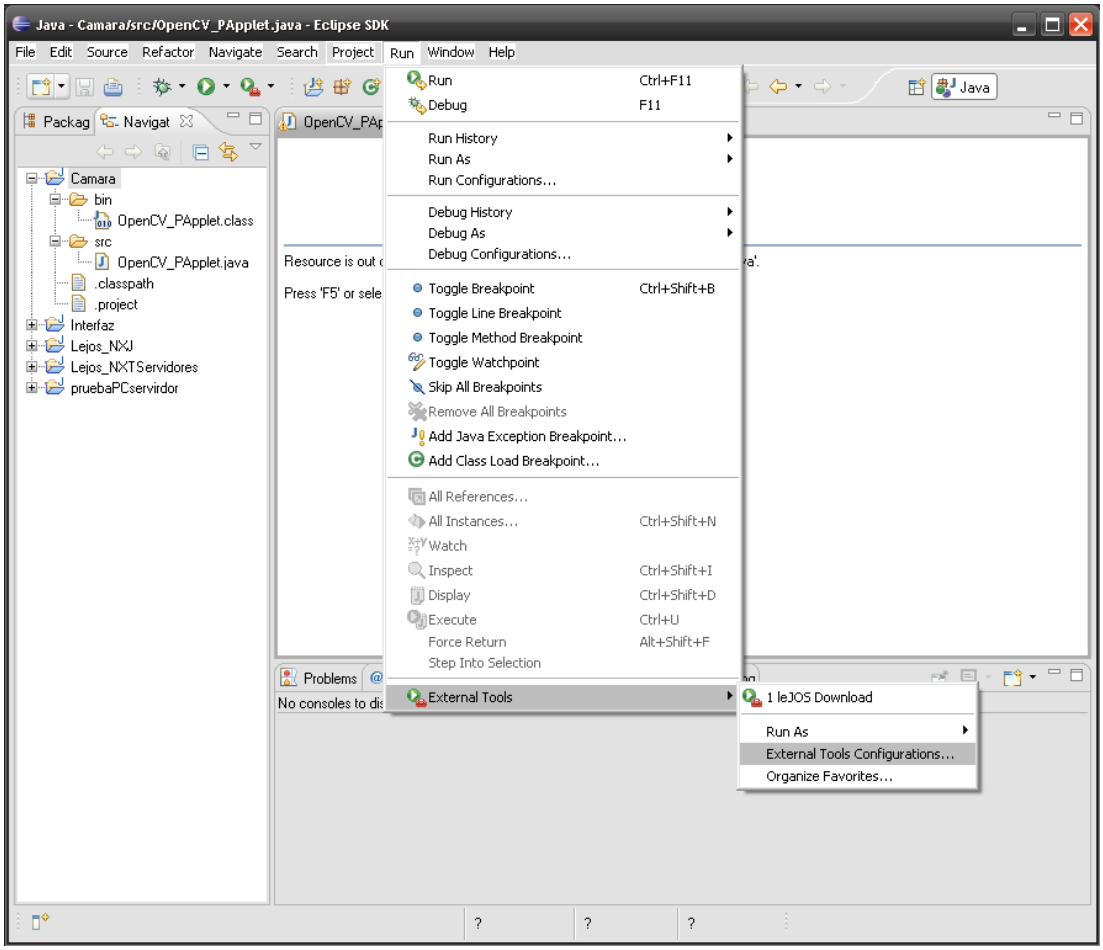


Figura 49: Configuración Eclipse

Se lanza una nueva ventana donde hay que seleccionar “Program” en la parte de la izquierda y el icono de “New launch configuration”. Como nombre se le puede dar “LeJOS Download” por ejemplo y en la pestaña “Main” hay que rellenar el campo “Location” con la ruta donde se encuentra el ejecutable “LeJOSdl.bat” proporcionado por LeJOS. (Deberá estar en %ruta_de_LeJOS_NXT% /bin). Este ejecutable es el que permite descargar directamente al NXT, lo que se está haciendo es vincular el ECLIPSE a esa aplicación para poder hacerlo directamente sin salir del entorno. Siguiendo con la configuración en el campo “Working Directory” hay que introducir “${project_loc}\bin” (sin comillas) y en el campo “Arguments” introducir “${java_type_name}” (sin comillas también). Y aplicar.

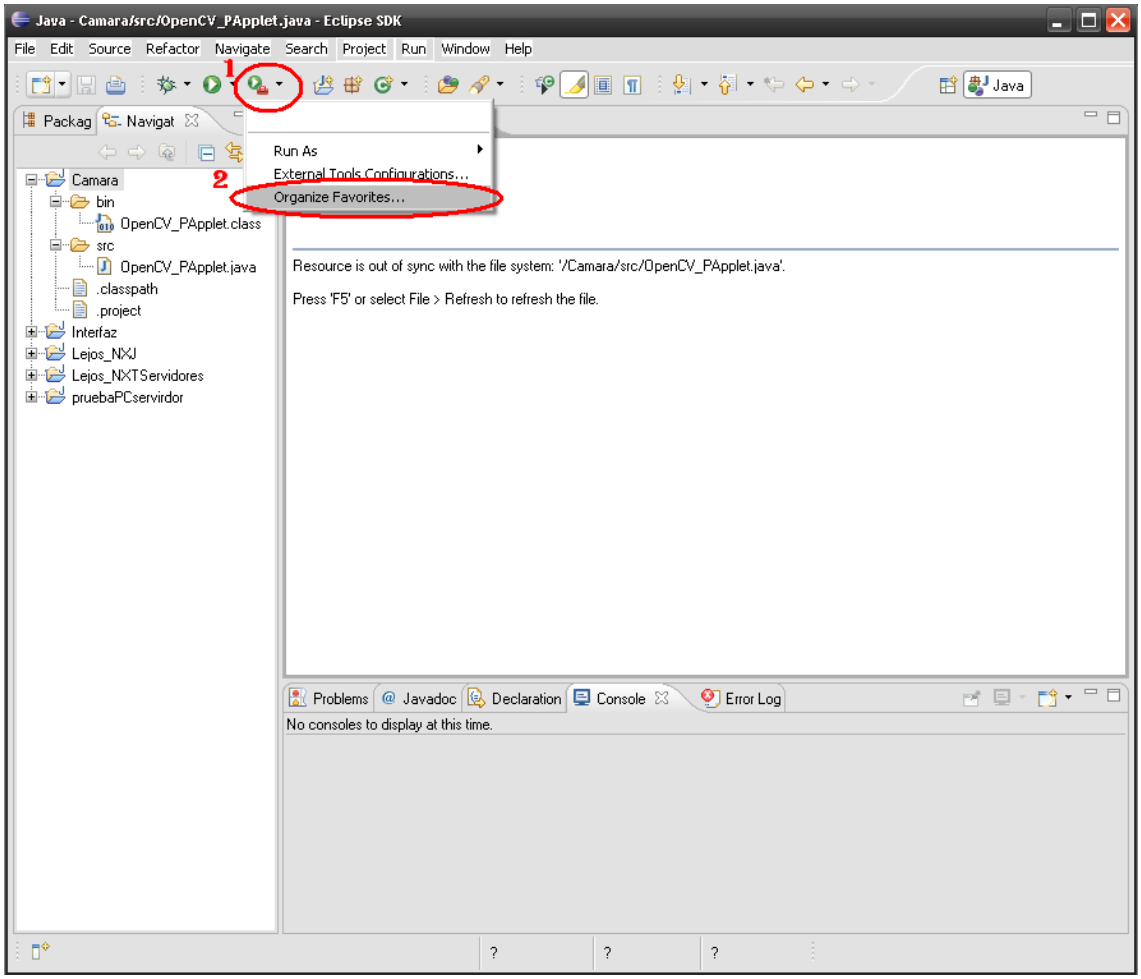


Figura 50: Configuración Eclipse 2

Dar clic en “Add…” en la ventana nueva que aparece y luego seleccionar la opción de “LeJOS Download” (o el nombre que se le haya querido dar a la configuración definida anteriormente). Presionar “Ok” y luego “Ok” de nuevo.

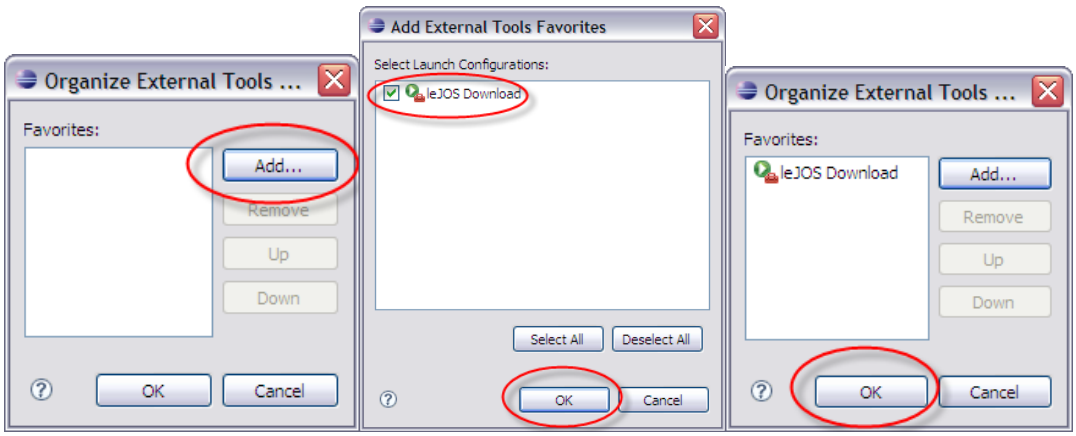


Figura 51: Configuración Eclipse 3

De este modo bastará con hacer clic en el botón “Run” que se ha configurado para indicar a Eclipse que debe compilar y descargar el programa en el NXT mediante USB.

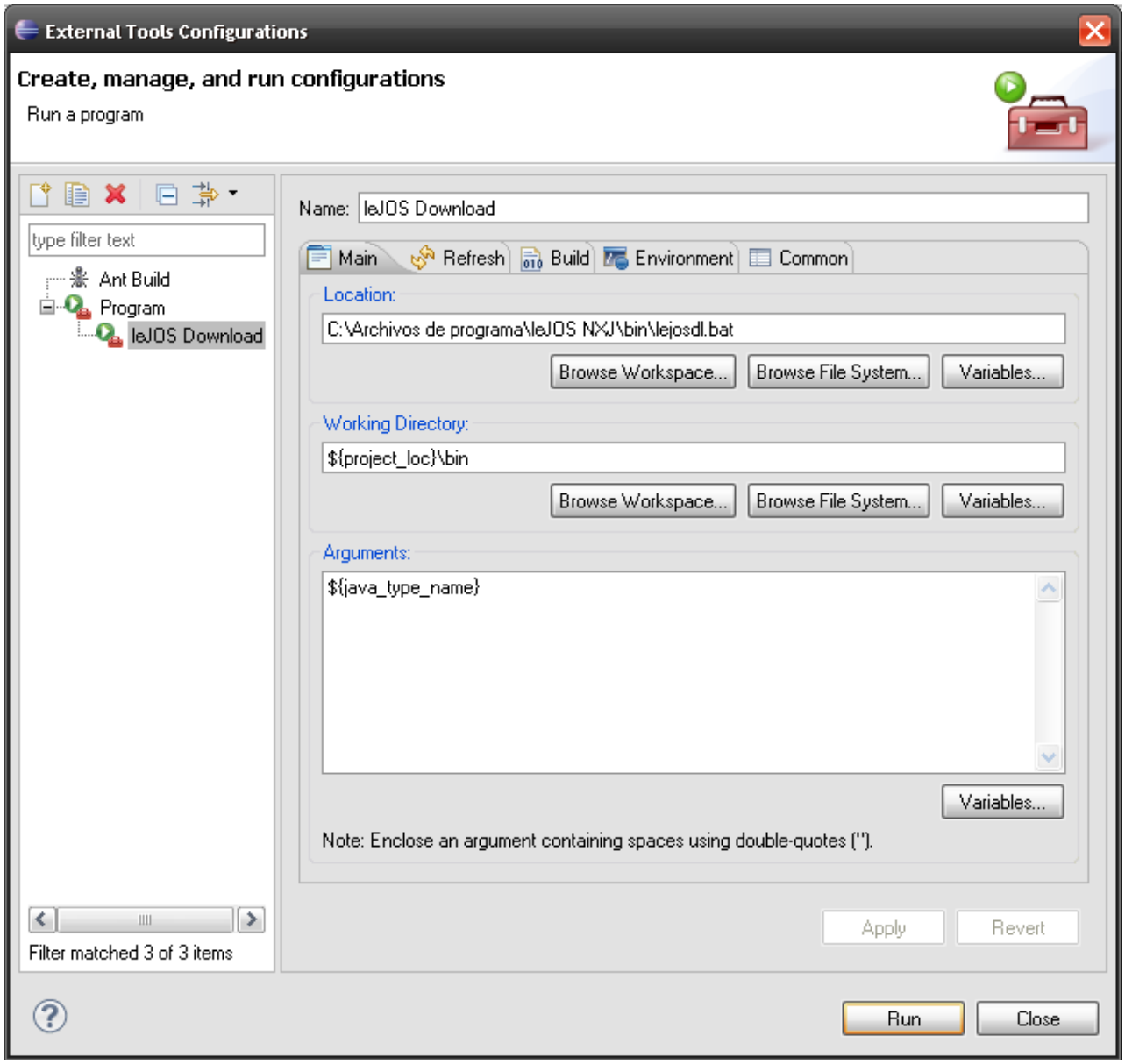


Figura 52: Configuración Eclipse 4

Cuando ya se ha creado la configuración, se puede poner un acceso directo haciendo clic en el icono de "Run"->"Organize Favorites…"

4.1.5.2. Compilación y ejecución de agentes JADE

JADE puede ser ejecutado bajo Eclipse como una aplicación de Java cualquiera. Sin embargo, es necesario llevar a cabo la configuración necesaria, así como incluir las librerías JADE que utilizarán nuestros agentes.

En primer lugar, y antes de configurar la interfaz para la ejecución de JADE, debemos incluir las librerías de JADE en el proyecto. Para ello, vamos a "Project"-

>”Properties”->”Java Build Path”->”Libraries”. En este momento nos aparecen varios botones. Tenemos que localizar el botón “Add External JARs” y seleccionar las librerías de JADE que hemos descargado previamente y almacenado en nuestro disco duro. Una vez están incluidas aceptamos con el botón “OK”.

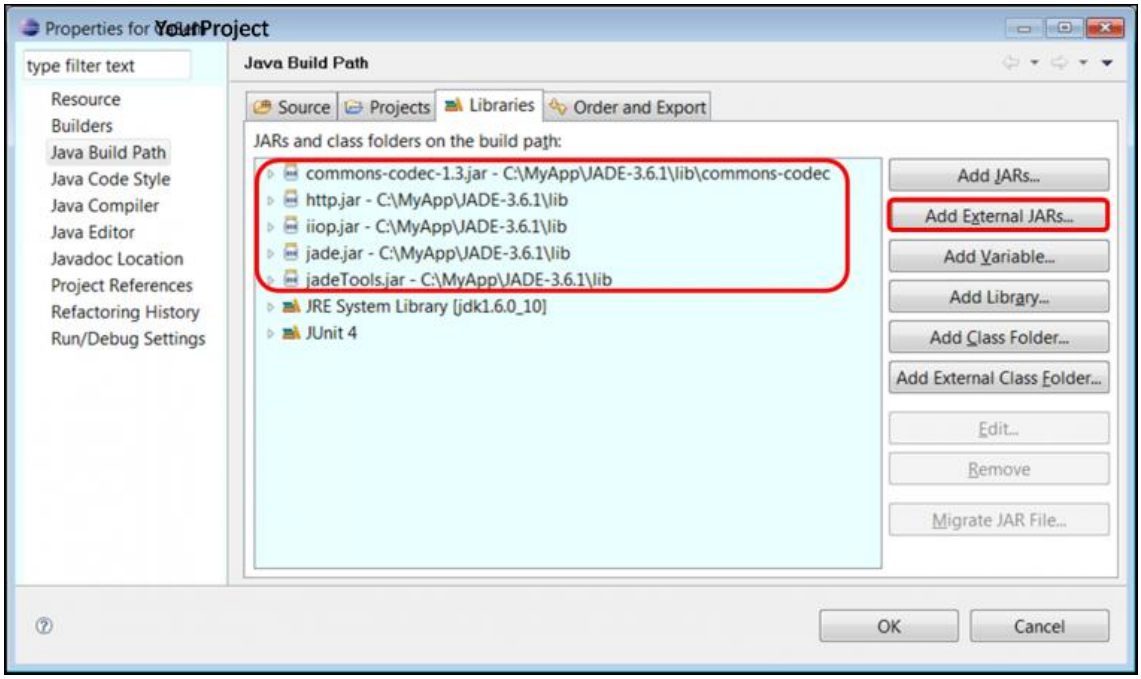


Figura 53: Inclusión de las librerías en Eclipse

Ya tenemos incluidas las librerías en nuestro proyecto, a partir de ahora podemos hacer uso de ellas haciendo llamadas a las funciones que definen.

