UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escuela Técnica Superior de Ingeniería Informática

Universitat Politècnica de València

Desarrollo de un simulador basado en agentes para la gestión inteligente de un entorno paisajístico

Trabajo Fin de Grado

Grado en ingeniería informática

Autor: Martin Taberner, Alberto

Tutores: Poza Luján, José Luis

Carrascosa Casamayor, Carlos

2016-2017

Resumen

Mantener un entorno de confort para una planta requiere una atención constante. Sería ideal poder contar con algún sistema automático que realizara las acciones necesarias para regular y mantener la planta en las mejores condiciones posibles. En este documento se va a realizar el diseño y un prototipo de un sistema multiagente para automatizar el cuidado.

Como evolución de un proyecto anterior donde se disponía de un robot con movilidad, actuadores y sensores para el cuidado de la planta, el sistema multiagente tomará el control de dicho robot para que, a través de un sistema de razonamiento basado en casos, tome las decisiones oportunas, para optimizar su cuidado.

Además, el sistema proporcionará un simulador que creará un entorno de realidad aumentada para los robots. Así como una comunicación bidireccional con una interfaz de usuario. Esta interfaz permitirá un cierto control sobre la movilidad del robot.

Palabras clave: Agente, Artefacto, Jardín, Simulación, Plantas, Chombo, Prototipo.

Abstract

Taking care of the comfort environment for a plant requires constant attention. It would be ideal to have some automatic system that perform the necessary actions to keep and regulate the best conditions for the plant. In this document will be done a design and a prototype of a multi-agent system to automatize the assistance.

As an evolution of an elder project where a robot with mobility, sensors and actuators was made to take care of the plant, the multi-agent system will take control of the mentioned robot, using a Case-based Reasoning to make the proper decisions and optimize the assistance.

Besides, a simulator will be provided that creates an augmented reality environment for the robots. As well as a bidirectional communication with a user interface. That will allow some control of the robot's mobility.

Keywords: Agent, Artefact, Garden, Simulation, Plant, Chombo, Prototype.

Resum

Mantenir un entorn de confort per a una planta requereix una atenció constant. Seria ideal poder comptar amb algun sistema automàtic que realitzarà les accions necessàries per a regular i mantenir la planta en les millors condicions possibles. En aquest document es va a realitzar el disseny i un prototip d'un sistema multi-agent per a automatitzar la cura.

Com a evolució d'un projecte anterior on es disposava d'un robot amb mobilitat, actuadors i sensors per a la cura de la planta, el sistema multi-agent prendrà el control d'aquest robot perquè, a través d'un sistema de raonament basat en casos, prenga les decisions oportunes, per a optimitzar la seua cura.

A més, el sistema proporcionarà un simulador que crearà un entorn de realitat augmentada per als robots. Així com una comunicació bidireccional amb una interfície d'usuari. Aquesta interfície permetrà un cert control sobre la mobilitat del robot.

Paraules clau: Agent, Artefacte, Jardí, Simulació, Planta, Chombo, Prototip.

Tabla de contenido

1. Introducción

1.1 Motivación

Según la RAE el paisajismo es el “estudio o diseño del entorno natural, especialmente de parques y jardines”. Siendo una actividad muy relacionada con el cultivo, aunque desde un punto de vista más estético. Las primeras presencias de esta actividad se remontan a la antigua Babilonia, con unos de las grandes maravillas del mundo antiguo como son los jardines colgantes.

En la actualidad es una actividad que está presente desde los jardines privados, hasta parques públicos. Y aun siendo tan extendida, el progreso en este campo continúa estancado, manteniendo procesos arcaicos en el cuidado de las plantas.

Las plantas son seres vivos que dependen del entorno para su supervivencia, dado que generalmente no tienen habilidad motriz.

Sería posible mejorar la capacidad de supervivencia de las plantas si se implementara un sistema autónomo que reaccionara de forma inmediata a sus necesidades, mediante acciones como pueden ser: dotarlas de movimiento, riego y fertilizante automáticos, entre otras.

Desde el ámbito de la ingeniería informática consideramos que existen herramientas que podrían dotar a las plantas de los mecanismos anteriores. De forma que se reducirá el trabajo de mantenimiento y la mortalidad de las plantas. Simplificando así la complejidad de la actividad paisajística.

1.2 Objetivos

La finalidad principal de este trabajo es el desarrollo, diseño e integración de diversas tecnologías a nuestro alcance para permitir la supervivencia de las plantas en diversos entornos y circunstancias. Y para poder llevar a cabo un objetivo tan difícil de abarcar, este proyecto se estructurará en cuatro proyectos diferenciados, los cuales deberán coordinarse para realizar cada uno sus propios sub-objetivos sin perder la visión del general.
Siendo los participantes de cada sub-proyecto:

Nombre	Ramón Ferrer Mestre
Rol	Desarrollador del sistema de persistencia y comunicaciones
Objetivo	Implementar la persistencia

Nombre	Sergio Rives Estellés
Rol	Desarrollador de la interfaz de usuario (IU) y el *render*
Objetivo	Implementación en Android de la IU

Nombre	Alberto Martín Taberner
Rol	Desarrollador del sistema multi-agente y *manager* (simulador)
Objetivo	Implementar el sistema de toma de decisiones

Nombre	Salva Pons
Rol	Desarrollador *hardware*
Objetivo	Implementar funciones del Arduino y montar el hardware

En este proyecto nos centraremos en el ámbito de la toma de decisiones a partir de un sistema multiagente, que permite una automatización predictiva del cuidado de las plantas.

Los sub-objetivos a desarrollar en este proyecto son los siguientes:

- El sistema debe ser capaz de tomar decisiones relativas a la planta que tiene a su cuidado.

- Los agentes deben ser capaces de comunicarse con la base de datos para tener un razonamiento basado en casos.

- El sistema debe poder trabajar en consonancia con el simulador. Pero sin dependencia absoluta, pudiendo realizar funciones básicas.

- Es necesario que el simulador proporcione información para el usuario de la situación real y virtual actual.

- El simulador debe permitir la comunicación bidireccional con el usuario.

2. Estudio de Mercado

2.1 Competidores potenciales/tecnologías similares

2.1.1 Parrot Pot

Maceta inteligente ya a la venta a un precio de 150 $, cuenta con 4 tipos de sensores: sensor de PH, de temperatura, de luz y de humedad. Si la planta tuviese alguna necesidad la maceta envía notificaciones vía bluetooth al móvil para poder atenderlas. Como función autónoma puede autorregularse si la planta necesita agua y está equipada con un tanque de agua a tal efecto. Funciona con 4 pilas AA que le dan una autonomía de 1 año. Utiliza la tecnología Flower Power.

Características:

- **SENSORS**
  - Capacitive soil moisture sensor (measurement range: 0 to 50%)
  - Fertiliser level sensor
  - Light sensor: 0 to 1000 µmol m-2. s-1
  - Air temperature sensor: 0°C to +55°C
  - Rain and water resistant (IPX5 rating)
- **SIZE**
  - Diameter x height: 20.5 cm x 31.2 cm
- **DETAILS**
  - Operating temperature: 0°C to 55°C
  - Material: ABS, PP, rubber
  - LED status indicator (red, green)
  - Built-in water tank: 2.2 litres
  - Soil volume: 2.4 litres
  - Batteries: 4 AA
- **CONNECTIVITY & APPS**
  - Connection via Bluetooth® Smart/Bluetooth V4.0 BLE
  - Free Parrot Flower Power app
  - Application programming interface (API)

http://www.odditycentral.com/technology/bad-with-plants-this-high-tech-flower-pot-can-keep-any-plant-alive.html

2.1.2 Flower Power

Tecnología usada por las macetas Parrot Pot, consistente en un dispositivo bluetooth adaptable a cualquier maceta de tamaño medio/grande que contiene 4 tipos de sensores: sensor de PH, de temperatura, de luz y de humedad. Gracias a una conexión con una base de datos de más de 8000 plantas, es capaz de enviar alertas al móvil por bluetooth si las condiciones de la planta no son óptimas.

Mediante su aplicación móvil es posible monitorizar el estado de una o más plantas, consultar un histórico con las condiciones de cada planta o planificar tareas a realizar para que las plantas estén en las mejores condiciones, siendo esta planificación automática.

El dispositivo funciona con una pila AAA y su precio es de 50 $.

Ilustración 2.1 Flower Pot

https://www.amazon.com/Parrot-Flower-Power-Bluetooth-dedicated/dp/B00FOM2Y6W

2.1.3 RoseRunner (2012)

Consiste en un robot capaz de moverse entre las macetas, fertilizarlas y moverlas de un punto a otro.

- Interfaz de control
- Joystick
- PC interfaz

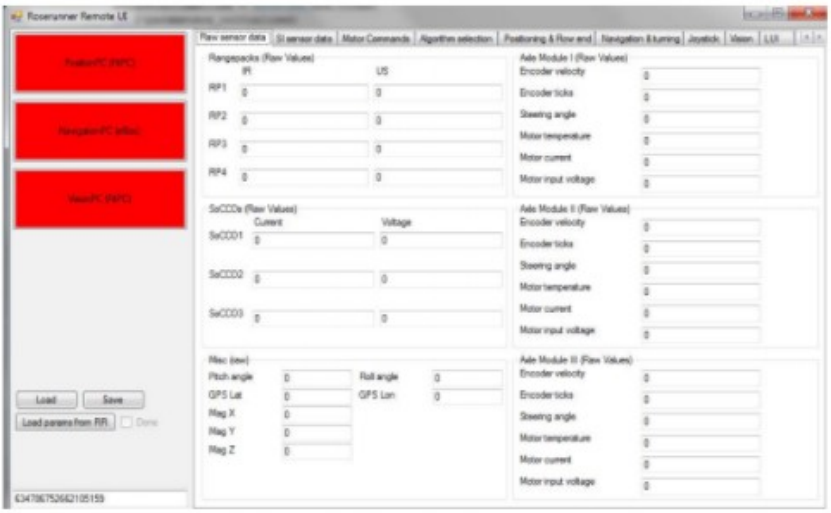

Ilustración 2.2. Rose Runner interfaz

Ilustración 2.3. Rose Runner

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.259.3801&rep=rep1&type=pdf

2.1.4 Jurema Action Plant

Máquina bio-interactiva capaz de transportar una planta sensitiva (Mimosa Púdica), intenta aumentar las funciones que puede realizar una planta habilitando que usen tecnologías similares a las que usan los humanos. Usa unas señales eléctricas que circulan a través de las células de las plantas como si de nervios humanos se tratase, y permite que la planta se aleje de un obstáculo al tocarlo, usando la propia planta y sus señales "nerviosas" como sensor. Adicionalmente dispone de un tanque de agua y una placa con los componentes electrónicos, lo cual interpreta las señales y necesidades de la planta para traducirlas en movimiento o dispensa de agua.

Ilustración 2.4. Jurema Action Plant

https://ivanhenriques.com/2011/06/02/jurema-action-plant/

2.1.5 Plantas nómadas

La Planta Nómada es un organismo vivo, constituido por un sistema robótico, una especie vegetal orgánica, un conjunto de celdas de combustible microbianas y fotovoltaicas. Para sobrevivir, este organismo toma agua contaminada y la procesa en sus celdas de combustible mediante una colonia de bacterias autóctonas de estas aguas, que se alimentan transformando los nutrientes en electricidad, para ser almacenada por su sistema de cosecha de energía. En este proceso de biodegradación mejora la calidad del agua y provee a la especie vegetal que también produce electricidad con su metabolismo. La liberación de oxígeno es el remanente de este ciclo energético. Por tanto, no solo es una especie adaptada al entorno modificado, sino que también restituye la energía que dispone de la tierra.

Ilustración 2.5. Plantas nómadas

https://youtu.be/kQwYEWaEHTs

http://www.plantasnomadas.com/

2.1.6 Fliwer

Sistema de riego inteligente para huertos, jardines y macetas. Es una tecnología que valora las necesidades de las plantas en tiempo real para proporcionar un riego adecuado, así como proporcionar información al usuario mediante Wifi o 3G sobre el estado de las mismas.

Ilustración 2.6. Fliwer

http://www.fliwer.com

2.1.7 Click & Grow Smart flowerbed

Semillero inteligente que proporciona los cuidados suficientes para que las plantas broten en 1-2 semanas, cuenta con un pequeño depósito y se encarga de que las semillas tengan las cantidades adecuadas de agua, oxígeno y nutrientes.

Utiliza 4 pilas AA y el precio oscila entre 20-60€ según el modelo.

Ilustración 2.7. Click & Grow Smart flowerbed

https://www.amazon.com/Click-Grow-flowerbed-Cockscomb-Indoor/dp/B008K95K80

2.1.8 A Control Method for a Swarm of Plant Pot Robots that Uses Artificial Potential Fields for Effective Utilization of Sunlight (CMSPR)

Es un proyecto de la universidad de Tokio de agricultura y tecnología, desarrollado por Masato Yuasa y Ikuo Mizuuchi.Que tiene como objetivo maximizar el uso de la luz solar en el cultivo de plantas. El sistema que implementan se basa en una serie de robots capaces de moverse y una serie de sensores instalados en la maceta, con el fin de monitorizar de la planta. El funcionamiento es realizado por medio de la transmisión de la información de estado de los sensores al ordenador de control, este calcula la orden y la envía a los robots para su ejecución.

