Proyecto de SI

SI-31

Miguel Ángel Seara Losada
David Simón Nóvoa
Mauro Zelenka Pedrosa
Luis Fernando Pérez Moure

Introducción

El presente documento describe el desarrollo de un sistema de asistencia domiciliaria basado en agentes inteligentes implementados en la plataforma **Jason**, cuyo objetivo es simular la interacción entre un robot asistencial (enfermera) y un usuario (owner) dentro de un entorno doméstico. Este sistema representa una evolución del ejemplo de robot doméstico trabajado en clase, y se ha rediseñado para cubrir funcionalidades más complejas relacionadas con el ámbito de la salud y el soporte automatizado en el hogar.

El sistema se construye siguiendo una arquitectura **orientada a agentes**, en la que los agentes interactúan con un entorno modelado mediante una cuadrícula y un conjunto de reglas de percepción y acción. El desarrollo se ha realizado de forma incremental, incorporando nuevas capacidades en sucesivas iteraciones a lo largo de varias semanas, lo que ha permitido extender y modularizar progresivamente la funcionalidad del sistema.

El entorno simulado reproduce una vivienda que incluye múltiples habitaciones diferenciadas (cocina, salón, baño, dormitorios, pasillo, etc.) y objetos interactivos como camas, sillas, sofá, mesa, nevera, lavadora y puertas. Esta estructura está modelada a través de la clase `HouseModel`, que extiende `GridWorldModel` y define tanto las dimensiones del entorno como la localización de cada objeto. Además, gestiona el estado del sistema, incluyendo variables clave como la cantidad de medicación disponible, el estado de apertura de la nevera, y si el robot está transportando o no una dosis de medicación.

La clase `HouseView`, basada en `GridWorldView`, se encarga de la representación visual del entorno y de los agentes. Utiliza imágenes personalizadas para representar gráficamente cada objeto del entorno, incluyendo distintos estados (por ejemplo, nevera abierta o cerrada). Esta vista permite observar en tiempo real las acciones de los agentes, facilitando tanto el desarrollo como la validación visual de los comportamientos esperados.

Por otro lado, `HouseEnv` extiende la clase `Environment` de Jason y actúa como interfaz entre el entorno gráfico y los agentes definidos en los archivos `.asl`. Esta clase gestiona las percepciones dinámicas que reciben los agentes según su posición y contexto, tales como `at(owner,sofa)` o `stock(drug,N)`, y permite ejecutar acciones del agente sobre el modelo físico, como `open(fridge)`, `get(drug)` o `move_towards(X)`. También se encarga de informar a los agentes

sobre las habitaciones en las que se encuentran ellos mismos, otros agentes, y los objetos relevantes, mediante percepciones como `atRoom(owner,bedroom1)` o `atRoom(fridge,kitchen)`.
El sistema está diseñado para que ambos agentes, owner y enfermera, puedan **moverse libremente por la casa**, esquivando obstáculos y utilizando puertas como puntos de transición entre habitaciones. La enfermera es capaz de gestionar la medicación siguiendo pautas indicadas por el owner, entregarla cuando corresponde, verificar su consumo y ajustar el stock de medicinas en el almacén. Además, se contempla que el owner pueda modificar dinámicamente sus pautas médicas, así como tomar la medicación por su cuenta, notificándolo posteriormente a la enfermera.

Este enfoque permite experimentar con conceptos fundamentales en sistemas multiagente, como la percepción compartida, la coordinación, la comunicación entre agentes y la planificación reactiva en entornos parcialmente observables. Esta memoria documenta cada uno de los elementos involucrados en el diseño del sistema, las decisiones de implementación adoptadas y la evolución funcional del proyecto, sirviendo como guía para comprender en profundidad el comportamiento de los agentes en este entorno domótico.

Objetivos

El objetivo general de este proyecto es diseñar, implementar y evaluar un sistema multiagente inteligente dentro de un entorno doméstico simulado, capaz de realizar tareas asistenciales relacionadas con la **gestión, entrega y consumo de medicación** en pacientes dependientes. Para alcanzar esta meta, se han definido una serie de objetivos específicos, tanto desde el punto de vista funcional como técnico, que guían el desarrollo de la primera fase del proyecto.

1. Simulación de un entorno doméstico estructurado

El sistema debe modelar un entorno doméstico lo más fiel posible a una vivienda real. Este objetivo se ha alcanzado mediante la implementación de la clase `HouseModel`, que extiende `GridWorldModel` y define una cuadrícula de dimensiones `2*GSize x GSize`. Dentro de este espacio se han instanciado múltiples objetos fijos (sillas, sofá, mesa, camas, puertas, electrodomésticos) y se ha realizado una segmentación del entorno en **habitaciones funcionales**, representadas mediante objetos `Area`:

- `kitchen`, `bath1`, `bedroom1`, etc., cada uno definido por un rango de celdas.
- Cada objeto del entorno se asocia a una habitación específica, lo que permite una contextualización espacial a nivel de percepciones.

El método `getRoom(Location)` permite determinar a qué habitación pertenece cualquier agente u objeto en un momento dado, proporcionando así información espacial rica para la toma de decisiones.

2. Navegación autónoma, consistente y sin colisiones

Tanto el **agente owner** como el **agente enfermera** deben ser capaces de desplazarse libremente por la casa, evitando obstáculos y sin colisionar entre sí. Este comportamiento se gestiona en el método `moveTowards(int Ag, Location dest)` de `HouseModel`, que analiza la viabilidad del movimiento en función de la presencia de objetos mediante el método `canMoveTo(int Ag, int x, int y)`.

La enfermera, por ejemplo, no puede atravesar celdas ocupadas por objetos voluminosos como el sofá, la lavadora o la mesa. Asimismo, se introdujo una **heurística simple de movimiento** con prioridad a ejes `x` e `y` en orden, mejorada para evitar que el agente quede bloqueado. Este movimiento es invocado desde el entorno mediante acciones Jason como `!go(fridge)` o `!move_towards(chair1)` y ejecutado en el método `executeAction` de `HouseEnv`.

3. Integración de un modelo de inventario para medicación

El sistema debe simular un punto de almacenamiento de medicamentos con control de stock. La **nevera** (`lFridge`) actúa como unidad central de almacenamiento, accesible por ambos agentes. Las variables `availableDrugs`, `fridgeOpen` y `carryingDrug` del modelo controlan el estado del inventario y las condiciones de acceso.