El siguiente paso es configurar Eclipse para la ejecución de aplicaciones basadas en agentes. Para ello, seguimos los siguientes pasos:

1. Vamos a “Run”->”Run Configurations”->”Java Aplication”

2. En la pestaña “Main” introducimos el nombre de nuestro proyecto, en “Main Class” introducimos jade.Boot y nos aseguramos de tener activada la opción que dice “include libraries when searching for a main class”.

3. En la pestaña “argument”->“Program Arguments” introducimos los argumentos que deben ser usados en la línea de comandos cuando se ejecuta JADE. Introducimos lo siguiente:

   -gui jade.Boot [nombre_agente:paquete_java.nombre_clase]

Donde *nombre_agente* es el nombre que le queremos dar a nuestro agente y que se mostrará en la GUI, *paquete_java* es el paquete donde se encuentra declarado nuestro agente y *nombre_clase* es el nombre de la clase que define al agente.

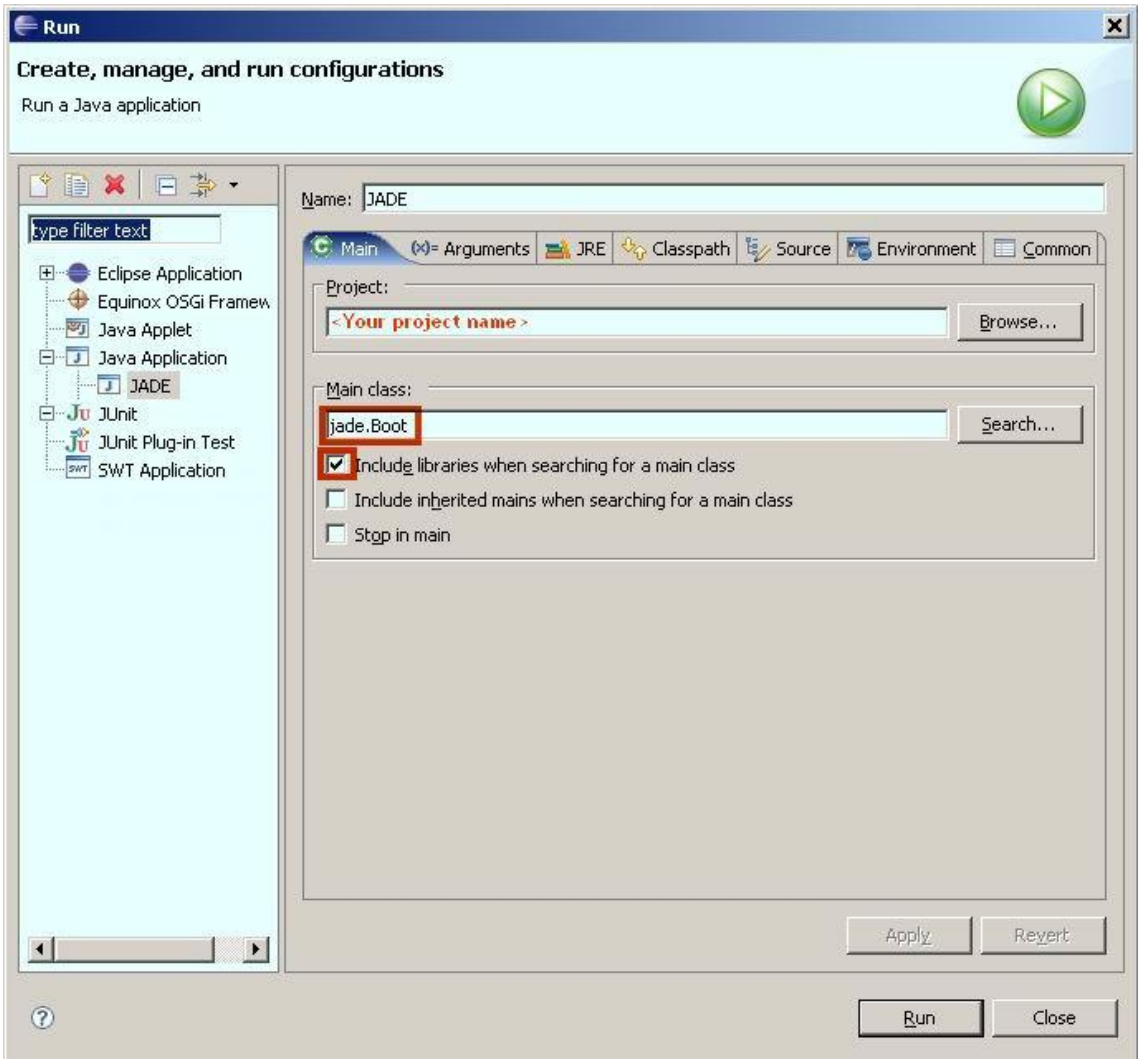


Figura 54: Configuración Eclipse y JADE 1

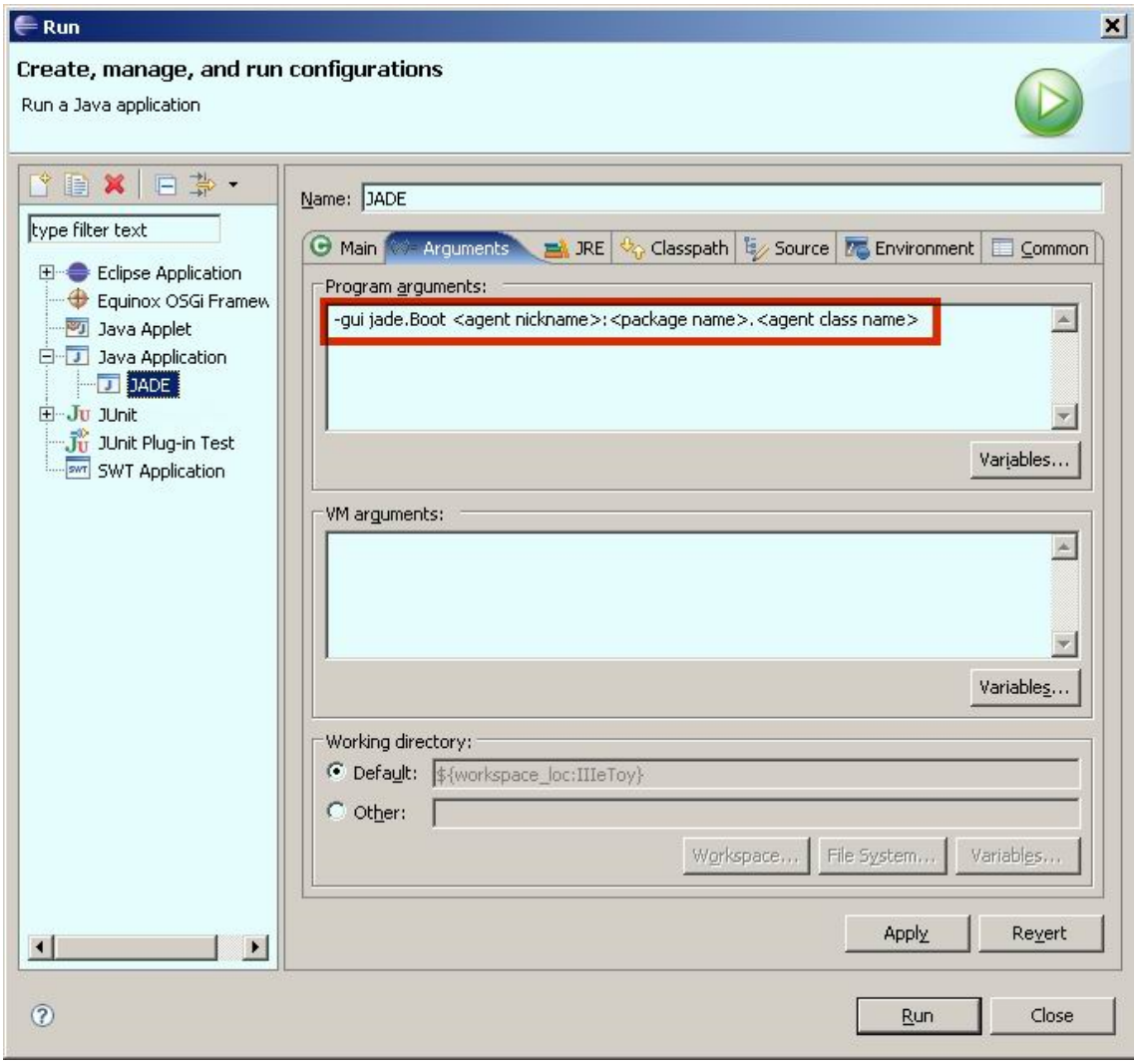


Figura 55: Configuración Eclipse y JADE 2

Una vez realizados estos pasos, si queremos lanzar a ejecución nuestra aplicación solo tenemos que ir al botón "Run", con lo que se crearían los agentes, se ejecutaría su método *setup()* y se mostraría la ventana de administración de JADE.

4.2. Herramientas de administración y depuración

La interfaz de administración y depuración que nos ofrece JADE, y que aparece cada vez que lanzamos nuestros agentes, nos abre un abanico de posibilidades para controlar y manejar a nuestros agentes en tiempo real. En esta sección hablaremos de cuáles son las principales funciones que podemos encontrar.

4.2.1. Lanzamiento de agentes usando al interfaz

Los agentes pueden ser lanzados de dos formas distintas: directamente desde la línea de comandos, que es como los lanzamos cuando utilizamos la configuración de eclipse explicada en el apartado superior, y desde la interfaz. En este apartado nos centraremos en el lanzamiento de agentes utilizando este segundo método.

Si queremos lanzar un nuevo agente, primero debemos decidir en que contenedor vamos a crearlo, por ello debemos seleccionar uno de nuestros contenedores disponibles (por defecto solo aparece el Main-Container en nuestra plataforma) y, sobre dicho contenedor, hacemos un clic derecho de ratón. Ahora, en el menú que nos ha aparecido debemos seleccionar la opción “Start New Agent”. En este momento, se nos abre un cuadro de dialogo como el que mostramos en la siguiente figura:

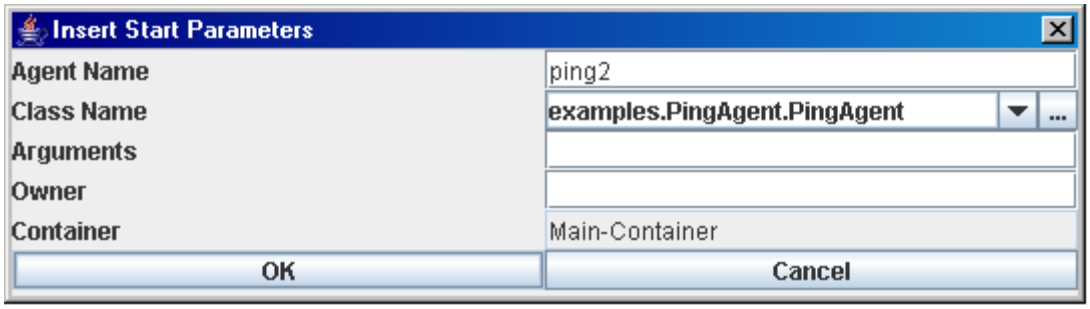


Figura 56: Lanzamiento de agentes utilizando la interfaz

Para lanzar el nuevo agente únicamente tenemos que rellenar los espacios necesarios. En este caso, indicamos cual es el nombre del agente en la celda “Agent Name” y la clase que define sus comportamientos en “Class Name”. Si el agente recibiese parámetros de entrada en su creación se especificarían en la celda “Arguments”. Una vez ya tenemos esto, le damos al boton “OK” y nuestro agente se creará. Para comprobarlo, podemos ver en el contenedor de la interfaz como aparece este nuevo agente.

4.2.2. Envío de mensajes ACL

La interfaz de JADE también nos permite crear mensajes de tipo ACL con los parámetros que nosotros deseemos y enviarlos a uno o más agentes destinatarios. Para ello, los pasos que debemos seguir son los siguientes.

Vamos al menú “Actions”->”Send Custom Message to Selected Agents”. En este punto, un cuando de dialogo especial es mostrado por pantalla, desde el cual

podremos redactar un nuevo mensaje y enviarlo a su destinatario. Para ello, deberemos rellenar los campos que se indican a continuación:

- Sender: donde indicaremos quien es el agente que envía dicho mensaje.
- Receivers: en este campo introducimos el destinatario del mensaje. Puede ser un agente o una lista de agentes, con lo que todos recibirían el mismo mensaje.
- Communicative act: dicho desplegable nos pide que seleccionemos cual es el acto comunicativo de este mensaje (INFORM, REPLY, REPLY_TO, CANCEL, etc.)
- Content: en esta celda introducimos es el contenido de nuestro mensaje. Lo que introducimos en este campo es recibido por los destinatarios como una cadena de caracteres (String).

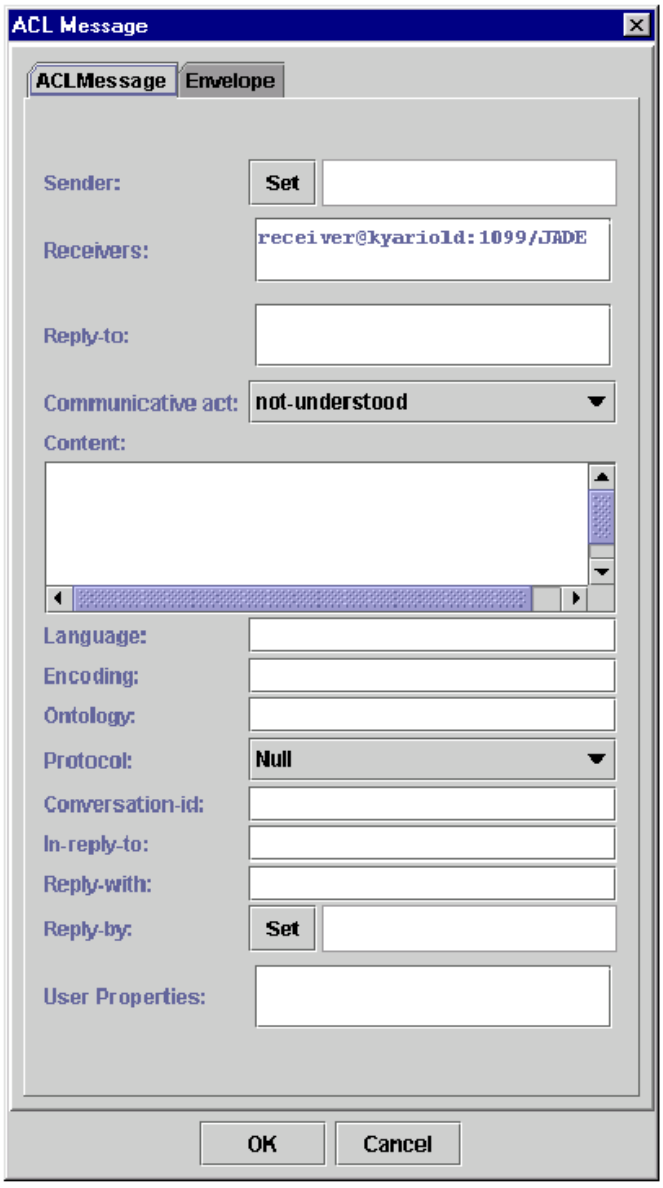


Figura 57: Envío de mensajes utilizando la interfaz

El resto de campos del mensaje no son obligatorios, por lo que solo deberemos rellenarlos cuando necesitemos enviar algun tipo de información determinada. Una vez los campos que nos interesan están completos hacemos click en el boton “OK” y el mensaje se crea y se envía a los destinatarios que hemos especificado.