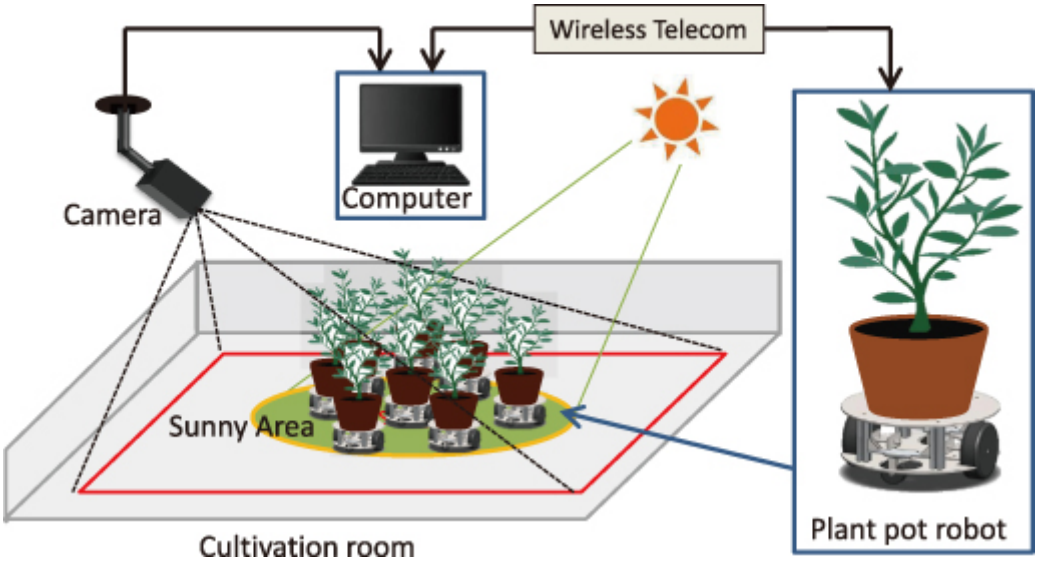

Ilustración 2.8. CMSPR

https://www.fujipress.jp/jrm/rb/robot002600040505/

2.1.9 PotPet

Este sistema tiene el objetivo de permitir a las plantas el moverse a hacia lugares soleados (sensor de luz) o detectar personas (sensor de movimiento), además puede advertir a las personas de la necesidad de riego y del punto de parada del riego mediante el sensor de humedad.

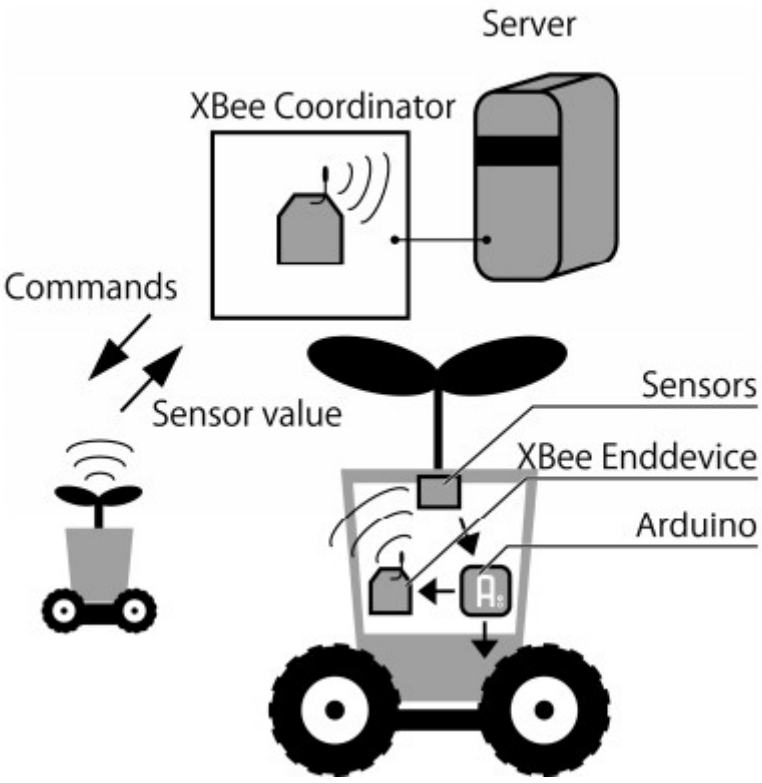

Ilustración 2.9. PotPet

http://sappari.org/pdf/potpet_tei2011.pdf

2.2 Tabla comparativa

	Parrot Pot	Flower Power	Rose Runner	Jurema Action Plant	Planta nómada	Fliwer	Click & Grow	Maceta intel.
Sensor humedad	✓	✓	✗	✓	✓	✓	✓	✓
Sensor fertilizante	✓	✓	✓	?	?	✓	✓	✓
Sensor luz solar	✓	✓	✗	?	✓	✓	✗	✓
Sensor temperatura	✓	✓	✗	?	✗	✓	✗	✓
Riego automático	✓	✗	✓	✓	✓	✓	✓	✓
Depósito agua	✓	✗	✓	✓	✗	✗	✓	✓
Movimiento	✗	✗	✓	✓	✓	✗	✗	✓
Alertas	✓	✓	✗	✗	✗	✓	✗	✓
Interior	✓	✓	✗	✓	✗	✓	✓	✓
Exterior	✓	✓	✓	✓	✓	✓	✗	✓
Simulación	✗	✗	✓	✗	✗	✓	✗	✓
Precio	150 $	50 $	?	?	?	1200 €	20-60 €	?
Disponible	✓	✓	✗	✗	✗	✓	✓	✗

Tabla 2.1. Comparativa de mercado

	Parrot Pot	Flower Power	Rose Runner	Jurema	Planta nómada	Fliwer	PotPet	CMSPR	Click & Grow	Maceta intel.
Sensor humedad	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓
Sensor fertilizante	✓	✓	✓	?	?	✓	✗	?	✓	✓
Sensor luz solar	✓	✓	✗	?	✓	✓	✓	✓	✗	✓
Sensor temperatura	✓	✓	✗	?	✗	✓	✗	✓	✗	✓
Sensor de movimiento	✗	✗	✗	✗	✗	✗	✓	?	✗	✗
Riego automático	✓	✗	✓	✓	✓	✓	✗	?	✓	✓
Depósito agua	✓	✗	✓	✓	✗	✗	✗	?	✓	✓
Movimiento	✗	✗	✓	✓	✓	✗	✓	✓	✗	✓
Alertas	✓	✓	✗	✗	✗	✓	✓	?	✗	✓
Interior	✓	✓	✗	✓	✗	✓	?	✓	✓	✓
Exterior	✓	✓	✓	✓	✓	✓	✓	?	✗	✓
Simulación	✗	✗	✓	✗	✗	✓	✗	✓	✗	✓
Disponible	✓	✓	✗	✗	✗	✓	✗	?	✓	✗

Tabla 2.2. Comparativa de mercado

2.3 Tecnologías

Para poder desarrollar un sistema multiagente que se adapte a las necesidades de nuestros objetivos se han contemplado las siguientes tecnologías, las cuales han sido recomendadas por el codirector, experto en la materia.

2.3.1 CArtAgO

CArtAgO [6] proporciona al sistema de agentes un entorno virtual, a base de "artefactos", los cuales permiten una conexión con objetos del entorno (por ejemplo, comunicación con una base de datos). Al asociarse a uno de dichos artefactos, la sincronización se hace de forma automática, permitiendo que el agente simplemente se quede a la espera de cambios en su entorno.

2.3.2 Jason

Jason [7] es un software *Open Source* que proporciona un intérprete de agentes, con el cual se puede desarrollar una plataforma multi-agente que sirve tanto para implementar agentes como para poder ejecutarlos.

2.3.3 Jade

Jade [9] es un *framework* para la implementación de sistemas multiagente. Jade sirve como infraestructura a Jason, y en nuestro caso nos va a servir además para gestionar la comunicación entre los agentes y el *manager*.

2.3.4 JaCalIVE

JaCalIVE [8] se base en el metamodelo MAM5 (Ilustración 2.10), con el cual provee un método para elaborar los IVEs mostrados en el modelo, junto con una plataforma para su ejecución. MAM5 describe entornos multiagente no sólo definidos por agentes, sino también por artefactos. Un IVE se describe entonces con un sistema de agentes, artefactos y simuladores físicos. En este proyecto es una tecnología para considerar para estructurar el sistema multiagente, así como para aplicar una simulación al movimiento de los agentes. [10] [11] [12]

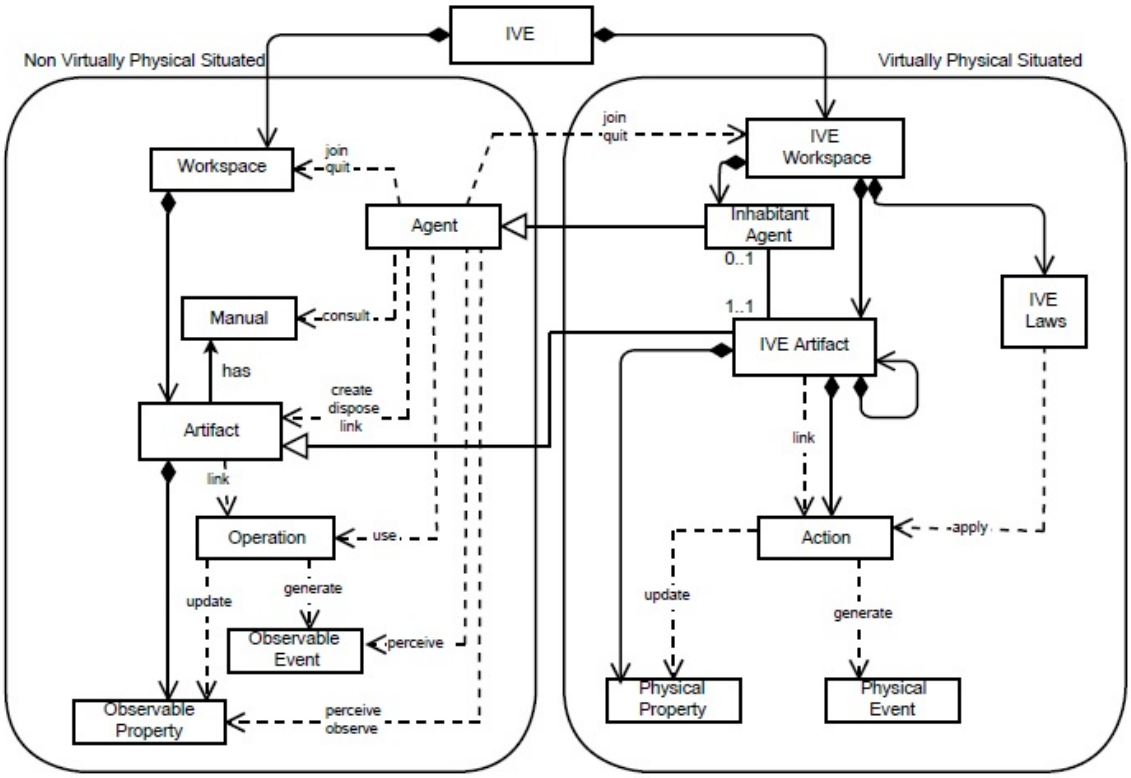

Ilustración 2.10. Metamodelo MAM5 de un IVE basado en A&A

3.1 Diseño completo

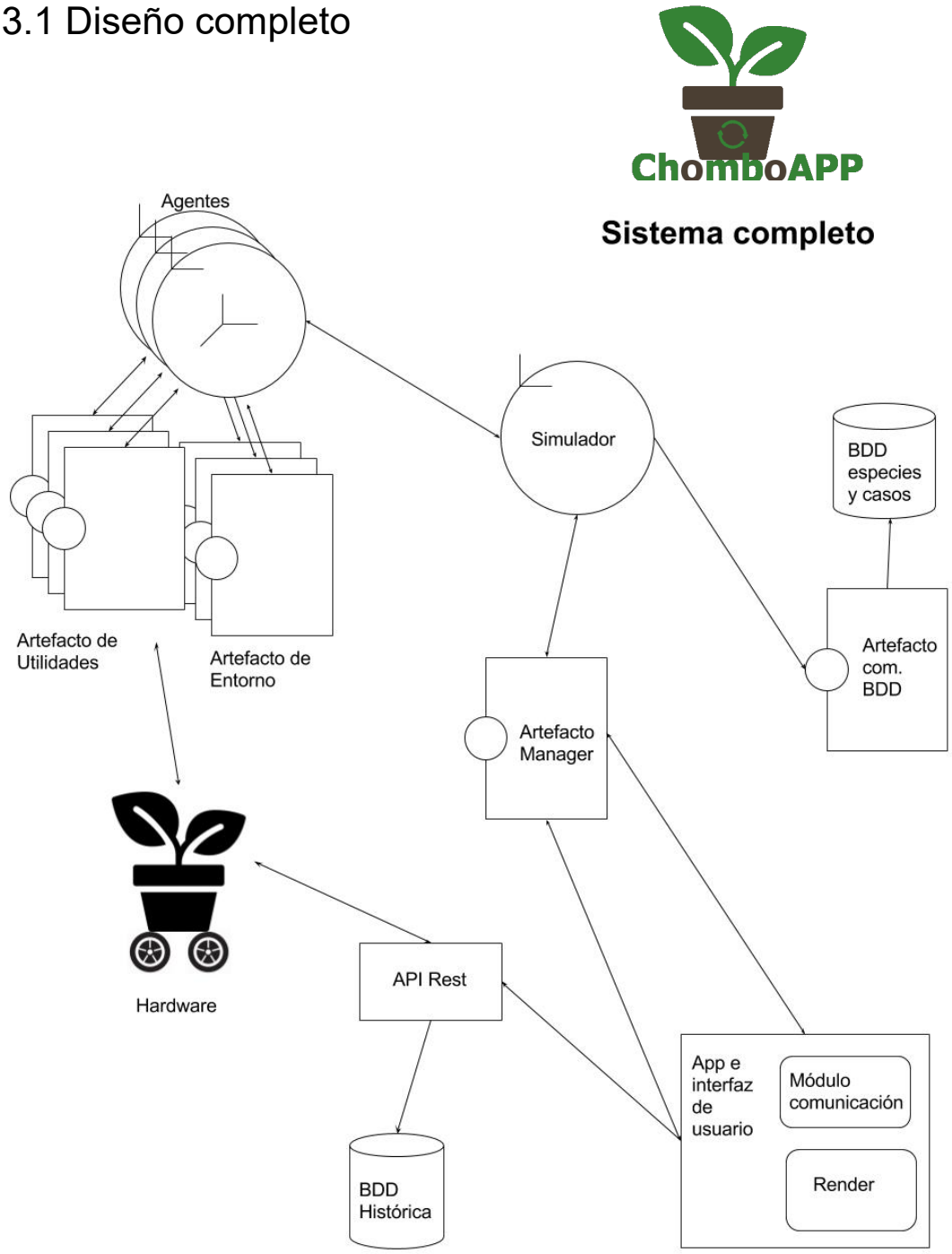

Ilustración 3.1. Diseño del sistema completo

En la imagen anterior (Ilustración 3.1), vemos la estructura del sistema que corresponde a todos los trabajos vinculados a este proyecto, a continuación, se explicará el diseño completo, dejando para más adelante el diseño de las partes tratadas en este trabajo.