Las acciones `open(fridge)`, `get(drug)`, `hand_in(drug)` y `sip(drug)` implementan una **máquina de estados de medicación**, permitiendo:

- Abrir la nevera (condición necesaria para acceder a la medicación).
- Extraer medicación (restringido a si hay disponibilidad y si el robot no transporta ya una dosis).
- Entregar la medicación al owner.
- Simular la ingesta de la medicación por parte del owner (`sipCount > 0`).

Estas operaciones se reflejan también visualmente mediante imágenes (`openNevera.png`, `beerBot.png`, etc.) en `HouseView`.

4. Coordinación y percepción distribuida entre agentes

Una de las metas clave del sistema es la **coordinación efectiva entre agentes**, basada en un modelo de **percepción distribuida y contextual**. La clase HouseEnv es responsable de mantener y actualizar estas percepciones, que incluyen:

- Posición relativa (at(enfermera,owner), at(owner,sofa), etc.).
- Ubicación de objetos (atRoom(fridge,kitchen)).
- Ubicación de agentes (atRoom(owner,bedroom1)).
- Estado del inventario (stock(drug,N)).
- Confirmación del consumo (has(owner,drug)).

Estas percepciones permiten a los agentes establecer **objetivos contextuales**, como !go(owner) si están lejos, !checkStock si no hay medicación, o !deliver(drug) si se requiere una entrega. La coordinación se logra mediante una combinación de **planificación reactiva** (basada en eventos) y **comunicación explícita** entre agentes Jason.

5. Entrega inteligente y validación del cumplimiento terapéutico

Otro objetivo fundamental es la entrega puntual y válida de la medicación al owner. La enfermera debe detectar si se encuentra próxima al owner (`distance == 1`), si tiene una dosis (`carryingDrug == true`) y si el owner la ha consumido (`sipCount` decreciente). Esta lógica es fundamental para asegurar que el agente robot no repita entregas innecesarias y que el sistema refleje con fidelidad el estado terapéutico del paciente.

Además, el sistema permite que el owner acceda directamente al stock, tome su medicación y notifique al robot mediante una percepción intencionada (`has(owner,drug)`). Este mecanismo refuerza la **autonomía del owner** en la simulación y la capacidad del robot para adaptarse a situaciones no planificadas.

6. Visualización en tiempo real y coherencia perceptiva

La clase HouseView permite observar el estado completo del sistema de forma **sincrónica con el modelo lógico**. A través del uso de imágenes representativas y posicionamiento en celdas, se puede verificar visualmente:

- Posiciones actuales de ambos agentes.
- Estado de puertas (abiertas o cerradas).
- Interacción con objetos (por ejemplo, el robot cambia de imagen si transporta medicación).
- Disponibilidad de stock (Fr (N) se muestra sobre la nevera).

Esto refuerza la trazabilidad del comportamiento y permite validar experimentalmente la consistencia entre lo que los agentes perciben y lo que el entorno representa.

7. Escalabilidad y adaptabilidad del sistema

Finalmente, el sistema está diseñado para ser **escalable y extensible**. La estructura modular de `HouseModel`, `HouseEnv` y `HouseView` permite la incorporación futura de:

- Nuevos agentes (por ejemplo, visitas o cuidadores).
- Nuevos objetos o sensores.
- Pautas de medicación dinámicas (a través de creencias en Jason).
- Gestión avanzada del plan de tratamientos, con prioridades o conflictos.

Esta capacidad de adaptación es clave para la segunda fase del proyecto, en la que se buscará incrementar la complejidad del sistema, dotarlo de inteligencia proactiva, y simular escenarios más realistas en los que los agentes respondan a cambios dinámicos del entorno o del estado del usuario.

Resumen de la solución

La solución desarrollada en esta primera fase del proyecto consiste en una simulación multiagente de asistencia domiciliaria en la que dos agentes interactúan en un entorno doméstico modelado como una cuadrícula dividida en habitaciones funcionales. El entorno está equipado con mobiliario, puertas y puntos de almacenamiento, y se gestiona mediante una arquitectura Jason sobre un modelo visual e interactivo.

Pauta médica y gestión de medicación

El agente owner posee una pauta de medicación compuesta por 5 principios activos, que deben ser administrados en horarios determinados a lo largo del día. Estas pautas están codificadas como creencias del tipo `medician(Medicamento, Hora)`, y son comunicadas inicialmente por el owner a la enfermera:

- `paracetamol`: a las 1h, 9h y 20h
- `ibuprofeno`: a las 11h
- `amoxicilina`: a las 15h
- `omeprazol`: a las 5h, 18h y 23h
- `loratadina`: a las 3h

Tanto el owner como la enfermera pueden acceder al punto de almacenamiento (la **nevera**) para obtener la medicación. Una vez obtenida, la enfermera se desplaza a la localización del owner para entregarla mediante la acción `hand_in(drug)`. El owner puede también tomar medicación por iniciativa propia y debe notificarlo explícitamente al robot con la percepción `has(owner,drug)`, que es gestionada por `HouseEnv`.

El modelo (`HouseModel`) incluye variables que controlan el stock (`availableDrugs`), el estado de la nevera (`fridgeOpen`), si el robot está cargando medicación (`carryingDrug`), y el seguimiento del consumo (`sipCount`), proporcionando una lógica completa de ciclo de entrega y validación.

Navegación autónoma con algoritmo A*

Uno de los componentes fundamentales para el correcto funcionamiento del sistema es el método de desplazamiento de los agentes dentro del entorno. Para lograr una movilidad autónoma, eficiente y realista —especialmente por parte de la enfermera— se ha diseñado una versión avanzada del método `moveTowards(int Ag, Location dest)` mediante el uso del **algoritmo A***.

El algoritmo A* permite calcular rutas óptimas dentro de la cuadrícula del modelo, evitando obstáculos fijos como muebles, paredes o electrodomésticos, y atravesando puertas según sea necesario. Esta implementación mejora sustancialmente el comportamiento del robot frente a la versión original, que se basaba en movimientos deterministas por ejes. En particular, A* permite:

- **Planificación de rutas mínimas** desde la posición actual al destino.
- **Evitar colisiones** con objetos no atravesables (sofás, camas, etc.).
- **Aprovechar los puntos de transición** entre habitaciones (puertas) de forma eficiente.
- **Adaptación dinámica** a la posición del owner, incluso si este se está desplazando.