4.2.3. Dummy Agent

La herramienta Dummy Agent la podemos encontrar en el menú “Tools”->”Dummy Agent”. Esta herramienta lo que nos permite es interactuar con los agentes JADE de forma más completa. En concreto, la interfaz no solo permite crear y enviar mensajes ACL sino que también mantiene una lista de todos los mensajes ACL que han sido enviados y recibidos por los agentes. Esta lista puede ser examinada por el usuario y cada mensaje de la lista puede ser visto en detalle e incluso modificado. Además, nos permite almacenar la lista de mensajes en el disco y para recuperarla más tarde.

El Dummy Agent puede ser ejecutado desde la interfaz o directamente desde la consola mediante el siguiente comando:

Java jade.Boot Dummy:jade.tools.DummyAgent.DummyAgent

Pudiendo además crear tantos agentes de tipo DummyAgent como necesitemos. La apariencia que nos muestra la interfaz es la que podemos ver en la siguiente imagen:

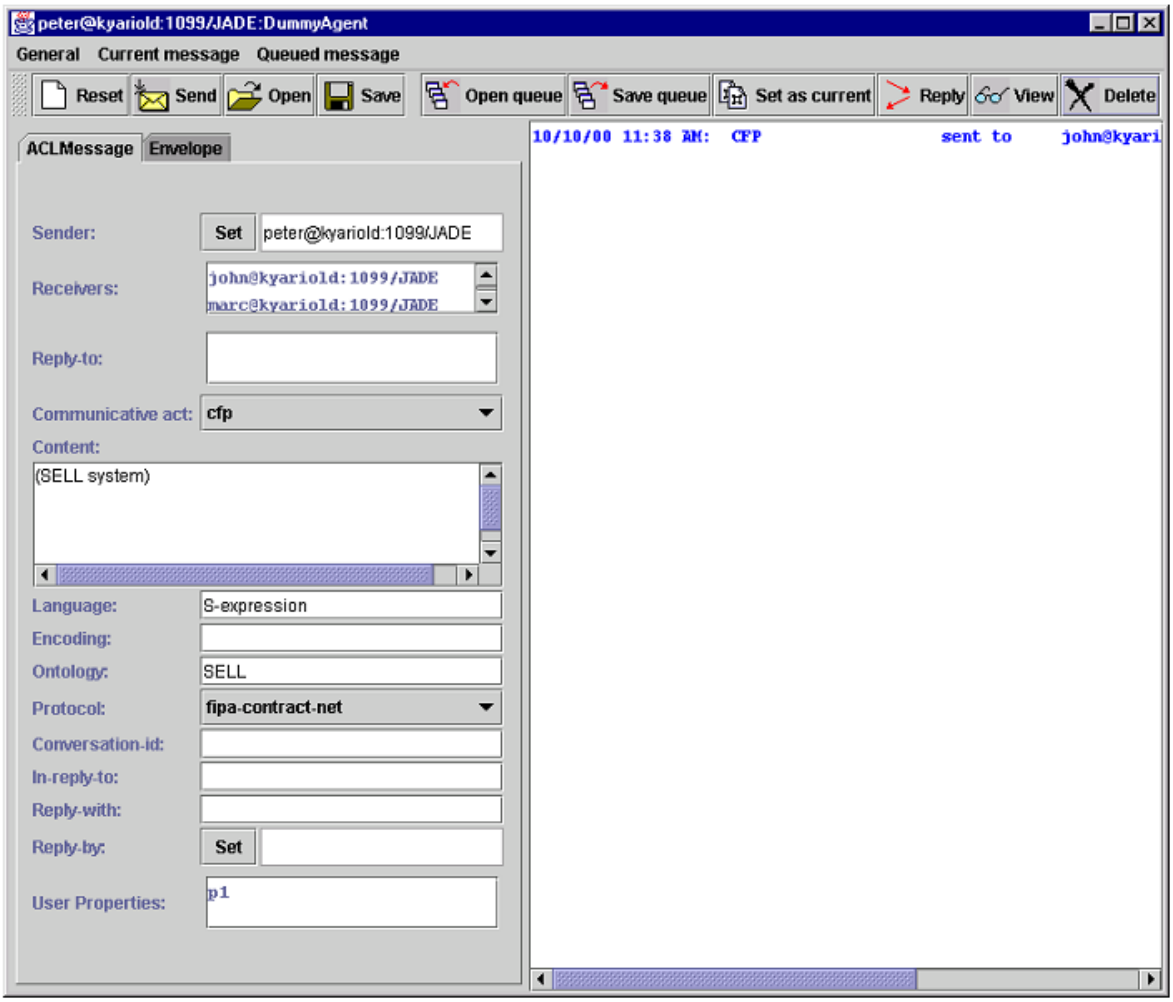


Figura 58: Herramienta DummyAgent

4.2.4. DF (Directory Facilator) GUI

Otra de las herramientas de gran utilidad que podemos encontrar en la interfaz de JADE es la GUI del DF. En ella, el usuario puede interactuar con el DF y realizar distintas acciones:

- Ver las descripciones de los agentes que están registrados
- Registrar y desregistrar agentes
- Modificar la descripción de agentes que ya están registrados
- Buscar a agentes con una determinada descripción

El funcionamiento del DF GUI se basa en la interacción con los agentes mediante envió de mensajes ACL para conocer cual es su estado actual, por lo que esta interfaz

puede ser lanzada únicamente desde el PC que contiene el Main-Container, ya que es donde se encuentra el agente DF.

Además, esta interfaz también permite registrar el DF con otros DFs de otras plataformas, con lo que fácilmente podemos tener una compleja red de dominios y subdominios del servicio de páginas amarillas.

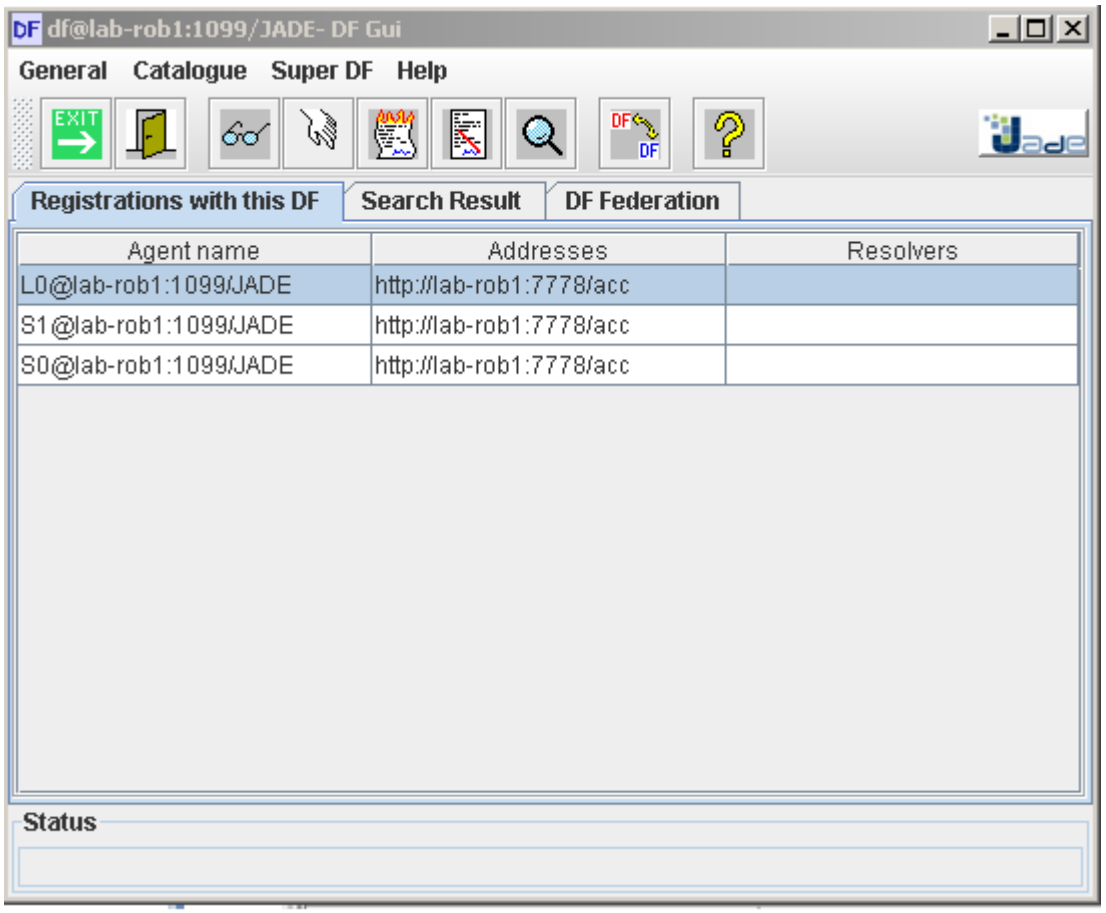


Figura 59: Herramienta DF GUI

Si hacemos doble click en cualquiera de los agentes que nos aparecen en la lista se abre una ventana donde podemos ver los servicios que tiene activos un determinado agente en ese momento, información que se actualiza constantemente y que nos permite conocer información de gran interés en tiempo real.

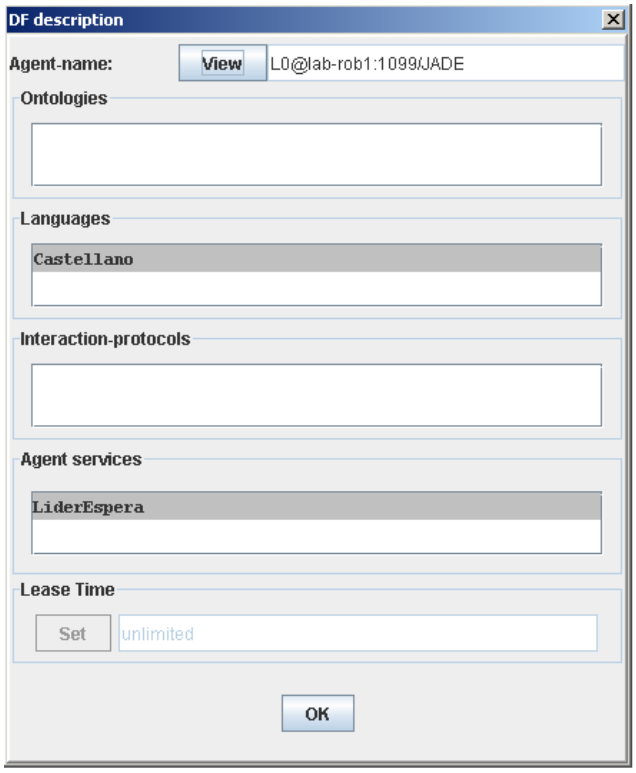


Figura 60: Consulta de servicios publicados con la herramienta DF GUI

4.2.5. Sniffer agent

Otra de las grandes herramientas de la interfaz de JADE es el Sniffer Agent. A través de este agente, un usuario puede realizar un "sniff" de un agente o de un grupo de agentes, con lo que todos los mensajes que viajan hacia o desde el agente sobre el que se realiza el "sniff" es inspeccionado y mostrado en dicha interfaz. Todos los mensajes que viajan pueden ser inspeccionados e incluso almacenados en el disco. El usuario puede también inspeccionar todos los mensajes y almacenarlos en un fichero para su posterior análisis y depuración.

Esta herramienta puede ser lanzada desde el menú "Tools"->"Sniffer Agent" o desde la línea de comandos a través del siguiente código:

Java jade.Boot sniffer:jade.tools.sniffer.Sniffer

La siguiente figura muestra una captura de dicha interfaz:

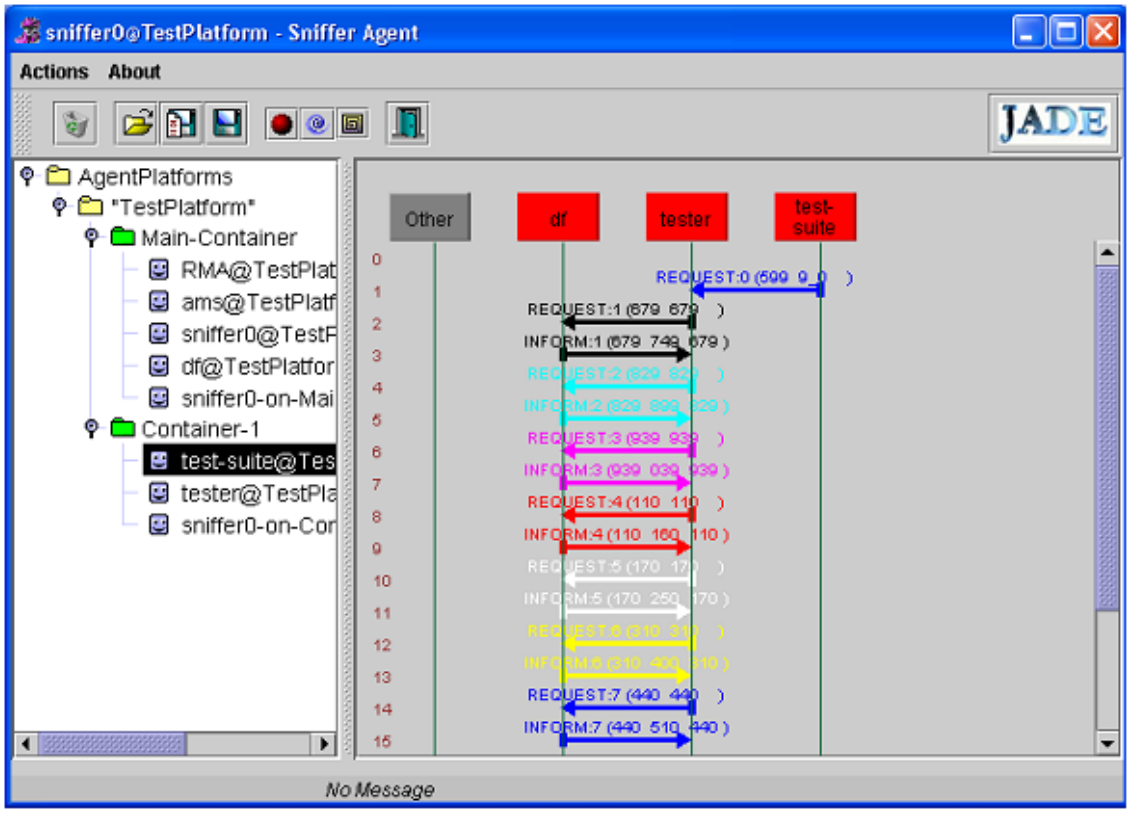


Figura 61: Herramienta SnniferAgent

4.2.6. Introspector Agent

Esta herramienta permite monitorizar y controla el ciclo de vida de un agente que esta en ejecución. Aquí podemos encontrar información muy útil, pudiendo hace un debug a cada uno de los agentes de nuestra plataforma. Para ejecutarlo, debemos ir a "Tools"->"Instrospector Agent".

En la interfaz que se nos muestra podemos hacer un debug individual a cada uno de los agentes que tenemos en ejecución. Para ello solo debemos seleccionar uno de los agentes que nos aparece en la lista de la izquierda y con el botón de la derecha seleccionamos "Debug On". Nos aparecerá una ventana con la siguiente información:

- Current state: estado en el que se encuentra el agente actualmente
- Incoming Messages: Lista de los mensajes recibidos por el agente en cuestión.
- Outgoing Messages: Lista de los mensajes que ha enviado un determinado agente

- Behaviours: Esta es la información más importante de la herramienta, en la que podemos ver cuales son los comportamientos que tiene activos un agente en un determinado instante.

En la siguiente imagen podemos ver una captura de pantalla con la información que acabamos de explicar.