A partir de las tecnologías usadas, se ha diseñado el sistema de la siguiente forma: Por una parte, disponemos del Servidor, el cual incluirá tanto el simulador (*manager*) como de la base de datos, así como del sistema multiagente virtual. Por otra parte, se dispone de un Agente en cada uno de los artefactos físicos. El simulador debe conectarse tanto a la base de datos a través de un artefacto, como a la interfaz de usuario, la cual generalmente se encontrará en dispositivos ajenos al sistema.

Se deberá comunicar con la interfaz de forma bidireccional, proporcionando al usuario la información de la situación de cada uno de los Chombos como recibiendo las diferentes acciones que el usuario decida tomar (crear nuevos Chombos o dar órdenes a los agentes).

Los tres trabajos diferenciados mencionados en el diseño son:

El diseño de la base de datos y comunicaciones, la interfaz de usuario y *render* y el trabajo que se desarrollará en esta memoria, que engloba el *manager* (Simulador) y el sistema multiagente. La parte *hardware* corresponde a otros proyectos que deberán unirse en futuros trabajos.

La comunicación entre el *Manager* y la interfaz de usuario tiene los siguientes propósitos:

- Permitir a los usuarios recibir la información de cada uno de los Chombos activos en el sistema, pudiendo ver el estado de cada planta, así como su posición y la actividad de los robots.

- Permitir a los usuarios modificar el mapa, permitiendo o negando que los Chombos puedan moverse por ciertas zonas (para delimitar su movimiento). Dichos mapas se deben adecuar a la realidad, por lo que se usarán posiciones GPS para marcar el tamaño del mismo.

- Permitir a los usuarios dar órdenes de movimiento a los Chombos, a través de puntos de encuentro que el usuario marcará en el mapa, para poder reunir a los robots en allí donde convenga, puede ser a todos de forma simultánea, o a cada uno de ellos por separado.

- Facilitar al usuario el crear nuevos Chombos, que una vez seleccionados los parámetros pertinentes (tipo de planta, características físicas del Chombo, etc.) se enviará al *Manager* para su creación en un fichero.asl (donde se diseña el agente). Se hace de forma tal y como se ve en el siguiente diagrama:

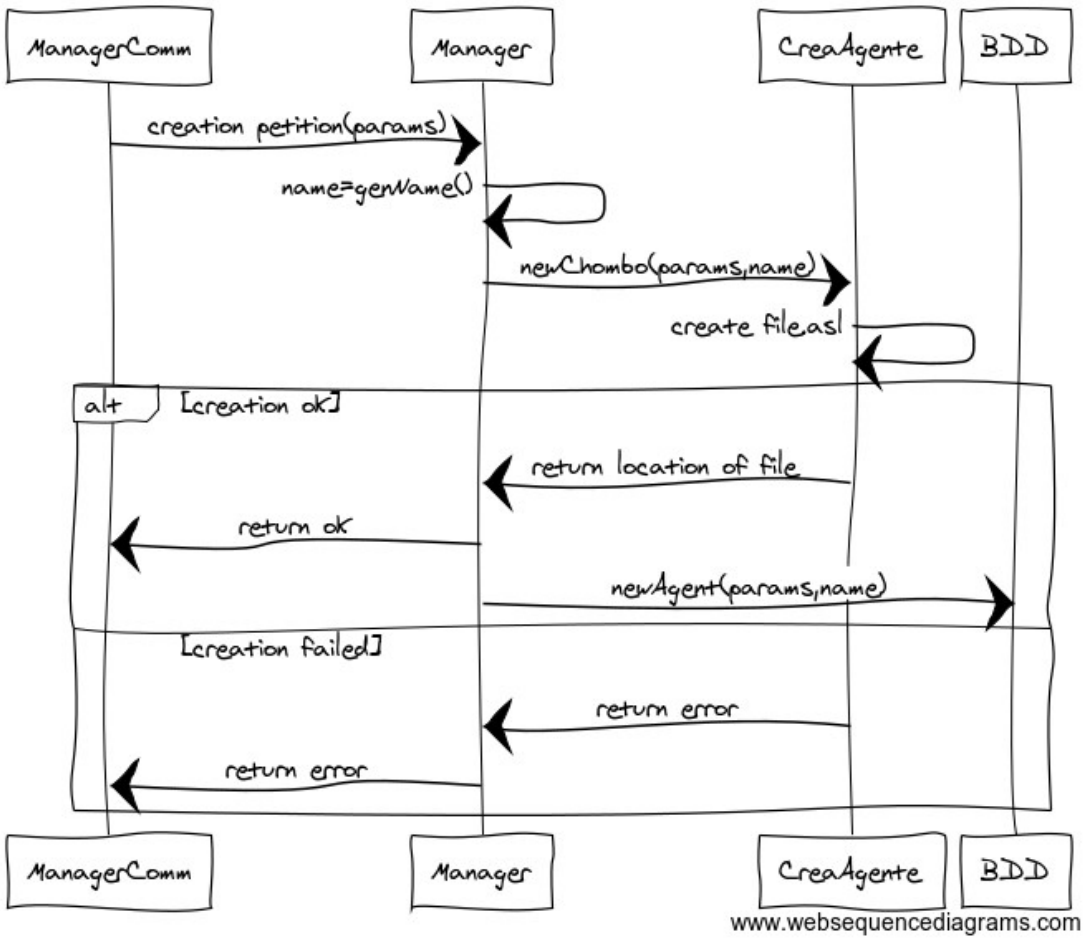

Ilustración 3.2. Diagrama de secuencia de creación de chombos

La comunicación con la base de casos está diseñada especialmente para los agentes, en la cual buscarán información de qué acciones son mejores (o peores) para el estado actual de cada agente. Cuando aparezca un caso nuevo, los agentes lo almacenarán para usos futuros.

Básicamente esto permitirá a los agentes poder tomar decisiones con conocimiento de casos pasados (y supuestamente fiables), con lo que disminuirá su tasa de fallo al tomar decisiones, además de que ampliará de forma dinámica la cantidad de casos contemplados y el conocimiento que se tiene del cuidado específico de cada planta.

3.2 Diseño de agentes y artefactos

Del sistema completo visto anteriormente, el fragmento que corresponde a este trabajo es el mostrado en la imagen a continuación:

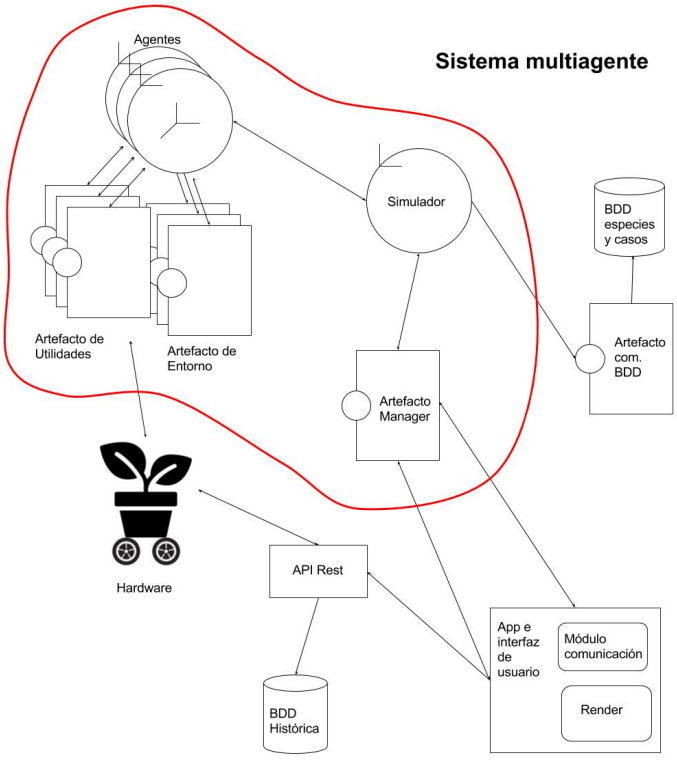

Ilustración 3.3. Especificación de las partes diseñadas en este trabajo

Ahora ahondemos en la parte en la que se centra este trabajo (Ilustración 3.3), el sistema multiagente y los artefactos que soportan.

3.2.1 Diseño de agentes

En este sistema, tal como se ve en la ilustración anterior (3.3), hay dos tipos de agentes: el agente Chombo y el Simulador (o *Manager*).

3.2.1.1 Agente Simulador / Manager

El agente simulador se encarga de gestionar el entorno virtual. Solo habrá una copia de este agente en el sistema. El agente se encargará de llevar un control de las posiciones de los distintos Chombos para evitar las colisiones. Además, como los Chombos reales reciben los datos directamente de los sensores, los virtuales requieren también que sus sensores estén simulados, de lo cual también se encarga el agente simulador, indicando a los agentes las siguientes propiedades:

- Se encarga de indicar a los agentes la cantidad de luz que hay en su entorno y donde se encuentran ellos.
- Se encarga de aumentar o reducir la cantidad de batería y agua almacenada.
- Se encarga de controlar la humedad en cada uno de los Chombos, aumentándola en caso de riego, o reduciéndola para tener consistencia con la evaporación natural.
- Su posición actual, así como el control sobre su movimiento.

El simulador dispone además de las siguientes funcionalidades:

- El simulador proporciona al sistema multiagente información de interés (como la posición de puntos de carga u obstáculos para la búsqueda de caminos eficientes).

- Como se ha mencionado, el simulador es el encargado de controlar el entorno, para ello, dispone de mapas (matrices) de la localización donde se desplieguen los artefactos físicos. En los cuales se guardan tanto las posiciones por las que pueden moverse (o las negadas ya sea por obstáculos o por el propio usuario) o de la luminosidad (la cual, con presencia de sensores reales se generan gradientes desde cada uno de los puntos sensorizados para intentar ser lo más precisa posible).

Para poder conseguir las funcionalidades antes mencionadas se requiere el artefacto *Manager* el cual se comentará más adelante, por ello el agente simulador también se encargará de crear dicho artefacto en el arranque del sistema. Hay otro artefacto que el simulador también se encarga de crear, el artefacto de comunicación con la base de datos, necesario para los agentes.

3.2.1.2 Agente Chombo

El agente Chombo se encarga de controlar el Chombo al que se esté asociado, habrá un agente por cada robot físico, así como uno por cada Chombo virtual que se haya generado. Cada agente tiene tanto capacidad reactiva como deliberativa, los propios robots disponen de sistemas de emergencia antichoque, por lo que no es algo que el agente deba tener en cuenta, en cambio sí reacciona a faltas de agua en el depósito interno, de batería y de humedad o luminosidad en la planta. Para ello es capaz de tomar las decisiones que considere oportunas para mantener las condiciones óptimas para la planta a su cuidado. Ya sea regar cuando la humedad esté baja o ir a las zonas más luminosas. Pero no solo debe cuidar la planta, sino también al robot al que esté asociado, moviéndolo, cuando sea necesario, a los puntos de carga de batería y agua, de forma que el robot siempre pueda cumplir con los requerimientos de la planta.

Para mejorar las decisiones del agente, se ha diseñado un sistema de razonamiento basado en casos, el cual sigue el siguiente esquema:

Para poder considerar que acción tomar ante las múltiples situaciones que se le pueden presentar, el agente consulta en la base de casos si ya se han tomado acciones para el estado actual, así como lo beneficiosa que resultó dicha acción tomada. De esta forma obtiene conocimiento de que debería hacer a continuación. Cuando se le indica al agente la acción anteriormente tomada puede ser tanto para indicarle qué acción debe realizar, como qué acciones no debe considerar (en el caso de que la acción anterior haya resultado perniciosa para la planta).

Como se ha comentado, la deliberación de una acción requiere la comunicación con la base de casos, en caso de fallo en la comunicación el agente se vuelve completamente reactivo (Ilustración 3.4), evitando así que se bloquee su función primaria (proteger a la planta). Cuando el fallo de comunicación sucede, como se ha mencionado antes, se siguen almacenando las posiciones de puntos de carga, así como los mapas de obstáculos, de forma que el agente pueda continuar su labor sin mayores problemas. Aunque no se renuevan hasta que el simulador no vuelve a estar activo.

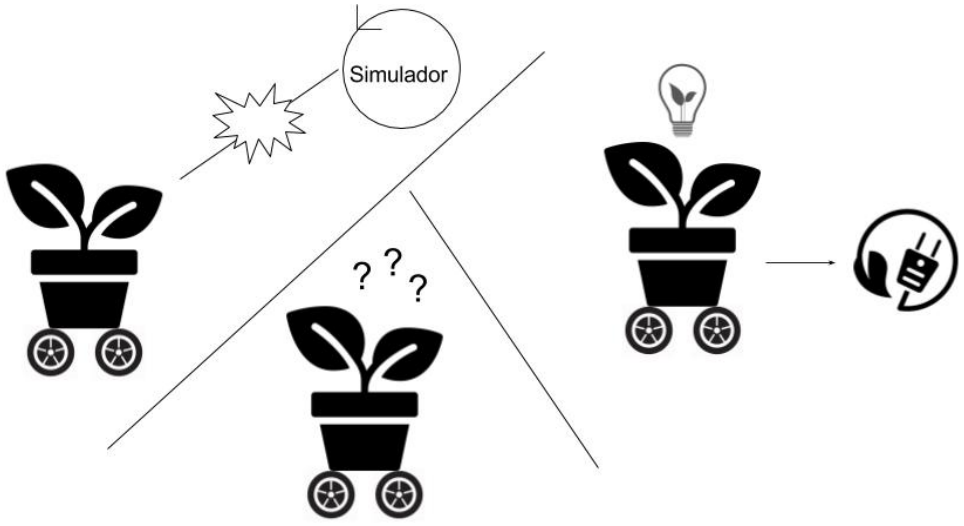

Ilustración 3.4. Secuencia de acción ante el fallo de comunicación

Por supuesto, si falla la comunicación, no habrá posibilidad de que el usuario vea el estado actual de la planta, ni darle órdenes al Chombo, por lo que se le alertará del error en su aplicación para que pueda tomar las medidas necesarias para solucionar el problema.