Internamente, la búsqueda A* trabaja sobre nodos que representan celdas del mapa, usando una función heurística basada en la distancia Manhattan. El algoritmo mantiene una cola de prioridad que ordena los nodos abiertos según su coste estimado total `f(n) = g(n) + h(n)` y permite reconstruir la ruta una vez alcanzado el destino. La ruta se almacena como una lista de ubicaciones, que los agentes siguen paso a paso.

Esta estrategia hace posible que el agente enfermera pueda buscar y localizar al owner en cualquier punto del entorno, cumpliendo con el requisito del proyecto de recorrer libremente el entorno **sin quedar atascado**, evitando obstáculos, y accediendo a habitaciones remotas para cumplir su rol asistencial.

Coordinación agente-agente y percepción contextual

La enfermera y el owner comparten información mediante un sistema de percepciones dinámicas gestionado por `HouseEnv`, que incluye percepciones como `atRoom(owner,bedroom2)` o `stock(drug, N)`, además de ubicaciones relativas (`at(enfermera,owner)`) y de objetos (`atRoom(fridge,kitchen)`).

Esto les permite ejecutar planes contextuales de forma reactiva y sincronizar tareas como entrega, consumo y actualización de stock.

La interfaz visual (`HouseView`) refleja en tiempo real el estado del entorno, el inventario de medicinas y la posición de los agentes, utilizando imágenes representativas personalizadas que permiten comprobar visualmente el comportamiento del sistema.

Arquitectura

La arquitectura del sistema se ha diseñado siguiendo el patrón de diseño **Modelo–Vista–Controlador (MVC)**, que permite estructurar el proyecto en tres componentes claramente diferenciados: la lógica del entorno (modelo), su representación visual (vista) y la coordinación entre agentes y entorno (controlador). Esta separación favorece la escalabilidad del sistema y facilita tanto su mantenimiento como su futura extensión.

En el contexto de este proyecto, el **modelo** es responsable de representar el estado interno del entorno y las reglas que rigen su evolución; la **vista** ofrece una representación gráfica sincronizada de dicho estado; y el **controlador** intermedia entre los agentes definidos en Jason y el modelo, gestionando las acciones solicitadas y actualizando las percepciones correspondientes.

El modelo (`HouseModel.java`)

El componente que representa el modelo del sistema es la clase `HouseModel`, que extiende `GridWorldModel` de Jason. Esta clase contiene toda la **lógica física y estructural del entorno**: define el tamaño de la cuadrícula, la distribución de las habitaciones (mediante objetos `Area`), y la ubicación de los elementos que componen la vivienda (camas, sillas, sofá, nevera, etc.).

Además, `HouseModel` gestiona variables clave que reflejan el estado interno del entorno, como la cantidad de medicación disponible (`availableDrugs`), si el robot está transportando una dosis (`carryingDrug`) o si la nevera está abierta (`fridgeOpen`). Las acciones que los agentes pueden ejecutar —como `openFridge()`, `getDrug()`, `handInDrug()`, `sipDrug()` o `moveTowards()`— están implementadas en esta clase.

Una característica especialmente relevante de esta implementación es la preparación del método `moveTowards` para incorporar el **algoritmo A***, que permitirá una navegación óptima entre habitaciones, evitando obstáculos y eligiendo trayectorias más eficientes. Este diseño refuerza el principio de responsabilidad única del modelo: se encarga exclusivamente de reflejar y modificar el estado del mundo simulado.

La vista (`HouseView.java`)

La clase `HouseView` representa la vista del sistema, y está encargada de ofrecer una **visualización en tiempo real del estado del modelo**. Esta clase extiende

`GridWorldView` y utiliza recursos gráficos personalizados para representar cada objeto del entorno (por ejemplo, `sofa.png`, `bot.png`, `openDoor2.png`, etc.), así como para mostrar las posiciones de los agentes y su estado (si están sentados, si transportan medicación, etc.).

`HouseView` actualiza dinámicamente la visualización cada vez que se produce un cambio en el modelo. Gracias a esta interfaz, es posible verificar visualmente el comportamiento del sistema y depurar errores funcionales observando directamente la evolución de la simulación. También permite comprobar visualmente eventos como la apertura de la nevera, el consumo de medicación, o la entrega de una dosis al owner.

El controlador (`HouseEnv.java`)

El componente controlador de esta arquitectura está representado por la clase `HouseEnv`, que extiende `Environment`, la interfaz que Jason proporciona para conectar el mundo físico con los agentes BDI definidos en archivos `.asl`. Esta clase es responsable de **interpretar las acciones emitidas por los agentes**, traducirlas en operaciones sobre el modelo (`HouseModel`) y **generar percepciones** que serán recibidas por los agentes en función del nuevo estado del entorno.

Por ejemplo, si la enfermera emite la acción `move_towards(owner)`, el controlador interpreta esta acción, invoca el método `moveTowards()` del modelo y luego actualiza las percepciones de los agentes para reflejar sus nuevas posiciones relativas (`at(enfermera,owner)`), su localización contextual (`atRoom(...)`), o el estado del inventario (`stock(drug,N)`). El controlador también se encarga de detectar situaciones como el cruce de puertas o la cercanía entre agentes, activando las percepciones correspondientes.

Al mantener separadas la lógica del entorno (modelo), la visualización (vista) y la coordinación con los agentes (controlador), esta arquitectura permite realizar mejoras o modificaciones en un componente sin afectar los demás. Por ejemplo, se puede modificar el algoritmo de navegación por A* en el modelo sin necesidad de alterar ni la vista ni el comportamiento de los agentes.

Este enfoque también hace posible incorporar nuevas funcionalidades en fases posteriores del proyecto, como alarmas de medicación, sensores simulados, planificación temporal más compleja o incluso la inclusión de agentes adicionales, sin comprometer la estabilidad del sistema actual.