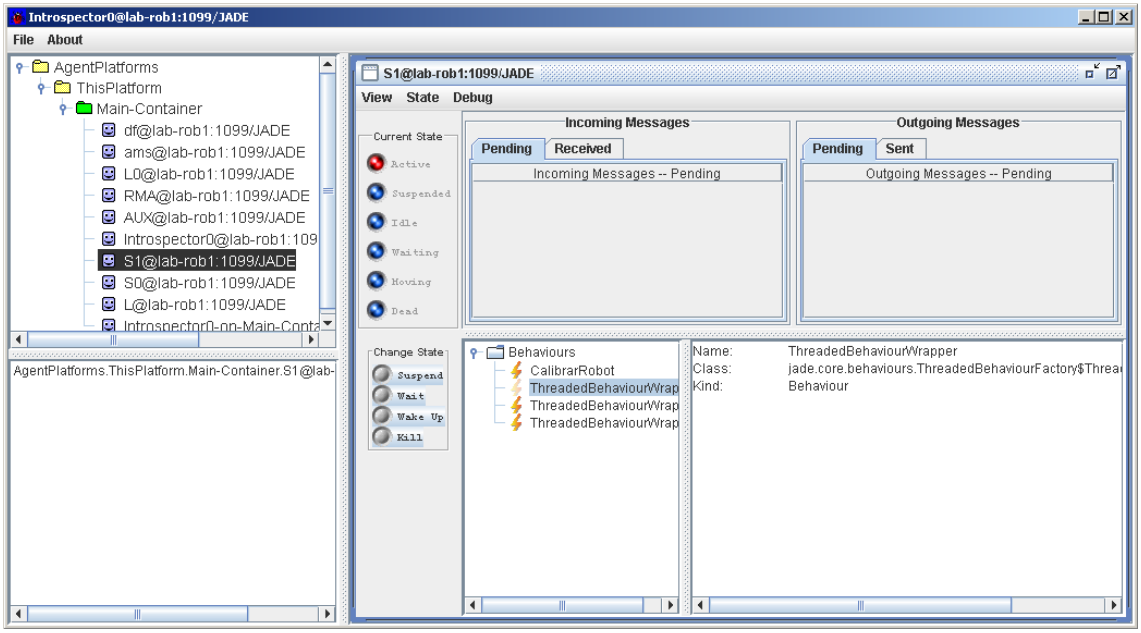


Figura 62: Herramienta Introspector Agent

4.3. Distribución de la aplicación en varios PCs

Uno de los aspectos más importantes en la utilización de la librería JADE es la facilidad que proporciona para distribuir la aplicación a través de la red. Podemos distribuir fácilmente nuestros agentes en varios computadores, con lo que la escalabilidad de una determinada aplicación está garantizada.

Al distribuir una aplicación, el host principal crearía los agentes AMS, DF y RMA en su contenedor principal o "Main-Container", añadiendo el resto de host un nuevo contenedor que se comunicaría con éste.

Como podemos observar solo tendremos una instancia de los agentes AMS y DF por lo que, cuando un robot en un contenedor remoto hace una búsqueda de un determinado servicio, se conectará a través de la red con el contenedor principal y a su vez con el agente DF. De esta forma, mientras la aplicación se encuentra distribuida, la publicación de los servicios está centralizada en el agente DF del host principal. El aspecto que quedaría después de la distribución es el siguiente:

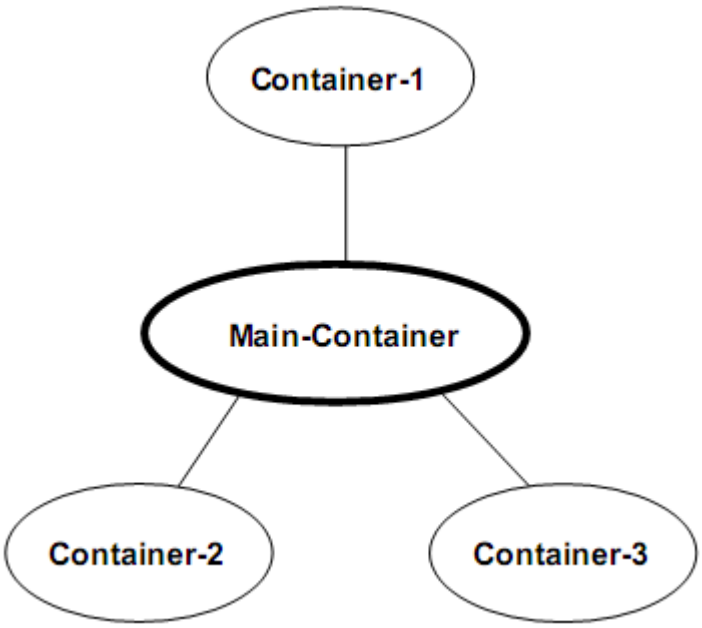


Figura 63: Contenedores en JADE

Lo primero que tenemos que hacer para distribuir la aplicación es seleccionar cual es el host que albergará el contenedor principal. Este host no tendrá que hacer nada especial, ya que son los otros host los que se conectan a éste, por lo que el lanzamiento de los agentes no varía. Si solo queremos poner en marcha la aplicación sin lanzar agentes para que esté disponible para conexiones remotas ejecutamos la siguiente instrucción:

Java jade.Boot

Una vez tenemos el contenedor principal en ejecución, si queremos agregar un nuevo host a esta arquitectura, debemos lanzar desde el nuevo host la siguiente instrucción:

Java jade.Boot –host IPMainContainer –container

La opción –container le indica a JADE que tiene que iniciar un nuevo contenedor en el mismo lugar que el contenedor principal. Podemos añadir la instrucción –port númeroPuerto para seleccionar el puerto por el que queremos que se realice la conexión. De lo contrario, JADE tomará el puerto por defecto, que en este caso es el 1099. Además, para que la distribución pueda funcionar sin problemas es necesario indicar a nuestro cortafuegos y/o antivirus que permita las conexiones entrantes y salientes en el puerto en cuestión, ya que de lo contrario no permitirá el intercambio de datos y la distribución fallará.

Si todo ha funcionado bien en la interfaz del host principal podemos ver la conexión de los contenedores remotos.

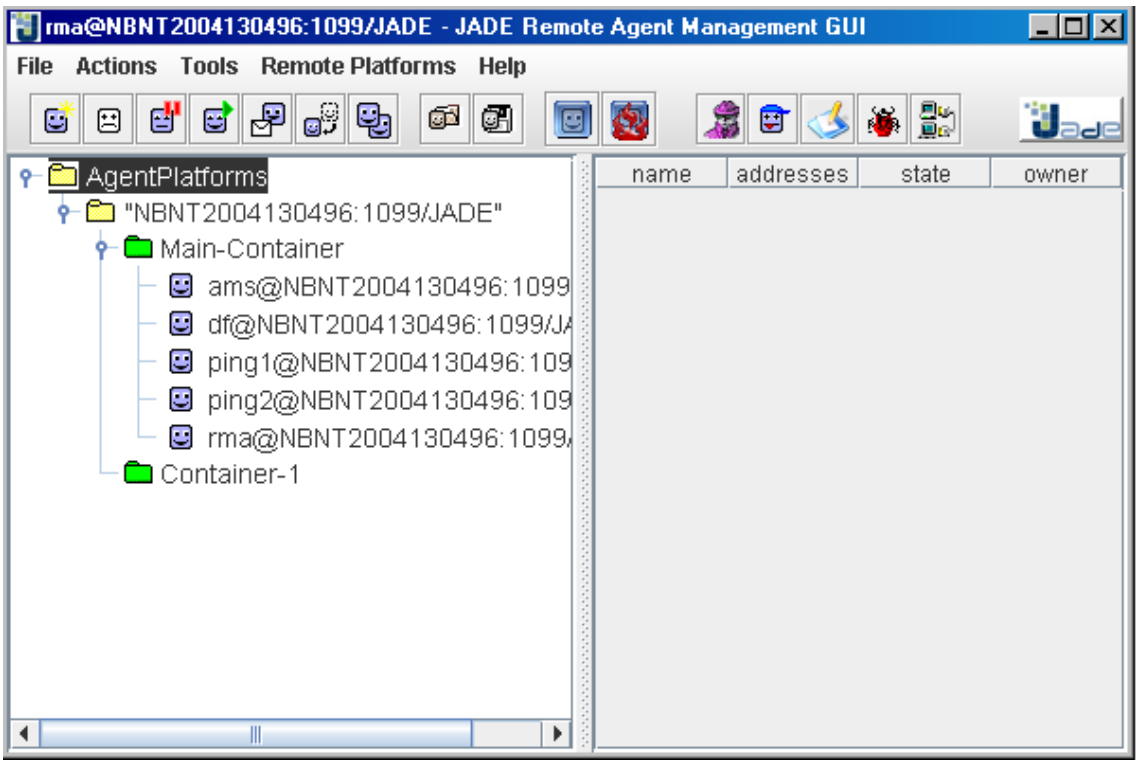


Figura 64: Aplicación distribuida en JADE

4.4. Java Threads: Concurrencia en Java.

Una de las principales características que se ha utilizado de la programación en JAVA es la creación de hilos de ejecución (también conocidos como Threads) para posibilitar la ejecución de tareas concurrentes, tanto en el robot como en los agentes (que se ha solucionado utilizando un comportamiento ThreadedBehaviour).

La máquina virtual de Java (JVM) es un sistema multi-thread. Es decir, es capaz de ejecutar varias secuencias de ejecución (programas) simultáneamente. La diferencia básica entre un proceso de Sistema Operativo y un Thread Java es que los Threads corren dentro de la JVM, que es un proceso del Sistema Operativo y por tanto comparten todos los recursos, incluida la memoria y las variables y objetos allí definidos. A este tipo de procesos donde se comparte los recursos se les llama a veces 'procesos ligeros' (lightweight process). Java da soporte al concepto de Thread desde el mismo lenguaje, con algunas clases e interfaces definidas en el paquete java.lang y con métodos específicos para la manipulación de Threads en la clase Object. Desde el

punto de vista de las aplicaciones los threads son útiles porque permiten que el flujo del programa sea divido en dos o más partes, cada una ocupándose de alguna tarea.

Para la mayor parte de las aplicaciones programadas ha sido necesario el conocimiento y el manejo de varios hilos de ejecución. Por ejemplo, si la idea es que el robot siga un comportamiento o una rutina y a causa de un cambio externo al robot y detectado por el ordenador, se desea modificar el comportamiento del mismo, se necesitará al menos dos hilos de ejecución en el robot: uno que lleve a cabo la rutina y otro que cada pequeño intervalo de tiempo escuche al ordenador para saber si debe cambiar su rutina debido a algún cambio externo a él. Estas técnicas se han utilizado en las dos aplicaciones que detallamos a continuación

4.5. Aplicación I: Robots limpiadores.

4.5.1. Introducción

El hombre, durante toda su existencia, ha convivido con el deseo de crear máquinas capaces de realizar tareas que le facilitasen el trabajo. Por esta razón, la primera aplicación que se ha llevado a cabo ha seguido por esta línea, buscando un modelo autónomo que se encargara de coordinar dos o más robots para realizar la limpieza de un determinado espacio. Este modelo holónico proporciona las directivas de comportamiento a seguir por cada uno de los robots limpiadores, consiguiendo que se coordinen entre ellos para así alcanzar un objetivo común, que en este caso sería mantener su espacio libre de manchas (pelotas).

4.5.2. Descripción del escenario

El escenario concreto que se ha utilizado para realizar las distintas pruebas consiste en una superficie de 130 cm de anchura y 170 cm de largura. En ella, se han introducido dos robots, los cuales se encargan de la limpieza de una determinada zona de la superficie. En concreto, un robot tiene asignada la zona norte mientras el otro se encarga de la zona sur. La distribución de robots en la superficie se puede apreciar en la siguiente imagen:

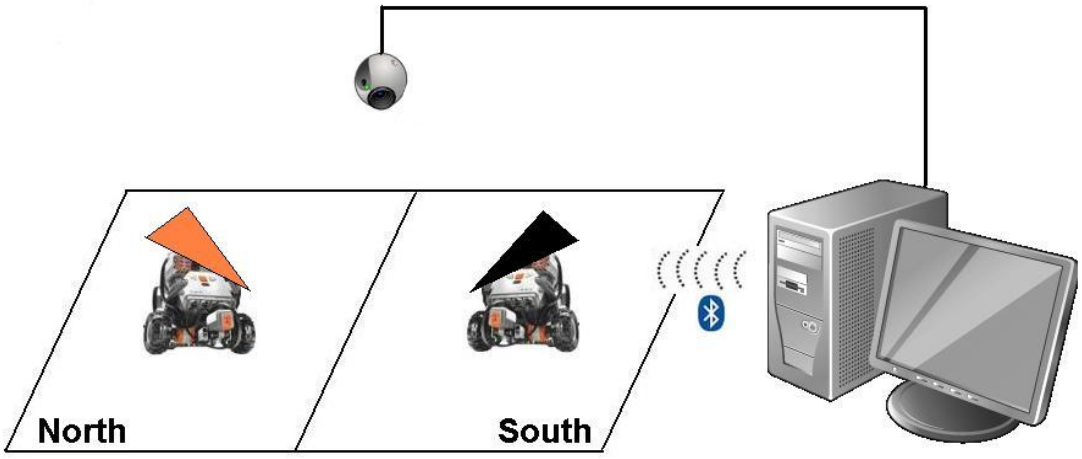


Figura 65: Escenario aplicación robots limpiadores

Para este ejemplo concreto se han utilizado los robots LEGO NXT con la versión de firmware LejOS, garantizando así la completa compatibilidad entre los robots y la plataforma de agentes JADE, funcionando todos ellos bajo el lenguaje de programación de alto nivel JAVA.

En el problema se ha introducido también una cámara cenital que está constantemente (a intervalos regulares de tiempo) obteniendo imágenes del escenario, a partir de las cuales se extrae toda la información necesaria. En concreto, la información que nos proporciona dicha cámara es utilizada para conocer en todo momento cuáles son las coordenadas reales de cada uno de los robots, así como la aparición en el escenario de nuevas manchas/pelotas, lo que nos permite comunicar al robot la aparición de estas y poder activar así sus comportamientos.

4.5.3. Captura y procesamiento de imágenes

Para el funcionamiento de la aplicación es necesario obtener toda la información posible de las imágenes que nos proporciona la cámara cenital que tenemos instalada. Para la obtención y filtrado de dichas imágenes se utiliza la librería OpenCV que funciona bajo Java. Se trata de una librería de visión de código abierto desarrollada por Intel bajo la licencia BSD. Es multiplataforma, con lo que puede funcionar en Windows, Linux y MAC OS X. Esta implementación no es una versión completa como la que se utiliza para otros lenguajes como C++, aunque tenemos acceso a todas las funciones necesarias para obtener la información de cada una de nuestras capturas en tiempo real. En concreto, las acciones que debe realizar el algoritmo encargado de controlar la cámara son:

- Obtener la posición y orientación de cada uno de los robots, para así poder calibrar su posición constantemente.

- Obtener la aparición de las manchas y sus coordenadas, con el fin de informar al robot pertinente de que debe llevar a cabo su limpieza.