A la hora de que el agente intente realizar un movimiento, primero se intenta efectuar de la forma más eficiente (ir recto), en el caso de que el robot encuentre un obstáculo o que el simulador le indique que debe detenerse (porque tiene delante un obstáculo virtual), se realiza una búsqueda del mejor camino usando Dijkstra 8-direccional (Ilustración 3.5), usando los mapas proporcionados por el servidor.

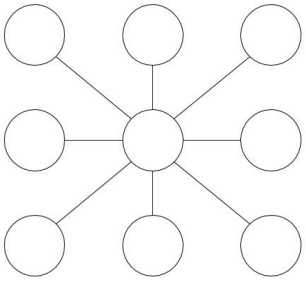

Ilustración 3.5. Nodos conectados 8-direccional para Dijkstra.

Para homogeneizar los estados iniciales, se ha decidido que cada vez que un Chombo real termine un movimiento vuelve a una posición inicial de rotación (para que siempre acaben mirando a una dirección determinada, lo que ayuda a la hora de poner puntos de carga de batería o agua).

El agente se debe encargar en su configuración (la cual inicia el simulador) de generar los artefactos que necesita, el de entorno y el de utilidades, los cuales se explicarán en su sección de diseño más adelante.

3.2.1.3 Diseño común de agentes para la ejecución

Una vez se ha cargado el sistema inicial, el simulador manda órdenes a los agentes para que inicien sus configuraciones (creando los artefactos de entorno y de utilidades propios de cada uno, y entregándoles el artefacto que les permitirá conectarse a la base de datos). Entonces el simulador obtiene de cada agente su artefacto de entorno (para poder comunicarse con él) y a través de este les envía la información inicial de los mapas, la posición de cada agente (para los virtuales), así como las posiciones de los puntos de carga de batería y agua. Una vez los agentes tienen todo lo necesario, el simulador les indica que ya pueden empezar a trabajar.

Cuando ya se ha configurado y arrancado el sistema, el simulador empieza la ejecución, recopilando las acciones que cada agente desee realizar (movimiento, riego o carga). Una vez se dispone de las acciones a realizar, las lleva a cabo e informando al agente si se ha podido desarrollar (en el caso de que se intenten mover a una posición ocupada) o comunicándoles el resultado (nueva posición alcanzada, cantidad de agua regada o cantidad cargada de batería o agua). Por último, reduce la batería y la humedad de cada uno, simulando el desgaste natural.

3.2.2 Casos de uso de los agentes

Ahora se va a explicar el diseño de las acciones que tanto el agente Chombo o el agente Manager pueden realizar. El siguiente diagrama de casos de uso muestra las acciones básicas que realiza cada una de las partes (Ilustración 3.6).

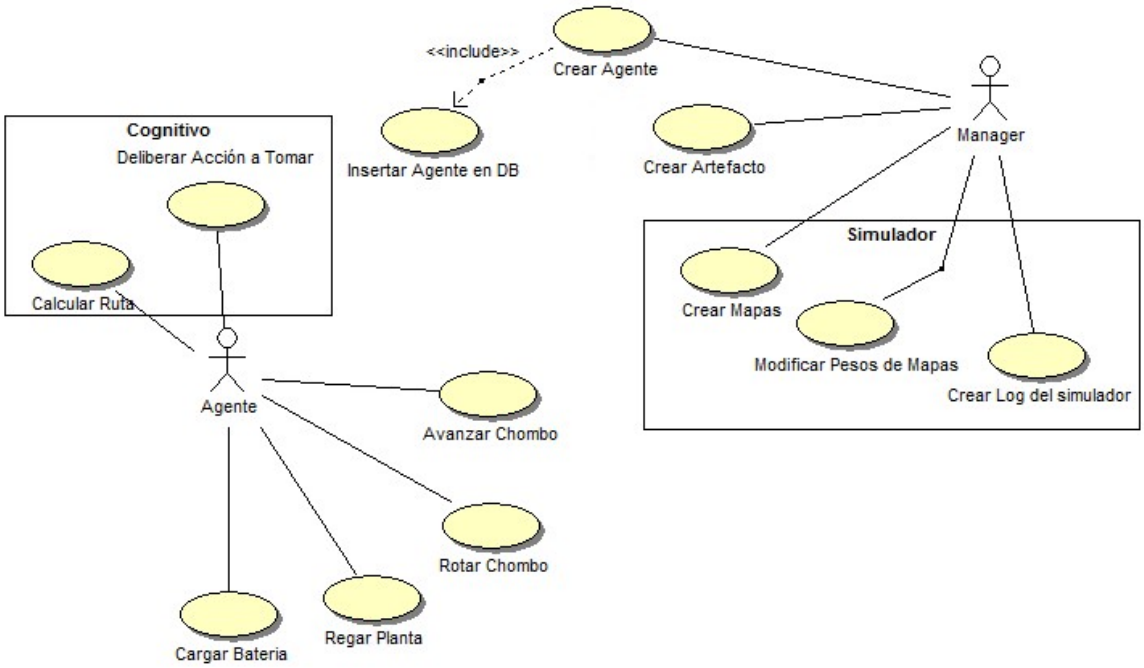

Ilustración 3.6. Diagrama de casos de uso del agente y el manager.

Como podemos observar en el diagrama, el Agente Chombo (Agente), dispone de dos tipos de acciones, las cognitivas y las acciones de control. Las

cognitivas se encargan de la toma de decisiones. Una de las decisiones es considerar que acción debe realizarse a continuación, con lo cual usa un sistema de razonamiento basado en casos, usando antiguas decisiones almacenadas en la base de casos. La otra implica la búsqueda del camino óptimo hacia el destino, esta acción se realizará siempre que se haya encontrado con un obstáculo, para encontrar la mejor forma de evitar dicho obstáculo. Tal como se ha indicado en el diseño del agente 3.2.1.

Las acciones de control son principalmente para usar los controladores del Chombo, hay acciones que varían en función de si el sistema al que están asociados es real o virtual, pero esto debe ser transparente al agente, por lo que los cambios de comportamiento se dejarán al artefacto de entorno. Las acciones son las que siguen:

- Rotar una serie de grados al robot para encarar la dirección a avanzar. Esto es necesario cada vez que se realiza un movimiento, pero solo va a afectar a los Chombos físicos, ya que los virtuales no requieren realizar dichas rotaciones (los moverá el simulador donde considere oportuno).

- Avanzar al Chombo, esta acción tiene dos versiones, en función de si el agente está a cargo de un sistema real o virtual. Si es real simplemente le dice al Chombo que avance una cantidad determinada de metros. Si es virtual le indica al simulador a donde quiere ir para que este tome la decisión de a dónde debe moverlo.

- Regar la planta, esta acción abre y cierra (pasado un cierto intervalo) la bomba de agua de la planta, el tiempo puede modificarse para que se pueda adaptar a las diferentes plantas. En el caso de los virtuales, esta acción le indica al servidor que se ha regado el tiempo del intervalo, para que este pueda modificar la cantidad de agua en el tanque y la humedad.

- Cargar batería/agua, esta acción sirve para ambos casos, simplemente se usa cuando el Chombo esté en una posición de carga, indica que el agente desea cargar, por si fuera necesario habilitar algún sistema en el robot (por si la válvula está cerrada o los conectores para la carga de batería están protegidos). Si es el sistema virtual le indica al servidor que se está realizando la carga, para que modifique los valores pertinentes (aumentando la batería o la cantidad de agua en el tanque).

El agente *Manager* por otro lado también dispone de dos tipos de acciones, relativas a la simulación o el resto, que son usadas principalmente al inicializar el sistema, como la acción de crear Artefacto, que crea tanto el artefacto de conexión a la base de datos, como el artefacto *Manager*. La acción de crear un agente nuevo se usa cuando algún usuario lo indica, esta acción corresponde a una de las conexiones con la interfaz, tal como se indica en la sección 3.1 y más concretamente en la ilustración 3.2.

Las acciones relativas al simulador tienen múltiples usos:

- La acción de creación de mapas sirve principalmente durante la carga del sistema, para tener la referencia a la localización donde se distribuyan los Chombos. Pero también sirve si el usuario desea cambiar de sistema, ampliarlo o reducirlo. Proporcionando nuevos márgenes se volvería a usar esta acción para crear el nuevo sistema.

- La acción de modificación de peso de los mapas sirve principalmente para posicionar a los Chombos en su posición aproximada. Pero también sirve para que el usuario pueda modificar ciertas localizaciones del entorno, permitiéndole negar que los Chombos puedan acceder donde desee.

- La acción de creación de log sirve para que el simulador informe de las acciones que los agentes van realizando, así como la información completa de la simulación (solo cuando se precise ya que es mucha más

carga que solo las acciones tomadas). Esta acción envía al usuario o usuarios toda la información, además de mostrarlo por consola por si se quiere llevar un registro en el propio servidor.

3.2.3 Diseño de artefactos

Como se ha comentado durante el diseño de los agentes, hay un total de cuatro artefactos que los agentes usan: el de entorno, el de utilidades, el *Manager* y el de conexión a la base de datos, pero como se puede ver en la Ilustración 3.3, este trabajo se encarga de diseñar los tres primeros, dejando el diseño del artefacto de conexión a la base de datos a la sección del proyecto correspondiente (el trabajo de desarrollo de la base de datos).

Los agentes se constituyen con una serie de creencias y reglas, las cuales cubren las funcionalidades exigidas anteriormente. Pero, aunque esto es útil para la toma de decisiones, no lo es tanto para poder procesar y computar los datos, por ello se requieren los tres artefactos mencionados.

3.2.3.1 Diseño de artefacto de entorno

El artefacto de entorno sirve tanto para comunicar la información de los sensores al agente como para controlar los actuadores del robot. Además, almacena los puntos de interés ofrecidos por el simulador para una mayor eficiencia a la hora de que un agente los requiera.

Aunque hay robots físicos y versiones virtuales de estos, los agentes deben ser iguales en ambos casos, ya que el comportamiento debe ser el mismo independientemente si tienen una interacción real o no. Por ello es el artefacto de entorno el que debe variar, a la hora de dar órdenes al Chombo al que pertenecen. Si es real son órdenes directas al Arduino que controla al robot, mientras que si son virtuales deben transmitir dichas órdenes al simulador para que las procese. Además, dicho artefacto sirve para que los datos de los sensores (o en el caso virtual de los datos de entorno que el simulador les

proporcione) se conviertan en creencias de los agentes, para poder tomar la decisión necesaria.

Como la situación del sistema (si es real o no) debe ser transparente al agente, tal y como se ha comentado anteriormente, aunque el artefacto de entorno deba ser modificado a la hora de adaptarse a cada sistema, la interfaz con el agente debe ser exactamente la misma, por lo que solo deberá cambiar la implementación de cada método.

Siendo que es este artefacto el que se encarga de la sensorización, el simulador deberá de disponer de acceso a cada uno de ellos, tanto para transmitir la información como para recuperar los datos del agente. Pero como las necesidades de sensorización si son distintas en un sistema real o virtual, el simulador si deberá saber que artefactos corresponden a los agentes reales o a los virtuales.

3.2.3.2 Diseño de artefacto de utilidades

El artefacto de utilidades se encarga de cualquier carga computacional que el agente requiera, tanto para el cálculo de caminos eficientes hasta su destino, como para lanzar eventos de control (para que el agente sepa cada cuánto tiempo deben realizarse ciertas acciones).

Para ello debe disponer de los mapas de posiciones actualizados cada vez que se requiera calcular el camino óptimo. Este cálculo debe ser lo más óptimo posible para evitar que el agente tarde demasiado en moverse.

Para evitar movimientos intermedios, el resultado de la búsqueda de caminos solo tiene que marcar las intersecciones donde se modifique el ángulo de movimiento, de forma que siempre que el movimiento siga una línea solo vea el punto final de esta y no se detenga a cada paso.

3.2.3.3 Diseño de artefacto Manager

El artefacto *Manager* gestionará la comunicación con la interfaz de usuario, así como de cargar la situación inicial (mapas tanto de luz como de posición), puntos iniciales de carga de batería y agua. Se requieren dos mapas de posicionamiento, uno estático y otro dinámico, de forma que cuando los agentes se vayan desplazando por el dinámico, se recuperen los valores anteriores de la posición que abandonan. Pero como esto es una funcionalidad interna, hacia los agentes solo existirá un mapa. el dinámico.

El artefacto *Manager* también almacena los nombres de cada agente del sistema, para que el simulador pueda disponer de ellos cuando quiera. El artefacto es el que separa los agentes virtuales de los reales, para las funcionalidades descritas anteriormente en el simulador de interacción con los agentes. Además, gestiona la posición inicial de los agentes añadiéndolos en el mapa (creando posiciones aleatorias para los agentes virtuales).

El artefacto *Manager* también gestiona el movimiento de los agentes, cuando el simulador le indica qué movimiento se va a realizar, el artefacto debe cuidar de que no haya colisión con el sistema virtual. Además, en el caso de los agentes virtuales, también es el encargado de, una vez se ha asegurado de que el movimiento es viable, realizar dicho movimiento, desplazándolos por el mapa de posiciones dinámico e informando al simulador de la nueva posición.

El artefacto *Manager* dispone además de un sistema para calcular las modificaciones a la humedad y a la cantidad de agua restante que se realizan al regar, funcionalidad disponible para los agentes virtuales. Es necesaria cuando dichos agentes envíen la petición de riego al simulador.

Por último, el artefacto *Manager* se encarga de la actualización del mapa lumínico, que debe variar en función de si hay sensores reales o no. Si hay sensores reales se distribuyen sus valores por el mapa, generando un gradiente entre las diferencias entre cada punto para una mayor precisión.

Mientras que, si no hay sensores, los valores lumínicos deben generarse de forma aleatoria.