Tecnologías e integración de productos de terceros

El desarrollo del sistema se ha basado exclusivamente en el uso de **Jason** y **Java**, dos tecnologías complementarias que permiten modelar agentes inteligentes e integrarlos con un entorno simulado programado de forma estructurada y eficiente. No se han utilizado librerías externas adicionales ni motores de simulación de terceros, lo cual garantiza un mayor control sobre la lógica de funcionamiento del sistema y favorece una comprensión más profunda de cada uno de sus componentes.

Jason: plataforma y lenguaje multiagente BDI

El comportamiento de los agentes owner y enfermera ha sido implementado utilizando **Jason**, una plataforma para el desarrollo de sistemas multiagente basada en el lenguaje AgentSpeak(L). Jason se fundamenta en el modelo BDI (**Belief-Desire-Intention**), lo que permite representar agentes con capacidades reactivas y deliberativas.

Mediante Jason, los agentes:

- **Perciben** su entorno a través de percepciones generadas por el entorno físico (HouseEnv.java).
- **Toman decisiones** en función de sus creencias (beliefs) y eventos internos o externos.
- **Ejecutan planes** codificados como reglas que definen cómo responder ante determinadas situaciones.
- **Se comunican** entre sí para intercambiar información, como la pauta de medicación.

Este marco de desarrollo permite una implementación declarativa y modular del comportamiento de los agentes, facilitando su comprensión y modificación.

Java: implementación del entorno y simulación visual

El entorno físico del sistema ha sido implementado íntegramente en **Java**, utilizando las clases proporcionadas por la extensión grid de Jason. Java ha permitido modelar de manera precisa:

- El **modelo del entorno** (HouseModel), incluyendo la lógica de movimiento, interacción con objetos, y almacenamiento de medicamentos.

- La **visualización en tiempo real** (HouseView), basada en AWT y Swing, que permite representar gráficamente habitaciones, objetos y agentes.

- El **controlador del entorno** (HouseEnv), que intermedia entre los planes de los agentes y el estado del modelo.

Java ofrece una base sólida para la manipulación de estructuras de datos, el control del flujo de ejecución, y la integración de recursos gráficos, lo cual ha sido clave para la representación visual y funcional del entorno domótico.

Especificación y análisis de requisitos

El desarrollo del sistema parte de un conjunto de requisitos derivados del enunciado del proyecto, centrados en la simulación de un entorno domótico asistencial mediante un sistema multiagente. Estos requisitos se han clasificado en dos categorías: **funcionales**, que definen lo que el sistema debe hacer; y **no funcionales**, que establecen condiciones sobre cómo debe hacerlo.

Requisitos funcionales

Los requisitos funcionales representan el conjunto de comportamientos esperados del sistema. En esta primera fase del proyecto, se han identificado y abordado los siguientes:

1. **Gestión de medicación pautada por el agente owner**
   El agente owner debe contar con una pauta médica definida, compuesta por múltiples medicamentos y horarios. Esta pauta debe ser comunicada al agente enfermera al inicio de la simulación.

2. **Movilidad autónoma del owner y la enfermera**
   Ambos agentes deben poder desplazarse libremente por la vivienda, evitando obstáculos y recorriendo distintas habitaciones según sus objetivos.

3. **Acceso al almacén de medicamentos (la nevera)**
   La medicación debe estar almacenada en un punto fijo accesible para ambos agentes, que puede ser abierto y cerrado según sea necesario.

4. **Entrega de medicación por parte de la enfermera**
   La enfermera debe buscar al owner y entregarle la medicación en el horario indicado por la pauta médica.

5. **Autoadministración de medicación por el owner**
   El owner debe poder acceder a la nevera y tomar la medicación por cuenta propia, informando a la enfermera del hecho posteriormente.

6. **Notificación de consumo al agente enfermera**
   Si el owner ha tomado la medicación, debe comunicarlo explícitamente a la enfermera, que a su vez debe verificar dicha acción.

7. **Actualización dinámica de la pauta médica**
   La pauta de medicamentos del owner debe poder modificarse durante la ejecución: se pueden añadir, eliminar o reprogramar dosis.

8. **Representación gráfica del entorno y sus elementos**
   El sistema debe ofrecer una visualización clara y precisa del entorno, los agentes, los objetos y los eventos que ocurren en la simulación.

Requisitos no funcionales

Los requisitos no funcionales se refieren a propiedades deseables del sistema más allá de sus funciones básicas:

1. **Modularidad y mantenibilidad**
   El sistema debe estar estructurado de forma modular, permitiendo modificar o ampliar cada componente (modelo, vista, controlador) de forma independiente. Esto se logra aplicando el patrón Modelo–Vista–Controlador (MVC).

2. **Eficiencia en la navegación del entorno**
   El sistema debe permitir desplazamientos óptimos entre habitaciones, lo que motiva la integración del algoritmo A* para el cálculo de rutas en moveTowards().

3. **Reactividad y consistencia perceptiva**
   Los agentes deben recibir percepciones actualizadas de forma coherente con el estado real del entorno, facilitando decisiones correctas basadas en información fiable.

4. **Claridad en la visualización del estado**
   Los usuarios deben poder comprender el estado actual de la simulación mediante representaciones gráficas simples pero expresivas, como imágenes, colores y etiquetas.

5. **Independencia de tecnologías externas**
   El sistema debe funcionar exclusivamente con tecnologías académicas (Jason y Java), evitando dependencias externas innecesarias.

Análisis de cobertura de requisitos

Todos los requisitos funcionales definidos para la primera fase han sido implementados con éxito. La gestión de la pauta médica se realiza mediante creencias Jason (medician(...)) y se integra con los planes de acción de los agentes. La movilidad libre se encuentra resuelta dentro de HouseModel y se encuentra en proceso de mejora con A*. Las acciones de interacción (get(drug), hand_in(drug), sip(drug)) están correctamente implementadas, y la comunicación entre agentes

está contemplada tanto en la transferencia inicial de información como en la retroalimentación tras la toma de medicación.

Asimismo, los requisitos no funcionales se han respetado mediante el diseño modular basado en MVC, la preparación del sistema para el uso de A*, la actualización constante de percepciones y una interfaz gráfica adaptada al dominio.

Diagramas

Diagrama entorno.