Para la detección de los robots se les ha añadido un triángulo de color naranja y otro de color negro en la parte superior a modo de señuelo, con lo que realizando un filtrado por color podemos averiguar de qué robot se trata y a que zona pertenece. Así mismo, el triángulo isósceles posibilita también que podamos obtener cual es la orientación del robot en todo momento

Se obtiene una respuesta rápida y en tiempo real frente a la aparición de las manchas, ya que la cámara esté constantemente tomando capturas del escenario y obteniendo la información. En nuestra aplicación en concreto se muestrea el escenario cada 50 milisegundos. En cada uno de esos muestreos las acciones que realiza la cámara son las siguientes:

1. Obtiene una captura del escenario

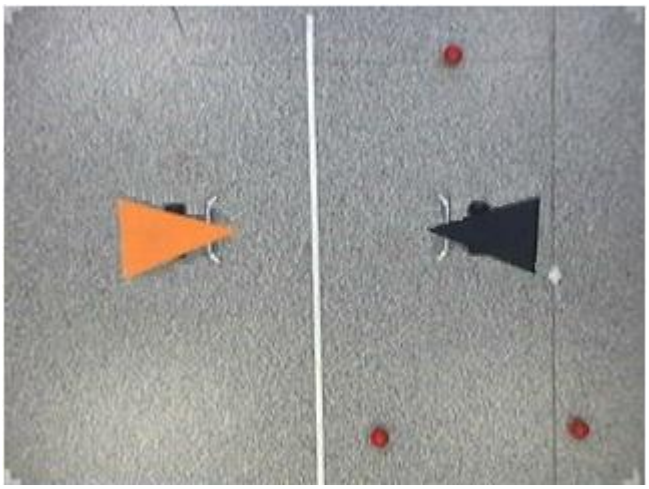


Figura 66: Captura del escenario

2. Realiza un filtrado de la imagen para eliminar toda la información no relevante en nuestro objetivo.

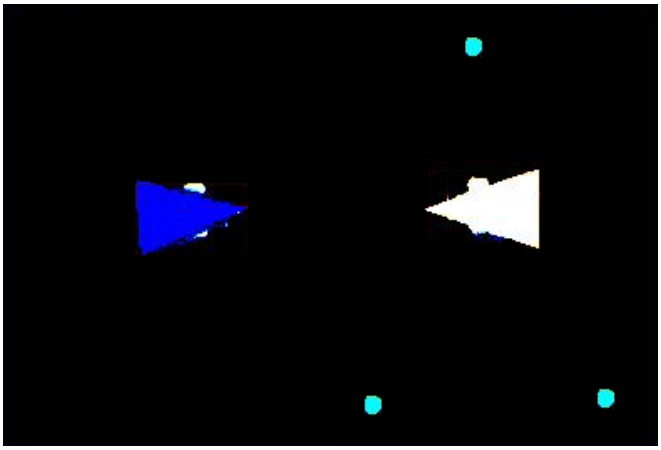


Figura 67: Filtrado de la imagen

3. Se obtienen cada uno de los bloobs.

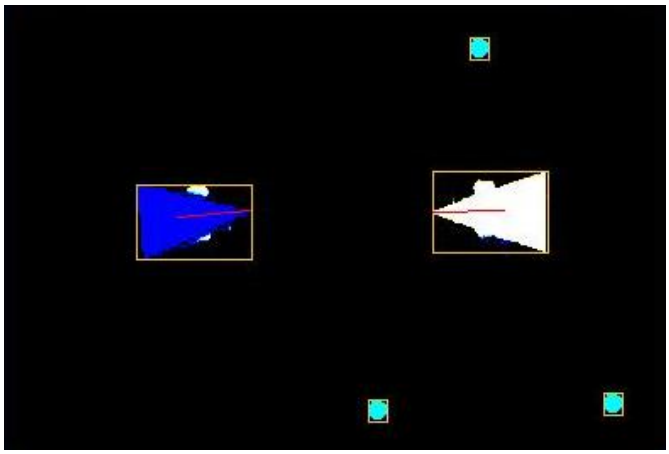


Figura 68: Obtención de la información

4. Para cada uno de estos bloobs:

    a. Se obtienen sus componentes RBG, con lo que podemos discriminar si se trata de una mancha, del robot norte o del robot sur.

    b. Se dibuja cada una de estas manchas en la interfaz para ayudar a su procesamiento.

    c. Para los robots, se obtienen no solo sus coordenadas, sino su orientación a partir de la información que nos proporciona el triangulo isósceles situado sobre ellos. La estrategia a seguir para conseguir la orientación del robot es obtener el vector que va desde el centroide del

bloob (que en este caso es el triángulo isósceles) hasta el punto más lejano de éste, que será el vértice de su ángulo desigual, el cual tiene la misma orientación que el propio robot. Podemos ver una explicación gráfica a continuación.

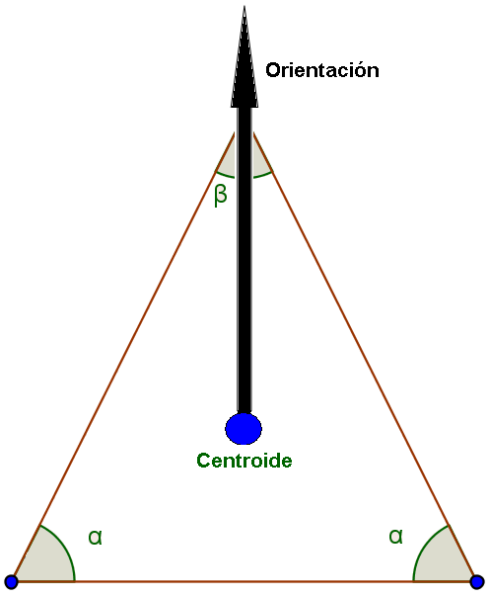


Figura 69: Calculo de la orientación

d. Una vez procesada toda la información, almacenamos tanto la posición de los robots como la de las manchas. Esta información será accedida por el agente que controla la cámara para conocer cual es el estado del escenario y así avisar a los robots pertinentes.

4.5.4. Programación de los algoritmos

Para la programación del algoritmo que soluciona este problema se han desarrollado varios agentes. En concreto, tenemos dos tipos de agentes, uno de ellos para el control de la cámara y otro para el control de los robots.

4.5.4.1. AgenteCámara

El agente CamAgent se encarga de la obtención de imágenes de la cámara a través de la librería OpenCV. Para cada una de estas imágenes, dicho holón se encarga realizar el procesado y obtener sus bloobs, a partir de los cuales puede detectar la

posición y orientación de ambos robots, así como la aparición o no de nuevas manchas. En el caso de que se encuentre alguna mancha, este agente realiza la distribución del trabajo entre los agentes que controlan los robots siguiendo un determinado patrón, encargándose de la distribución y coordinación de éstos.

El objetivo es crear un grupo de robots limpiadores que realizan sus tareas independientemente, pero que sin embargo pueden colaborar entre si en momentos específicos. En principio, cada uno de los robots debe encargarse de limpiar su propia zona. Sin embargo, si se da la situación en que un determinado robot esta ocioso por no tener manchas en su zona activa y, por el contrario, el otro robot tiene mucho trabajo, el primer robot ayudaría a éste en la limpieza de su zona. Para ello, el agente cámara realiza una búsqueda de servicios ofrecidos por los agentes siguiendo el flujo que podemos ver en el siguiente diagrama:

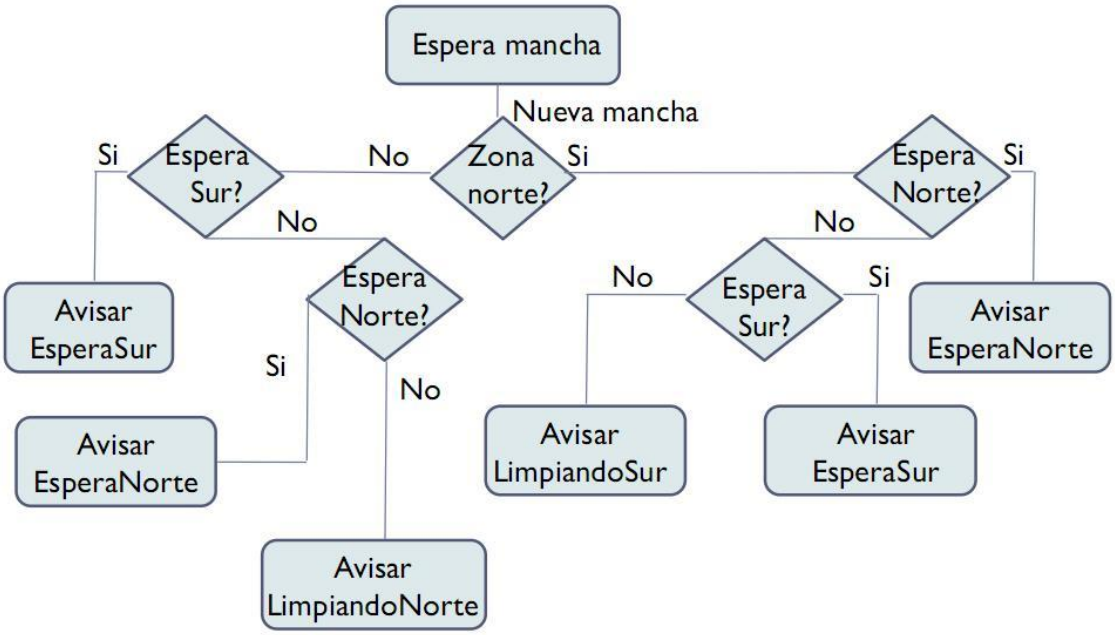


Diagrama 1: Búsqueda de estados

Como podemos apreciar en diagrama anterior, se prioriza que cada robot limpie la zona que le ha sido asignada mediante la búsqueda del servicio EsperaNorte o EsperaSur. Si dicho servicio no está disponible significará que no hay ningún robot en estado de espera, por lo que se buscará robots pertenecientes a otra zona en estado ocioso que, en caso de estar disponibles, se desplazarían a esta zona para ayudar en la limpieza de las manchas de otros robots. Si en este punto tampoco hay agentes que ofrezcan el servicio necesario, se entregará la mancha al robot encargado de limpiar la zona en la que ha sido encontrada.

4.5.4.2. AgenteLimpiador

Además del agente encargado de controlar la cámara, cada uno de los robots LEGO NXT está controlado por un agente JADE. Este agente se comunica con el robot a través de una conexión Bluetooth, mediante la cual se consigue enviar las acciones a realizar de forma inalámbrica. Su función consiste en:

1. Esperar la recepción de una determinada mancha. En este estado el servicio que oferta este robot sería Esperando[Sur, Norte].
2. Si se recibe una mancha se desplaza hasta sus coordenadas para realizar la limpieza. Durante este proceso el robot actualiza su servicio a Limpiando[Norte, Sur].
3. Una vez se ha realizado la limpieza comprueba si han recibido nuevas manchas. En caso afirmativo volvería al punto 2 para realizar las tareas de limpieza.
4. Si, por el contrario, no ha recibido ninguna mancha, vuelve a su posición inicial, actualizaría su servicio a Esperando[Norte, Sur], y vuelve de nuevo al punto 1.

A continuación podemos ver una sucesión de imágenes en las que se ilustra el proceso que se ha comentado anteriormente.

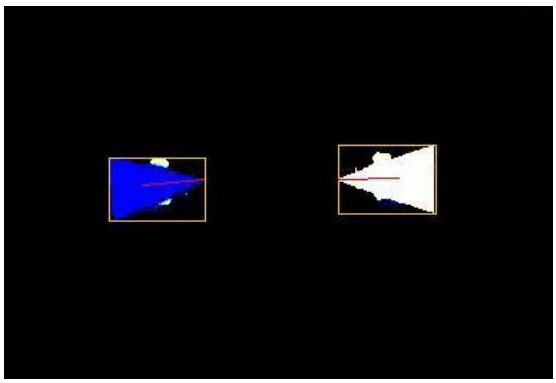


Figura 70: Captura 1

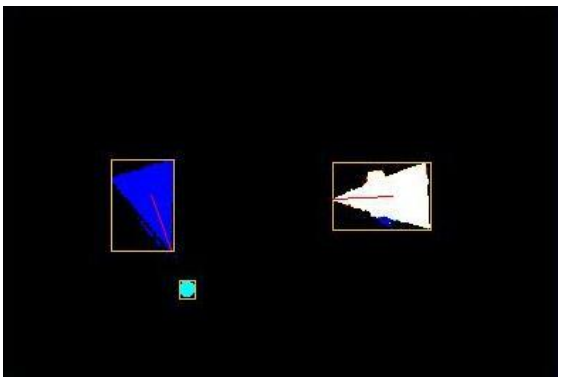


Figura 71: Captura 2

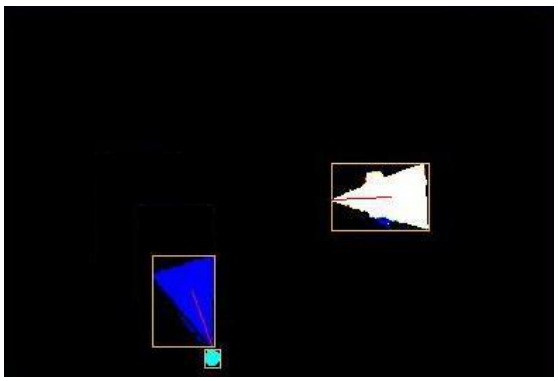


Figura 72: Captura 3

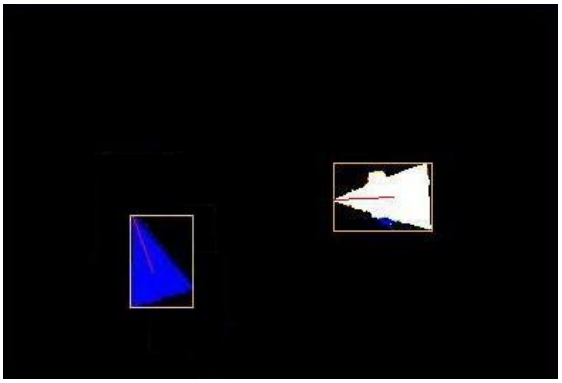


Figura 73: Captura 4

4.5.4.3. Código robots

En este caso, el código que se ha implementado en cada uno de los robots se limita a recibir y alcanzar cada una de las coordenadas donde aparece el objeto a limpiar, dejando la parte de razonamiento y coordinación en el código de cada uno de los agentes.

4.6. Aplicación II: Generación de convoyes

4.6.1. Introducción

El objetivo de la presente aplicación es la generación de convoyes de robots LEGO NXT, los cuales se desplazan a las coordenadas destino que se les ha indicado. Durante su desplazamiento a una nueva posición, el convoy puede sufrir distintos cambios, tanto en su trayectoria como en su estructura, perdiendo alguno de los robots que le siguen si es necesario. Además, cada uno de los convoyes tiene asignado un nivel de prioridad, el cual se utilizará a la hora de tomar decisiones como, por ejemplo, la división del convoy por la aparición de otro más prioritario que éste y para el cual no hay suficientes robots libres.

4.6.2. Descripción del escenario

Para la implementación de la aplicación se ha utilizado un espacio acotado por el cual los robots pueden desplazarse libremente. Ocasionalmente, dentro de dicho espacio pueden colocarse distintos obstáculos (a priori desconocidos por el convoy) que tendrán que ser sorteados utilizando distintas técnicas de evasión de obstáculos. En cuanto a los robots, de nuevo se comunican mediante la tecnología Bluetooth con el PC, a través de la cual reciben las acciones a realizar por el comportamiento que están ejecutando. Esta información también se utiliza para enviar información hacia el

PC como puede ser la presencia de un determinado obstáculo, la llegada de un convoy a su destino o el nivel de batería.

Uno de los aspectos más importantes de la implementación que cabe destacar es la escalabilidad, ya que podemos agregar tantos robots como queramos sin apenas modificar el código, distribuyendo la aplicación en varios host si es necesario.