3.2.1 Otras consideraciones

Otra opción que estuvo en consideración en la creación del simulador era usar JaCaIIVE, pero tiene muchas configuraciones físicas que no resultan útiles (como la gravedad), y solo podríamos usar movimiento y colisiones, con lo que aún se necesitaría crear una simulación del resto de parámetros (agua, batería, luz…) por lo que se ha decidido simular todo en el mismo sistema., ya que resulta demasiado complejo para este primer prototipo que se está realizando.

4. Desarrollo

En este capítulo se tratará la implementación de los agentes y artefactos vistos durante el apartado anterior. Como se ha comentado y se puede ver en la imagen siguiente (Ilustración 4.1), hay dos tipos de agentes, el simulador (que solo habrá una copia del mismo en el sistema) y los agentes Chombo (de los cuales puede haber cualquier cantidad de agentes de forma simultánea). Estos agentes se van a desarrollar en Jason [explicado en el apartado de tecnologías 2.3.2]. Para el desarrollo de los artefactos se usará Java con una extensión de CArtAgO [tecnologías 2.3.1]. Como se ha comentado anteriormente, habrá tres artefactos distintos: artefacto Manager, del cual solo habrá una copia en el sistema (ya que va asociado al simulador) y los artefactos de entorno y utilidades, de los cuales habrá una copia por cada agente Chombo en ejecución.

El sistema multiagente, tal y como se ve en la imagen siguiente (4.1) tiene conexión con todos los sistemas, por lo que requiere una interfaz de comunicación con cada uno de ellos

Por el momento solo se ha podido probar la interfaz con la base de datos, ya que no se dispone aún de la interfaz de usuario ni del Chombo físico, pero en esta sección se explica que modificaciones serán necesarias para la adaptación a estos sistemas.

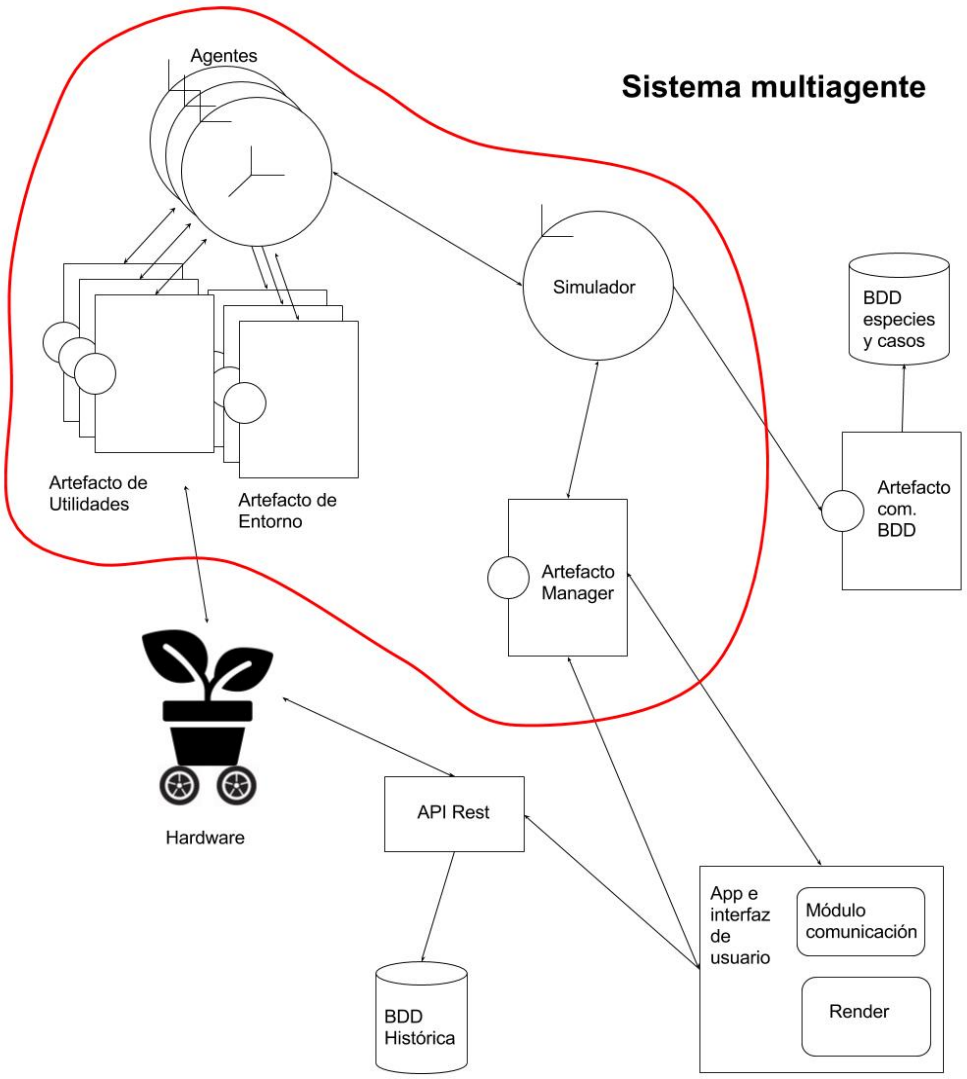

Ilustración 4.1 Sistema completo

4.1 Desarrollo de Agentes

En este punto se va a explicar la implementación final del prototipo. El cual consistirá en los dos agentes y tres artefactos mencionados. Para el agente Chombo se debe tener en cuenta que debe ser siempre igual indistintamente a si se asocia a un artefacto físico o virtual. En cambio, el artefacto de entorno, aunque no debe variar su interfaz, si tiene que modificarse su implementación a la hora de conectarse a un sistema real o virtual.

4.2.1 Desarrollando el agente Chombo

A la hora de hacer el agente se ha tenido en cuenta que, en Jason, las reglas son prioritarias según el orden en el fichero (o, dicho de otro modo, las reglas que aparecen primero se revisarán y ejecutarán primero), en un ciclo de ejecución solo se resolverá la primera regla posible, por lo que hay que tener cuidado con qué regla se ha de poner antes, para que no puedan ocurrir bloqueos inesperados o malas funcionalidades.

Como se ha visto en la parte de diseño, este agente se encarga del control del Chombo. Por lo que este agente representa la inteligencia artificial en el sistema. Para poder desarrollar dicha inteligencia, se usa el sistema de razonamiento basado en casos, que permite al agente evolucionar durante la ejecución del mismo, al poder almacenar nuevos casos que vayan surgiendo, para tener cada vez un conocimiento más amplio de las acciones óptimas a realizar. Esto mejora el comportamiento futuro de todo el sistema.

En esta sección se explicará cada una de las reglas que el agente requiere para poder cubrir las funcionalidades descritas en el apartado de diseño, como son el control de los actuadores del Chombo, la recepción de los sensores o cómo se realiza la conexión con la base de casos para poder desarrollar el sistema cognitivo

Aparte de las reglas que se desarrollarán más adelante, primero se tiene una regla de inicialización, que solo debe ejecutarse una vez, y a petición del simulador, en la cual se crean los dos artefactos que requiere el agente (el de entorno y el de utilidades), además de obtener el artefacto de conexión a la base de datos que crea el simulador.

Teniendo esto en cuenta, las reglas se han establecido en el siguiente orden:

- Primero se han establecido las reglas de bloqueo:

- !chargingW: regla para bloquear al agente siempre que este se encuentre cargando agua. La regla se anula cuando alcanza el 70% de capacidad del tanque. Pero no tiene porque detener la carga.
- !chargingB: regla para bloquear al agente siempre que este se encuentre cargando batería. La regla se anula cuando alcanza el 80% de capacidad de batería. Pero no tiene porque detener la carga.

- Segunda van las reglas de las condiciones de parada !stop, ya sea porque se ha encontrado un obstáculo físico, porque se ha encontrado uno virtual o porque simplemente ha llegado a su destino. Con estas reglas se mira si ha terminado el movimiento o si por el contrario es necesario hacer una búsqueda del camino óptimo para intentar llegar al destino marcado.

- Tercera va la regla de búsqueda de situación en la base de datos !bdd, como la toma de decisiones se basa en un razonamiento basado en casos, esta regla es esencial para la parte cognitiva del agente. En esta regla se usa el artefacto de comunicación con la base de casos, con el cual obtiene que acción debería realizar o que acciones no son apropiadas para dicho caso (En función del índice de confort del caso). Si el agente no encuentra una solución para el estado actual, realiza otra búsqueda con situaciones similares. Mientras se esté realizando la acción obtenida o no se hayan encontrado acciones, la regla no puede repetirse mientras se esté ejecutando la acción propuesta o si no se ha encontrado acción alguna (creando un nuevo caso).

- Cuarta va la regla de movimiento !move, la cual, una vez se le ha pasado la posición a la cual ha de moverse, primero ejecuta un algoritmo trigonométrico para saber cuántos grados ha de rotar el Chombo (para los casos físicos) y después avanza hacia esa posición la distancia necesaria.

- Quinta va la regla de riego !water, la cual lanza dos acciones al Chombo, que son abrir y cerrar la bomba de agua (después de un cierto tiempo establecido, el cual se puede modificar durante la ejecución).

- Por último, van las reglas reactivas !idle, una vez ya se ha buscado en la base de datos y, o no se ha tenido éxito, o el resultado han sido solo acciones que no deben hacerse, se contemplan las necesidades del Chombo y de la planta, mirando tanto si tiene suficiente batería o agua en el tanque, como si las condiciones de humedad y temperatura están dentro de las marcadas para la planta en cuestión. Si alguna de las necesidades no se cubre, se plantean las acciones para resolverlo. En el caso de que sean múltiples necesidades de forma simultánea, se considera cuál es más urgente en ese momento.

4.2.4 Desarrollando el simulador

El simulador es otro agente del sistema, necesario para comunicarse con el resto de agentes, el cual se requiere en la inicialización y proporciona un apoyo de realidad aumentada para los agentes reales. Mientras que es esencial para los agentes virtuales, ya que no pueden realizar acción alguna sin el mismo.

El simulador divide su regla principal !start en las siguientes partes:

- Inicialmente hay una etapa de inicialización, en la cual crean los dos artefactos que requiere, tanto el *Manager* como el artefacto de conexión con la base de datos.

- Después obtiene los nombres de todos los agentes del sistema y los almacena en el artefacto. Avisando a cada agente de que realice su fragmento de código de inicialización. Una vez se hayan inicializado, les envía a cada uno de ellos los mapas, su posición y las posiciones de carga. Cuando ya tienen todos estos parámetros se les da la orden de iniciar la ejecución.

- A continuación, comienza la etapa de simulación, para tener un control temporal preciso, primero se marca el tiempo en el artefacto *Manager* en el cual se comienza cada ciclo de simulación, entonces se comienza obteniendo de cada agente las órdenes que han realizado entre ciclos. Las órdenes pueden ser:

    - De movimiento, con la cual deben cambiar la posición del agente como se explicará más adelante en el artefacto *Manager*, y se devuelve al agente la nueva posición, o un error en el caso de que el movimiento no esté permitido (porque la posición a la que desean moverse esté ocupada). En el caso de que sea un error, se le envía también el mapa de posiciones para que pueda ejecutar la búsqueda de caminos con los datos más actualizados posibles.

    - Abrir la bomba de agua, la cual solo indica por el *log* que se ha comenzado a regar la planta (para que en el futuro el usuario pueda observar qué acción está realizando)

    - Cerrar la bomba de agua, que viene a su vez con el tiempo que ha estado abierta, con el cual se calcula la cantidad de agua gastada y la humedad que ha aumentado con dicha acción.

    - Carga de batería, la cual aumenta la batería disponible en el agente que efectúe dicha acción.

    - Carga de agua, que al igual que la batería aumenta la cantidad de agua en el tanque cada ciclo que pase cargando.

- Por último, se le transmite el mapa de luminosidad y se le modifica el desgaste por batería y humedad, intentando simular lo que gastaría en una situación real. Además, calcula el tiempo que ha gastado y se detiene para que entre ciclo y ciclo siempre haya el mismo tiempo de espera, en el caso de que una iteración completa gaste más tiempo que

el tiempo de espera de ciclo, esto no podrá ser posible y no se esperará nada.

4.2 Desarrollo de Artefactos

4.2.2 Desarrollando el artefacto de entorno

Como se ha mencionado anteriormente, el artefacto de entorno se encarga tanto de controlar los valores de los sensores, de controlar los actuadores, como de almacenar valores de interés para optimizar búsquedas y evitar errores por fallos de comunicación.

Para ello este artefacto almacena, además, todos los valores que le transmite al agente (para que puedan ser accesibles al *manager* cuando los requiera el usuario).

Los siguientes métodos sirven para controlar los actuadores, los cuales deben implementarse de manera distinta si están conectados a un robot o a un sistema virtual:

- rotate(amount) (solo útil para Chombos físicos): cuando se conecta a un sistema real indica la cantidad de grados que debe rotar sobre sí mismo el robot, siendo un ángulo positivo el giro a la derecha y negativo hacia la izquierda. Como se menciona sólo es útil para hacer que el Chombo rote. No tiene sentido que en un sistema virtual se efectúe dicha rotación, ya que el movimiento es directo al estar simulado.
- advance(x, y): hace avanzar una cantidad $\sqrt{(x-x')^2+(y-y')^2}$ u.m. al robot, o le indica al servidor a qué punto desea ir el agente. Siendo x' e y' los valores de la posición actual del agente, los cuales se encuentran almacenados en el propio artefacto.

- openWater() / closeWater(): ambos métodos sirven para controlar el riego a la planta (para abrir la bomba o para cerrarla), en el caso del Chombo físico simplemente transmite la orden al *hardware* mientras que, en el caso virtual, el método de abrir inicia un contador de tiempo que se para con el de cerrar. Una vez se cierra, envía al servidor la cantidad de tiempo que ha estado regando, tanto para vaciar el tanque como para indicar cuánto aumenta la humedad.

Los siguientes métodos sirven para optimización y para que el agente pueda seguir yendo a cargar agua o batería en el caso de que pierda la comunicación con el simulador.