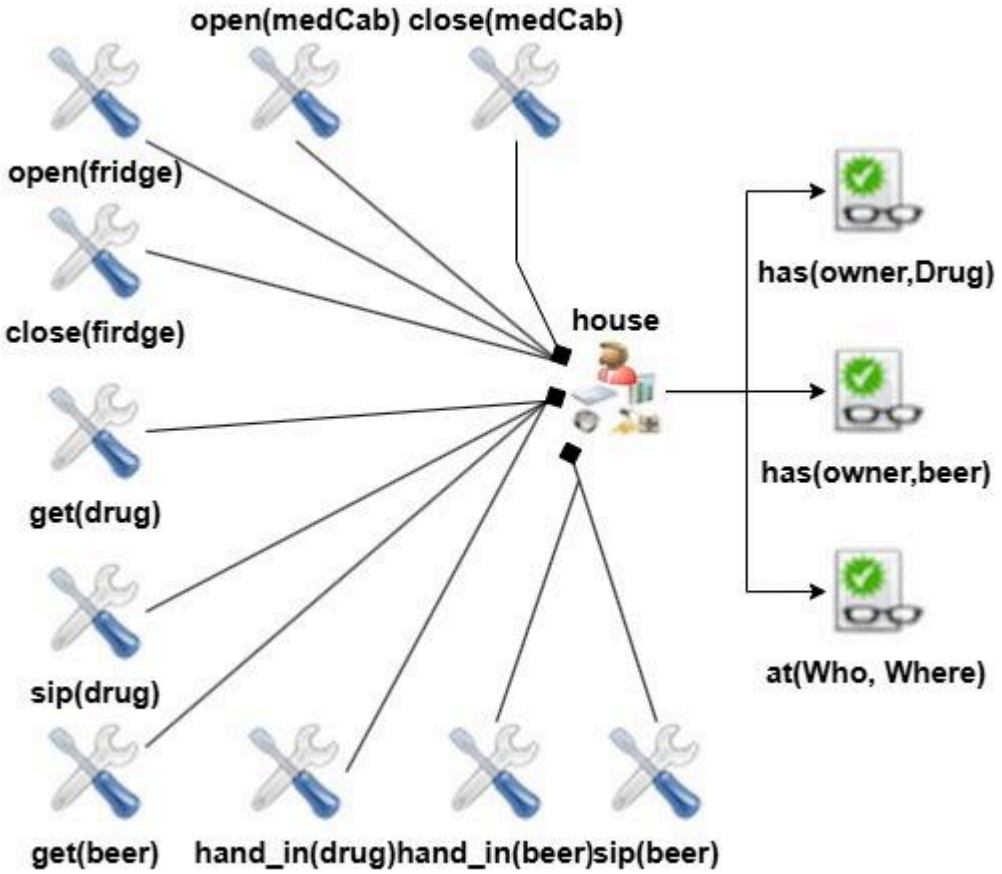


Diagrama organización para robot doméstico

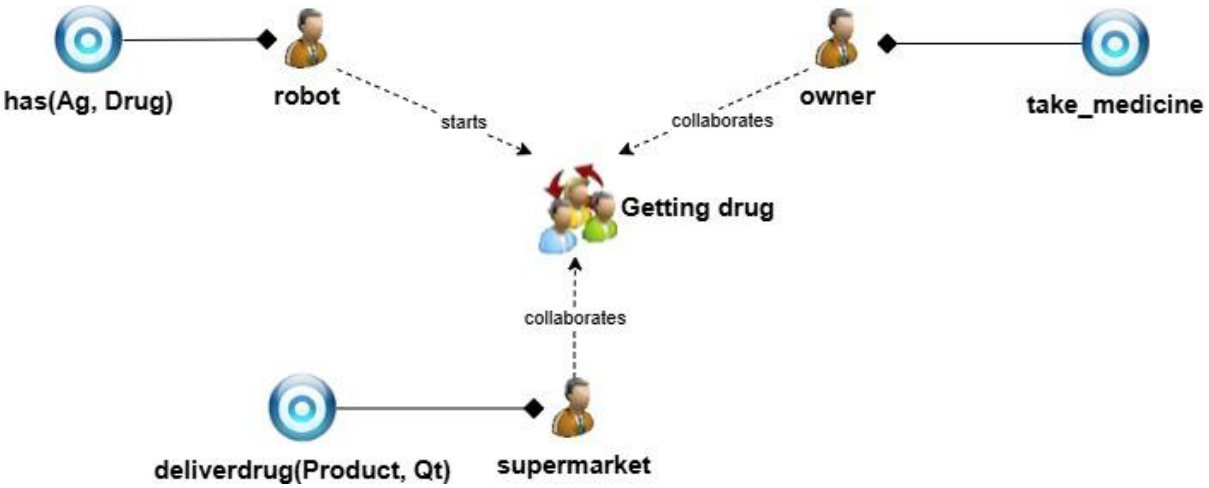


Diagrama de objetivos ideal para robot doméstico

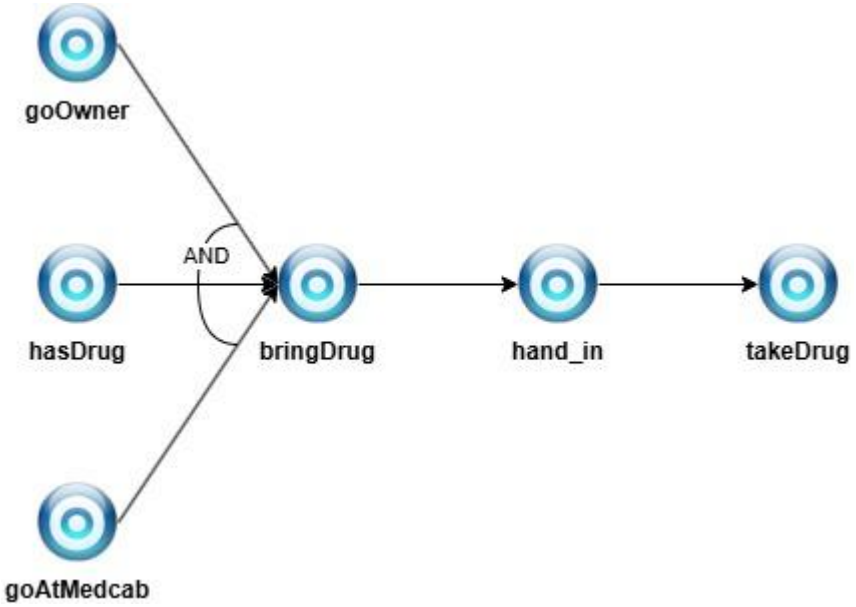


Diagrama de objetivos ideal para robot doméstico