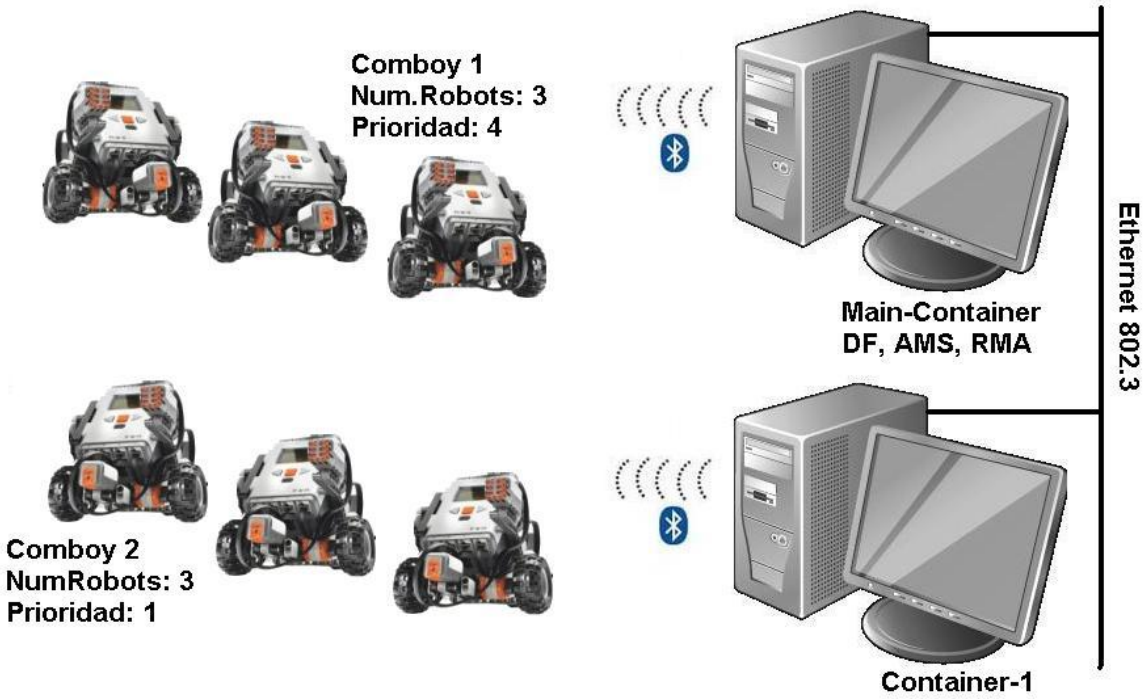


Figura 74: Escenario aplicación convoyes

En cuanto a los robots existen dos configuraciones con distinta sensorización, una para el robot líder del convoy y otra para los robots seguidores

4.6.2.1. Configuración del robot líder del convoy

El robot líder del convoy, al ser el encargado de dirigir al resto de robots por el escenario evitando cualquier tipo de colisión, va provisto de un sensor de ultrasonido móvil, con el cual se puede obtener la distancia hacia un cierto obstáculo desde distintos ángulos. Esta configuración es necesaria para la implementación de los algoritmos de evasión de obstáculo que se explicarán más adelante.

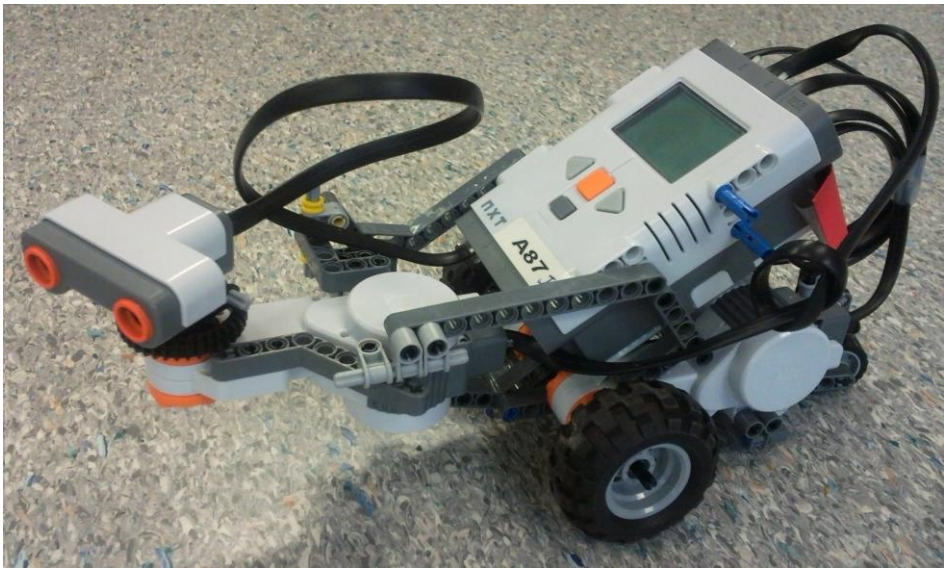


Figura 75: Configuración robot líder

4.6.2.2. Configuración del robot seguidor

El robot seguidor es aquel cuya única función se limita a seguir a un líder o a otro robot seguidor, por lo que su sensorización se limita a incorporar una cámara NXTCam, la cual es capaz de detectar colores y devolver su posición relativa. En nuestro caso, el color que se ha programado es el rojo, por esto todos los robots poseen en la parte trasera una especie de señuelo rojo, el cual es utilizado por el robot seguidor para ejecutar su algoritmo de persecución.

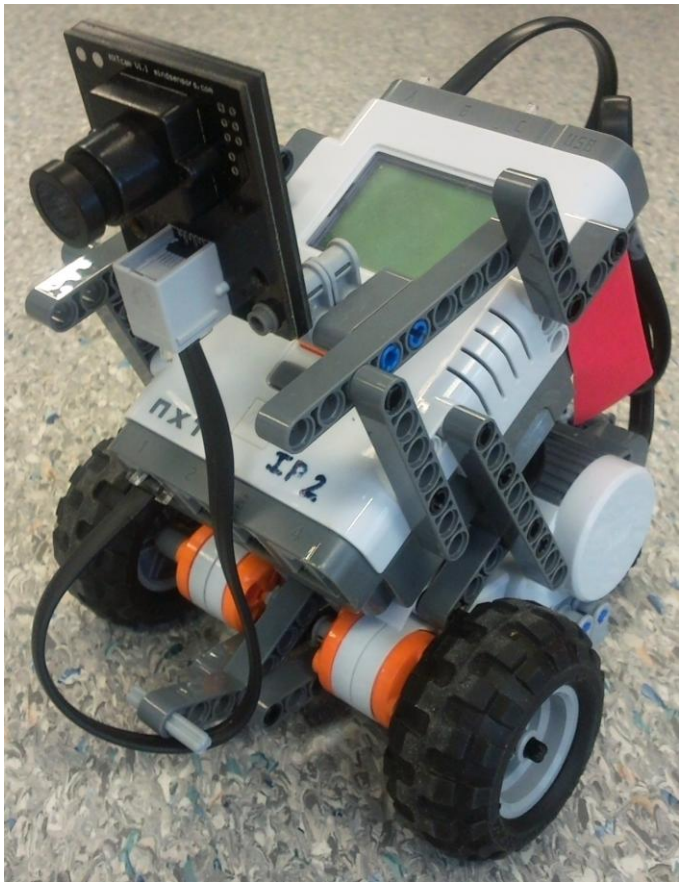


Figura 76: Configuración robot seguidor

4.6.3. Programación del algoritmo

En la programación de la aplicación se ha buscado añadir la funcionalidad necesaria para dotar a los convoyes de la autonomía y capacidad de negociación necesaria para responder a diversas situaciones, en las que se requiere toma de decisiones y colaboración entre varios robos o convoyes.

4.6.3.1. AgenteLider

Este agente se encarga del control del robot líder, controlando a un determinado robot para que ejerza las acciones propias de un líder del convoy. En concreto, este agente realiza las siguientes funciones:

1. Creación de un convoy, después de recibir una determinada solicitud y de comprobar la disponibilidad de los robots seguidores que se requieren. Si hay suficientes robots seguidores en estado SeguidorEsperando se creará directamente. Sino, buscará un convoy menos prioritario con el número de robots suficientes para crear el nuevo convoy. Si lo hay, solicitará estos robots a su agente líder, sino espera una nueva solicitud.

2. Desplazamiento a las coordenadas destino evitando los posibles obstáculos que puedan aparecer en la trayectoria.

3. Negociación y cesión de robots en caso de aparición de un nuevo convoy más prioritario.

4. Negociación con el resto de robots del abandono del convoy por un determinado robot al no tener batería suficiente para proseguir.

Además de los comportamientos propios para la realización de estas acciones, se han definido dos comportamientos de tipo ThreadedBehaviour, los cuales se ejecutan en un hilo dedicado de forma concurrente al resto de comportamientos. Estos dos comportamientos se encargan del manejo de la interfaz Bluetooth: uno se encarga del envío de datos y el otro de la recepción, para evitar problemas que se comentan en secciones posteriores.

4.6.3.2. AgenteSeguidor

El agente seguidor posee los comportamientos necesarios para formar un determinado convoy cuando éste sea solicitado. Su funcionalidad se resume básicamente en tres tipos distintos de acciones:

-Crear y seguir un determinado convoy.

-Escuchar posibles solicitudes del robot líder para abandonar el convoy actual y formar parte de uno más prioritario.

-Conocer continuamente el estado de la batería y decidir si tiene que abandonar el convoy, solicitando al robot líder su abandono y esperando confirmación por parte de este.

Del mismo modo que en el caso anterior, se han definido dos comportamientos de tipo ThreadedBehaviour para la recepción y envío de datos a través del túnel Bluetooth.

4.6.3.3. AgenteSolicitador

Este agente se limita a solicitar un determinado número de convoyes con una prioridad entre los valores 1 a 9, siendo prioridad 1 la prioridad más alta y 9 la menos prioritaria.

Su funcionamiento se basa en recibir, por parte de una capa superior, la solicitud de una creación de convoy. Es en este momento cuando el AgenteSolicitador busca

holones con el servicio LiderEsperando para solicitar la creación de un robot con las características requeridas.

4.6.3.4. Programación de los robots

La programación de los robots ha sido menos laboriosa que la de los agentes, ya que son estos últimos los encargados de llevar a cabo la coordinación entre los robots. El robot es capaz de satisfacer todas las necesidades del agente que lo controla y adaptarse a las situaciones cambiantes que le afectan. Para ello se han creado un código que controla al robot líder y otro que controla al robot seguidor.

En concreto, las acciones que puede realizar un robot líder y que son recibidas a través de la interfaz Bluetooth son las siguientes:

- CalibrarRobotLider: Este método se encarga de informar al robot de cual es su situación actual (posición y orientación) al iniciar la aplicación.
- IniciarConvoy: Con esta orden se sitúa al robot líder en la posición necesaria para comenzar a construir un convoy. Las coordenadas en las que se creará también las recibe desde su agente.
- IrADestino: Mueve el robot líder a cada uno de los destinos que ha recibido, informando de las posibles situaciones que pueden suceder (aparición de obstáculos, fin de la navegación).
- EvitarObstaculo: Después de localizar un obstáculo en el escenario y trayectoria del robot, este código se encarga de realizar la evasión del obstáculo, pudiendo elegir entre el algoritmo Bug2 y VFH.
- IrAConvoy: En este caso, el robot se desplaza hasta un nuevo convoy por ser más prioritario y no disponer de los robots necesarios para su creación. En este método se realiza también evasión de posibles obstáculos.
- Parar: Se limita a detener el convoy para satisfacer necesidades como la aparición de un convoy más prioritario, el agotamiento de la batería de un robot seguidor, etc.

Por su parte, el robot seguidor puede realizar las siguientes acciones:

- CalibrarRobotSeguidor: De forma análoga al robot líder, este método calibra al robot indicándole su posición inicial y sus coordenadas dentro del escenario.

- IrAConvoy: Mueve el robot seguidor hasta su posición en el convoy para que pase a formar parte de éste.

- SeguirConvoy: Código que se encarga de desplazar el robot en el escenario siguiendo a su predecesor. Este método se explica en profundidad en la sección siguiente.

- Parar: Detiene al robot para satisfacer situaciones como el agotamiento de batería, la aparición de un convoy más prioritario, etc.

- GetPosiciónRobot: Este método devuelve cuales son las coordenadas y orientación del robot sobre el que se ejecuta. Es necesario para que, si aparece un convoy mas prioritario, el nuevo robot líder pueda calcular la posición a la que debe acudir para crear el nuevo convoy.

- Giro: Realiza un giro de 180º sobre si mismo para que el robot que lo ejecuta pase a formar parte de un nuevo convoy más prioritario.

- AbandonoBatería: Si se da la situación en la que un robot no tiene la suficiente batería para continuar, se ejecutará este comportamiento para conseguir que el robot pueda abandonar el convoy.

En ambos casos, tanto para el robot líder como para el robot seguidor, se ejecutan concurrentemente los hilos encargados de escuchar la interfaz Bluetooth.

4.6.3.5. Programación de la cámara NXTCam

Para dotar a los robots seguidores de la capacidad de seguir a un determinado robot se ha utilizado el sensor NXTCam. Este sensor realiza capturas a través de su objetivo para más tarde procesarlas. Básicamente lo que hace es buscar en la imagen zonas con el color o los colores que se le han programado para luego devolverlos en un vector, añadiendo a cada muestra información útil para su procesamiento. En nuestro caso, la cámara ha sido programada para obtener objetos de color rojo, ya que serán las marcas que poseerán cada uno de los robots en su parte trasera. Para ello, debemos de seguir estos pasos:

1. Conectamos la cámara al PC, lanzar la aplicación NXTCamView y pulsamos el botón “Connect”.

2. En este punto ya tenemos la cámara conectada al PC y podemos tomar capturas, así que enfocamos la cámara a la zona o color de interés y presionamos el botón “Capture”.

3. Sobre la imagen obtenida, seleccionamos el color que queremos que la cámara detecte y ajustamos los umbrales a nuestro antojo, permitiendo así un mayor margen de detección (por si cambian variables externas como la luz ambiente, etc.).

4. Una vez establecidos los colores, y con el fin de que la cámara sea capaz de obtener la información de dicho color, guardamos el color en cuestión presionando el botón "Upload".

5. Para comprobar el correcto funcionamiento podemos lanzar la opción de "Tracking" y ver como detecta el color programado. En la siguiente imagen se muestra una captura de la aplicación.

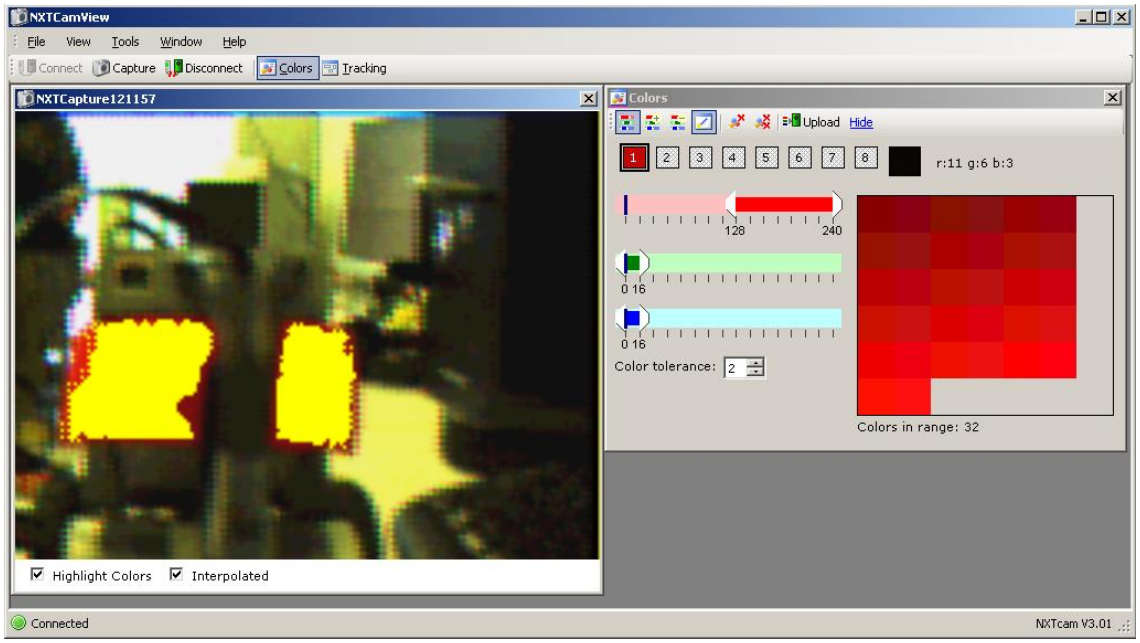


Figura 77: Aplicación NXTCamView

Una vez configurada la cámara explicaremos la estrategia que se ha seguido. El robot seguidor es capaz de seguir un determinado robot (con el correspondiente señuelo rojo) adaptando su velocidad a la del robot guía, aumentando la velocidad si detecta que se aleja el señuelo, deteniéndose si se detiene el señuelo y realizando giros para evitar perder el señuelo.

La cámara se ha instalado inclinada hacia delante, con lo que, después de obtener el centroide de la marca, tenemos los siguientes casos:

a) La marca se detecta en la parte superior de la imagen, lo que indica que se está alejando y que conviene aumentar la velocidad de las ruedas en función de la distancia.

b) Si la marca se detecta en la parte inferior de la imagen significará que el robot esta cerca, con lo que conviene reducir la velocidad o incluso detener los motores.

c) Si el señuelo se desplaza a uno de los lados será necesario modificar la velocidad de las ruedas para realizar giros, a mayor velocidad de giro cuanto mayor sea el desplazamiento lateral de la marca.