- nextBPoint() / nextWPoint(): ambos métodos son para indicarle al agente el punto más cercano de carga, ya sea de batería (BPoint) o de agua (WPoint). Para que este método pueda funcionar, primero debe recibir del simulador dichos puntos de carga. También requiere los mapas de posicionamiento, ya que el método no contempla los puntos de carga ocupados (para evitar que un agente se atasque intentando ir al punto de carga más cercano donde no podrá efectuar dicha carga, lo cual es peligroso ya que puede hacer que el agente pierda toda la batería o agua durante la espera).

Los siguientes métodos son exclusivos del servidor:

- moveTo(x, y): este método sirve para indicar al agente que debe moverse hacia la posición (x, y). Se usará cuando el usuario desee desplazar al Chombo a dicha posición.

- stop(): este método sirve para indicar al agente que se ha detenido por un obstáculo virtual (y en el caso de que sea un agente físico, detener al robot). Con este método es con el cual el agente se verá forzado a realizar un cálculo de búsqueda del camino óptimo.

- getOrders(): este método es además exclusivo para los agentes virtuales, sirve para que los agentes puedan transmitir al simulador que acciones pretenden llevar a cabo. Las cuales se almacenarán en una lista que se transmitirá al artefacto *manager* para su procesamiento.

Por último, van todos los métodos de set/get, para fijar los valores de los parámetros sensorizados, así como para obtener dichos parámetros (para el servidor). Se obviarán dichos métodos al ser repetitivos y de información redundante.

4.2.3 Desarrollando el artefacto de utilidades

Este artefacto sirve principalmente de apoyo a los agentes, ejecutando funciones complejas o gestionando los eventos necesarios, ya que el sistema de creencias y la forma en la que se tratan los datos no facilita estos dos casos.

Los métodos que requiere el agente para sus funciones son los siguientes:

- path(mapa[][], ix, iy, fx, fy): este método calcula el camino óptimo usando Dijkstra 8-direccional. Aunque Dijkstra está pensado para grafos, se puede usar aquí si consideramos que cada posición (x, y) en el mapa (matriz) es un nodo, el ser 8-direccional es porque se han unido no sólo los nodos cercanos verticales y horizontales, sino también los diagonales. Para evitar colisiones laterales, se le niega a la búsqueda un camino diagonal siempre que cualquiera de los espacios adyacentes (vertical u horizontal) sean inaccesibles. El camino se devuelve entonces como una serie de puntos que marcarán el camino del agente, pero solo se usan los puntos en los que el agente deba detenerse para rotar (siempre que un camino sea recto, no es necesario almacenar todos los puntos del mismo). Este método requiere un mapa de posiciones, la posición inicial del agente y el destino.

- nextMove(path[]): este método le indica al agente cual es el siguiente punto al que debe moverse, se debe llamar una vez se dispone de un camino generado por Dijkstra. Además, almacena el punto al que se acudió la última vez en el que se llamó al método, de forma que si durante el camino se encontrara con algún obstáculo (otros agentes que se hayan movido, por ejemplo), volverá a recalcular Dijkstra. Este método requiere un *array* con las posiciones de los siguientes destinos.

- lightEvent(): este método inicia un Timer, el cual activa un ActionListener interno cada cierto intervalo para que se le avise al agente de que debe almacenar la luz actual. Este método es necesario para el siguiente.

- lightAmount(amount): este método sirve para almacenar la luminosidad actual y llevar una cuenta de la cantidad de luz recibida. La luz almacenada no es una sustitución de la anterior (de eso se encargan los sensores), si no que se va acumulando a lo largo del día. De esta forma el agente puede ser consciente de la cantidad de luz que lleva hasta el momento, para saber si la planta que cuida requiere más o menos luz en ese instante. Se almacena hasta un máximo de veinticuatro horas, teniendo un margen de un día completo y así también puede saber si el día anterior ha recibido una cantidad de luz excesiva o escasa.

- trigo (ix, iy, fx, fy, r): este método sirve para calcular cuánto debe rotar un agente para encararse hacia el objetivo. El cálculo sea realiza con funciones trigonométricas. Facilitando así el movimiento, necesitando simplemente una orden de avance. Para ello requiere la posición inicial, el destino, y el acumulado de rotación hasta el momento.

- giveOrder(actions[]): este método sirve para que, dada una lista con las acciones propuestas por la base de casos, se le indique al agente que acción debe realizar o que acciones no debería considerar. En el caso de que haya un -1 como único elemento, implica que no hay un caso lo suficiente similar. Este método requiere un *array* con las acciones propuestas.

4.2.5 Desarrollo del artefacto Manager

Este artefacto tiene múltiples funciones, entre ellas sirve de apoyo para el simulador, calculando las variaciones que suceden por las acciones de los agentes. Otra función es el control del movimiento de los agentes reales, o la simulación de los virtuales, para evitar las colisiones (creando un mundo de realidad aumentada). Además, sirve de intermediario entre todo el sistema multiagente (simulador y agentes) con el usuario, enviando la información de la situación del sistema, así como de las acciones que cada agente irá realizando. Por último, también sirve para que el usuario pueda indicar las zonas de interés (puntos de carga, zonas de acceso prohibido o el punto de reunión donde se le puede indicar qué agentes deben acudir a él), u obtener la información de cada una de los Chombos de forma que pueda observar su estado (cantidad de batería o agua en el tanque, humedad, temperatura, luminosidad, etc.).

Para poder realizar todo esto, en el artefacto se han implementado los siguientes métodos:

- init(): Este método es de inicialización, en el cual se deben cargar todos los mapas (mapas de posicionamiento estático y dinámico, donde los agentes solo aparecen en el dinámico y así poder recuperar el valor de cada posición durante el movimiento de estos, así como el mapa de luminosidad) y los puntos de carga iniciales.

- Varios métodos para almacenar los agentes y los artefactos de entorno asociados a cada uno de estos, usando simplemente un array para los agentes y un *Map* implementado como *HashMap* para los artefactos. Así como los métodos para recuperar ambas cosas. Siendo los métodos:
    - saveAgentsName(agents[]): almacena en dos listas todos los agentes que le proporcione el simulador. Una lista para los agentes reales y otra para los virtuales.
    - setArtifact(agent, artifact): almacena en un *HashMap* los artefactos asociándolos al agente al que corresponda.

- Para que el simulador mantenga siempre el mismo tiempo de ciclo (a no ser que un ciclo se alargue más que un tiempo marcado), se han creado los dos siguientes métodos:
    - startTime(): este primer método sirve que se almacene el instante actual, el cual se ejecuta al iniciar el ciclo.
    - getWaitingTime(): Este segundo método obtiene, al igual que el primero, el tiempo en el que se llama, restándolo al instante almacenado anteriormente y restando el resultado a un tiempo de ciclo marcado. El resultado de la consecución de restas anterior es el tiempo que falta hasta que un nuevo ciclo deba ser iniciado. El intervalo obtenido se entrega al simulador para que espere dicho tiempo. Si la diferencia es negativa, se le envía un cero, para que no espere nada y comience el siguiente ciclo de forma inmediata.

- nextPosition(ix, iy, fx, fy): este método sirve para el movimiento virtual de agentes, que, dando la posición inicial (ix, iy) y final (fx, fy) a la que el agente quiera acudir, se devuelve la siguiente posición a la que el agente se ha movido (más cerca de su objetivo). Una vez calculada, el artefacto modifica en el mapa la nueva posición (que es la forma en la que los agentes realmente se mueven). En el caso de que no pueda moverse a dicha posición por un obstáculo, se le indica al agente que debe recalcular el camino óptimo (y por supuesto no se modifica la posición del agente). Para poder realizar el movimiento, el artefacto debe tener en cuenta el vector de movimiento entre el punto final e inicial y debe tomar la decisión de a qué casilla debe moverse, ya que el vector no necesariamente caerá siempre sobre los centros del mapa cuadriculado. Para resolver este problema se ha tomado la siguiente decisión:

    - Primero se calcula el vector, que es tan sencillo como restar al punto final el inicial en ambas dimensiones (x = fx - ix, y = fy - iy).

- Entonces se contempla si el movimiento es colineal (x = 0 o y = 0), para evitar errores de divisiones por cero. Si resulta serlo, directamente se intenta realizar el movimiento en dicho eje hacia la dirección del vector previamente calculado.

- Por último, sabiendo que no es colineal, se calcula el ángulo del vector, usando la regla trigonométrica de tan(ángulo) = y/x. Como ya disponemos de la dimensión de x e y, se ha tomado la decisión de que, si el ángulo está en el intervalo de treinta y sesenta grados, se realiza un movimiento diagonal, si no, se realiza el movimiento horizontal o vertical. De esta forma se busca la mayor homogeneidad posible, ya que con esta decisión acaban siendo un tercio para cada movimiento posible, como se ve en la siguiente imagen (Ilustración 4.2). Para evitar tener que contemplar el cuadrante al que hay que realizar el movimiento, se usa el absoluto de la división |y/x|, para evitar cálculos innecesarios, se precalculan tan(30) y tan(60), y para el movimiento se usa el signo de x e y, de forma que siempre se moverá hacia el cuadrante correspondiente sin tener que averiguar cuál es.

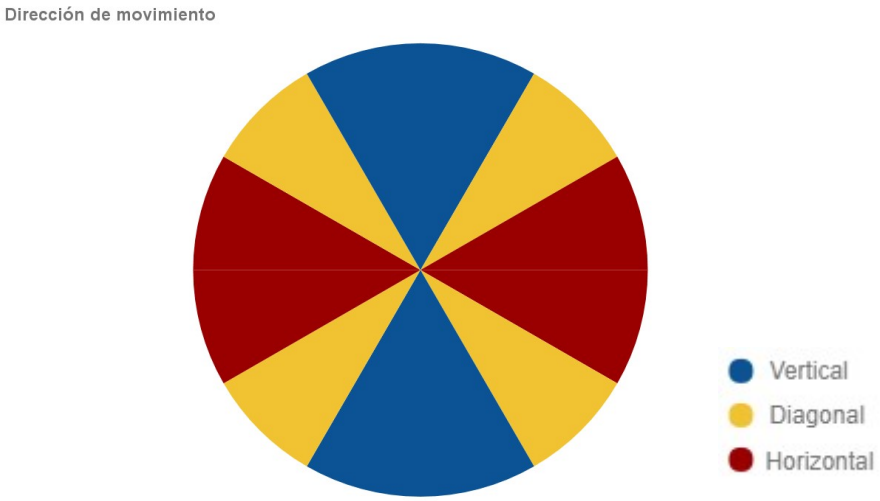

Ilustración 4.2 Elección de movimiento respecto al ángulo

- initAgentPos(num): este método sirve para iniciar la posición de los agentes virtuales de forma aleatoria, repartiéndolos por el mapa en posiciones factibles. Requiere saber la cantidad de agentes para hacer una distribución homogénea.

- updateLightMap(): este método sirve para que, a falta de sensores lumínicos y para que los agentes puedan tener a su disposición un mapa de luz, se genere en función de la hora del día un mapa lumínico genérico (Ilustración 4.3). La hora del día modificará la intensidad de la luz almacenada. El mapa se genera con un foco de luz en el centro (un cuarto del mapa) con la misma intensidad en todos sus puntos, que va degradándose a razón de 5% por casilla hacia los bordes, esto permite que los agentes virtuales puedan simular su búsqueda de luz. Creando un mapa similar al siguiente:

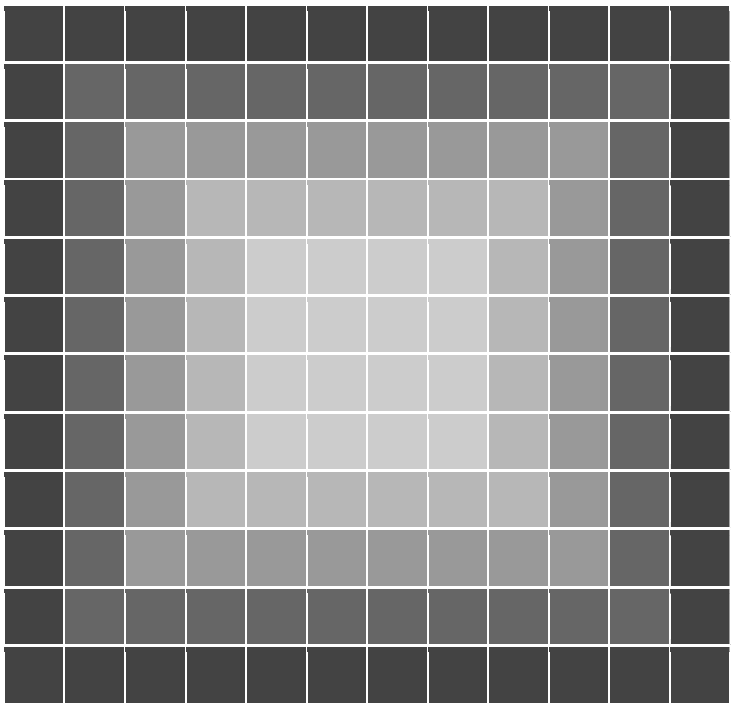

Ilustración 4.3 Mapa de iluminación para el prototipo

El mapa se ha diseñado en propósito del prototipo simulado, y una vez se disponga de sensores de iluminación reales, deberá ser completamente sustituido por los datos que estos proporcionen. El método debería tener un gradiente entre las diferentes posiciones de los sensores, para intentar tener un mapa lo más preciso posible.

- Por último, están los métodos de envío de acciones, los cuales irán comunicando a los usuarios las acciones que van realizando los agentes, para que puedan visualizar en el *render* las modificaciones en el sistema. Como este trabajo se ha terminado antes que el correspondiente de la interfaz de usuario, no se puede visualizar el sistema de la forma mencionada. Por lo tanto, para poder visualizarlo se ha modificado temporalmente el código, de forma que se van emitiendo las acciones que se han realizado y la situación de los mapas por consola. Los métodos son los siguientes:

    - sendActionMove(agent, ix, iy, nx, ny): este método informa de que el agente *agent* se ha movido de (ix,iy) a la posición (nx,ny).
    - sendActionOpenW(agent): este método sirve para que se informe de que el agente *agent* ha abierto la válvula de agua para empezar a regar, por si se desea mostrar de alguna forma al usuario.
    - sendActionCloseW(agent , water, humity): este método sirve para informar de que el agente *agent* ha cerrado la válvula de agua con el resultado de disminuir *water* litros el tanque y aumentar *humity* % la humedad.
    - sendActionChargeB(agent): este método sirve para informar que el agente *agent* ha empezado a cargar la batería.
    - sendActionChargeW(agent): este método sirve para informar que el agente *agent* ha empezado a cargar el agua.
    - sendMaps(): este método sirve para enviar el estado del sistema, como esto puede resultar muy pesado (los mapas pueden ser muy grandes), este método solo debe ser ejecutado cuando el usuario lo requiera (generalmente al iniciar la visualización).