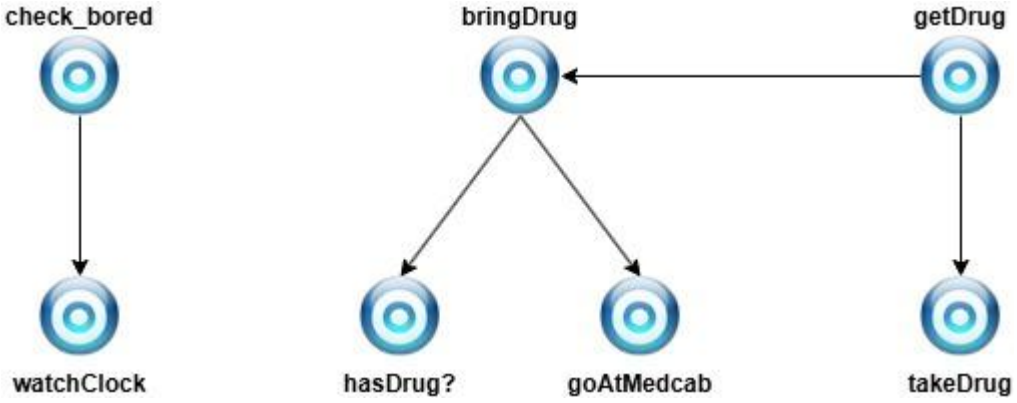
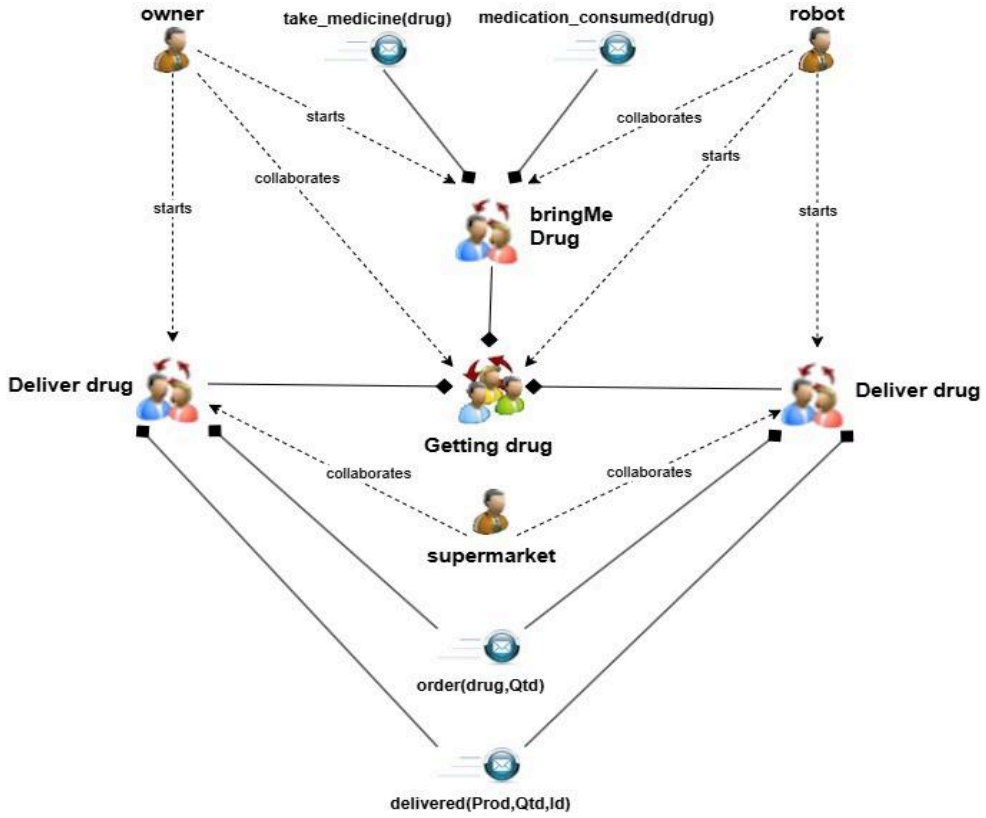


Diagrama de interacción para robot doméstico



Owner

Diagrama agente.

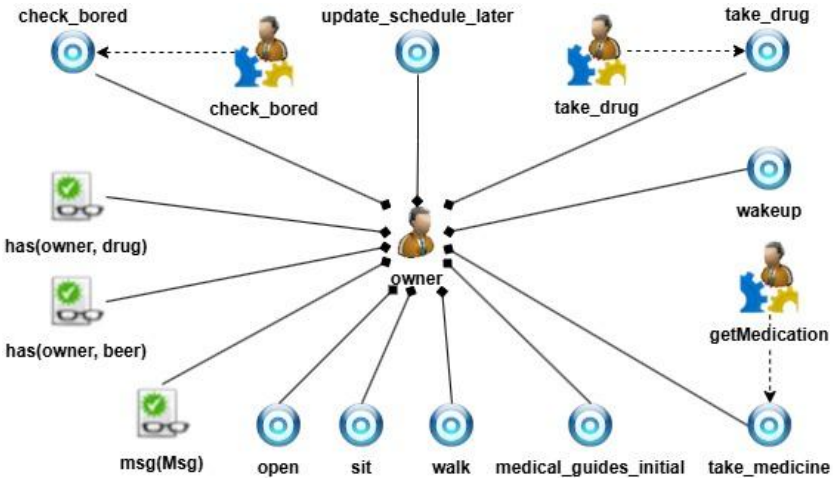
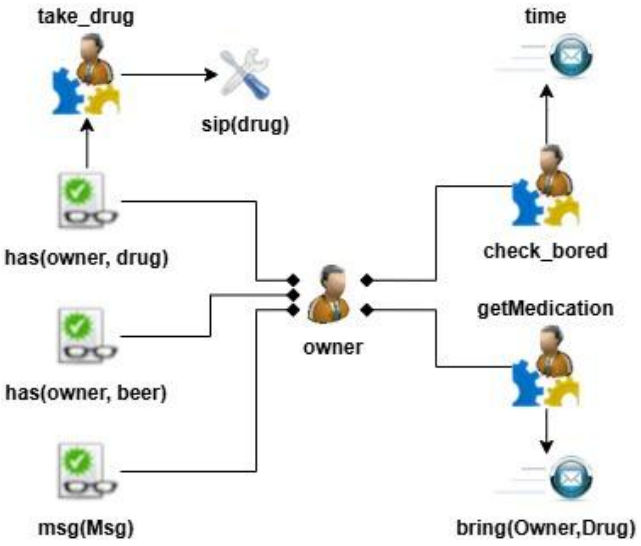