Podemos tener simultáneamente dos casos, por ejemplo, que la marca aparece en una esquina superior, con lo que será necesario realizar un giro pero aumentando también la velocidad lineal para no perder el robot predecesor.

4.6.3.6. Algoritmos de evasión de obstáculos

Los algoritmos de evasión de obstáculos se encargan de cambiar la trayectoria del robot en base a la información que reciben de sus sensores durante el desplazamiento y con el fin de evitar posibles colisiones. La respuesta del robot en este aspecto debe ser tomar una decisión en función de la lectura de sus sensores y de la posición destino. En esta aplicación se han implementado dos algoritmos de evasión de obstáculos, pudiendo el robot líder elegir entre cualquiera de ellos dos. En concreto, los algoritmos que se han desarrollado son:

-Bug2: Es el algoritmo que más ampliamente se utiliza en el campo de la robótica móvil. Representa una técnica en la que solo la lectura actual de los sensores afecta a la hora de tomar decisiones, con lo que su estrategia se basa en detectar un obstáculo para entonces bordear su contorno. El algoritmo finaliza inmediatamente cuando es posible moverse directamente a la posición destino sin producir una colisión, es decir, cuando el robot se encuentra orientado hacia su posición destino de nuevo y ya no hay ningún obstáculo que impida su circulación. En la siguiente imagen podemos observar cual sería la trayectoria a seguir con un obstáculo dado:

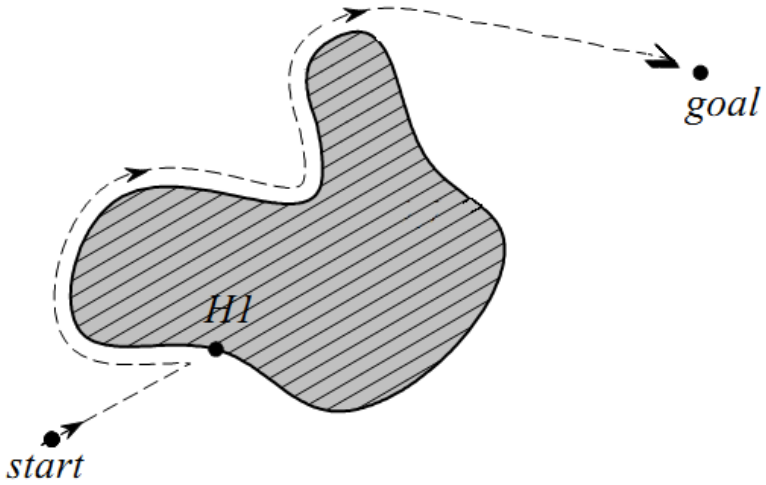


Figura 78: Algoritmo de evasión Bug2

-**Vector field histogram (VFH):** Algoritmo desarrollado por J.Borenstein y Y.Koren. El principal inconveniente del algoritmo anterior es que toda decisiones a cada instante teniendo en cuenta únicamente las últimas lecturas de sus sensores. El algoritmo VHF soluciona este problema creando una especie de mapa local del entorno del robot. Su funcionamiento se basa en construir un histograma a partir de los datos que lee de sus sensores, calculando su dirección a partir de esta información. En concreto, se aplica una función de coste para cada una de las opciones, eligiendo siempre la de menor coste. La función de coste es la siguiente:

G= a*direcciónObjetivo + b*orientaciónActualRobot + c* orientaciónAnteriorRobot

En la formula, el término 'a' toma valores mucho mayores que el resto, con lo que se busca minimizar los cambios de trayectoria y alcanzar el destino por la ruta más corta. En nuestro caso, los valores de a, b y c eran 10, 2 y 2 respectivamente.

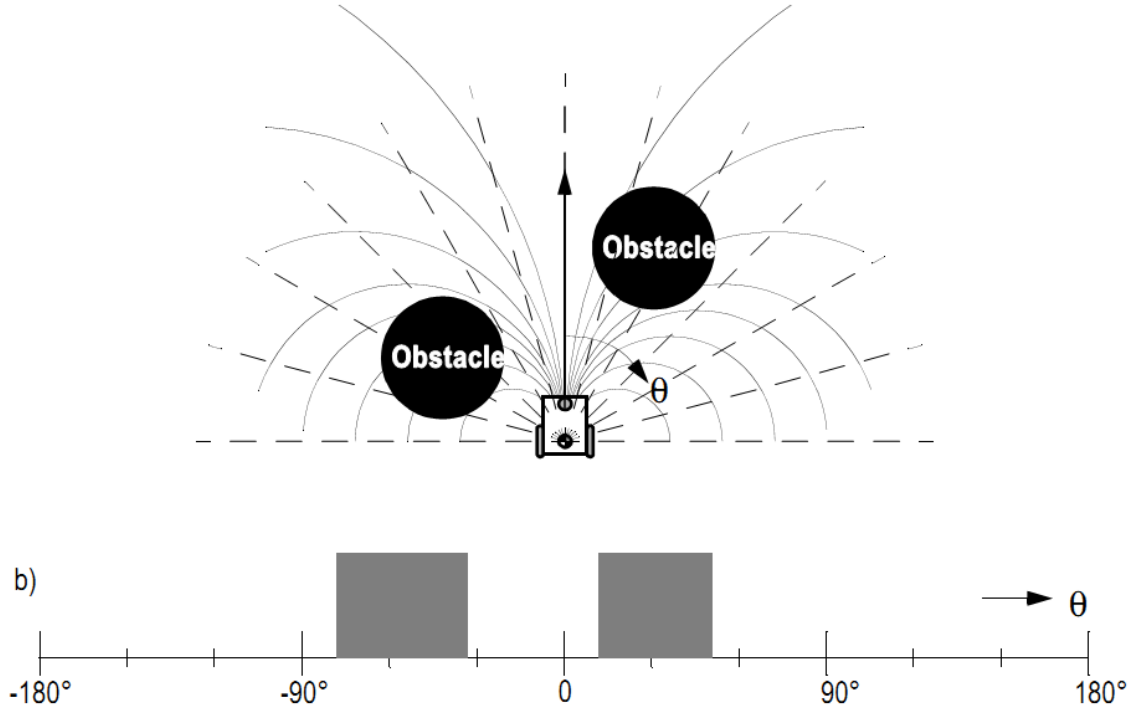


Figura 79: Algoritmo de evasión VFH

4.6.3.7. División de convoyes

Una de las características que presenta la aplicación es la posibilidad de dividir un determinado convoy durante su desplazamiento si aparece un convoy con una prioridad mayor. Esta condición se hace verdadera cuando un líder necesita crear un convoy y no hay suficientes robots seguidores en estado de espera, con lo que necesitará solicitar robots a otro convoy que, aunque se ha creado con anterioridad a este, tiene una prioridad menor. La negociación de los robots que forman un determinado convoy es realizada por ambos líderes siguiendo el siguiente patrón:

1. El nuevo líder pregunta a todos los convoyes su longitud y prioridad, para así poder seleccionar a uno de ellos, siendo el elegido aquel que tenga los suficientes robots y una prioridad menor.

2. Una vez seleccionado, informa al líder de dicho convoy la cantidad de robots que necesita.

3. Este envía un mensaje a cada uno de los robots que deben abandonar el convoy para que dejen de ejecutar el comportamiento de seguidores. Así mismo, solicita las coordenadas al último de ellos.

4. Informa al nuevo líder de las coordenadas del último robot que forma el convoy para que pueda ir a buscarlos.

5. El nuevo líder calcula a que coordenada debe acudir para posicionarse justo detrás del último robot (a unos 25 cm de distancia más o menos) y, una vez la ha alcanzado, avisa al robot líder de ello. Para obtener la coordenada exacta a la que acudir el agente calcula, en función de la orientación y la posición del último robot del convoy, su posición de acuerdo a la siguiente fórmula:

   *incX = Math.sin (angulo * Math.PI / 180) * 25 ; //ángulo en radianes*

   *incY = Math.cos (angulo * Math.PI / 180) * 25 ;*

   xDestino = xUltimoRobot - incX ;

   yDestino = yUltimoRobot - incY;

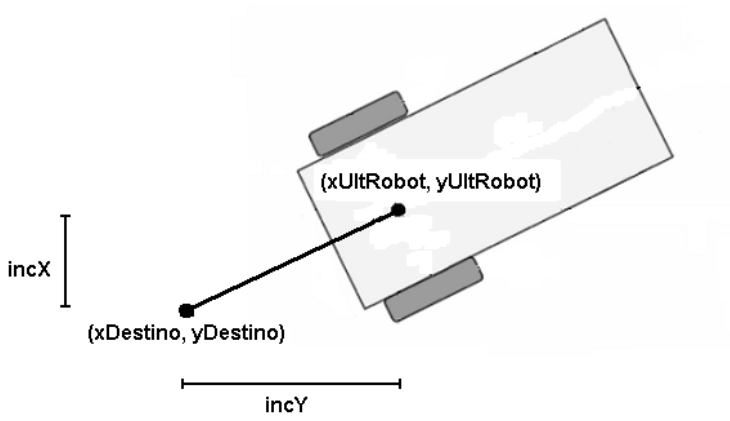


Figura 80: Cálculo de la posición destino

6. El antiguo robot líder informa a los robots que deben abandonar el convoy que deben realizar un giro de 180º para así pertenecer al nuevo convoy. Estos, cuando han acabado de dar el giro y están preparados, avisan al viejo líder y ejecutan de nuevo el comportamiento de seguidores.

7. Una vez todos los robots están preparados el viejo líder informa al nuevo de que está todo listo, con lo que los dos empiezan a moverse en dirección a sus coordenadas objetivo.

8. Cada uno de los líderes actualiza su información acerca de los robots que forman el convoy.

A continuación podemos ver una serie de capturas que ilustran este proceso.

Se ha resuelto también la situación en la que puede aparecer un obstáculo en el trayecto de un robot líder mientras éste se desplaza a un convoy menos prioritario para robar robots. En este caso, el algoritmo de evasión de obstáculos utilizado es el mismo. Sin embargo, es necesario comprobar que el obstáculo que se detecta no sea el propio robot que se está intentando alcanzar. Para evitar ejecutar el algoritmo de evasión de obstáculos en esta situación lo que hace el robot es comprobar cual es su distancia a la posición destino mediante la siguiente formula:

$$Distancia = \sqrt{ (xActual - xDestino)^2 + (yActual - yDestino)^2 }$$

Si la distancia es menor de 30 cm indicará que esta cerca de su posición destino, con lo que el objeto que se está detectando es el propio robot que se quiere que forme parte del convoy, con lo que en este caso no se lanzaría el comportamiento de evasión de obstáculos. De lo contrario, significaría que se ha detectado un obstáculo que conviene ser evadido.

4.6.3.8. Abandono del convoy por batería agotada

Actualmente, la duración de las baterías es uno de los principales problemas que padecen todos los dispositivos, ya sean móviles, portátiles, reproductores de MP3 o robots. Este fenómeno ha desembocado en un reto de la ciencia actual y que consiste en prolongar la duración de dichas baterías el máximo tiempo posible.

Este problema tampoco es ajeno a los robots Lego NXT. Por este motivo, el problema que hemos abordado contempla posibles problemas que pueden aparecer durante la ejecución de la aplicación, proporcionando una solución. En concreto, se ha resuelto el problema que puede aparecer si un robot seguidor se queda sin batería durante su pertenencia a un convoy, ya que no solo él dejaría de seguir al convoy, sino que todos los robots que estuviesen detrás suyo perderían el convoy. En este caso todos los robots posteriores al que agota su batería se detendrían detrás de éste mientras el líder desconocería por completo esta situación.

Para solucionar esta situación, cada uno de los robots está monitorizando (a intervalos regulares de tiempo) cual es el estado de su batería. Si se da la situación de que un robot posee poca carga se pondría en contacto con su líder para abandonar el convoy y seguir con el resto de robots. El proceso de comunicación que se seguiría en este caso es el siguiente:

1. El robot con problemas de batería informa a su líder de su estado de baja carga.

2. El líder detiene el convoy y envía un mensaje de finalización de comportamiento de seguidor al robot inmediatamente posterior al que no tiene batería.

3. El líder envía un mensaje de confirmación al robot con baterías agotadas de que puede abandonar el convoy.

4. Una vez abandonado el convoy, el robot líder le comunica al robot posterior al que no tenía batería que ejecute de nuevo el comportamiento seguidor, con lo que avanzará hasta la posición del anterior robot.

5. Una vez formado de nuevo el convoy el robot líder sigue desplazándose hacia sus coordenadas objetivo.

4.7. Problemas encontrados y soluciones

A continuación detallaremos los problemas que se han encontrado a lo largo de la implementación y las soluciones que se han llevado a cabo para solventar dichas situaciones.

4.7.1. Problemas por luz ambiental

El principal inconveniente que se ha encontrado al realizar la implementación de los robots limpiadores ha sido el comportamiento de la cámara cenital. En concreto, dependiendo de las condiciones de luz, se modificaba la respuesta de la cámara pudiendo no detectar alguna de los estados. Los principales casos que se han dado es la no detección de una mancha (al tratarse de objetos pequeños es más fácil perder su posición) y la detección errónea de alguno de los robots, devolviendo una orientación que no es la real y causando un mal comportamiento.

La posición de los tubos de luz artificial y las ventanas que dejan pasar gran cantidad de luz ambiental han sido las principales culpables de dichas situaciones.

Una de las soluciones era modificar el algoritmo para que trabajara con dos capturas y calculara la diferencia absoluta entre ellas. Una de las imágenes se tomaría antes de colocar los robots en el escenario, la cual se almacenaría internamente. La otra captura sería la que se captura constantemente mientras los robots están trabajando. El método se basa en restar la diferencia entre ambas imágenes, con lo que, en la imagen resultado de dicha resta, únicamente tendríamos la información correspondiente a los robots y las manchas. Así, se eliminan los problemas de las luces, ya que se restarían con la primera imagen y el algoritmo funcionaría correctamente en cualquier situación.

OpenCV dispone de una función llamada *AbsDiff()* la cual, si se le proporcionan las dos imágenes, devuelve la diferencia absoluta entre ellas. Con este procedimiento se conseguirían eliminar los posibles errores por luces y se conseguiría un algoritmo mucho más robusto. En la siguiente imagen podemos ver una captura de dicho funcionamiento:

Figura 81: Ejemplo de función AbsDiff()

4.7.2. Conexión Bluetooth y Threads

Uno de los principales problemas encontrados durante la implementación de la aplicación ha sido la compartición de la conexión Bluetooth por los distintos Threads, tanto en el software que se ejecuta en los agentes como en el que se ejecuta en los robots. En concreto, cuando dos o más hilos de ejecución están escuchando simultáneamente la interfaz Bluetooth a la espera de una determinada acción, solo uno de los hilos recibe la información. En concreto, solo el hilo que ha comenzado a escuchar antes en la tubería recibirá la información. Esta información, al no ser la esperada por un hilo en concreto se descartaría, con lo que el hilo que realmente esta esperando para la recepción de esta nunca la recibiría, siendo esto uno de los motivos de los errores que han ocurrido a lo largo del proyecto.

En concreto, en un mismo instante de tiempo, uno de los robots puede estar escuchando por su interfaz Bluetooth información con distintos fines y en distintos hilos de ejecución, como puede ser:

- Escucha de la orden de detención de división del robot por la aparición de un convoy más prioritario que este.
- La escucha de la orden de detención por el abandono de un robot por agotamiento de baterías.
- Recepción de un nuevo paquete con las coordenadas a seguir.

La solución que se ha llevado a cabo consiste en la implementación de un objeto donde almacenar la información y un par de hilos, uno para la recepción de datos y otro para el envío. El hilo de recepción se encarga de estar constantemente escuchando la tubería de llegada de información y almacenarla en la cola

perteneciente. Por su parte, el hilo de envío se encarga de comprobar si la cola no esta vacía, para entonces obtener el valor y enviarlo por la interfaz Bluetooth.