4.3 Pruebas

Ahora pasamos a la última etapa de desarrollo, en el cual se explicarán las pruebas realizadas y las pruebas que no se han podido hacer.

4.3.1 Pruebas realizadas

Para comprobar que se cumplen las funcionalidades y conexiones descritas en los anteriores apartados, se han creado tres sistemas distintos en los cuales probar los siguientes testeos:

N.º Test	Descripción	Comprobación
Test 1	Conexión de simulador con Artefacto BDD	Comprobación con éxito
Test 2	Conexión de simulador con Artefacto *Manager*	Comprobación con éxito
Test 3	Conexión agente con Artefacto Utilidades	Comprobación con éxito
Test 4	Conexión agente con Artefacto Entorno	Comprobación con éxito
Test 5	Seleccionar caso de BDD	Comprobación con éxito
Test 6	Calcular índice de caso	Comprobación con éxito
Test 7	Ver información de la especie para la creación de agente	Comprobación con éxito
Test 8	Comprobar paso de mensajes entre simulador y agente	Comprobación con éxito
Test 9	Comprobar ciclo de ejecución de simulador	Comprobación con éxito

El primer sistema que se realizó fue para comprobar que la comunicación con la base de casos se realizaba de forma efectiva. Esta prueba se realizó tal y como se describe a continuación:

Para comprobar la comunicación con la base de datos, se creó un agente temporal adicional para comprobar la conexión. Se desarrolló un agente que no pertenece al sistema general, muy simple, que únicamente hacía una petición a la base de datos y comprobaba que se habían modificado los datos. Para poder realizar la prueba también se creó la primera versión del artefacto de comunicación con la base de datos. Esta primera aproximación tenía la siguiente estructura (Ilustración 4.4):

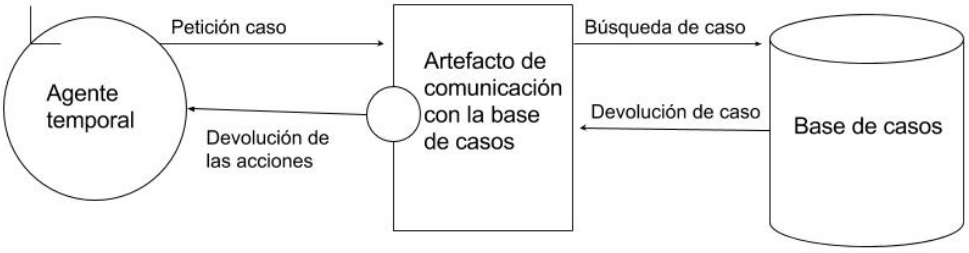

Ilustración 4.4. Sistema de conexión a la base de casos

Con ello comprobamos que la comunicación era efectiva, ejecutando múltiples veces el agente. Con este sistema se comprueban los test relacionados con la base de datos (Test 1, 5, 6, 7)

Los dos siguientes sistemas corresponden ya a la ejecución de los agentes y artefactos del sistema completo. Pero como ya se ha explicado, como no disponemos de la interfaz de usuario, al no estar aún completa, se han redirigido los mensajes por consola, para poder realizar estas comprobaciones.

El siguiente sistema corresponde a una ejecución del sistema completo con ocho agentes, para comprobar que se crean tanto los agentes como los artefactos de forma satisfactoria, además también se comprueba la comunicación entre agentes (Test 8), y la comunicación de los agentes con sus artefactos (Test 2, 3, 4).

La prueba se ha efectuado creando los agentes (ocho Chombo y un *manager*), y entonces es el *manager* el que se encarga de crear sus dos artefactos (BDD y Manager) y de avisar a cada agente que inicie su configuración, creando así todos los artefactos de Utilidades y de Entorno.

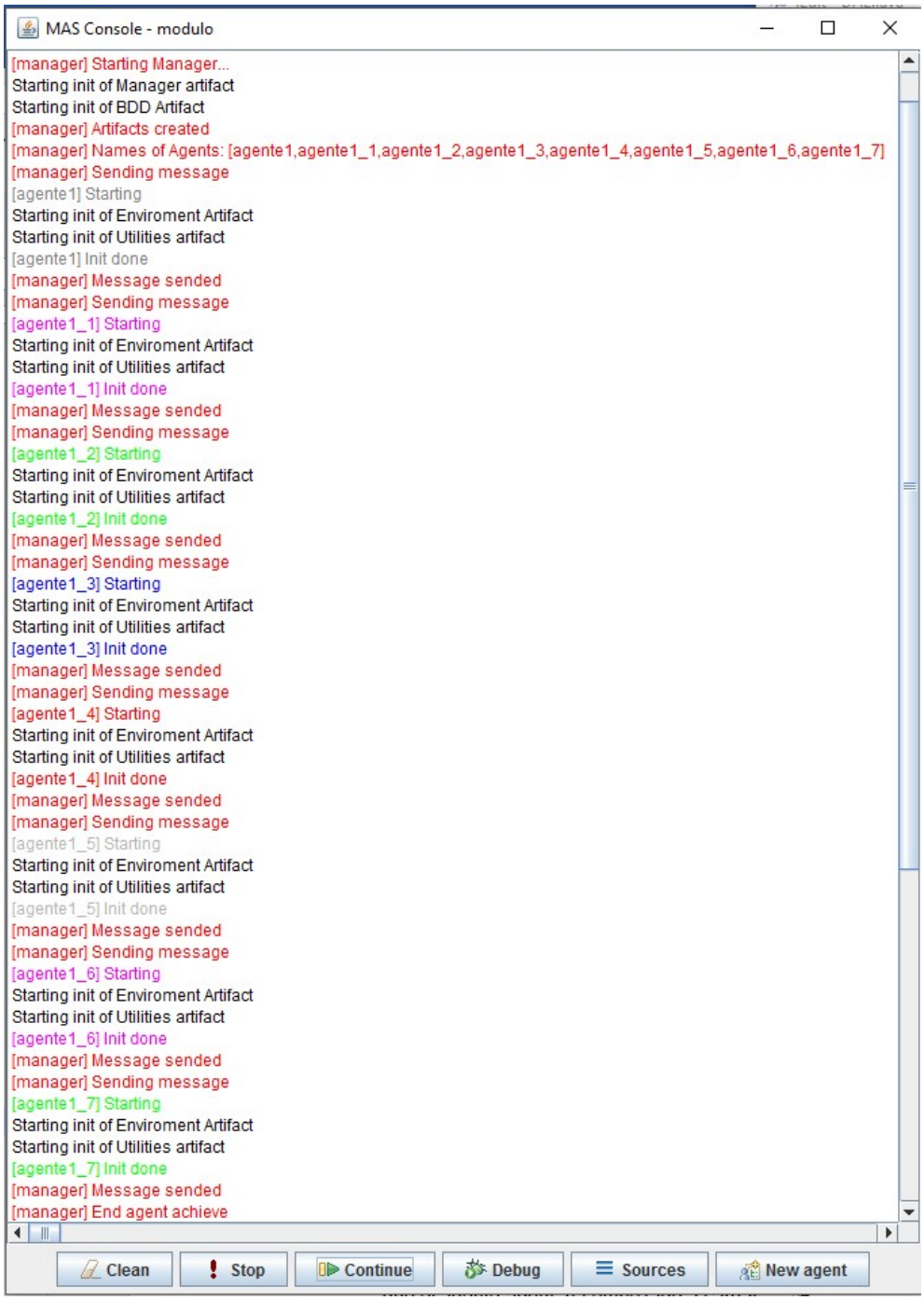

Ilustración 4.5. Log de prueba de arranque

Como se ha explicado durante el diseño, la creación de artefactos requiere la comunicación entre el simulador y los agentes Chombo, inicializándose durante el arranque, como se ve en el anterior log (Ilustración 4.5). Un ejemplo con ocho agentes Chombo y un Manager.

La última prueba corresponde a un *manager* con un único agente Chombo, para comprobar la ejecución de los ciclos del simulador (Test 9). Se ha simplificado a la salida de sólo un agente, ya que, habría demasiados mensajes como para poder ver múltiples ciclos de simulación.

Durante la creación se emiten por consola las creaciones de cada módulo para poder ver si algo falla, pero durante el ciclo del simulador, debe verse solo la propia simulación y las acciones que efectúan los agentes, ya que es lo que se transmitirá a la interfaz de usuario. A continuación, probaremos dicho ciclo de simulación (Ilustración 4.6).

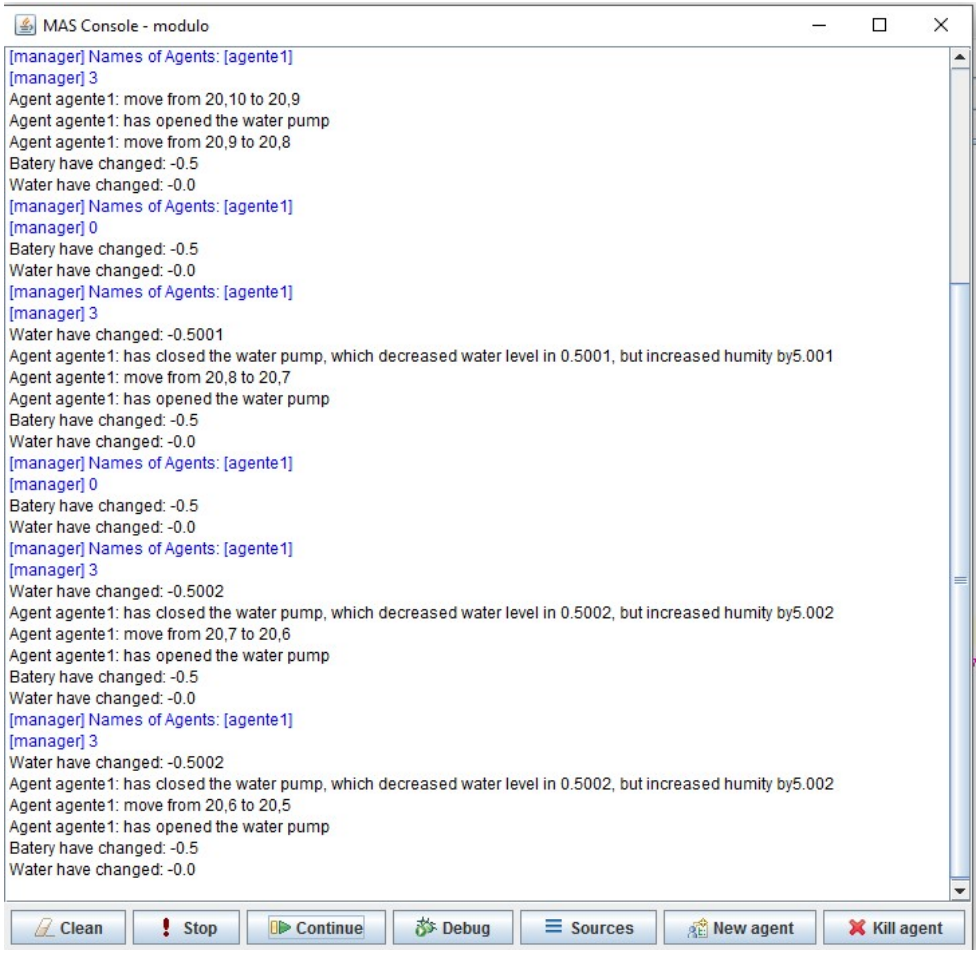

Ilustración 4.6 Log de prueba de ciclo de ejecución

Como podemos observar en la imagen, se ven múltiples ciclos de ejecución del *manager* en los cuales aparecen la cantidad de acciones que el agente1 desea realizar, y la resolución de dichas acciones, con las modificaciones de agua al regar o las posiciones a las que el agente va avanzando.

Las pruebas de conexión con los artefactos son implícitas, ya que para que el agente pueda recibir los cambios (las salidas por consola que no van asociadas a un agente las efectúa el artefacto de entorno de dicho agente, no el simulador, por lo que vemos que si le llegan los datos), el simulador primero debe conectarse con el artefacto *Manager* el cual se conecta con el artefacto de Entorno del agente que tiene almacenado (recordemos que se almacena en un *HashMap* en el artefacto *Manager*) y en la conexión entre artefactos se emite la modificación para que pueda ser visible al agente.
¿Y cómo estamos seguros de que la comunicación entre el artefacto de Entorno y el agente funciona? Porque como podemos ver en la prueba anterior, el agente también va modificando su posición (de la inicial 20,10 acaba en la 20,5 después de todos los ciclos), y la decisión de a dónde debe ir la toma el agente y para conocer su posición actual para decidir hacia donde avanzar, debe recibir las nuevas posiciones que le envía el simulador a través del artefacto de entorno. Por lo que comprobamos que la comunicación con el artefacto de entorno funciona.

Por último, faltaría la comunicación con el artefacto de Utilidades, el cual es totalmente transparente al usuario, por lo que es complicado verlo en los logs, pero sigue exactamente el mismo sistema que el de Entorno. Además, en la prueba 4.3 se puede observar que efectivamente el artefacto de Utilidades se crea, esa salida por pantalla se efectúa en la sección correspondiente de inicialización del propio artefacto de Utilidades, por lo que vemos que la comunicación entre el agente y el artefacto de Utilidades funciona.

4.3.2 Plan de pruebas de integración

1. Al no haberse implementado por el momento la interfaz, no se ha podido probar la comunicación con el usuario. Por ello se ha creado la salida por consola en vez de poder visualizar los movimientos y las acciones en un *render*. Cuando el trabajo de la interfaz de usuario finalice, deberán probarse las siguientes conexiones:

    - Que el render visualice de forma efectiva las acciones de los agentes.
    - Que el usuario pueda obtener y visualizar los datos de cada Chombo.
    - Que el usuario pueda dar órdenes de desplazamiento a los agentes.
    - Que el usuario pueda negar desplazamientos por determinadas zonas del mapa.
    - Que el usuario sea capaz de crear e insertar nuevos agentes al sistema.