check_bored
check_bored
update_schedule_later
take_drug
take_drug
wakeup
has(owner, drug)
owner
has(owner, beer)
getMedication
msg(Msg)
open
sit
walk
medical_guides_initial
take_medicine

Diagrama de tareas.



Supermarket

Diagrama agente.

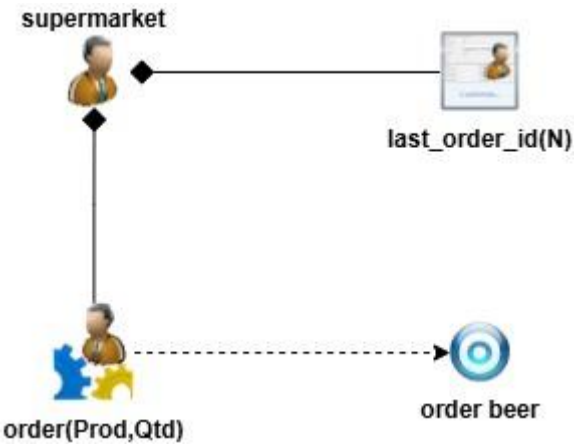
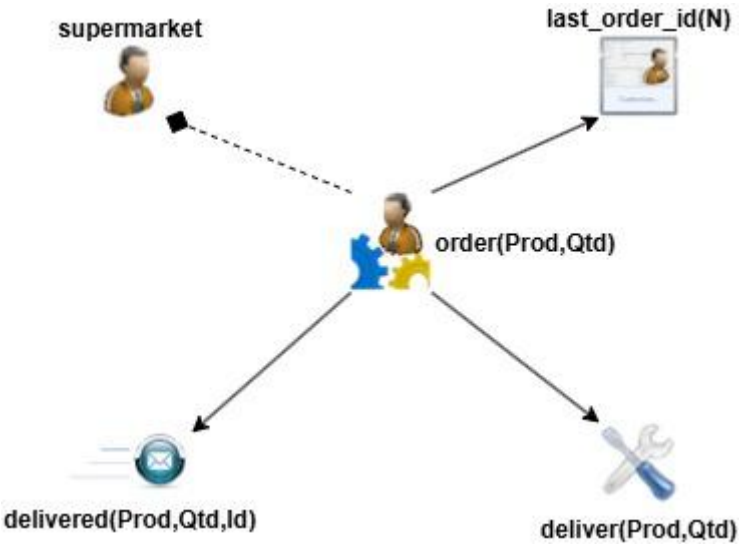


Diagrama tareas.



Robot enfermera.

Diagrama agente.

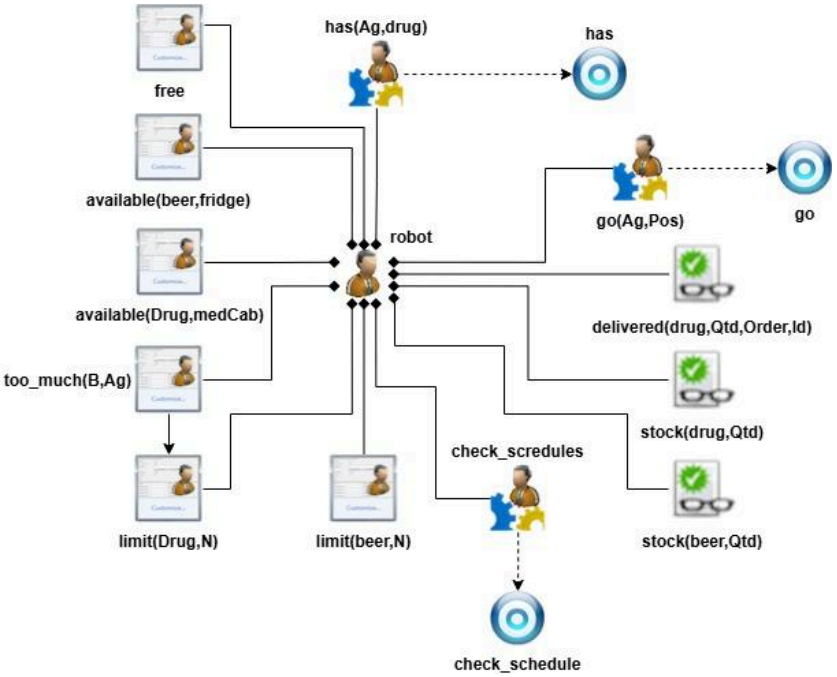
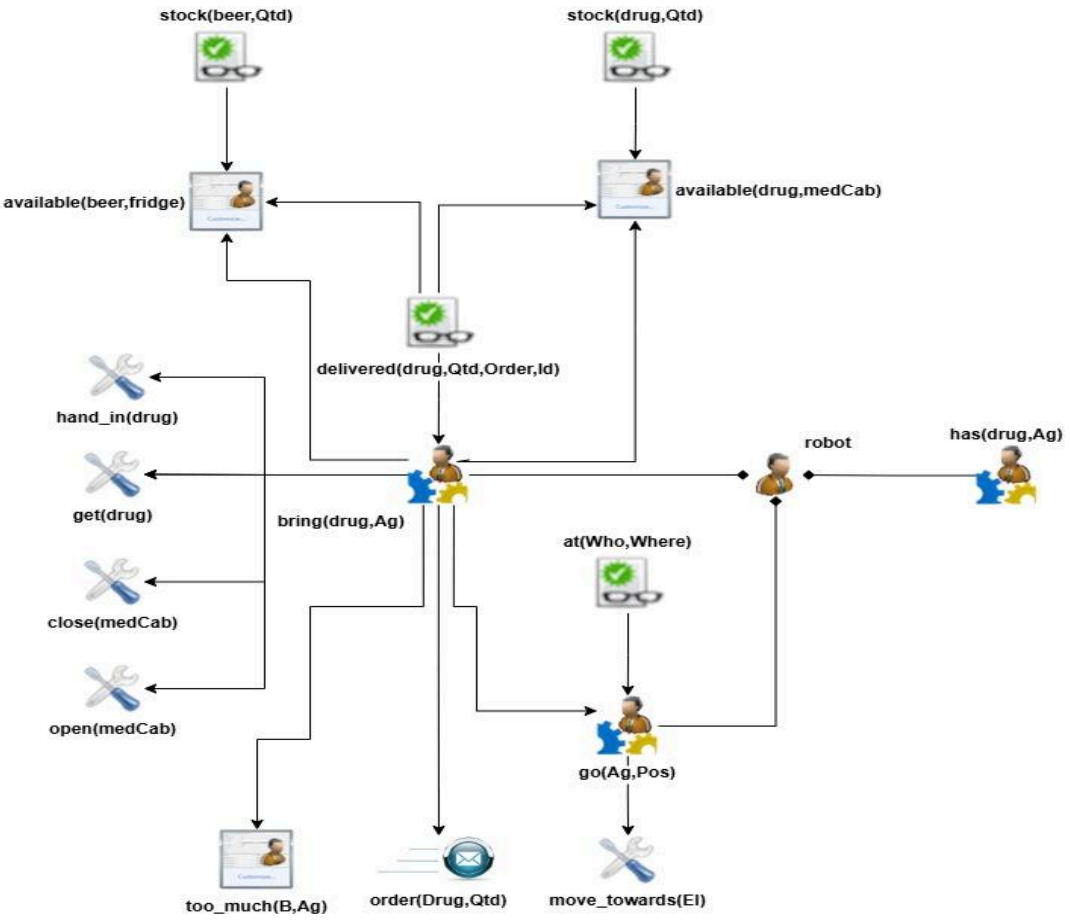