Por su parte, el hilo de ejecución que necesitaba recibir algún tipo de dato lo que hace es consultar al objeto que almacena toda la información recibida. Si encuentra información destinada a él obtiene su valor y la elimina de la cola de recepción, para que no sea utilizada por otro hilo de ejecución erróneamente.

5.- CONCLUSIONES

A la finalización del presente proyecto se han conseguido alcanzar todos los objetivos que se habían propuesto y los cuales se describen en el primer apartado de la memoria. A través de este desarrollo se ha logrado obtener los modelos holónicos necesarios para el control de organizaciones de robots, realizando tareas de forma sincronizada y colaborativa. Esta organización es capaz de reorganizarse en tiempo de ejecución para obtener de la mejor forma posible los objetivos que han sido establecidos, todo ello mediante la ejecución de los comportamientos y del intercambio de mensajes necesarios.

Además, se ha documentado toda la información necesaria para poder ejecutar una plataforma agente, facilitando la futura ampliación de algunas de las aplicaciones implementadas. También se ha especificado el uso de la interfaz de administración de JADE, a partir de la cual podemos controlar nuestras plataformas, inicializar o detener agentes, consultar los posibles estados y servicios y se han descrito los pasos necesarios para realizar tanto la depuración como la distribución de una aplicación basada en agentes.

La utilización de la plataforma JADE en el ámbito de la robótica proporciona numerosas ventajas. Entre las más importantes tenemos:

- Facilita la colaboración y autonomía de robots en un determinado entorno, permitiendo que se adapten al medio en el que trabajan a partir de los comportamientos disponibles en cada uno de los agentes.

- Permite implementar aplicaciones con un alto grado de escalabilidad y de distribución en la red. El sistema se puede ampliar fácilmente distribuyendo los agentes en varios computadores de una misma red.

- Al estar basada en el lenguaje de programación Java ofrece todas las ventajas de este lenguaje de programación. Al tratarse de un lenguaje multiplataforma podemos ejecutar nuestros agentes en cualquier sistema operativo (Windows, Linux y Mac) que disponga de la maquina virtual de Java.

Como futuros proyectos se propone la integración de otro tipo de robot móvil en la aplicación, dejando la parte de inteligencia artificial al agente Jade, el cual se programaría con JADE, y que se puede comunicar con un robot mediante distintas conexiones (Bluetooth, Wi-Fi, ZigBee, etc.). En el robot solo sería necesario reescribir las funciones que controlan el robot, teniendo en cuenta el lenguaje de programación adecuado la configuración del robot.

6.- BIBLIOGRAFÍA

[1] Oracle Java. http://www.oracle.com/technetwork/java/index.html

[2] LeJOS: Java for Lego Mindstorms. http://lejos.sourceforge.net/

[3] Wikipedia. http://es.wikipedia.org/wiki/Wikipedia:Portada

[4] OPENCV: Processing and Java Library.
http://www.ubaa.net/shared/processing/opencv/

[5] JADE. http://jade.tilab.com/

[6] Wikispaces: Programación JADE.
http://programacionjade.wikispaces.com/Comunicaci%C3%B3n

[7] La plataforma de agentes JADE. Botía, Juan A.
ants.dif.um.es/juanbot/page_files/escuelaAgentes2005jade.pdf

[8] Agents. Vaucher, Jean.
http://www.iro.umontreal.ca/~vaucher/Agents/Jade/primer4.html

[9] JADE API. http://jade.tilab.com/doc/api/

[10] Diseño y programación de algoritmos para robots móviles. Soriano, Ángel.

[11] Una metodología multi agente para sistemas holónicos de fabricación. Giret, Adriana.

[12] An Introduction to Software Agents. Bradhsaw, J. M.

[13] Multiagent Systems. Weiss, G.

[14] Historia de la robótica: de Arquitas de Tarento al robot Da Vinci.

Sánchez Martín, F.M., Millán Rodríguez, F., Salvador Bayarri, J., Palou Redorta, J.,

Rodríguez Escovar, F., Esquena Fernández, S., Villavicencio Mavrich, H.

[15] MindSensors. http://www.mindsensors.com

[16] leJOS Forums, Java for LEGO Mindstorms. http://lejos.sourceforge.net/forum/

[17] Developing Multi-Agent Systems with JADE. Luigi Bellifemine, F., Caire, G., Greenwood, D. ISBN: 978-0-470-05747-6

[18] Desarrollo de agentes software. Fernandez, C., Gómez, J., Pavón, J. www.fdi.ucm.es/profesor/jpavon/doctorado/sma.pdf

[19] Software Agents: An overview. Hyacinth, N. http://agents.umbc.edu/introduction/ao/

[20] Running JADE under Eclipse. http://wrjih.wordpress.com/2008/11/29/running-jade-under-eclipse/

[21] Robots industriales. http://cfievalladolid2.net/tecno/cyr_01/robotica/industrial.htm

[22] Las generaciones de la robotica y la informática industrial. http://orbita.starmedia.com/~eniak/genera.htm

[23] Introduction to autonomous mobile robots. Siegwart, R., NourBakhsh, I.,

[24] Enfoque holónico basado en agentes para el control de organizaciones de robots móviles. Cervera, A., Soriano, A., Gómez, J., Valera, A., Valles, M., Giret, A. Jornadas de Automática 2011, Sevilla.

[25] Application and evaluation of Lego NXT tool for Mobile Robot Control. Valera, A., Vallés, M., Marín, L., Soriano, A., Cervera, A. International Federation of Automatic Control. Congreso Milan, Italia, 2011.

Anexo A. Programación de agentes con comportamientos

En este anexo se explicará detalladamente cual es el procedimiento a seguir para implementar un agente que ejecuta un comportamiento con el código que nosotros necesitemos.

Para implementar un determinado agente en la plataforma JADE es necesario crear una clase de Java que extienda de la clase *jade.core.Agent* y darle código al método *setup(),* el cual se ejecutará al lanzar nuestro agente. A continuación podemos ver un ejemplo:

```
import jade.core.Agent;

public class demo extends Agent{

    protected void setup(){

        //Aqui introducimos el código que debe ejecutar nuestro agente

        System.out.println("Hello World!");

    }

}
```

Este ejemplo mostrará por pantalla el mensaje "Hello World" y terminará su ejecución.

Un agente debe ser capaz de tener las capacidades necesarias para responder a los posibles eventos externos que aparecen. Si queremos que nuestro agente ejecute un determinado comportamiento debemos primero darle código. Para ello, dentro de la clase donde hemos implementado el método *Setup(),* creamos una nueva clase, la cual en este caso deberá extender a la clase que implementa el tipo de comportamiento que necesitamos. Si queremos un comportamiento que solo se ejecute una vez debemos extender de la clase *jade.core.behaviours.OneShotBehaviour*. Si por el contrario queremos que se ejecute de forma cíclica extenderíamos de la clase *jade.core.behaviours.CyclicBehaviour.* Cuando se invoca un determinado comportamiento se ejecuta se método *action(),* por lo que es aquí donde debemos incluir nuestro código. A continuación mostramos un ejemplo de este desarrollo:

```
private class ComportamientoEjemplo extends OneShotBehaviour{

    public void action(){

        //Aquí el código del comportamiento

        System.out.println("Soy un comportamiento!");
```

```
    }
}
```

Éste comportamiento mostraría por pantalla el mensaje “Soy un comportamiento!”. Para añadir este comportamiento a la pila de comportamientos de un determinado agente debemos llamar al método *addBehaviour()* de la siguiente forma:

```
addBehavior(new ComportamientoEjemplo());
```

El planificador de comportamientos sigue un algoritmo Round-Robin, ejecutando el primer comportamiento de la cola y siguiendo hasta llegar al último. Si por ejemplo añadimos esta ultima instrucción al método *setup()* del agente que hemos definido más arriba y lo ejecutamos lo que obtendríamos por pantalla sería lo siguiente:

Hello World!

Soy un comportamiento!

Si, en lugar de extender de la clase *OneShotBehaviour* extendemos el comportamiento de la clase *CyclicBehaviour* el comportamiento se añade a la pila de comportamientos automáticamente cada vez que finaliza su ejecución, por lo que en nuestro caso lo que obtendríamos por pantalla sería lo siguiente:

Hello World!

Soy un comportamiento!

Soy un comportamiento!

Soy un comportamiento!

Soy un comportamiento!

…

También se pueden eliminar comportamientos de la cola de comportamientos con el método *removeBehaviour()*. Como se puede observar este comportamiento se ejecuta en el mismo hilo que el método *setup()*, por lo que hasta que no finalice la ejecución de éste no se empieza a ejecutar el código del comportamiento. Si, por el contrario, lo que queremos es ejecutar un comportamiento de forma concurrente, será necesario que este sea invocado en un nuevo hilo. Para ello, este comportamiento deberá hacer uso de la clase *jade.core.behaviours.ThreadedBehaviourFactory* e invocar al método *wrap()*, el cual se encarga de lanzar el comportamiento. Para ejecutar el comportamiento anterior en un nuevo *Thread* hacemos lo siguiente:

private ThreadedBehaviourFactory hiloComportamiento = new ThreadedBehaviourFactory();

addBehaviour(hiloComportamiento.wrap(new ComportamientoEjemplo()));

De esta forma, se creará un nuevo hilo de ejecución para ejecutar el comportamiento ComportamientoEjemplo.

Estos comportamientos son los que más se han utilizado a lo largo del proyecto. Si lo necesitamos podemos encontrar más información acerca de la invocación de distintos tipos de comportamientos en la guía de programación avanzada de JADE, la cual se descarga directamente desde el siguiente enlace:

http://jade.tilab.com/doc/programmersguide.pdf

Para el lanzamiento de cada uno de nuestros agentes podemos utilizar dos métodos, desde la interfaz de administración (como se explica en la sección 4.2.1) o mediante código Java. El código que tenemos que utilizar en este segundo caso, y que se complementa con la sección 4.1.5.2 (donde explica el lanzamiento de agentes mediante Eclipse), es el siguiente:

```
AgentContainer aContainer = getContainerController();

AgentController aController;

try {

        String sName = "";

        synchronized(this) {

/*Establecemos el nombre de nuestro agente en la variable sName*/

                sName="L0";

        }

/*Mediante el array param podemos proporcionarle argumentos a nuestro agente en el momento de su creación. Estos argumentos serán accedidos desde el agente mediante el método getArguments()*/

        Object[] param = {"A87", "00:16:53:0c:10:63", 0, 0, 0};

/*La función createNewAgent() se encarga de crear un agente con los datos que se requieren (nombre del agente, clase que implementa el agente y parámetros). Este agente se lanza desde el método start(), el cual ejecuta el código del método setup() de nuestro agente.*/
```

```
        aController = aContainer.createNewAgent(sName, "src.AgenteLider", param);
        aController.start();
}
catch(jade.wrapper.StaleProxyException e){
        e.printStackTrace();
}
```

Anexo B. Programación y utilización del sensor NXTCam

El objetivo del presente anexo es explicar el funcionamiento del sensor NXTCam. El firmware LejOS proporciona la librería *lejos.nxt.addon.NXTCam* para poder controlar el sensor a través de sus métodos y que podemos encontrar en el siguiente enlace a su API:

http://lejos.sourceforge.net/nxt/pc/api/index.html

En concreto, los métodos que más nos interesan para realizar el procesado de los datos obtenidos son los siguientes:

- *NXTCam(SensorPort)*: Creamos un objeto de tipo NXTCam con el que controlar nuestro sensor cámara indicando el puerto en el que ha sido conectada.
- *sendCommand(char cmd)*: Este método nos permite enviar distintos comandos (inicialización, activación, etc.) a nuestro sensor.
- *getNumberOfObjects()*: Devuelve el número total de objetos detectados, independientemente del color que sean.
- *getObjectColor(int id)*: Indica el color del objeto que se le pasa como argumento, devolviendo el índice de uno de los colores que se le ha programado.
- *getRectangle(int id)*: devuelve un objeto de tipo *java.awt.Rectangle* con las medidas y coordenadas del objeto que se le pasa como argumento.

La resolución de las capturas que nos proporciona la cámara son 170x143, situándose la coordenada 0,0 en la esquina superior izquierda como se indica en la siguiente figura:

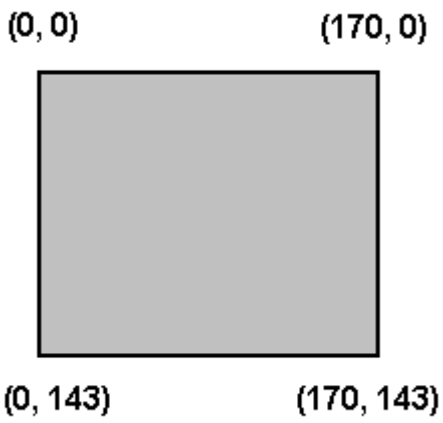


Figura A1: Resolución NXTCam

El código que se ha implementado para la persecución del convoy se resume en obtener a cada iteración la mancha de mayor área, y en función de su centroide, acelerar, frenar o girar como ya ha explicado en la sección 4.6.3.5. El código que se ha utilizado en la aplicación de generación de convoyes es el siguiente:

```
/*Creamos el objeto cámara y lo incializamos.*/
    NXTCam cam=new NXTCam(SensorPort.S1);
    cam.sendCommand('A');
    cam.sendCommand('E');

    int increment, numObjetos, x, y, velDerecha, velIzquierda;
    while(true){
    /*Obtenemos el número de manchas que hay en la captura actual.*/
        numObjetos=cam.getNumberOfObjects();
        if(numObjetos>0){
    /*Si se detectan manchas obtenemos la de mayor tamaño con el método obtenerArea y nos guardamos la referencia.*/
            rectMax=cam.getRectangle(0);
            for(int i = 1; i<numObjetos; i++){
                rect = cam.getRectangle(i);
                if(obtenerArea(rect)>obtenerArea(rectMax))
                rectMax=rect;
            }
    /*Para la mayor de las manchas obtenemos su centroide en las coordenadas x e y.*/
            x=(int)rectMax.getCenterX();
            y=(int)rectMax.getCenterY();
```

```
/*En función de la posición del centroide aplicamos la velocidad necesaria a
cada uno de los motores. Si la coordenada y es menor del valor 55 indica que
estamos cerca del robot, por lo que debemos detenernos.*/

        if(y<55){

            if(x>90 && x<110){ //Vamos hacia delante

/*La mancha esta centrada, el robot se desplaza en línea recta.*/

            velDerecha=300;

            velIzquierda=300;

        }

        else if(x>110){

/*La mancha esta desplazada a la derecha, por lo que, en función de su
desplazamiento, hacemos que el robot realice un giro disminuyendo la
velocidad del motor derecho.*/

            velDerecha=(int)(300-((x*10)-1100)/1.7);

            velIzquierda=300;

        }

        else if(x<90){

/*El caso contrario al anterior, giramos a la izquierda*/

            velDerecha=300;

            velIzquierda=(int)(300-((900-(x*10))/2));

        }

        incremento = (50-y)*4;

/*En función de la coordenada y calculamos la distancia a la mancha, por lo que
aumentamos la velocidad de los motores sumando el siguiente incremento.*/

        velDerecha+=incremento;

        velIzquierda+=incremento;

/*Aplicamos la velocidad al motor derecho. Si es negativa cambiamos el signo e
invertimos el sentido de giro.*/
```

```
            if(velDerecha>0){
                Motor.A.setSpeed(velDerecha);
                Motor.A.forward();
            }
            else if(velDerecha<0){
                velDerecha=-velDerecha;
                Motor.A.setSpeed(velDerecha);
                Motor.A.backward();
            }
/*El mismo caso que el anterior pero en el motor izquierdo. Si la velocidad es
positiva se aplica directamente, sino se invierte signo y sentido de rotación del
motor izquierdo*/
            if(velIzquierda>0){
                Motor.C.setSpeed(velIzquierda);
                Motor.C.forward();
            }
            else if(velIzquierda<0){
                velIzquierda=-velIzquierda;
                Motor.C.setSpeed(velIzquierda);
                Motor.C.backward();
            }
        }
/*No se detecta ninguna mancha, por lo que en este caso detenemos los
motores.*/
        else{
            Motor.A.stop();
            Motor.C.stop();
```

```
        }
    }
```