2. Al no disponer del sistema físico, no se ha podido probar la implementación de los agentes en un robot. Cuando el sistema físico esté disponible, se debe implementar el artefacto de Entorno (al ser las órdenes al Arduino muy específicas de cada sistema, haberlo implementado ahora sería pura especulación) y con él se deben realizar las mismas pruebas que se han hecho a los agentes virtuales, pero en el mundo real. La última prueba será ver si la interacción del mundo virtual con el real funciona, poniendo obstáculos virtuales que el robot deba evitar.

5. Conclusiones

En este proyecto se han presentado las soluciones que mejor se han considerado en base a las decisiones tomadas en diseño e implementación. En este capítulo se presentan los objetivos logrados y propuestas de ampliaciones futuras.

5.1 Objetivo individual

Se ha logrado crear un prototipo del sistema para el mantenimiento y supervivencia de plantas, a partir de macetas robotizadas (Chombos), con un sistema multiagente que controle las decisiones pertinentes a estos temas. Para comprobar el comportamiento de dicho sistema, se genera un simulador que gestione tanto los artefactos físicos como unos virtuales creados para el propósito de evaluación de las distintas situaciones.

Como resultado de la coordinación con el resto de los trabajos para el proyecto general, se han llevado a cabo una serie de decisiones y modificaciones constantes sobre el diseño, llegando así al esquema definitivo visto en el apartado 3. Este esquema responde a los siguientes hitos:

El sistema es capaz de tomar decisiones relativas al cuidado de cada planta. A través de agentes que tienen en consideración los diferentes parámetros obtenidos de sensores o derivados de estos (como la iluminación, nitratos, temperatura, humedad, pH, entre otros).

Los agentes reciben información de relevancia del simulador, pero sin requerir asistencia constante, pudiendo seguir cuidando de la planta, aunque la comunicación falle. De forma que este sistema tenga tolerancia al particionado de red.

Los agentes son capaces de comunicarse con la base de datos para tener un razonamiento basado en casos, mejorando la efectividad de sus decisiones.

El sistema está preparado para la comunicación bidireccional con el usuario. Con métodos con los que poder ordenar movimientos a los distintos agentes, métodos para la obtención de datos de cada uno de los Chombos de forma individual o colectiva, así como métodos para emitir la información por paso de mensajes. Pero a falta de la finalización del trabajo de la interfaz de usuario, por el momento las emisiones de datos se realizan por consola.

El agente está preparado para la conexión con los artefactos físicos, teniendo como base el artefacto de entorno. El cual implementa toda la funcionalidad con artefactos virtuales, permitiendo la modificación a través de la herencia de dicho artefacto. Pero con la necesidad de cambiar los métodos de acciones físicas (rotar, mover, regar, etc.) y de sensorización, manteniendo la misma interfaz hacia el agente, para que sea transparente el hecho de si controla un sistema virtual o real.

5.2 Trabajo futuro

Dadas las circunstancias que han estado presentes durante la realización del proyecto, han aparecido ideas que no han podido ser implementadas, pero que serían de gran interés para un futuro, ya que mejorarían sustancialmente algunos aspectos del objetivo final.

5.2.1 Trabajo futuro del sistema multiagente

Para una mayor capacidad de análisis del entorno, se podría desarrollar un sistema de aprendizaje automático, para que con un sistema en funcionamiento y con los datos que este provee, se pueda mejorar la toma de decisiones. Así como hacer un análisis estadístico de los datos históricos, para hallar patrones que permitan una mejor adaptación.

Para evitar que el servidor sea un único punto de fallo (aunque los agentes físicos sigan trabajando, se corta toda la comunicación con los usuarios), podría establecerse de forma distribuida, ya sea usando replicación o completa distribución entre los agentes.

Si se dispone del sistema distribuido anterior, podría llevarse al siguiente nivel, en el cual toda la comunicación se cedería a los propios agentes, mejorando su coordinación al informar estos directamente de las condiciones de entorno individuales al resto del sistema, sin necesidad de usar al simulador como intermediario.

5.2.2 Trabajo futuro del sistema global

Sería interesante la realización de un sistema de persistencia de la información que no dependa de un único nodo para evitar convertirlo en un punto único de fallo. Por lo tanto, es interesante realizar una división de la información más acercada a los interesados que van a utilizarla. De forma que quede un diseño del sistema con información local útil replicada para cada Chombo que la necesite, pero con una interfaz global a la que se puedan realizar consultas sobre toda la información. Siendo así una base para un sistema de almacenamiento federado.

Una forma de facilitar el despliegue del sistema completo podría ser el uso de la plataforma Docker, dado que nos permite automatizar la inyección de dependencias del sistema y poner en marcha todos los componentes mediante Docker-compose.

Otro punto de interés para el futuro es dotar de mayor funcionalidad a las bases de datos mediante comunicaciones reactivas a las operaciones de las alteraciones de las entidades. De forma que se puedan advertir de ciertos comportamientos o patrones de alteraciones de información coherente pero incorrecta o que usualmente genere errores. Por ejemplo, se podría aplicar a la

modificación constante de la valoración de algún caso, dado que en teoría los casos rara vez deberían modificarse.

Implementar una gestión de usuarios dentro de la aplicación, de forma que se puedan crear usuarios desde la propia aplicación e identificarse con las credenciales correspondientes. Esto permitiría gestionar un mismo jardín desde distintos dispositivos siempre y cuando se disponga de las credenciales del usuario propietario de la red.

En este punto también podrían incluirse distintos niveles de usuario, ya que el usuario experto puede necesitar mayor control sobre su jardín (por ejemplo, alterando los pesos de los parámetros de las plantas) mientras que un usuario con menos conocimientos no iría más allá de la monitorización de sus chombos.

La posibilidad de parametrizar y gestionar distintas redes de chombos, pues un mismo usuario puede tener la necesidad de gestionar dos jardines independientes el uno del otro y en distintas localizaciones.

Disponer un render en 3 dimensiones que represente de forma más fiel el movimiento de los chombos dentro del jardín, con posibilidad de hacer zum y rotar la vista para obtener una mejor perspectiva.

Dotar a la aplicación de características que mejoren su usabilidad y funcionalidad para así resultar atractiva a un mayor número de usuarios: introducción de gestos para la navegación y realización de acciones, filtros de búsqueda, subir fotos propias de plantas, visionado en *streaming* de la planta a través de su cámara cenital, conexión con redes sociales, multilenguaje, etc.

6. Referencias

1. Delgado Sanchis, A. (2016). Comunicación entre módulos para la construcción de jardines inteligentes. Universidad Politécnica de Valencia, ETSINF.

2. Guzmán Godia, A. (2016). Informatización de un módulo para la construcción de jardines inteligentes. Universidad Politécnica de Valencia, ETSINF.

3. Gil Rodríguez, C. (2016). Desarrollo de aplicación web para gestionar módulos inteligentes para la construcción de jardines. Universidad Politécnica de Valencia, ETSINF.

4. Ferrando Ferragut, L. (2016). Monitorización y control de los módulos de un jardín inteligente. Universidad Politécnica de Valencia, ETSINF.

5. Ferrer Mestre, R. (2017). Desarrollo del gestor de datos y casos para el soporte a un sistema multi-agente para un entorno paisajístico. Universidad Politécnica de Valencia, ETSINF.

6. http://cartago.sourceforge.net. (2010). server support. [online] Available at: http://cartago.sourceforge.net/?page_id=47 [Accessed 10 Jan 2017].

7. http://jason.sourceforge.net server support. [online] Available at: http://jason.sourceforge.net/api/ [Accessed 15 May 2017].

8. http://jacalive.gti-ia.dsic.upv.es/ server support. [online] Available at: http://jacalive.gti-ia.dsic.upv.es/example.php [Accessed 5 Jul 2017].

9. http://jade.tilab.com/ server support. [online] Available at: http://jade.tilab.com/doc/api/index.html [Accessed 20 May 2017].
10. Barella, A. Ricci, O. Boissier, and C. Carrascosa. (2012) MAM5: Multi-Agent Model For Intelligent Virtual Environments. In 10th European Workshop on Multi-Agent Systems. EUMAS. p.16-30.

11. J.A. Rincon, C. Carrascosa and Emilia Garcia. (2014) Developing Intelligent Virtual Environments using MAM5 Meta-Model International Conference on Practical Applications of Agents and Multi-Agent Systems pp. In Press.

12. J.A. Rincon, Emilia Garcia, V. Inglada and C. Carrascosa. (2014) Developing Adaptive Agents Situated in Intelligent Virtual Environments International Conference on Hybrid Artificial Intelligence System pp. In Press.

13. https://docs.oracle.com/javase/8/docs/api/. (2016). server support. [online] Available at: https://docs.oracle.com/javase/7/docs/api/java/awt/event/ActionListener.html [Accessed 15 Jun 2017].

14. http://www.emse.fr/~boissier/enseignement/maop11/doc/cartago-main-api/index.html. server support. [online] Available at: http://www.emse.fr/~boissier/enseignement/maop12/doc/c4jason-api/cartago/invoke_obj.html [Accessed 25 Jun 2017].

15. http://www.telegraph.co.uk. [online] Available at: http://www.telegraph.co.uk/gardening/tools-and-accessories/the-best-apps-to-identify-unknown-plants-and-flowers/ [Accessed 1 Dec 2016].

16. http://blog.parrot.com. (2015) [online] Available at: http://blog.parrot.com/2015/01/05/ces-2015-flower-power-pot-the-smart-pot-that-grows-healthy-plant/ [Accessed 1 Dec 2016].

17. https://www.alibaba.com [online] Available at: https://www.alibaba.com/showroom/smart-flower-pot.html [Accessed 1 Dec 2016].

18. http://es.aliexpress.com [online] Available at: http://es.aliexpress.com/popular/smart-flower-pot.html [Accessed 1 Dec 2016].

19. https://www.google.com [online] Available at: https://www.google.com/patents/US3961444 [Accessed 2 Dec 2016].

20. https://www.google.com [online] Available at: https://www.google.com/patents/US4403443 [Accessed 2 Dec 2016].

21. https://www.google.com [online] Available at: https://www.google.com/patents/US6070359 [Accessed 2 Dec 2016].

22. https://www.google.com [online] Available at: https://www.google.com/patents/US4108350 [Accessed 2 Dec 2016].

23. https://www.google.com [online] Available at: https://www.google.com/patents/US3069807 [Accessed 2 Dec 2016].

24. https://www.google.com [online] Available at: https://www.google.com/patents/US20060272210 [Accessed 2 Dec 2016].

25. https://www.google.com [online] Available at: https://www.google.com/patents/US6243986 [Accessed 2 Dec 2016].

26. http://www.actahort.org [online] Available at: http://www.actahort.org/members/showpdf?booknrarnr=417_4 [Accessed 4 Dec 2016].

27. http://www.fliwer.com [online] Available at: http://www.fliwer.com/# [Accessed 4 Dec 2016].

28. http://delivery.acm.org [online] Available at: http://delivery.acm.org/10.1145/1360000/1357335/p1797-consolvo.pdf?ip=158.42.174.22&id=1357335&acc=ACTIVE%20SERVICE&key=DD1EC5BCF38B3699%2E016407C0B79CBB66%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35&CFID=677700695&CFTOKEN=86003516&__acm__=1475767424_edd7e7a77bbde675bf3f0ef7fc1ec83d [Accessed 4 Dec 2016]

29. Ministerio de cultura y pesca. (2013) Proyecto sigAGROasesor. Available at: http://agroasesor.es/es/ [Accessed 7 Dec 2016]

30. Emmi Pizzella, L. A. (2011). Entorno de simulación para flota de robots orientado a la gestión de malas hierbas en cultivos. Universidad Complutense de Madrid, Máster en Investigación en Informática, Facultad de Informática, Departamento de Ingeniería del Software e Inteligencia Artificial.

31. https://myflowerpower.parrot.com. (2016) [online] Available at: https://myflowerpower.parrot.com/#plantdb [Accessed 10 Dec 2016].

7. Anexo

7.1 Glosario

- **Actuador**: Dispositivo capaz de transformar energía eléctrica en la activación de un proceso con la finalidad de generar un efecto sobre un proceso automatizado.

- **Agente**: Aplicación informática con capacidad para decidir cómo actuar para alcanzar sus objetivos.

- **APP**: Acrónimo de la palabra aplicación, al que se recurre para referirse a la aplicación móvil que se encargará de la monitorización e interacción del sistema.

- **Base de casos**: Conjunto estructurado de información que representa los casos del sistema.

- **Chombo**: Contenedor de una o varias plantas, con sensores, actuadores y capacidad de movimiento, que es controlado por un agente.

- **Fuentes externas**: Origen de los datos que no son obtenidos directamente por el sistema.

- **Fuentes internas**: Componentes del sistema que generan y entregan acceso a información de este.

- **Integración de datos**: Combinación coherente de datos originarios de diferentes fuentes en una vista unificada.

- **Razonamiento basado en casos**: Metodología de la inteligencia artificial con la finalidad de resolver problemas recordando situaciones previas similares y reutilizando la información y el conocimiento sobre esa situación.

- **Sensor**: Dispositivo capaz de detectar magnitudes físicas y transformarlas en valores binarios.

- **Simulador**: Reproducción del sistema en un entorno con unas características específicas.

- **Sistema**: Se usa este término para referirse globalmente a todas las partes que componen el proyecto, desde la APP y el sistema de agentes hasta los sensores y actuadores, pasando por las conexiones y el sistema de persistencia.

- **Sistema de agentes**: Modelo de programación en el cual entidades virtuales (llamadas agentes) toman el papel de los usuarios, tomando control de las acciones necesarias que se deben llevar a cabo para cumplir los objetivos marcados. Pudiendo crear nuevos objetivos y variaciones para cada caso particular, en función del entorno.

- **Sistema gestor de base de casos(SGBC)**: Software utilizada para la extracción e inserción de casos en la base de casos.