Diagrama de tareas



Conclusión

En esta primera fase del proyecto se ha logrado transformar el ejemplo original de robot doméstico en un sistema domótico asistencial multiagente, capaz de simular la gestión de una pauta médica en un entorno estructurado y dinámico. El sistema desarrollado permite observar la interacción entre un agente paciente (`owner`) y un agente robot (`enfermera`), dentro de una vivienda compuesta por habitaciones funcionales, objetos interactivos y restricciones físicas realistas.

A nivel funcional, se han implementado con éxito todos los requisitos definidos en el enunciado: el owner dispone de una pauta médica compuesta por múltiples medicamentos y horarios, la cual es transmitida inicialmente a la enfermera. Ambos agentes pueden desplazarse por el entorno sorteando obstáculos, interactuar con objetos como la nevera o las sillas, y ejecutar acciones contextuales para la administración de la medicación. El sistema contempla tanto la entrega por parte del robot como la autoadministración por parte del paciente, incorporando mecanismos de notificación y validación del consumo.

Desde el punto de vista técnico, el proyecto se ha estructurado siguiendo el patrón **Modelo–Vista–Controlador (MVC)**, lo cual ha permitido mantener una separación clara de responsabilidades. El modelo (`HouseModel`) encapsula el estado y la lógica del entorno, la vista (`HouseView`) ofrece una representación visual intuitiva y en tiempo real, y el controlador (`HouseEnv`) intermedia entre el entorno físico y los agentes definidos en Jason. Asimismo, se ha preparado la infraestructura para integrar un algoritmo de búsqueda heurística A* que mejore la navegación de los agentes, reforzando la autonomía y eficiencia del sistema.

El uso de **Jason** como plataforma multiagente, en combinación con **Java** para la implementación del entorno, ha demostrado ser una solución eficaz para simular sistemas inteligentes con capacidades reactivas, deliberativas y colaborativas. Gracias a esta base, el sistema queda preparado para futuras fases en las que se podrán incorporar mejoras como la planificación temporal compleja, alarmas de medicación, incorporación de nuevos agentes o reglas de prioridad terapéutica.

En definitiva, esta fase ha permitido sentar las bases de un sistema de asistencia inteligente que combina inteligencia artificial distribuida con simulación visual, cumpliendo los objetivos marcados y abriendo un abanico de posibilidades para su evolución en etapas posteriores.

Bibliografía

Bordini, R., Hübner, J., & Wooldridge, M. (2007). *Programming Multi-Agent Systems in AgentSpeak Using Jason*. John Wiley & Sons.
Obra de referencia para el desarrollo de sistemas multiagente con Jason. Aporta los fundamentos del modelo BDI y del lenguaje AgentSpeak(L).

MOOVI – Plataforma de docencia virtual de la Universidad de Vigo
Plataforma utilizada para el acceso al enunciado del proyecto, documentación de apoyo, recursos docentes y entregas.

OpenAI. *ChatGPT*. https://chat.openai.com
Herramienta de asistencia basada en inteligencia artificial utilizada para apoyar la redacción técnica, explicación de código, análisis de requisitos y estructuración de la memoria.

Google DeepMind. *Gemini*. https://deepmind.google/technologies/gemini
Utilizado puntualmente como modelo de lenguaje alternativo para contraste de ideas y verificación de enfoques técnicos durante la fase de planificación del proyecto.

Oracle Java Documentation. *Java Platform, Standard Edition 8 Documentation*.
https://docs.oracle.com/javase/8/docs/
Fuente oficial para la consulta de documentación sobre librerías estándar de Java utilizadas en la implementación del entorno simulado.

Red Blob Games. (2014). *Introduction to the A* Algorithm*.
https://www.redblobgames.com/pathfinding/a-star/introduction.html
Recurso visual y técnico de alta calidad para comprender y adaptar el algoritmo de búsqueda A* en entornos discretos como el simulado en este proyecto.

Russell, S., & Norvig, P. (2016). *Artificial Intelligence: A Modern Approach* (3rd ed.). Pearson.
Texto académico clásico que aborda los fundamentos de la inteligencia artificial, incluidos los modelos de agentes, heurísticas y algoritmos de planificación.